Implementing audio feature extraction in live electronic music

Jamie Bullock

Birmingham Conservatoire

Birmingham City University

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

October 25, 2008

For Harry and Jake.

Acknowledgements

I would like to thank my supervisor, Lamberto Coccioli for changing the way I think about software design. Thanks also to Simon Hall for invaluable support and guidance throughout my research and to Peter Johnson for supporting us in this crazy artform we call live electronic music. Thanks to Rivka Golani for input into *Fission*, and to Larissa Brown and David Matheu for really 'going for it' in the performance. Thanks to Katharine Lam for a beautiful performance of *Variations*.

Many thanks to Miller Puckette for writing Pure Data, one of the most concise pieces of software ever developed, and the Pure Data community for continuing to prove its value. Thanks to Richard Stallman for the GPL license, the single most intelligent contribution to computing in the 20th century, and to Linus Torvalds for adopting it for Linux. Thanks to Ichiro Fujinaga, Tristan Jehan and Tae Hong Park whose seminal work started me thinking about the potential of audio feature extraction in live electronics. Thanks to the team at RootDown FM. I couldn't have survived 'the writeup' without a soundtrack of Jazz, Funk, Hip-Hop, Soul, Latin, Reggae, Afrobeat, Leftfield, Downtempo and Beats! Thanks to Aki Shiramizu, Martin Robinson, Andrew Deakin and Jonty Harrison for original and influential thinking, and to Henrik Frisk for helping me realise the originality of my own work. Thanks to James Stockley for 'beer and scrubs', Paul Broaderick and Bev Tagg for great friendship, and the team at Birmingham Women's hospital Neo-Natal unit for giving my son medical care whilst I wrote my dissertation in the 'parents room'.

Thanks to my family for teaching me to believe I could achieve anything, and just 'being there' when times were bad. Most of all I would like to thank my wife Cheryl, the love of my life.

Related Publications

Chapter 2 may include fragments of text from Bullock (2007).

Abstract

Music with live electronics involves capturing an acoustic input, converting it to an electrical signal, processing it electronically and converting it back to an acoustic waveform through a loudspeaker. The electronic processing is usually controlled during performance through human interaction with potentiometers, switches, sensors and other tactile controllers. These tangible interfaces, when operated by a technical assistant or dedicated electronics performer can be effective for controlling multiple processing parameters. However, when a composer wishes to delegate control over the electronics to an (acoustic) instrumental performer, physical interfaces can sometimes be problematic. Performers who are unfamiliar with electronics technology, must learn to operate and interact effectively with the interfaces provided. The operation of the technology is sometimes unintuitive, and fits uncomfortably with the performer's learned approach to her instrument, creating uncertainty for both performer and audience. The presence of switches or sensors on and around the instrumental performer begs the questions: how should I interact with this and is it working correctly?

In this thesis I propose an alternative to the physical control paradigm, whereby features derived from the sound produced by the acoustic instrument itself are used as a control source. This approach removes the potential for performer anxiety posed by tangible interfaces and allows the performer to focus on instrumental sound production and the effect this has on the electronic processing. A number of experiments will be conducted through a reciprocal process of composition, performance and software development in order to evaluate a range of methods for instrumental interaction with electronics through sonic change. The focus will be on the use of 'low level' audio features including, but not limited to, fundamental frequency,

amplitude, brightness and noise content. To facilitate these experiments, a number of pieces of software for audio feature extraction and visualisation will be developed and tested, the final aim being that this software will be publically released for download and usable in a range of audio feature extraction contexts.

In the conclusion, I will propose a new approach to working with audio feature extraction in the context of live electronic music. This approach will combine the audio feature extraction and visualisation techniques discussed and evaluated in previous chapters. A new piece of software will be presented in the form of a graphical user interface for perfomers to work interactively using sound as an expressive control source. Conclusions will also be drawn about the methdology employed during this research, with particular focus on the relationship between composition, 'do-it-yourself' live electronics and software development as research process.

Contents

List of Figures

Epigraph

An act of bravery was required to break out of tonality's coffin earlier in this century. The rule-inventing and rule-breaking of that era unblocked new musical thoughts. Today, electronics require similar courage to approach and offer the same liberation. Atonalism also offered a distraction; suddenly, in order to be a composer one had to study mathematics. An equivalent distraction value comes today out of using electronics, particularly live electronics. Fear and distraction are an important part of electronic music today. Perhaps in the future the computer will be as easy to configure and use, and as reliable, as a piano is now. If this happens, composers and performers will seek new sources of uncertainty.

(Puckette and Settl, 1993)

CHAPTER 1

Introduction

Already very early I thought of a microphone as a musical instrument, like a bow or like a percussion instrument, whatever you use.

— Karlheinz Stockhausen

1.1 Research Context

Before proceeding with an outline of research questions and methodology, it is important that some contextual background is given in order to orient the reader and clarify the aims and scope of the research.

1.1.1 Integra

This research has been conducted in the broader context of Integra, a project with which I have been involved throughout the course of my Ph. D. as both researcher and software engineer. Many of the themes of this thesis have developed in parallel with the evolution of Integra.

Integra is an international project led by Birmingham Conservatoire in the UK and supported by the Culture programme of the European Union. It has a wide range of objectives including the development of a new software environment to make music with live electronics, and the modernisation of works that use old technology. The

project members are comprised of researchers based at universities and research centres, and performers, some of whom come from established contemporary music ensembles. Project outcomes include the production of new works, concerts, workshops, software releases, knowledge transfer and a database of new and 'modernised' repertoire created using the Integra:Live software[1].

The common theme that brings these diverse elements of the Integra project together is the notion of 'fusing music and technology'[2]. This is embodied in the workflow of the project, which aims to create synergy between scientific research and artistic practice, and proceeds as shown below.

- Integra:Live software for live electronics composition and performance is developed and released
- Composers are commissioned to write pieces for new music ensembles and live electronics using Integra:Live
- New software modules are developed through collaboration between composers, research centres and performers
- New modules are incorporated into Integra:Live software as reusable components
- The commissioned works are performed in concert using Integra:Live with the new modules incorporated

This concept of reusability and the reciprocal relationship between artistic ideas and technical innovation is central to Integra and also to my own research. The desire to produce humane, user-friendly, generalised (not work-specific) software for live electronics is of foremost importance. It is for this reason that my thesis focuses on the practical *implementation* of audio feature extraction, using my own musical practice as a test bed. The key, as in Integra is to establish a process of software development and user testing in real composition and performance situations, resulting in a sustainable software product.

[1] `http://www.integralive.org`

[2] Integra website strapline at `http://www.integralive.org`, accessed 7 June 2008

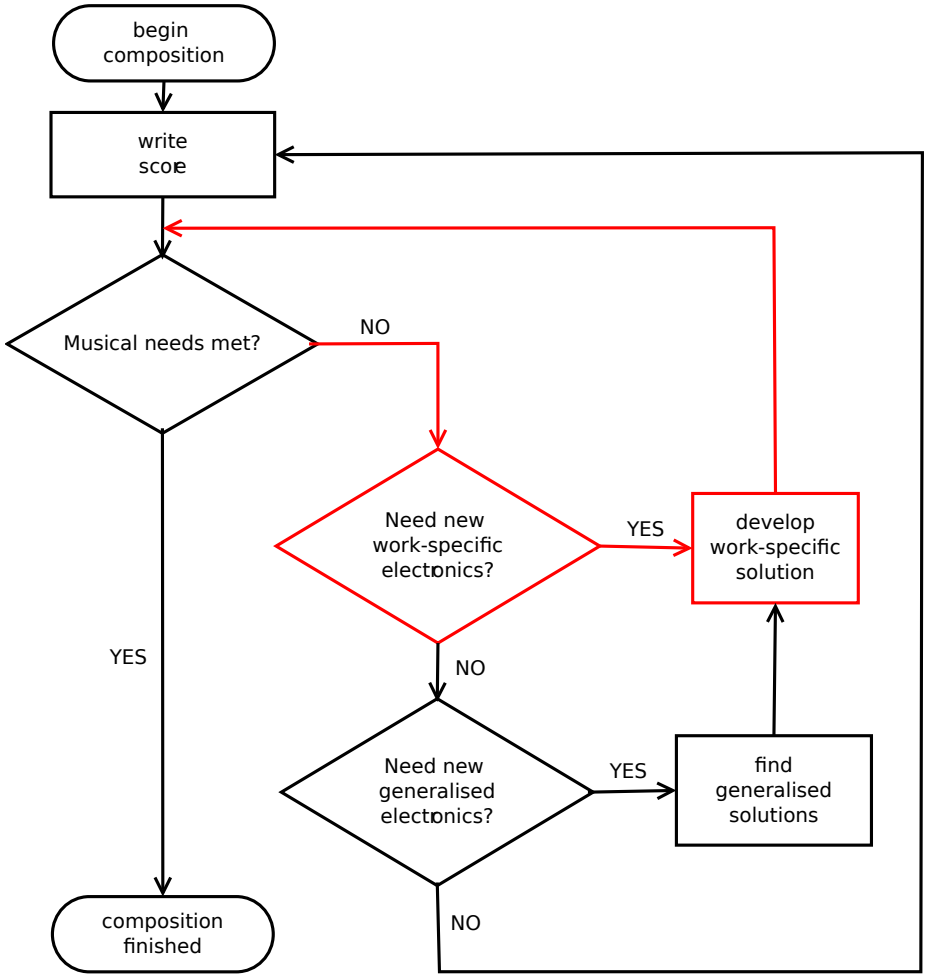


Figure 1.1: Composition with live electronics work flow diagram

1.1.2 Vision for live electronics software

Integra challenges the conventional partnership of 'composer and musical assistant' established at research centres such as the Institut de Recherche et Coordination Acoustique/Musique (IRCAM), for developing music with live electronics(Guercio, Barthelemy, and Bonardi, 2007). Instead Integra seeks to empower composers and performers by providing user-friendly software that can be accessed and used by musicians with no programming or digital signal processing (DSP) skills. Currently, composers tend to develop 'ad hoc' technical solutions for their works either through self-teaching and experimentation, or through collaboration with a technical expert. This process is shown in figure 1.1. Typically composers will spend a large portion of their time in the central part of the diagram (shown in red). This 'DIY' approach has resulted in software developed in environments such as Max[3] and Pure Data (cf. section 1.7.3), which is designed to be used exclusively in the work for which it was developed. That is, it can't be used by other composers in other pieces without significant extra work in extracting reusable components from the software.

It is for this reason that I have focused much of this thesis on the outer part of figure 1.1 – the development of generalised solutions. My hypothesis is that this will ultimately lead to more flexible and sustainable systems that are of greater long-term benefit to the live electronics and research communities.

1.1.3 Composer as software developer

It will become apparent when I outline my methodology in section 1.3 that in the research process I have assumed three distinct roles:

1. Composer
2. Musical assistant (cf. section 4.4)
3. Software developer

Due to my inter-disciplinary background, with formal training in musical composition and sonic art, and career experience in computer programming, I have felt qualified to assume these disparate identities. The synergy created by connecting on a very personal level the worlds of software engineering and musical composition and performance has seemed appealing. It represents an embodiment of the union of 'artistic' and 'scientific' thinking central to the Integra project (cf. section 1.1.1), and to my own

[3] `http://www.cycling74.com`

philosophy as a composer. However, this approach presents many challenges, not least to the individual undertaking them. The methodological challenge of assuming very different roles in one's own research process will be revisited in chapter 5.

1.2 Research questions

This research explores the following question as its central theme:

> "what are the possibilities for the implementation of audio feature extraction techniques in live electronic music?"

The many implications of this question are presented throughout this thesis, with research outcomes consisting of the thesis itself, release-quality computer software for audio feature extraction, and a folio of compositions that explore the possibilities of the software. The relationship between these elements is shown diagrammatically in figure 1.2. The thesis itself, documents and contextualises the research process, with its central argument presented as a transition between the two central areas of investigation: development of computer software and musical composition. In the final chapter, conclusions are drawn about the relationship between these elements.

The central theme of *implementing* of audio feature extraction subdivides into two main areas of investigation:

- The implementation of an audio feature extraction system for use in live electronic music
- The implementation of feature extraction in the context of composition and performance in live electronic music practice

The research methods used to explore these areas draw upon techniques from the fields of Musical Information Retrieval (MIR), Auditory Scene Analysis (ASA), statistical analysis and Artificial Intelligence (AI).

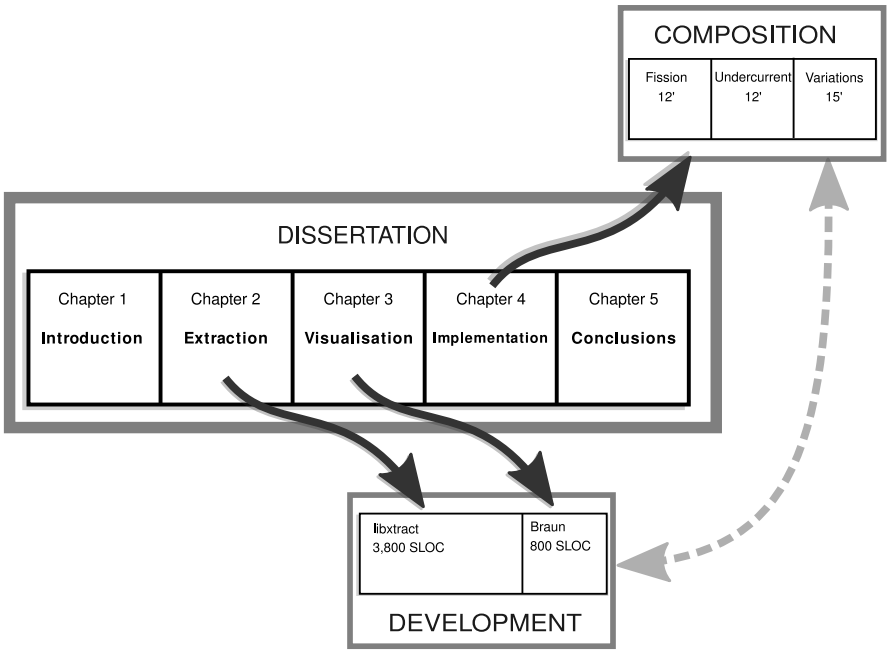


Figure 1.2: Diagrammatic representation of the relationship between the written thesis, compositions and software development components of the submission

1.2.1 Objectives

The main objectives of this research are as follows:

- To develop stable and usable software for real-time audio feature extraction in the context of live electronic music
- To develop a framework for the study, comparison and selection of features from a library of audio files
- To develop as part of the framework, suitable visualisation tools and techniques to enable performers and composers to explore audio features in a visual manner
- To develop software systems for the extraction of higher-level features from lower-level features
- To develop software systems that use audio feature extraction software in conjunction with other live electronic elements to form a musically meaningful interactive system

- To compose new musical works that make use of audio feature extraction software according to the musical requirements of the composition
- To work with performers in testing and adapting interactive software based on audio feature extraction
- To give performances of newly composed music using the software developed for this thesis

1.2.2 Evaluation criteria

Whilst the software developed in the course of this research is intended to solve or address various research questions posed by the thesis, the software in turn forms part of a new hypothesis. That is, whilst it is anticipated that the software developed will demonstrate an implementation of various feature extraction techniques suitable for use in live electronic music, the existence of this software poses the fundamental question: how useful is this software in composition and performance practice?

The research outcomes will therefore be assessed on two levels:

- The success or otherwise of the implementation of a general system for audio feature extraction and selection
- The usefulness of such a system in composition and performance with live electronics

Common to both of these are:

- The inclusion of a representative and broad range of features
- The demonstration of applicability in bespoke interactive software for an individual piece

Ultimately, as suggested by Collins 'the testing ground of real concert pieces is seen as a vital proof-of-concept [...] Technology that works in real situations is a true test of engineering'.

Evaluation of this technology will therefore be conducted through critical analysis of given use-case scenarios involving my own compositional work, and the work of other composers. This research will be presented through a series of case studies in chapter 4.

1.3 Research methodology

The purpose of this research is to produce musically useful software for real-time audio feature extraction and to test the software in musical composition and performance. The methodology employed to achieve this end has involved a reciprocal process of composition and software development. In some cases software has been developed in a pseudo-experimental way, and then incorporated into part of a musical work. In other cases the musical idea has preceded the software, and the software has been developed in response. These processes are discussed in greater detail in chapter 4.

1.3.1 Practice-led vs practice-based methodology

The terms practice-led and practice-based are often used interchangeably, and there is no consensus on their meaning. In their online funding guides, the UK arts and humanities research council (AHRC) describe practice-led research as:

> [...]research where practice is an integral component; or where it is specifically undertaken with a view to generating outputs and outcomes with a defined application beyond the education sector; and/or where it theorises contemporary practice in order to then inform the Principal Investigator's own individual practice[4]

This interpretation of 'practice-led' is corroborated by Haseman:

> There has been a radical push to not only place practice within the research process, but to lead research through practice. Originally proposed by artists/researchers and researchers in the creative community these new strategies are known as creative practice as research, performance as research, research through practice, studio research, practice as research or practice-led research. In this paper, to clarify, performative researchers are constructed as those researchers who carry out practice-led research. Practice-led research is intrinsically experiential and comes to the fore when the researcher creates new artistic forms for performance and exhibition, or designs user-led, online games or builds an online counselling service for young people.

[4] `http://www.ahrc.ac.uk/FundingOpportunities/Documents/Research%20Funding%20Guide.pdf`, accessed 9 July 2008

> (Haseman, 2006)

By contrast he defines 'practice-based' research as:

> practice-based research strategies and include: the reflective practitioner (embracing reflection-in- action and reflection-on-action); participant research; participatory research; collaborative enquiry, and action research. Invariably these strategies re- interpret what is meant by 'an original contribution to knowledge'. Rather than contribute to the intellectual or conceptual architecture of a discipline, these research enterprises are concerned with the improvement of practice, and new epistemologies of practice distilled from the insider's understandings of action in context.
>
> (Haseman, 2006)

An almost completely contradictory interpretation is given by Biggs who asserts that 'practice-based research prioritises some property of experience arising through practice, over cognitive content arising from reflection on practice' (Biggs, 2004).

In this thesis Haseman's interpretation of practice-based research is taken, and the research methodology is primarily practice-led according to Haseman's definition. That is, the outcomes of the creative practice of composition and performance, and the creative developmental practice of software development are presented as key components of the research 'contribution to knowledge'. The written component serves as documentation of the research process, and a rational contextualisation of the non-textual outcomes.

1.3.2 The role of software development in the project

Unlike similar research projects, for example Collins (2006), the software developed during the research is not presented as a 'solution' to the research questions in the thesis. In Collins (2006), the research questions are set out as 'guidelines for investigation or directives to be fulfilled', and include outcomes such as 'To deliver computationally feasible real-time machine listening from recent and novel automatic audio analysis research' and 'To produce interactive music systems which take as their input an audio signal alone, and yet also involve symbolic reasoning on extracted sound objects; to manifest hybrid systems'. In my own research the software is developed as an hypothesis to be tested through artistic practice. As evidenced through the case studies presented

in chapter 4 an iterative process emerges from the research method where the software development both reciprocates and anticipates musical requirements.

1.4 Ethical considerations

Free, Libre and Open Source Software (FLOSS), is software that is distributed under a license that is designed to guarantee the user a number of essential freedoms. There are several established variations on the definition of FLOSS, the Open Source Initiative's 'Open Source Definition' (see appendix D), and the Free Software Foundation's 'Free Software Definition', which significantly pre-dates it.

The following text is an excerpt from the Free Software Definition[5] outlining the four essential freedoms of free software.

> "Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer."
>
> Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:
>
> - The freedom to run the program, for any purpose (freedom 0).
> - The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
> - The freedom to redistribute copies so you can help your neighbour (freedom 2).
> - The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

A decision was made early in the research process that all software developed for the project would be released under a free and open source (FLOSS) software license

[5] Source `http://www.gnu.org/philosophy/free-sw.html`.

Software	License	Purpose
Pure Data	BSD Style	Live electronics realisations
Csound	LGPL	Live electronics realisations
GNU Lilypond	GPL	Typesetting of musical scores
LaTeX	LPPL	Typesetting of thesis
vim	GPL-compatible	Text editing
gnuplot	gnuplot license	Graph plotting
gcc and GNU autotools	GPL	Application compilation
Python	PSFL	Application interpretation
wxPython	wxWindows license	GUI development

Table 1.1: FLOSS software used in the research process

namely the GNU GPL[6], and that only FLOSS software would be used for the construction of the live electronics systems and the production of scores. It is the view of the current author that the freedom guaranteed by FLOSS licensing is not only ethically sound, it also serves as a basis for improved collaboration, knowledge sharing, cumulative innovation and software sustainability. Table 1.1 lists some of the free software used in this research.

1.5 Background

I will now present a contextual review of live electronic music, with a focus on works and techniques involving audio feature extraction in musical composition and performance.

1.5.1 Live electronic music

Some of the earliest experiments using electrically powered technology in music were made in the context of live performance. The first of John Cage's *Imaginary Landscapes* (1939) involved the use of variable speed turntables in concert and is often cited as being the first composed piece of 'live electronic music' (Griffiths, 1978). It wasn't until the mid-1960's that the use of live electronics in composition and performance started to evolve further. This emerged from close collaborations between performing ensembles such as the Sonic Arts Union, and composers, such as Stockhausen and Cage.

[6] Available at `http://www.gnu.org/licenses/gpl.html`.

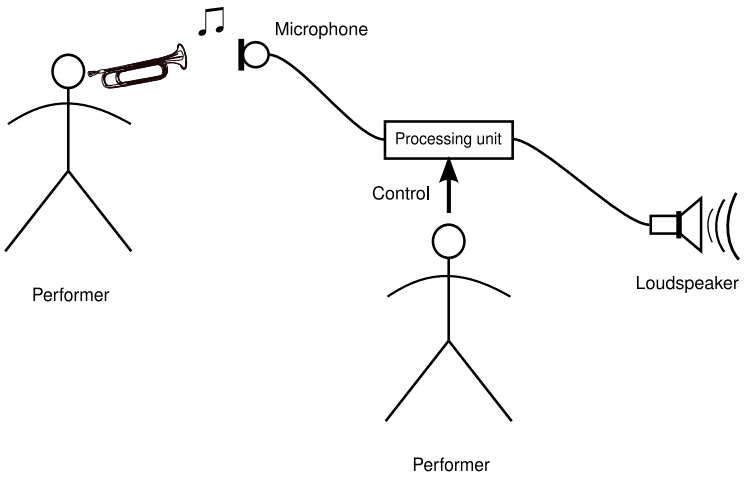


Figure 1.3: Archetypal live electronics configuration

Stockhausen's *Mikrophonie I* (1964-5) epitomises some of the hallmarks of what later became idiomatic writing for acoustic instruments and live electronics. The piece uses two microphones and two filters to amplify and modify the sound of a large tam-tam to produce 'strange and subtle noises from an instrument which might have appeared limited' (Griffiths, 1978). The model established in this work (figure 1.3) has subsequently been used and adapted for a wide range of instrumental and processing combinations and serves as a basis for what could be considered an archetypal live electronics configuration.

In the entry on 'electro-acoustic music' in The New Grove Dictionary of Music and Musicians, Emmerson and Smalley give the following explanation of live electronic music:

> In live electronic music the technology is used to generate, transform or trigger sounds (or a combination of these) in the act of performance; this may include generating sound with voices and traditional instruments, electro-acoustic instruments, or other devices and controls linked to computer-based systems.
>
> (Emmerson and Smalley, 2001)

Whilst this suffices as a technical definition of live electronic music, several distinct aesthetic approaches have emerged since its inception in 1939. Although the tech-

niques presented in this thesis are applicable in various 'styles' of live electronic music, the primary aesthetic concern is a specific subset of live electronic music where 'sound produced by the performer [is] modified electronically at the time of production in a manner controlled by the instrumentalist or another performer (often at the mixing console)' (Emmerson and Smalley, 2001).

In the following section I will outline the details of this 'classic' model of composition and performance with instruments and live electronics, drawing upon prominent works from the live electronic repertoire. I will the proceed to outline an extension of this model involving real-time audio feature extraction.

1.5.2 The control-processing paradigm

The generalised model of performance for systems for live electronics as shown in figure 1.3 is based on the following stages:

1. Sound is produced by a performer, often using a musical instrument
2. Acoustic energy is converted to electrical energy using a transducer, usually a microphone
3. The electrical signal is processed electronically
4. The signal is converted to an acoustic waveform using a loudspeaker

Stage 3 is the element that is usually referred to as 'the electronics'. This represents a 'black box' with one or more inputs and outputs, and enables the nature of the signal processing to be changed during the course of a performance. Physical control over the processing parameters can be provided by hardware inputs from potentiometers or other sources of voltage change. These can be operated by the performer controlling the mechanical sound production (musical instrument), or by an additional performer who specialises in live electronics. There are many variations on this model that involve additional instrumental performers, parallel electronic processing elements, and multiple channels of speakers. The schematic diagram for a sophisticated live electronics system is shown in figure 1.4.

1.5.3 Music N

In 1957 Max Matthews developed MUSIC I, the first computer software for sound generation to be adopted for musical composition. This was followed by what has come to be known as the 'MUSIC N' series of applications (MUSIC II, MUSIC III, MUSIC IV, MUSIC V, MUSIC 360)(Manning, 2004). Of these applications, MUSIC III set the precedent for the implementation of the control-processing paradigm for live electronics in the digital domain. It did this by establishing the notion of individual audio processing modules or 'ugens'[7] which could be combined in an 'orchestra', with a 'score' sending control data to change their processing parameters over time (Chadabe, 1997).

MUSIC III was designed for non-real-time synthesis, but its modular approach of using interconnected 'ugens' has culminated in more recent applications such as Csound, Max/MSP, SuperCollider and ChucK all of which draw at least a certain amount of 'influence' from the MUSIC N series, with Csound resembling it the most closely. All of these applications make a conceptual (and often technical) separation between control data and signal processing, whereby control messages such as MIDI input or 'Score' events are treated differently to the signal processing components such as waveform generators, filters and delays. As computers have become more powerful, real-time (low-latency) control over processing parameters has become possible meaning that the 'processing unit' functionality shown in figure 1.3 is increasingly being performed by a general-purpose personal computer running 'audio programming environment' software such as Max/MSP.

[7] Unit generators.

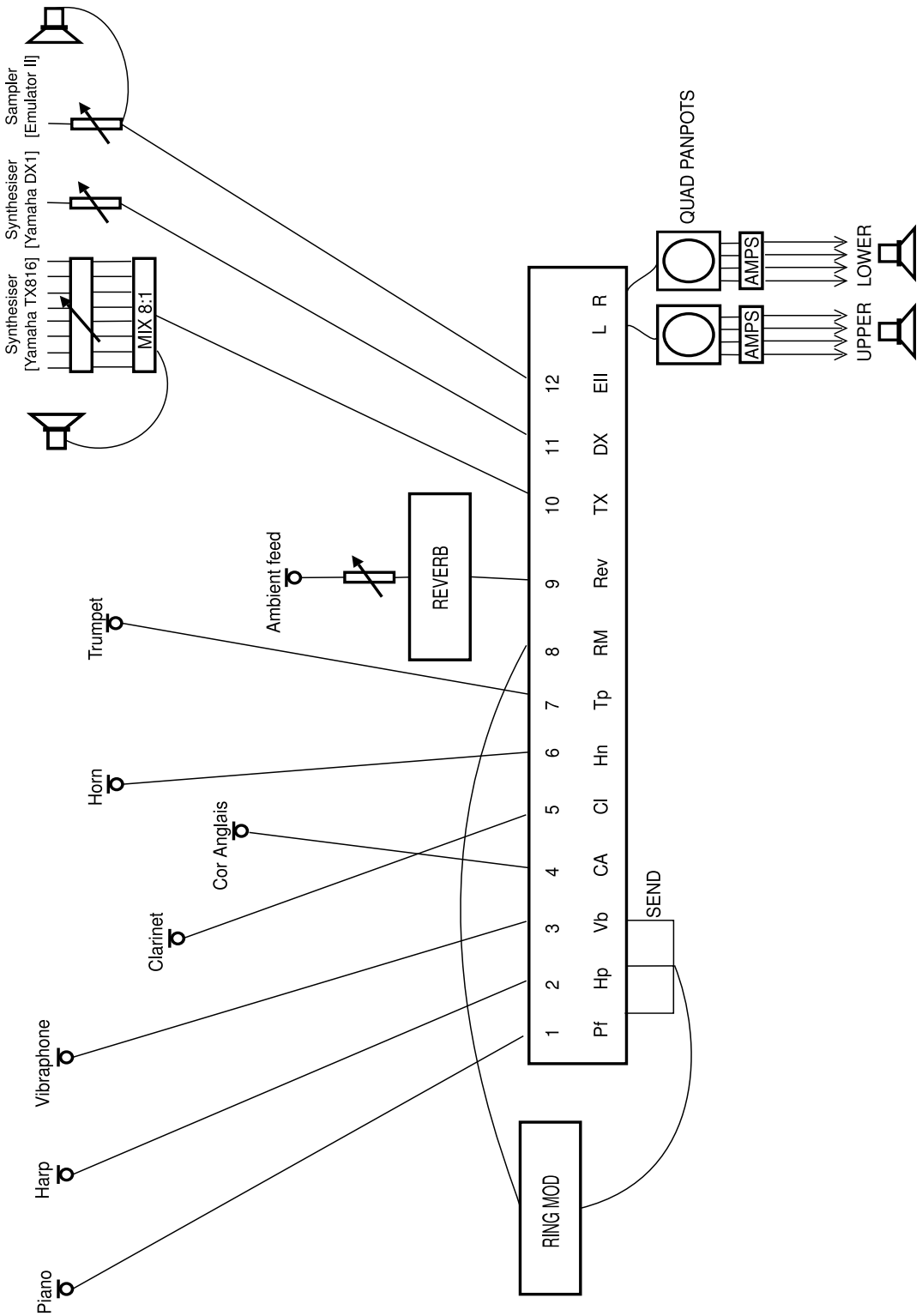


Figure 1.4: Live electronics setup for *Madonna of Winter and Spring* by Jonathan Harvey (1986)

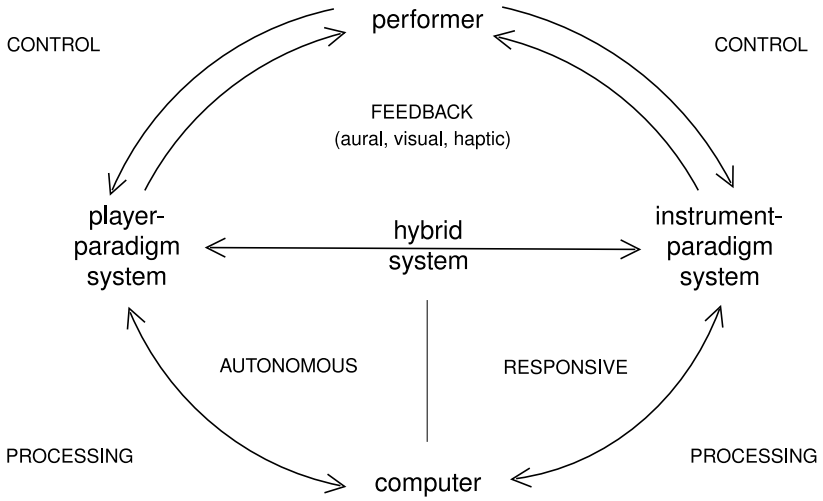


Figure 1.5: Instrument-player *continuum* that extends Rowe's player-paradigm and instrument-paradigm(Rowe, 1992)

1.5.4 Models of interaction

In Rowe (1992), two distinct models of interaction in live electronic music are described. In *instrument paradigm* systems, the electronics 'extend and augment the human performance' through direct response to input generated by the performer via sound or physical control(Rowe, 2004). In contrast, the *player paradigm* refers to systems that provide 'a musical presence with a personality and a behaviour of its own'(Rowe, 1992). Player-paradigm systems have a certain degree of autonomy and independence with respect the input of the human performer – the performance is more like a duet. According to Rowe's models many archetypal works for live electronics are based on the instrument paradigm, where an instrumentalist's acoustic sound is modified by simple and direct processing with control over the parameters taking immediate effect (cf. section 1.5.1). The 'player paradigm' can be observed in more recent systems (Beyls, 1988; Collins, 2006; Martin, Scheirer, and Vercoe, 1998) made possible due to advances in computer processing power and high-level domain specific languages from the late 1980s onwards.

These classifications, whilst useful for gaining an 'overview' of the type of system used seem a poor fit for the diversity and richness of live electronics practice. An alternative model is presented in fig 1.5, which shows a *continuum* of possibilities incorporating both instrument and player paradigms.

McNutt describes live electronic performance as 'a kind of chamber music, in which some ensemble members happen to be invisible'(McNutt, 2003). This is a shrewd observation, because it acknowledges the true complexity (and difficulty) of the interaction between performers and electronics. It is the current author's view that many of the key issues in current live electronics practice should centre on improving interactivity between performer, composer, audience and technology. Questions relating to these issues will be explored further in chapters 3, 4 and 5.

1.5.5 Why audio feature extraction?

According to McNutt, performance with live electronics implies prosthesis:

> From the performer's point of view, a composer's use of electronics will always involve some prosthetic elements that complicate the practice of her art. These can stand in the way of the ideal collaboration between composers and performers. Often issues that seem self-evident to performers appear to be invisible to electro-acoustic composers, and vice versa.
> (McNutt, 2003)

My primary motivation for developing live electronics systems based on audio feature extraction is to minimise the number of prosthetic elements, and to provide a seamless sense of interaction for the performer where sound becomes both the source of control and the means of gaining auditory feedback. Using sound as a medium for interaction removes the requirement for sensors, switches and other physical controllers in order to convey gestural information and performer intention. In the following sections existing technology for live electronic music will be briefly discussed, and some of the possibilities of audio feature extraction will be presented.

The hypothesis of this research is that a bespoke audio feature extraction system will provide a number of musical uses including, but not limited to:

- Direct control over live performance parameters in a one-to-one or one-to-many relationship
- Generating control data for statistical analysis and machine learning
- Creating mappings between multiple features and N-dimensional output for control or storage

- Musical gesture capture and analysis
- Source recognition including instrument and playing technique recognition
- Chord recognition
- Non-real-time analysis of recorded audio including instrumental performance
- Score following in non-pitch-based music

It is very common for the computer or electronics components in live electronic music to involve some form of physical interaction from one or more "technicians" or from the instrumental performers themselves. Early examples of this include Stockhausen's *Mikrophonie I* (1964) and *Mikrophonie II* (1965), which involve real-time parameter change of filters and ring modulators respectively. More recent examples include numerous compositions written at IRCAM (Institut de Recherche et Coordination Acoustique/Musique) including Manoury's *Jupiter* for Flute and electronics (1987), and Boulez' *Anthèmes II* for Violin and electronics (1997). Significant non-IRCAM examples include Berio's *Ofanìm* for children's chorus, ensemble, female voice and live electronics (1988; revised 1997), and Nono's *Caminates... Ayacucho* for contralto, flute, choirs, orchestra, and live electronics (1986-1987). However *Jupiter* and *Anthèmes* are particulaly relevant to this research for the notable advances in the relative autonomy vested in the instrumental performer at their time of composition by providing performers with an element of control through the use of score following (Orio, Lemouton, and Schwarz, 2003).

As the possibilities for live signal processing and synthesis have expanded, the number of parameters available for real-time control have increased. It is not uncommon for pieces written between 1980 and the present day to have 10 to 100 parameters available for control either optionally or on an obligatory basis during the course of a performance. Clearly controlling all of these parameters directly through a textual or graphical on-screen interface, or even through a set of control faders presents a significant problem for the performer of the electronics part. This approach is evidenced in works such as Berio's *Ofanìm* and *Cronaca del luogo*, and Boulez' *Répons* which require at least two or more 'assistants' to setup and operate the electronics.

Approaches to solving the problems of multi-parameter control have involved the development of strategies for mapping and combining multi-dimensional parameter sets into lower-dimensional representations. In addition, sensor-based systems, whilst also providing avenues for the exploration of correlation between musical and physical gesture, and sonic output, provide a convenient means of multi-parameter control. These sensors are often attached to the acoustic instruments and/or performers, with their outputs ultimately being fed into a computer for processing. This creates possibilities for systems where physical performance gesture is closely coupled with the audio output in a piece (cf. Jensenius (2007)).

An extension of this sensor-based approach is the bespoke 'gestural controller'. Here a new interface is constructed using a combination of various sensors and/or moving parts and other hardware to produce a device that generates multi-dimensional data for controlling synthesis or signal processing parameters. With a few notable commercially-successful exceptions such as the Eigenharp[8] from EigenLabs, gestural controllers tend to be developed on an ad-hoc basis for specific works, performances or installations (cf. Paradiso (2003)). Beyond conventional electronic 'keyboard', 'guitar' and 'wind' controllers, availability of existing gestural controllers is limited and new controllers can be expensive or time-consuming to develop. It is perhaps for this reason, along with a lack of standardisation that gestural controllers are not widely used in live electronic music outside a few specialist research centres such as IDMIL[9] and STEIM[10].

1.5.6 Sound as a control source

An alternative (or addition) to physical controllers as a source of control data is to use data derived from the acoustic sound of an instrumental performance. Early examples of sound being used as a source for real-time control data are provided by Subotnick's works involving "ghost" boxes. The earliest of these was *Two Life Histories* (1977). The "ghost" boxes contained an envelope follower and pitch detector whose output was recorded to tape (and later programmable memory), and subsequently used to control live processing of the instrumental sound.

In 1987, Phillipe Manoury combined pitch detection with score following to control progress through a series of musical cues that affect changes in a single sideband modulator, a bank of pitch shifters, and a reverberator (Puckette and Lippe, 1994). Many other

[8] http://www.eigenlabs.com/
[9] http://www.idmil.org
[10] http://www.steim.org

composers have used such pitch-based score following techniques including Cort Lippe in his Music for Clarinet and ISPW (Lippe, 1993), and Boulez in *... explosante-fixe...* , a work that has been adapted to take advantage of recent advances in score following technology (Schwarz, Cont, and Schnell, 2005). The notion of pitch as a control source will be discussed in more detail in chapter 4.

1.6 Audio feature extraction

The primary concern of feature extraction is a reduction in the dimensionality of a given data set. Audio feature extraction is a process that involves transforming *audio* data from a high to low dimensional representation. Many audio features such as fundamental frequency, RMS amplitude and spectral centroid have only one dimension, these will be referred to a scalar features. Often these features have perceptual correlates such as pitch, loudness and brightness, although in many cases there is no standard mapping between the measured and perceived features. These so-called perceptual features will be discussed in greater detail in section 1.8. There are many other types of audio features including 'musical' features such as rhythm, tempo, harmony and texture. It is also possible to define completely new 'semantic descriptors', which describe an arbitrary aspect of the sound in a semantically-meaningful way (Herrera, Bello, Widmer, Sandler, Celma, Vignoli, Pampalk, Cano, Pauws, and Serra, 2005).

The focus of this thesis is on Low Level Descriptors (LLDs), which may form the building blocks for extracting higher-level features (see chapter 4). Amatriain (2004) provides a useful distinction between LLDs and high-level descriptors stating that 'low-level descriptors are those related to the signal itself and have little or no meaning to the end-user' whereas 'high-level descriptors are meaningful and might be related to semantic or syntactic features of the sound'. He also states that 'some descriptors can be viewed as either low or high-level depending on the characteristics of the extraction process and the targeted use' (*ibid.*).

The MPEG-7 standard defines an 'Audio Description Framework' that includes an LLD interface, which defines seventeen descriptors divided into the following groups:

- Basic: Instantaneous waveform and power values

- Basic Spectral: Log-frequency power spectrum and spectral features including spectral centroid, spectral spread and spectral flatness
- Signal Parameters: Fundamental frequency of quasi periodic signals and harmonicity of signals
- Timbral Temporal: Log attack time and temporal centroid
- Timbral Spectral: Specialized spectral features in a linear-frequency space, including spectral centroid, and spectral features specific to the harmonic portions of signals including harmonic spectral centroid, harmonic spectral deviation, harmonic spectral spread and harmonic spectral variation
- Spectral Basis representations: Two features used in conjunction primarily for sound recognition, but generally useful as projections into a low-dimensional space to aid compactness and recognition

(Lindsay, Burnett, Quackenbush, and Jackson, 2003)

In addition to these categories, MPEG-7 provides a set of tools for content description, these include:

- Descriptors (D): A descriptor is a representation of a feature, where a feature is a distinctive characteristic of the data that signifies something to somebody
- Description Schemes (DS): A description scheme specifies the structure and semantics of the relationship between its components, which may be both descriptors and description schemes
- Description Definition Language (DDL): a language that allows the creation of new description schemes and, possibly descriptors

(Avaro and Salembier, 2002 [in Salembier and Sikora (2002)])

The DDL is a declarative language based on an extension of the W3C's XML Schema Language and is used to define the structure and content of MPEG-7 documents. However, Amatriain (2004)[11] argues that 'MPEG7 would benefit from being an extensible software framework instead of just a set of tools and definitions'.

[11] Online version only: `http://www.iua.upf.es/~xamat/Thesis/html/`.

Herrera and Serra propose an alternative description scheme which draws upon both the MPEG-7 descriptor and work done on the Sound Description Interchange Format (SDIF), although the latter increasingly addresses a different set of problems to MPEG-7 (Herrera, Serra, and Peeters, 1999). The focus of their work is on what they term low- and mid-level descriptors, providing a hierarchical approach that allows for macro- and micro-structural searches for example 'search for timbres with similar variations in the partial amplitudes along time' (*ibid.*).

In general, the distinction between features and non-features is not straightforward, and likewise the distinction between high-level and low-level features is not standardised, with different authors providing different interpretations. Amatriain, for example only uses two categories 'spectral' and 'temporal' for LLDs. It could therefore be argued that it only makes sense to consider the *relative* level of abstraction between features rather than trying to create an absolute scheme. For the purposes of this research, the result of any DSP operation will be considered as a candidate feature, and the terms 'higher-level' and 'lower-level' will be used in favour of the more absolute 'high-level' and 'low-level'.

1.7 Precedents

A review will now be presented exploring selected existing research that is of particular relevance to this thesis. The focus will be on examples where feature extraction leads to a musically meaningful or useful mapping onto real-time performance parameters. In the majority of cases the scope of the review will be restricted to examples that have resulted in generalised solutions and publicly available software rather than bespoke or ad-hoc solutions created on a 'one off' basis for individual works. This is consistent with the aim of this research in creating sustainable generalised tools for live electronics practice.

1.7.1 Early precursors

In 1978, David Wessel conducted a series of experiments to obtain 'representations of sounds that suggest ways to provide low-dimensional control over their perceptually important properties' (Wessel, 1979). His research involved the development and use of computer software for taking perceptual similarity judgements about a group of sounds and representing the similarities as proximities in a 2-dimensional space. The hypothesis of the research was that it would be possible to create a musically meaningful mapping between locations within this space and additively synthesised audio. Wessel writes:

> The basic idea is that by specifying coordinates in a particular timbre space, one could hear the timbre represented by those coordinates. If these coordinates should fall between existing tones in the space, we would want this interpolated timbre to relate to the other sounds in a manner consistent with the structure of the space.
>
> (Wessel, 1979)

Although there is no evidence in the article that Wessel actively developed an implementation of the system he describes, it is apparent from Grey (1975), that such a system should be technically feasible. Wessel suggests that although he had previously been successful in using a 2-dimensional graphical user interface (GUI) to control the Di Giugno oscillator bank at IRCAM, in order to accomplish real-time interpolation between timbral zones, an 'efficient computer language for dealing with envelopes [would be] needed' and that 'With such a language it will be possible to tie the operations on the envelope collections directly to the properties of the perceptual representations of the material' (Wessel, 1979).

The system described by Wessel proposes a feature extraction process as executed by a human, and not by computer (the computer is used only to organise the dissimilarity judgements). It could be summarised by the diagram shown in figure 1.6. Following the collection of similarity judgements, the data is subject to Multi-dimensional scaling (MDS) analysis in order that the instrumental timbral differences can be visualised on a two-dimensional graph, with the horizontal axis representing the 'bite' of the sounds, and the vertical axis representing their brightness. It is suggested that the brightness percept correlates to the 'spectral energy distribution' audio feature, and the 'bite' percept relates to the 'nature of the onset transient'(Wessel, 1979).

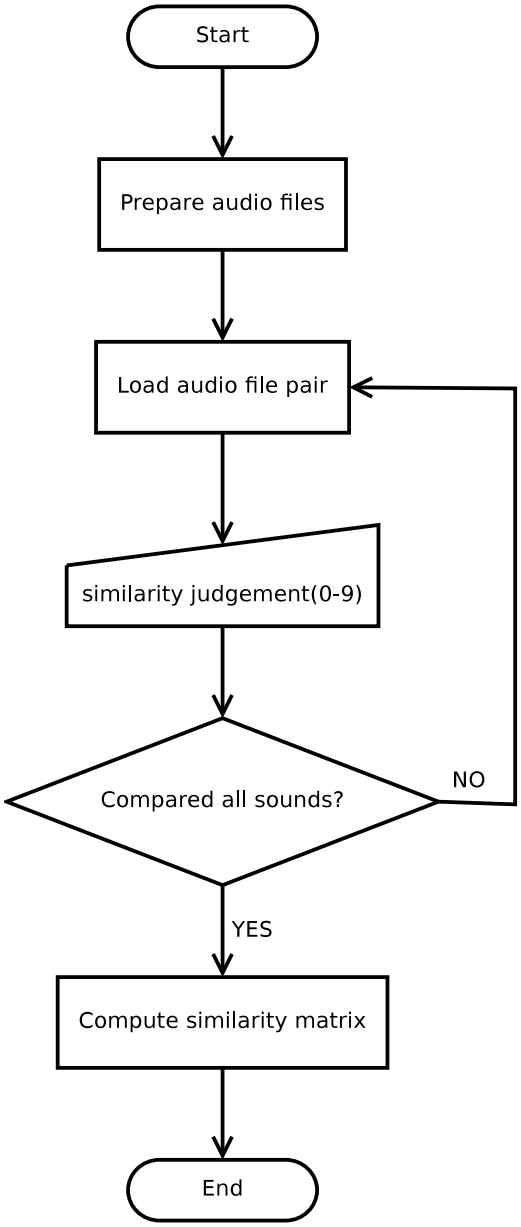


Figure 1.6: Process employed by Wessel (1979) for 'extracting' timbre similarity judgements made by a human listener

The basic model of audio analysis (collection of features and/or judgements) and mapping/scaling used by Wessel can be observed in much of the work that followed including research presented in chapter 4.

1.7.2 libtds

libtds[12] was developed by Eric Métois between 1992 and 1996 during his term as Research Assistant at the Music and Media group under the supervision of Prof. Tod Machover (Métois, 1991). libtds implements a range of feature extraction functionality including forward and inverse Fourier transform with optional Hann windowing, real-time time domain pitch detection, brightness estimation, formant following and harmonic analysis. The system was used in a range of musical compositions and performances at MIT including Machover's Hyperstring piece *Forever and Ever*, premiered in 1993, *Improvisation* for Cello, Keyboards and a Disklavier (1995) and Machover's The *Brain Opera* (1996). All of these works used libtds to provide real-time feature extraction, mainly on Silicon Graphics workstations, although in the latter case a Windows PC was also used.

Of particular relevance to this research is the 'Singing Tree', a 'novel interactive musical interface which responds to vocal input with real-time aural and visual feedback' (Olivier, Yu and Métois, 1997). The 'Singing Tree' was developed using Métois' software, and 'loudness, pitch contour, noisiness and formantic structures are analysed on the fly, leading to a set of parameters that characterises the audio input'(Métois, 1991). Métois proceeds to describe the success of the system achieved through immediacy of gestural correlation between vocal utterance and resultant 'response' from the system:

> The simplicity of their purpose and their responsiveness are the major keys to the singing trees' success. The choices that were made concerning the analysis of the incoming audio stream and the mapping to the musical engine turned out to be appropriate. Audience members have no trouble figuring out how to use this installation as they become quickly involved with the musical experience it provides. Yet, the same simplicity has some drawbacks as people sometimes feel they have explored to whole range of the experience too quickly. The trade-off between building self-explanatory setups and systems that provide a sustained degree of involvement is a difficult compromise that one encounters endlessly in the context of interactive

[12] libtds by Métois is unrelated to the freeTDS and libTDS projects that implement the TDS (Tabular DataStream) protocol.

installations.

(Métois, 1991)

The dichotomy Métois presents here highlights one of the central problems of live electronic music: that of creating systems where there is a causal relationship between human input and the system's output, which is obvious to performer and audience, but not so predictable as to become unmusical or 'tiring'. This issue will be discussed further in chapters 4 and 5.

1.7.3 fiddle~ and bonk~

In the mid-to-late 1990s, several software platforms became available that integrated audio and control data processing for the purpose of real-time composition and performance of live electronic music. In 1996 Miller Puckette publicised and released Pure Data (Pd), an 'integrated computer music environment' based on his Max software, which was in turn based on IRCAM's Patcher (Puckette, 1988, 1996). The release of Pd was significant because previous real-time software such as Max, dealt only with low-bandwidth control data (such as MIDI), and widely available synthesis solutions such as Csound were not originally designed for real-time use. James McCartney's SuperCollider software was also released around the same time, but primarily as 'an environment for real-time audio synthesis' (McCartney, 1996), that is, it did not have as broad a scope as Pd. SuperCollider at that time was also closed-source, whilst Pd had a BSD-style license[13]. The MSP extension to the Max language was also made available shortly after the release of Pd, and was significant as being the first commercially supported, widely available graphical programming environment for audio and control data.

In their basic form both Max/MSP and Pd are capable of spectral analysis using fft-based objects. However, extracting audio features from the outputs of these objects using the Max or Pd languages is non-trivial due to the absence of functionality for programmatically iterating over audio blocks[14]. However, both languages are extensible using external dynamically-linked libraries (externals), written in C. In 1998 Puckette, Abel and Zicarelli presented two such externals, written specifically for the purpose of

[13] BSD-style licenses are permissive software licenses that allow changes to the source code, and redistribution in source or binary form.

[14] This is provided by the expr~ external but its functionality is limited and it is too computationally expensive for extensive use.

extracting features from real-time audio, namely pitch, sinusoidal components and onset locations (Puckette, Apel, and Zicarelli, 1998). The work of Puckette et al. in this area is significant because of the wide availability of the software they produced. The two externals `fiddle~` and `bonk~` were simultaneously released with source code for Max/MSP (Macintosh) and Pd ('Wintel', SGI, Linux), with `fiddle~` also running under FTS, which was also available on several platforms (*ibid.*). Although previous pitch detection solutions had been provided by Brown and Puckette (1993), and beat detection had been achieved by Vercoe (1984), `fiddle~` and `bonk~` made feature extraction technology available to anyone with a home computer and a simple audio card. As a result, these externals have been used in many musical compositions and performances since their release (cf. Trueman (1999), Schiesser and Traube (2006) and Ciufo (2002)). They are also being used in the modernization of musical works that previously used proprietary software (cf. Bullock and Coccioli (2006)).

1.7.4 Audio-driven synthesis

In his 2001 paper *An Audio-Driven Perceptually Meaningful Timbre Synthesiser*, Tristan Jehan describes a system that brings together audio feature extraction, timbral mapping and synthesis in the context of live electronics performance. Based around the `analyzer~` external for Max/MSP, developed by Jehan, and the efficient Cluster Weighted Modelling technique (Gershenfeld, Schoner and Métois, 1998), Jehan proposed a system developed in Max/MSP whereby 'continuous changes in articulation and musical phrasing' lead to 'highly responsive sound output' (Jehan, 2001). In terms of feature extraction, Jehan only uses pitch, loudness, and brightness, however, he does note that in future work the system could be extended with more perceptual features (*ibid.*). At the time of writing `analyzer~` incorporates the additional features: Bark bands, and a 'noisiness' value which uses the bark-based spectral flatness feature. One of the main strengths of the system is that Jehan makes available the results of the research in the form of several reusable publicly downloadable components for Max/MSP. The `analyzer~` external is discussed in more detail in section 2.3.5.

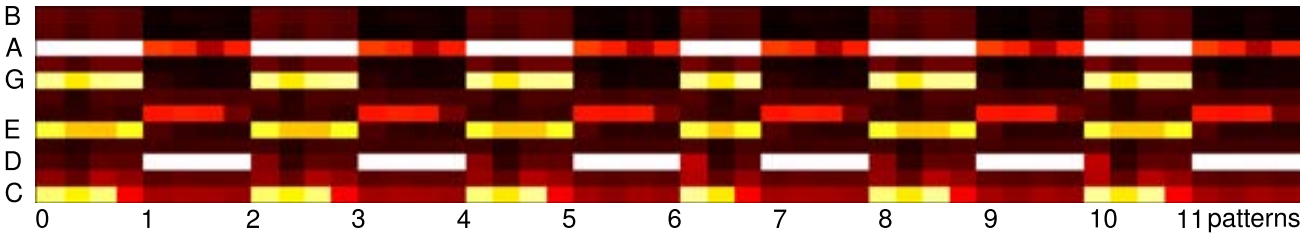


Figure 1.7: Pitch chromagram from Jehan (2005) showing a series of eight different chords (four Am7 and four Dm7) looped 6 times on a piano at 120 BPM

1.7.5 Context-aware automated montaging

Jehan's more recent work involves a much wider range of features including pitch chroma (figure 1.7), and also takes feature change over time into consideration. His design paradigm has shifted from one where the machine is responsive, to one where the machine is reactive, even proactive, with the final goal being to 'automate the creation of entirely new compositions' (Jehan, 2005). Essentially a system has been developed, which derives structure through feature extraction from a database of musical fragments, and then uses 'knowledge' about these structures to synthesise new works through concatenative synthesis (*ibid.*).

A similar approach is taken by Collins in his BBCut software(Collins, 2006). However Collins' approach has a far greater emphasis on suitability for *live* performance, and is therefore of greater relevance to the work presented in this thesis. Collins research involves a 'set of interactive systems developed for a range of musical styles and instruments, all of which attempt to participate in a concert by means of audio signal analysis alone' (*ibid.*). Collins' use of feature extraction is manifold. Onset detection, zero crossings detection, and a power metric are used for audio segmentation, and once a segment has been found additional features such as median pitch, perceptual attack time, loudness and spectral centroid are added to a database, which can then be used as the basis for algorithmic recombination of stored events. Functionality for tempo analysis and beat tracking are also provided. The BBCut2 software is primarily implemented as a set of Classes, and UGens for the SuperCollider system, but it is also available in the form of the LiveCut VST plugin by Rémy Muller[15]. It has applications in on-the-fly 'drum 'n' bass' or 'broken beat' style drum slicing as evidenced by the *CutProc11* and *SQPusher* cutting procedures provided with the package. It has also been used in Collins' own

[15] `http://mdsp.smartelectronix.com/`.

compositions including *Ornamaton* for harpsichord and baroque recorder (2006) and *Sat at Sitar* for sitar (2005) (*ibid.*).

1.7.6 Hsu and Butcher

Since their early collaborations at STEIM in 2001, William Hsu and John Butcher have been developing a 'machine listening' system for Butcher to improvise with, in particular Hsu has been exploring ways in which the system can respond to timbral change in the Saxophonist's playing (Hsu, 2005). Hsu seeks to avoid the emphasis placed on pitch in systems such as Rowe's *Cypher* (Rowe, 2004) and George Lewis's Voyager (Lewis, 2000) by extracting a range of relevant audio features. Hsu approaches the system's design 'from a listener's perspective', focusing on 'classifications that a human musician might perceive and respond to', the implication being that he worked closely with Butcher to achieve this end(Hsu, 2005). His studies result in the following timbre categories:

- noisy (vs not noisy)
- containing harmonic partials (vs. inharmonic partials)
- containing a sharp attack (vs. no sharp attack)
- containing multiphonics (vs. no multiphonics)
- rough (vs. smooth)

Hsu's research is particularly enlightening in highlighting the problems that can occur when unconventional or extended playing techniques are used. He notes that when using `fiddle~` (cf. section 1.7.3) 'pitch and partial estimations are also unreliable and less usable when the saxophone tone itself is noisy or has a complex and rapidly changing spectrum'. In addition to using sinusoidal magnitude coefficients from `fiddle~` the initial system developed by Hsu uses relative spectral centroid (absolute centroid divided by fundamental frequency), zero crossing count and peak energy distribution. A more recent version of the system has the potential to store feature profiles for the classification of gesture and acoustic roughness (Hsu, 2006).

1.8 Perception

This thesis seeks to avoid making an assumption that a close correlation between human perception of a sound's qualities, and the computer's classification 'decisions' inherently means that the computer's classification will be musically useful. Rather, it seeks to establish how feature extraction and classification might be musically useful regardless of whether the machine classification matches an existing perceptual model. For example some studies place an emphasis on the psychoacoustic correlation of certain features (Collins, 2005a; Klapuri, 1999; Slaney and Lyon, 1990) either because they believe that incorporating psychoacoustic information into their study will improve the robustness of their algorithms or because such features have been shown to give good results in practice. However, the current author's research makes no assumptions that because an audio feature 'decision' correlates well with our own perception it will by default be more useful in the context of live electronic performance.

This is not to say that perceptually informed features such as Mel Frequency Cepstral Coefficients (MFCC) and Bark Bands, which use perceptually determined frequency scaling are not useful as sources. It just means that there is no evidence of an implicit or explicit relation between the proximity of a computer classification to human judgement, and its musical usefulness as a control parameter in live electronics.

For example the fact that Spectral Flatness (cf. section 2.5.1.11) corresponds reasonably well with perceptual noise content may or may not make the feature useful for use in live electronics. For the purpose of feature extraction in live electronic music, the following features may be given greater significance than perceptual correlation:

1. How stable the feature is (is the implementation prone to NaNs, infs, denormals)?
2. Is the feature algorithm easy to implement?
3. Is the feature computationally efficient?
4. Does the feature computation create significant latency? How much?
5. Does the feature give output that is predictable in relation to its input?
6. Does the feature output have a linear correspondence with the audio input, if not, is the non-linearity useful?

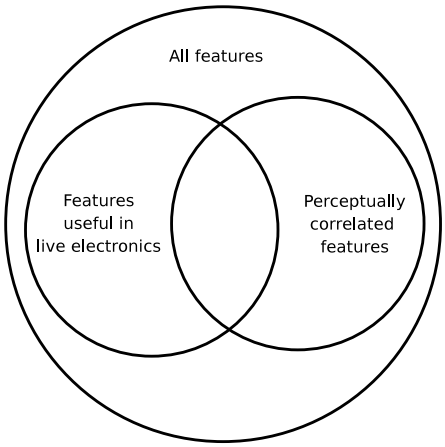


Figure 1.8: Venn diagram of the hypothetical relationship between musically useful features and perceptually correlated features

Some of these are properties of the implementation rather than the features themselves. However, from an end-user perspective the implementation is of high importance, especially if a robust real-time system is to be built around the feature extraction routines.

This thesis is not concerned with human perception of sound per se, but rather with the machine's perception of sound, and how this perception can be musically useful in usable live electronics systems. For example, if we present the system with a Violin sound, we are interested in the application of the machine classification in a musical context, and not how the machine's classification of the sound compares to our own. A Venn diagram showing the hypothetical relationship between the measured features and perceptually correlated features is shown in figure 1.8. A good example of the contrast between these two classes of audio features is given by Collins when comparing various onset detection methods:

> There are differences between perceptual segmentation (finding event boundaries as a human observer would function in real-time) and physical segmentation (breaking up events as fast and as accurately as possible for digital editing purposes).
>
> (Collins, 2005a)

More specifically, Collins suggests that for the purposes of audio segmentation discrimination functions using 'fuller psychoacoustic models of loudness' were 'less effective' for the purposes of segmentation than 'relatively simple' ones (*ibid.*).

1.8.1 Pitch, Loudness and Timbre

Pitch, timbre and loudness are perceptual qualities of sound, and carry with them a certain amount of subjectivity. The terms themselves are semantically ambiguous, and culturally informed – having several layers of meaning. As implied in the previous section, perceptually correlated feature extraction and classification are not of primary relevance to this research, however, it is worth discussing them briefly in order to further clarify the relationship between the perceptual and the scientifically measurable aspects of sound.

1.8.1.1 Pitch

Many papers on pitch detection (cf. de la Cuadra, Master, and Sapp (2001)) use the words pitch and fundamental frequency interchangeably, whilst others imply that pitch is based on a 'musically normalised' version of the fundamental (Collins, 2006). For the purposes of this thesis, a transformation between fundamental frequency (f_0 hereafter) and perceptual pitch is often given by:

$$p = 12 * log_2(f1/f2) \tag{1.1}$$

With the inverse relation given by:

$$f = 2^{p/12} * f2 \tag{1.2}$$

Where p gives the difference in semitones between two frequencies $f1$ and $f2$ in Hertz. However there are several other methods for conversion between measurable frequency in Hertz and perceptual pitch, which are based on perceptually derived scales of measurement. These include the Mel and Bark scales discussed further in Chapter 2.

When describing the use of 'pitch detection' in the context of a musical work, Lippe (1993) makes no reference to 'fundamental frequency' at all. In contrast, Slaney and Lyon (1990) are precise in their use of terminology, making a clear distinction between "perceptual pitch" and fundamental frequency.

In the author's current research the fundamental frequency is considered simply as an audio feature with possibility for musical application. Auditory based frequency-scale warping is used for calculation in certain features such as Bark bands and MFCC.

1.8.1.2 Loudness

Fletcher and Munson (1933) define loudness as 'a psychological term used to describe the magnitude of an auditory sensation' (Fletcher and Munson, 1933). They proceed with an attempt to make a scientifically reproducible system for measuring loudness based on the concept of intensity, where:

> The sound intensity of a sound field in a specified direction at a point is the sound energy transmitted per unit of time in the specified direction through a unit area normal to this direction at the point.
>
> (Fletcher and Munson, 1933)

This resulted through a process of empirical investigation in a set of 'loudness level contours' (Fletcher and Munson, 1933), (figure 1.9), where each contour represents the sound pressure level (SPL) in decibels (dB) for which a human listener perceives a steady-state tone to be of equal perceived loudness over a range of frequencies. The unit of the 'phon' is used to measure the loudness, which is achieved through reference to the perceptually-derived curves. There are now a number of different versions of the equal-loudness contours where the measurements have been updated.

There are several other definitions of loudness, which take human perception into account. For example, Zwicker proposes the division of the audible frequency spectrum into a series of critical bands (25 1/3 octave bands between 20Hz and 12500Hz). This method takes into account the characteristics of the middle and outer ear, in particular the phenomenon of auditory masking (Fastl and Zwicker, 2006). It is this model that is used as the basis (through Moore and Peeters) for the 'loudness' feature described in section 1.8.1.2.

1.8.1.3 Timbre

Regardless of whether one considers perceptual aspects of loudness and pitch, or their underlying measurable correlates, the metrics for these features are regarded as one-dimensional. Any sound 'S' can be considered to have a loudness or pitch value on a given scale in a single dimension (see figure 1.10). However, one might consider timbre in terms of words like 'roughness', 'brightness' and 'noisiness'. These dimensions of timbre can coexist, with the dimensions being strongly or weakly correlated depending on the features in question. In addition some of these dimensions (e.g. 'roughness') are time dependent and require a transient spectrum for their variability, whilst others (e.g. 'brightness') can be observed in sounds whose spectrum stays constant.

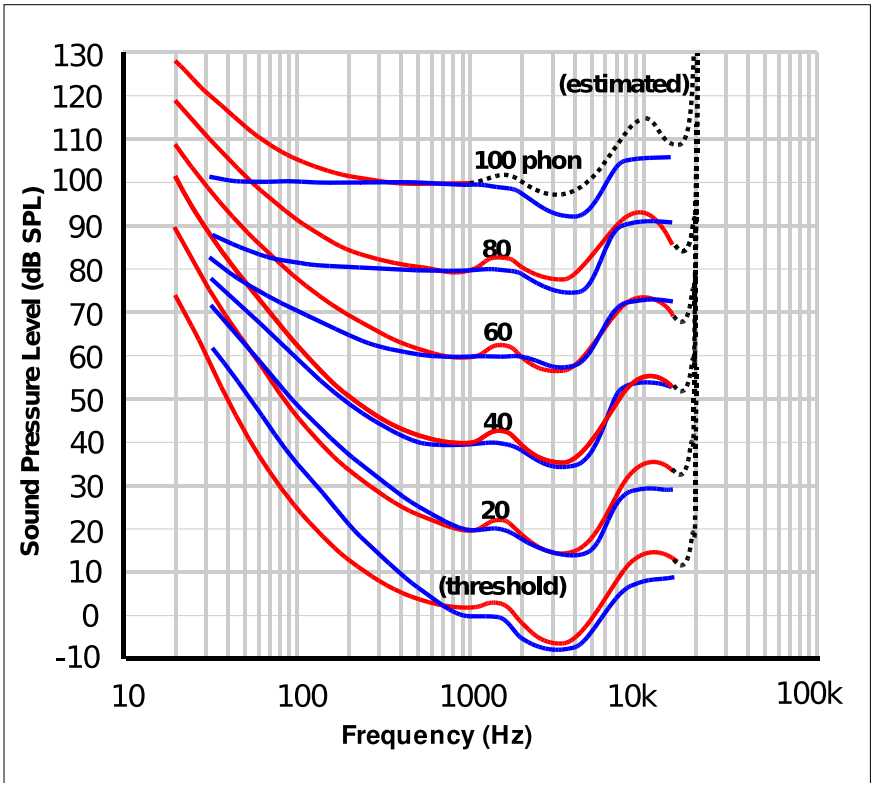


Figure 1.9: Equal-loudness contours (red) (from ISO 226:2003 revision) Fletcher-Munson curves shown (blue) for comparison

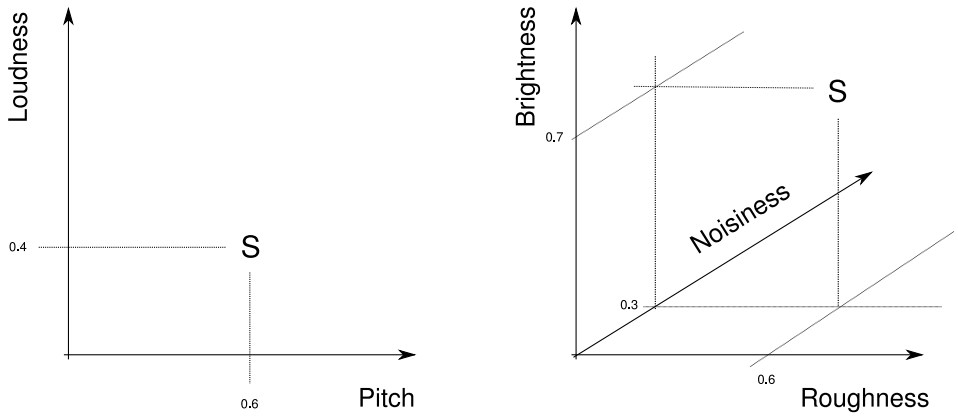


Figure 1.10: The multidimensional (vector) nature of timbre compared to the scalars pitch and loudness

1.8.1.4 Timbre and frequency

Studies into the relationship between timbre and frequency have yielded differing and sometimes contradictory results. Marozeau, de Cheveigné, McAdams, and Winsberg (2003) suggest that timbre is perceived relatively independently from pitch other than for instruments where fundamentally different resonance (e.g. through changes in resonator geometry) is invoked through the f_0 change. However, their research only considers timbral change for f_0 differences smaller than an octave, it also only considers instrument class, and not more subtle qualities such as perceived 'brightness'. Research by Schubert and Wolfe demonstrates that 'brightness is dependent upon f_0 to the extent that increasing f_0 also increases spectral centroid' (Schubert and Wolfe, september/october 2006). Hong-Park also suggests that whilst there might not be conclusive evidence to suggest a strong relation between f_0 and timbre, f_0 can be used as an elimination tool to demarcate timbral classes, for example instrumental groups, through their f_0 range (Park, 2004).

Despite their perceptual relatedness (or otherwise), the *measurement* of timbre and f_0 are very separate. The purpose of a fundamental frequency estimator is to measure the presence of a fundamental frequency and to tell us how high or low it is in Hertz. It is a matter of design in the pitch detection algorithm how the 'confidence' of the output f_0 value is represented, and this will be discussed further in section 2.5.1.20, however an f_0 detector explicitly does not measure the noisiness of a signal or any other timbral quality.

1.8.1.5 Timbre and loudness

The relationship between playing intensity (effort), and changes in frequency spectrum can be observed by comparing measured spectra. However, the precise relationship between loudness and timbre is not clearly defined. Melara and Marks (1990) give an account of research conducted in order to establish the interrelatedness or correlation effect between pitch, amplitude and timbral information perceived by listeners in two listening tests. Their tests show that attributes from one dimension were classified faster when paired with 'congruent' attributes from the other dimension (Melara and Marks, 1990), suggesting that we make a perceptual 'association' between certain changes in loudness and related timbral change. Research by Clark and Milner reinforces this notion. In a series of experiments designed to measure the relatedness of playing intensity and timbre, they asked a number of musically competent subjects to identify the dynamic marking with which tones were reproduced at a common loudness (Clark and

Milner, 1964). The following conclusions were drawn from the results:

1. The timbre is very weakly dependent upon dynamic marking and thus intensity, except possibly for the flute and trombone played *pianissimo* and the trumpet and violin played *fortissimo*.
2. The attack transient cannot be relied upon as a cue for the dynamic marking or intensity with which an instrument is originally played.

(Clark and Milner, 1964)

In addition, the research of Penttinen (2006) demonstrates that an increase in playing dynamic on a harpsichord has an effect on the perceived timbral characteristics of the instrument (Penttinen, 2006). It could be concluded that the precise nature of the relationship between playing 'intensity' and timbre is complex, but that a loudness feature could give some indication of timbre when combined with other features.

1.8.1.6 Loudness and Frequency

Perceived pitch tends to increase as intensity increases above 3000Hz, and decreases as intensity increases below 2000Hz. Spectral richness also tends to reinforce a sense of pitch, so the more sine-like a spectrum is the more distinct the sense of frequency, but the less distinct the sense of pitch (cf. Gerhard (2003)).

The non-linear frequency response of human ear gives rise to strong correlations between frequency and perceived loudness. This has been researched extensively and can be observed in numerous versions of the 'equal loudness contours', several of which are shown in figure 1.9.

1.8.2 Morphological features

'Acousmatic music' is a nebulous term with several different meanings(McFarlane, 2001). Pierre Schaeffer is considered to have introduced the word 'acousmatique'[16] in reference to the fact that in musique concrete the acoustic sound (heard through loudspeakers) is detached from its original (mechanical) source(Schaeffer, 1966). Chion elaborates on this:

> Acousmatic sound is sound one hears without seeing [its] originating cause - [an] invisible sound source. Radio, phonograph and telephone, all which transmit sounds without showing their emitter are acousmatic media.
>
> (Chion, Gorbman, and Murch, 1994)

[16] Derived from the Greek word 'akusmatik'.

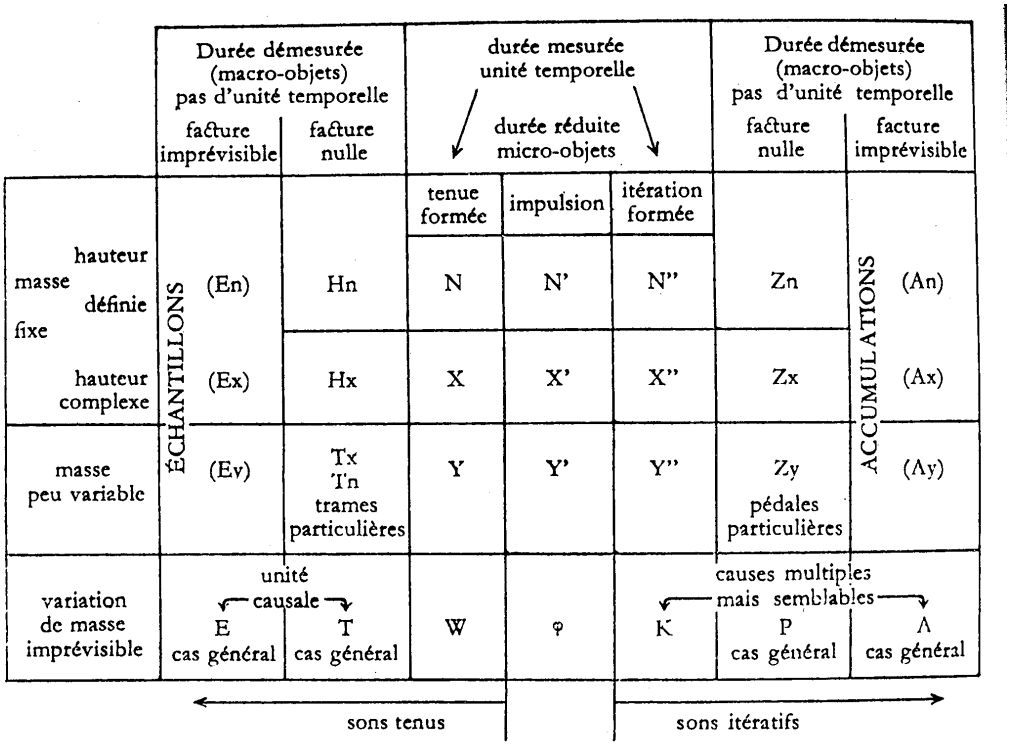


Figure 1.11: Pierre Schaeffer's TARTYP diagram(Schaeffer, 1966, p. 459)

There is debate amongst 'acousmatic music' practitioners about whether the term refers to a compositional style (based on musique concrete) or a way of listening to sound in which we listen to sound's abstract structures without consideration of its origin.

In his *Traité des objets musicaux* Schaeffer describes a form of listening — écoute réduite (reduced listening) — in which the listener focuses on the timbral, rhythmic, textural and structural properties of sound rather than its acoustic origin or symbolic associations. In order to facilitate this abstract mode of listening Schaeffer developed a classification system for better understanding the nature of sonic types. Central to Schaeffer's argument is the notion of *typomorphology*, a taxonomy for identifying sound type and shape as summarised in the TARTYP diagram as shown in figure 1.11.

Unlike the audio features discussed so far TARTYP doesn't present a scientifically grounded taxonomy where the features correlate to measurable acoustic or auditory phenomena. Neither does it present a set of 'semantic' features since the terminology invented by Schaeffer has no established meaning outside of its lexical scope in Scha-

effer's work. Instead Schaeffer takes a phenomenological approach, where Schaeffer's own personal experience of sound as composer and listener is documented as a set of axioms. As Thoresen writes:

> Schaeffer's approach to the world of sound is characterised by a phenomenological attitude: It seeks to describe and reflect upon experience, rather than explain; it posits the actual life world experience of sound as its primary object of research ("la primauté de l'oreille"); it clarifies a number of different listening intentions by which the same physical object may be constituted as various objects in the listener's mind. The capacity to shift between different listening intentions becomes a true sign of the virtuoso listener, and Schaeffer insists that the listener should train his listening even as a musician would train his instrument!
>
> Unfortunately, and for a number of reasons, one of the major achievements of Schaeffer's work, his codification of all sound categories into a grand, unified diagram, remained without much practical consequence.
>
> (Thoresen, 2002)

Thoresen proposes an adaptation of Schaeffer's typomorphology that could serve as a tool for practical aural analysis of electro-acoustic music, proposing a number of new symbols and terms (see figure 1.12). Such a symbolic notation could prove extremely useful — for example in graphical or diffusion scores for Acousmatic music and for notating the *effect* of live electronic processing. However, despite the qualifications Thoresen gives in terms of spectral properties, insufficient information is provided to construct an absolute mapping between symbolic (textual or graphical) and sonic representations.

Figure 1.12: Expanded typology diagram showing novel notation, from Thoresen (2002)

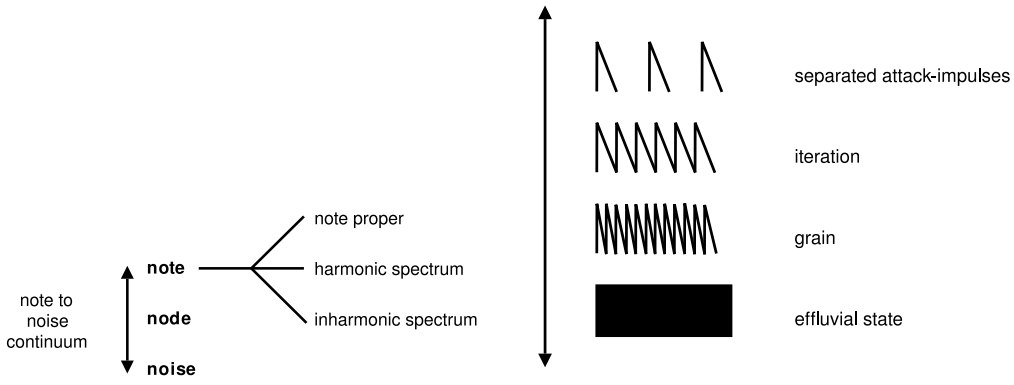
1.8.2.1 Spectro-morphology

In the seminal paper 'Spectromorphology and Structuring Processes' Smalley proposes an alternative approach to Schaeffer's écoute réduite, which shifts the taxonomic emphasis to the sound's measurable spectrum. Like Thoresen and Schaeffer, the analysis of sound and the proposed terminology are approached from a listener-composer perspective, and seek to address problems of vocabulary in acousmatic art:

> The lack of shared terminology is a serious problem confronting electroacoustic music because a description of sound materials and their relationships is a prerequisite for evaluative discussion. In searching for appropriate words we are obliged to borrow non-musical terms because the circumscribed vocabulary invented for purely musical explication is too limited for spectro-morphological purposes. Such semantic borrowings immediately indicate that music involves mimesis: musical materials and structures find resemblances and echoes in the non-musical world. These links with human experience may be obvious, tangible and conscious, or covert, elusive and unconscious. However, to find out what happens in the life of a sound or sound structure, or what attracts us about a sound quality or shape, we must temporarily ignore how the sound was made or what caused it, and concentrate on charting its spectro-morphological progress.
>
> (Smalley, 1986)

Smalley is referring primarily to terminology for describing artistic works in the tradition of *musique concrète* — music for fixed media — however spectro-morphological theory is equally applicable to live electronic music and 'mechanical' instrumental music where pitch is not the primary structural determinant.

Central to Smalley's theory of spectro-morphology is the notion of continua between fixed or clearly defined states. The pitch-effluvium continuum refers to a continuum of states between a point where discrete pitch components are clearly discernible in a sound to a state where 'aural interest is forced away from charting the pitch behaviour of internal components to follow the momentum of external shaping'(Smalley, 1986). In this context, the effluvial state is the point at which individual pitches can no longer be perceived, represented by the 'node' in figure 1.13(a). Another useful continuum identified by Smalley is the 'attack-effluvium continuum'. This describes the transition between the perception of discrete sonic events heard sequentially in time, and



(a) Spectral typology (b) Attack-effluvium continuum

Figure 1.13: Smalley's spectro-morphological continua, from (Smalley, 1986, p. 65-72)

continuous sound with no discernible internal attacks (see figure 1.13(b). This transition can be reproduced using granular synthesis techniques by increasing the *grain rate* from a setting where individual grains can be heard (2 grains per second) to a setting where a continuous sound is perceived (50 grains per second). Unfortunately straightforward mappings between terminology and practical examples do not exist for many of Smalley's other terms. However the spectro-morphological concept still serves as a useful starting point for relating compositional structures and measurable audio features.

There is currently no general agreement on standard terminology or symbolic notation for describing sound on an abstract level in electro-acoustic music. The community would certainly benefit from a descriptive vocabulary of symbols and or terms especially if the definitions were clear and linked to acoustically measurable features.

1.8.3 Representing time

All of the features mentioned so far: low-level descriptors and perceptual features are to a certain extent a function of time. Certain aspects of timbre for example can't be measured if one assumes that the spectral content of a sound stays constant throughout its duration. Change over time is also inherent in typo-morphology and spectro-morphology. The approach taken in the author's research is to develop a robust system for the extraction of pseudo-instantaneous features in real-time and then to use these features as a basis for measuring morphology as required. For example, a system that uses the second order derivative of a number of features over time is described in section 4.4.

1.8.4 'Higher level' features

In general 'higher level' feature extraction is taken to mean that the feature being extracted carries more semantic meaning than 'lower level' features (Zhang and Chen, 2003). In addition, Herrera et al. (1999) propose a distinction between 'syntactic' and 'semantic' features where the syntactic level is concerned with the specification of physical structures and the signal properties, and the semantic level is concerned with audio events and audio objects.

Examples of semantic descriptors typically include adjectival descriptions such as 'cheerful', 'sad', 'dynamic' and 'harmonious' (Lesaffre, Leman, Devoogdt, De Baets, De Meyer, and Martens, 2006). A conceptual framework illustrating the complexity of the relationship between various feature levels, and their symbolic meanings is shown in table 1.2.

Despite the fact that some authors (Lesaffre et al., 2006; Lesaffre, Leman, Tanghe, Baets, Meyer, and Martens, 2003) have shown that listeners' perception of musical content can be inter-subjectively consistent, such descriptors certainly make a range of (possibly invalid) socio-cultural assumptions (Baumann, Pohle, and Shankar, 2004). These assumptions could include factors about the age, musical education, nationality and general cultural experience of the listener. Leman, for example acknowledges that

> High level concepts form part of the expressive gestural semantics which people use when they speak about music. The structure is still badly understood, and the inter-subjective consistency of this semantics has rarely been investigated on a large scale.
>
> (Leman, Vermeulen, Voogdt, Taelman, Moelants, and Lesaffre, 2003)

Indeed, the framework proposed in table 1.2 does make a number of assumptions about what constitutes 'musical content', categorising features into *melody*, *harmony*, *rhythm*, *source*, and *dynamics* at the 'formal' concept level. Assuming that *spectral envelope* in table 1.2 corresponds approximately to low-level spectro-morphology (cf. Smalley (1986)), it would be interesting to explore where *gesture*, *texture*, *spatio-morphology* and other of Smalley's concepts fit into the high-level feature space of Leman et al. (2003).

On a purely technical level, 'higher level' features can be extracted from vectors of 'lower level' features by using an additional dimension reduction stage. This process is explored further in chapter 4.

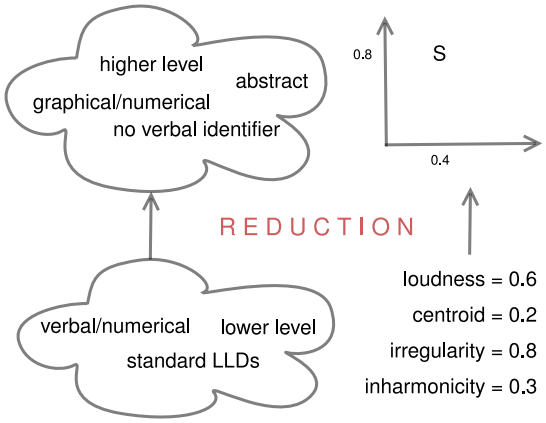


Figure 1.14: Dimension reduction for the purpose of live electronics control

1.8.5 Feature similarity

For the purposes of using audio as a real-time control source it will be investigated in this thesis how applicable semantic and morphological taxonomies really are. For example, in the majority of cases the semantic domain will be bypassed altogether because all we need is a musically meaningful mapping between audio and resultant control data. It might however, be fruitful to pursue purely visual, non-verbal means of representing the data. This has at least two central areas of investigation:

- Data visualisation
- Composer/Performer > computer interaction

Rather than rely on the concept of semantic or 'high level' features as discussed in section 1.8.4, which describe a correlation between a perceived sound and an inter-subjective judgement about its 'qualities', or a morphological description as outlined in section 1.8.2, it is proposed that a more abstract metric is used for the purposes of musical control. A hypothetical dimensionality reduction relationship between lower and higher level features is shown in figure 1.14.

STRUCT		CONCEPT LEVEL		MUSICAL CONTENT FEATURES				
CONTEXTUAL	global >3 sec	HIGH II	EXPRESSIVE	cognition | emotion | affect = *syntactic+semantic concepts*				
				melody	*harmony*	*rhythm*	*source*	*dynamics*
		HIGH I	FORMAL	key profile	tonality cadence	rhythmic patterns tempo	instrument voice	trajectory articulation
	global <3 sec	MID	PERCEPTUAL	successive intervallic pattern	simultane intervallic pattern	beat IOI	 spectral envelope 	dynamic range sound level
NON-CONTEXTUAL	local + spatial	LOW II	SENSORIAL	*pitch*		*time*	*timbre*	*loudness*
				periodicity pitch pitch deviations		note duration onset	roughness spectral flux	neural energy
	local + temporal	LOW I	PHYSICAL	fundamental frequency		offset	spectral centroid	peak
				frequency		*duration*	*spectrum*	*intensity*

Table 1.2: Conceptual framework used within the context of audio mining. A distinction is made between low-level, mid-level and high-level audio descriptors(Leman et al., 2003)

1.9 Conclusions

In this chapter I have identified audio feature extraction as a possible alternative to other sources of control data in systems for live electronic music. A review of available features has been conducted along with a critique of existing approaches. It has been stated that the primary concern of this thesis is to investigate the applicability of audio feature extraction to musical usage. It has also been stated that part of the research outcomes for this project will be free and open source software relating to audio feature extraction.

In the following chapter I will describe the process of developing a new system for audio feature extraction specifically for use in music with live electronics.

CHAPTER 2

Extraction

The most important single aspect of software development is to be clear about what you are trying to build.

— Bjarne Stroustrup

2.1 Introduction

Feature extraction could be defined as a form of data processing that takes a set of values and returns a more compact representation of those values. The compact representation is called a feature, and the initial set of values could be referred to as the input vector. The process of feature extraction is a form of dimension reduction, because it involves the mapping of an input vector of dimension N onto an output scalar or vector that has dimensionality that is smaller than N.

Feature extraction is a process of abstraction, and in practice the abstracted feature is supposed to describe some useful aspect of the original data. For this reason, features are sometimes referred to as 'descriptors' (cf. section 2.6.2). There are many formally defined audio features (cf. section 2.5), and new features emerge as authors combine existing features or develop new algorithms. One of the key tasks for researchers is therefore to determine which features are the most salient or useful in a given context.

This has been explored in Jensenius (2002) and Aucouturier and Pachet (2004), and a practical example of feature selection for a specific live electronics setup is given in section 4.2.2.3.

In this chapter I will discuss the libXtract feature extraction library, developed as an integral part of this research. It will be compared to existing systems, and the rationale for its development will be given along with detailed description of the feature extraction functions it includes. Chapter 4 presents case studies in which libXtract has been used in practical composition and performance of live electronic music.

2.2 LibXtract

LibXtract is a cross-platform, free and open-source software library that provides a set of feature extraction functions for extracting low-level descriptors (LLDs) from audio data blocks. It is intended to be fast enough to be used for real-time audio, and flexible enough to extract features from non-audio data. The eventual aim is to provide a superset of the MPEG-7 and Cuidado audio features (cf. section 1.6). The library is written in ANSI C and licensed under the GNU GPL so that it can easily be incorporated into any program that supports linkage to shared libraries. The full API documentation is provided in appendix C.

2.3 Existing systems

There are a number of existing systems that either provide functionality that overlaps with that of LibXtract, or a similar functionality, but with a different emphasis in usage. I will discuss some of these systems, providing a context to the motivation for the development of LibXtract, and evaluating the contribution it makes.

2.3.1 Aubio

One project related to libXtract is the Aubio library[1] by Paul Brossier. Aubio is designed for audio labelling, and includes excellent pitch, beat and onset detectors(Brossier, 2006). libXtract doesn't currently include any onset detection, this makes Aubio complimentary to libXtract with minimal duplication of functionality. Front-ends to Aubio include the Audacity audio editor, the Aubio Vamp plugin and the Aubio Python bindings.

[1] `http://aubio.piem.org`

Aubio's functionality is available as a set of Pd externals making it suitable and easily available for use in live electronics systems.

2.3.2 jAudio

libXtract has much in common with the jAudio project (McEnnis, McKay, Fujinaga, and Depalle, 2005), which seeks to provide a system that 'meets the needs of MIR researchers'[2] by providing 'a new framework for feature extraction designed to eliminate the duplication of effort in calculating features from an audio signal'. jAudio provides many useful functions such as the automatic resolution of dependencies between features, an API which makes it easy to add new features, multidimensional feature support, and XML file output. Its implementation in Java makes it cross-platform, and suitable for embedding in other Java applications, such as the promising jMIR software. However, libXtract was written due to a need to perform real-time feature extraction on live instrumental sources, and jAudio is not designed for this task. libXtract, being written in C, also has the advantage that it can be incorporated into programs written in a variety of languages, not just Java. Examples of this are given in section 2.7.

2.3.3 CLAM

According to McEnnis et al. (2005), CLAM[3] is 'a framework that aims at offering extensible, generic and efficient design and implementation solutions for developing Audio and Music applications as well as for doing more complex research related with the field'. However, as noted by McEnnis et al. (2005), 'the [CLAM] system was not intended for extracting features for classification problems', it is a large and complex piece of software with many dependencies, making it a poor choice if *only* feature extraction functionality is required.

2.3.4 Marsyas

Marsyas[4] started as a free software framework for building and integrating audio analysis tools (Tzanetakis and Cook, 2000). Subsequently functionality for a wide range of 'Computer Audition'-related tasks has been added to the framework as shown in table 2.1. The architecture of Marsyas is designed to provide a flexible, modular and abstracted approach providing a range of 'building blocks' that can be combined in various ways (Tzanetakis, 2002). It presents a useful balance between a large framework like CLAM, which has a number of superfluous components if only feature extraction is required,

[2] MIR (Musical Information Retrieval) is an interdisciplinary field concerning the retrieval of information from music-related data

[3] `http://clam.iua.upf.edu`

[4] `http://marsyas.sf.net`

	Marsyas	LibXtract
Analysis functions	>=37	>=60 (17 of these are common to Marsyas)
Language	C++	ANSI C
Software type	Framework	Library
File I/O	Yes	No
Network I/O	Yes	No
MIDI I/O	Yes	No
Filters/delays/envelopes	Yes	No
Windowing	Yes	Yes
Scheduling	Yes	No
Machine Learning	Yes	No
Application domain	Computer Audition	Feature extraction
Soft real-time	Yes	Yes
Language bindings	Lua, Python, Java, Ruby	Python, Java, Pure Data, Max/MSP
Dependencies	libMAD (optional)	libfftw (optional)
FFT	Built-in	external

Table 2.1: Comparison between libXtract and marsyas

and libXtract, which takes a 'lean and mean' approach by providing *only* feature extraction functionality. Table 2.1 illustrates the distinctions between the two projects.

Marsyas is a promising project providing a 'one stop' low-dependency framework for computer audition, and is used in a number of other projects including SndPeek (cf. section 3.3.3 and Marsyas3D(Tzanetakis and Cook, 2001).

2.3.5 analyzer~

`analyzer~` is a Max/MSP object written by Tristan Jehan (cf. section 1.7.4. It outputs the perceptual features: pitch, loudness, brightness, noisiness, onsets, and Bark scale decomposition (Jehan, 2001). The pitch detection method is based on the `fiddle~` algorithm described in detail in section 4.2.2.1. One of the advantages of the `analyzer~` object is its ease of use, since signal windowing, FFT analysis, spectrum computation and feature extraction are all done 'inside' the object without the user necessarily needing to know about their workings. The popularity of `analyzer~` is evidenced by references in a number of papers including (but not limited to) Hsu (2005), Borchers and Muhlhauser (1998) and Bell et al. (2007). The success of `analyzer~` is evidence of the utility of feature extraction functionality in the context of environments for live electronic music (Max/MSP), however it is unfortunate that `analyzer~` is a 'closed source'

project and therefore it is not easily adaptable to other systems.

2.3.6 FEAPI

FEAPI (Feature Extraction API) is described by its authors as 'an easy-to-use platform-independent plugin application programming interface (API) for the extraction of low level features from audio in PCM format in the context of music information retrieval software' (Lerch, Eisenberg, and Tanghe, 2005). The FEAPI distribution consists of host and plugin SDKs and a number of example plugins that extract features such as 'Maximum Value', 'Zero Crossings', 'Spectral Flux' and 'Spectral Centroid' from PCM audio. FEAPI seeks to resolve a number of problems that exist within the MIR (musical information retrieval) software development community such as code duplication between projects, incompatible APIs and subtly different implementations by providing a simple and robust plugin architecture. There exists a FEAPI host called `feapi~` for Max/MSP which is available in the FEAPI subversion repository[5]. There is little evidence to suggest that FEAPI has been widely adopted outside of the FEAPI development community, and anecdotal evidence suggests that it may be superseded by the Vamp API (cf. section 2.3.7).

2.3.7 Vamp

The Vamp analysis plugin API was devised by Chris Cannam for the Sonic Visualiser[6] software. Sonic Visualiser is described in detail in section 3.3.2.

Vamp is an analysis plugin format that draws some influence from LADSPA[7] and DSSI[8]. However, whilst these and other formats take audio or control data as input, and output audio data, Vamp outputs analysis data. According to Cannam, Landone, Sandler, and Bello (2006) Vamp plugins are dynamic link libraries that process sampled audio data, returning complex multidimensional data with labels representing semantic observations. Example applications include audio feature extraction, onset detection and beat tracking with labelling. Current Vamp plugin hosts include Audacity[9], Ardour[10] and Sonic Visualiser.

[5] `http://feapi.svn.sourceforge.net/viewvc/feapi/trunk/FEAPI/`

[6] `http://www.sonicvisualiser.org`

[7] LADSPA (Linux Audio Developer's Simple Plugin API) is plugin and plugin host development API originally developed for, but not limited to the Linux platform.

[8] DSSI (Disposable Soft Synth Interface) is an extension of LADSPA that allows for plugins with with graphical user interfaces and MIDI control.

[9] `http://http://audacity.sourceforge.net/`

[10] `http://ardour.org/`

Name	Description
Queen Mary plugin set	Note onset detector, beat tracker, tempo estimator, key estimator, tonal change detector, structural segmenter, timbral and rhythmic similarity estimator, chromagram, constant Q spectrogram and MFCC calculation
Vamp Aubio plugins	Onset detection, pitch tracking, note tracking and tempo tracking plugins using Paul Brossier's aubio library
Mazurka plugins	Spectral visualisation and feature extraction plugins from the Mazurka project
Vamp libXtract plugins	Low-level feature extraction plugins using Jamie Bullock's libXtract library to provide around 50 spectral and other features
MATCH Vamp plugin	Vamp implementation of the MATCH audio alignment algorithm from Simon Dixon. Sonic Visualiser v1.2 can use this for automatic alignment
OFA Vamp plugin	Plugin that performs audio fingerprinting and lookup using the MusicIP OFA library
Vamp example plugins	A small set of simple plugins as included with the Vamp developers kit. Amplitude tracker, simple percussion onset detector, spectral centroid, and zero-crossing counter

Table 2.2: Some of the currently available Vamp plugins and libraries (Source `http://www.vamp-plugins.org/`)

2.3.7.1 Vamp plugins

Compared to FEAPI, Vamp plugin development has been relatively prolific, with at least five plugin libraries and two sophisticated singleton plugins made available since the project was registered (with SourceForge) in March 2006[11]. Some of the more significant plugins are shown in table 2.2.

2.3.7.2 Vamp caveats

One of the main problems with Vamp, with regard to its use in live electronic music is that it was not designed for real-time use. As noted in Cannam (2008):

> Plugins are not supplied with the whole of their audio input at once, but instead are fed it as a series of short blocks: in this respect they resemble real-time audio processing plugins, but in all other respects they are executed "off-line", meaning that they are not expected to run in real-time and are not obliged to return features as they occur.
>
> (Cannam, 2008)

And later in the same document:

> Vamp plugins do not actually do any display or user interaction: they just return data. In most cases, these data are not the final result of the work the

[11] Figures recorded in May 2008

> host is doing, but are useful for something else – either for a host to display and the user to interact with in some way, or as intermediate calculations for a particular purpose such as semantically meaningful audio editing. Vamp plugins do not do swirly visualisation and are typically not useful in real-time contexts such as interacting with musical performers.
> (*ibid.*)

It can therefore be concluded that whilst Vamp provides a useful format for 'pre-analysis' in the context of live electronic music, it is not an ideal solution for live interaction.

2.3.8 MatLab implementations

There currently exist a number of MatLab feature extraction libraries, which are designed for research purposes, particularly in the field of music information retrieval (MIR). These include RPextract[12], Goldsmiths College MPEG-7 Audio Reference Software Toolkit[13], CATbox[14] and the Cuidado audio features (cf. Peeters (2004)). Many of these projects provide usful reference implementations, for example the Goldsmiths project claims ISO-IEC 15938-4 conformance. However since MatLab is rarely used in live electronics systems, these projects are unlikely to gain usage in the context of live electronic music other than (possibly) for comparison purposes.

2.3.9 FASTlab

The FASTLab Music Analysis Kernel[15] provides an extensive set of functionality for audio feature extraction, segmentation, between-frame tracking, labelling and integration with a database for persistent storage. The initial target application for the system was a mapping expert system that used derived features to plan several stages of signal processing for music mastering and post-production(Pope, Holm, and Kouznetsov, 2004). FFT, LPC and Wavelet based features are included.

In Pope et al. (2004), the authors suggest that the emphasis in 'feature extraction and storage' research should shift from MIR - that is explicitly 'information retrieval' to a more generic field of 'music meta-database' (MDB) systems. They conclude that 'a

[12] `http://www.ifs.tuwien.ac.at/mir/muscle/del/audio_extraction_tools.html#RPextract`

[13] `http://mpeg7.doc.gold.ac.uk/`

[14] `http://music.ucsd.edu/~sdubnov/CATbox/CATbox.htm`

[15] `http://fastlabinc.com/`

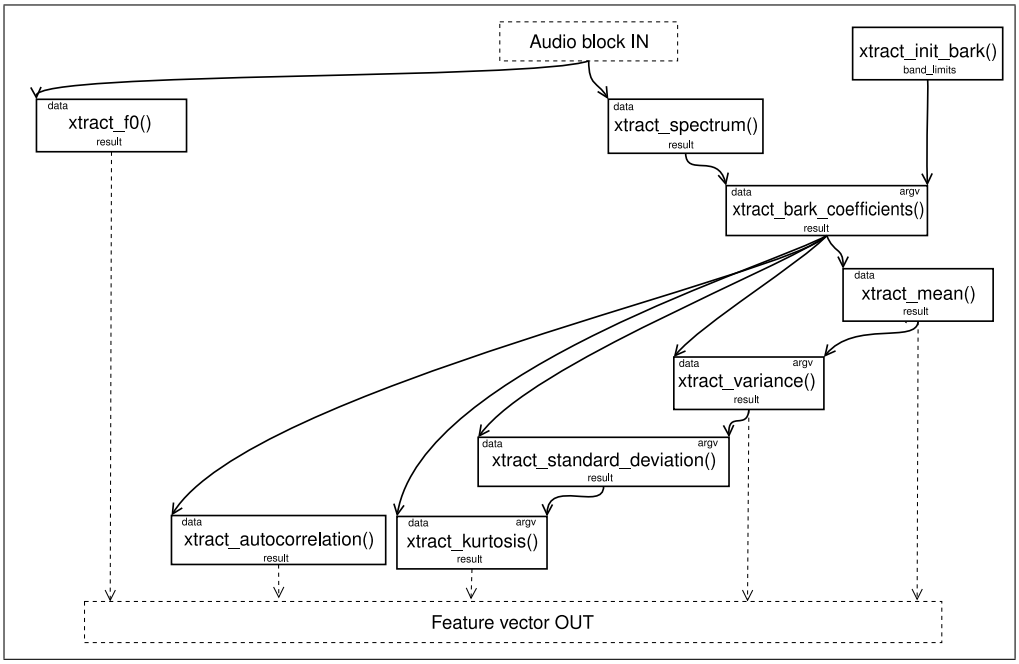


Figure 2.1: Example libXtract call graph showing quasi-arbitrary passing of data between functions

formal engineering discipline of feature vector design is needed for the field of music/-sound databases to progress beyond its present ad-hoc stage'. In the following section I will outline some of the design considerations in making the libXtract library that go some way to moving away from an ad-hoc approach.

2.4 Library design

The main idea behind libXtract is that the feature extraction functions should be modularised so they can be combined arbitrarily. Central to this approach is the idea of a cascaded extraction hierarchy and a simple example of this is shown in Figure 2.1. This approach serves a dual purpose: it avoids the duplication of 'sub-features', making computation more efficient, and if the calling application allows, it enables a certain degree of experimentation. For example the user can easily create novel features by making unconventional extraction hierarchies.

LibXtract seeks to provide a simple API for developers. This is achieved by using an array of function pointers as the primary means of calling extraction functions. A consequence of this is that all feature extraction functions have the same prototype for

their arguments. The array of function pointers can be indexed using an enumeration of descriptively-named constants. A typical libXtract call in the DSP loop of an application will look like this:

```
1 xtract[XTRACT_FUNCTION_NAME](
2     input_vector, blocksize, argv, output_vector);
```

This design makes libXtract particularly suitable for use in modular patching environments such as Pure Data and Max/MSP, because it alleviates the need for the program making use of libXtract to provide mappings between symbolic 'xtractor' names, and callback function names, significantly reducing the amount of 'wrapper' code required. Since the primary goal of the library is that it should be used within systems for composition and performance of live electronic music, support for these (commonly used) environments is especially important.

LibXtract divides features into *scalar* features, which give the result as a single value, *vector* features, which give the result as an array, and *delta* features, which use two data frames as input and provide either a scalar or vector as output. To make the process of incorporating the wide variety of features (with their slightly varying argument requirements) easier, each extraction function has its own function descriptor. The purpose of the function descriptor is to provide useful 'self documentation' about the feature extraction function in question. This enables a calling application to easily determine the expected format of the data pointed to by `*data` and `*argv`. It also enables the 'donor' functions for any sub-features (passed in as part of the argument vector) to be determined allowing the programmatic construction of feature dependency graphs.

The library has been written on the assumption that a contiguous block of data will be written to the input array of the feature extraction functions. Some functions assume the data represents a block of time-domain audio data, others use a special spectral data format, and others make no assumption about what the data represents. Some of the functions may therefore be suitable for analysing non-audio data.

2.4.1 Why write a library?

The libXtract project was started due to the limited availability of feature extraction functionality for commonly used environments for audio and multimedia such as Pd, Max/MSP, SuperCollider and Chuck. Some of the functions provided in libXtract started off as Pd 'externals' in the flib[16] library. However, it became apparent that developing a large amount of functionality for one specific environment would not be beneficial to the broader live electronic music community. An alternative to this 'bespoke' approach (also taken in Marsyas and jMIR) would have been to create a feature extraction plugin API, and then to write the feature extraction functions as plugins (an approach subsequently taken by FEAPI and Vamp projects). However it was decided that the best use of time, providing maximum flexibility (including the ability to later wrap the library as plugins) would be to develop a C library of functions with a very 'lightweight' API. This has subsequently proved to be a good decision since libXtract is now available in a variety of publicly available implementations (cf. section 2.7).

2.5 Feature List

In the following subsections, an explanation of each feature included in the libXtract library will be provided along with the mathematical basis for each feature. Where appropriate I will also discuss implementation issues, and the features' perceptual correlates. Features will be presented in the order that they appear in the API (enumeration xtract_features_) see appendix C.

2.5.1 Scalar features

Scalar feature extraction functions are functions that return a single floating point value as their result. In general the scalar extraction functions fall in to one of two categories: *standard* scalar functions, which take an arbitrary array of floating point values as input and *spectral* scalar functions, which expect the data to be in a specific format. LibXtract's spectral data format consists of an array of $N/2$ magnitude, power, log power, or log magnitude coefficients followed by $N/2$ corresponding frequency values where N is the size of the input array. This data type trades memory for speed by storing the frequency values rather than computing them on-the-fly.

[16] 'flib', short for 'feature extraction library' is still available in the Pd CVS repository, but is now deprecated in favour of xtract˜

2.5.1.1 Standard statistical features

The following features are based on standard statistical functions, that is the input vector is assumed to represent a 'distribution' of length N. Because of the flexible nature of libXtract, these functions can be used arbitrarily on the vector outputs of other feature extraction functions. For example, the mean of the magnitude spectrum, or the skewness of the first five bark coefficients could be taken.

In the following equations x_k is the value of the kth element in the input vector, N is the number of elements considered.

Mean

$$\bar{x} = \frac{1}{N}\sum_{k=1}^{N} x_k \tag{2.1}$$

The mean value in libXtract is the arithmetic mean (average) of a range of values. It has the same unit as the original values. The mean of a distribution is shown graphically in figure 2.2. When used in conjunction with xtract_spectrum () supplied with the XTRACT_POWER_SPECTRUM constant it can be used to compute the 'average energy' feature.

Variance

$$Variance = \frac{1}{N-1}\sum_{k=1}^{N}(x_k - \bar{x})^2 \tag{2.2}$$

The variance of a distribution of values is a measure of the extent to which values in a distribution are spread or scattered(Rencher, 2002). It is sometimes referred to as the 'second order moment'. The unit of a variance metric is the square of the unit of the original values. The variance of a distribution is shown graphically in figure 2.2.

Standard Deviation

$$\sigma = \sqrt{Var(x_1 \ldots x_N)} \tag{2.3}$$

Like the variance, standard deviation indicates the 'spread' of a distribution. However, standard deviation is often more convenient because the unit is the same as that of the original values. The standard deviation is shown graphically in figure 2.2.

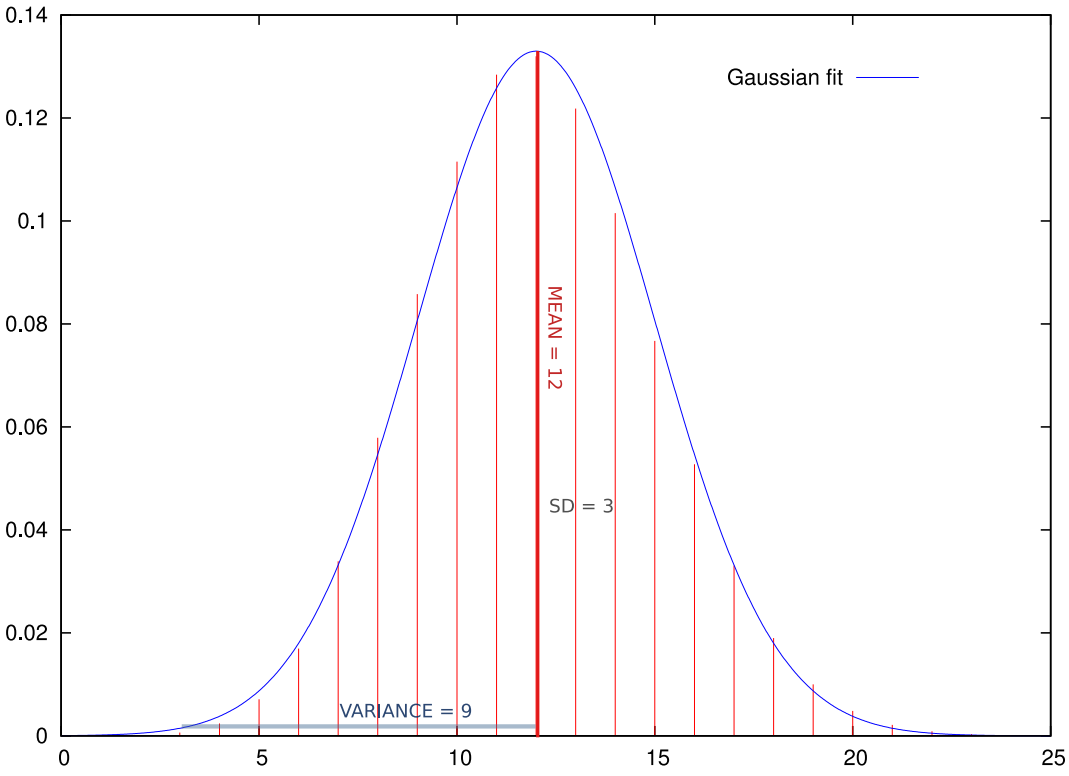


Figure 2.2: Distribution graph showing mean, standard deviation, and variance

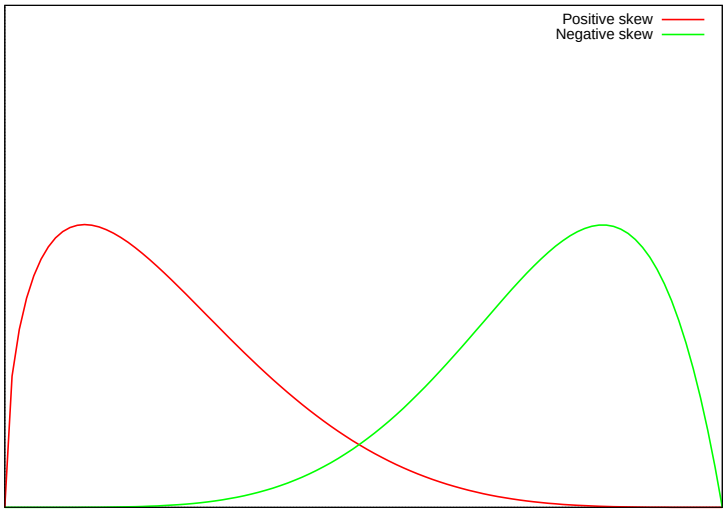


Figure 2.3: Distributions with positive and negative skewness

Average Deviation

$$AverageDeviation = \frac{1}{N}\sum_{k=1}^{N}|x_k - \bar{x}| \tag{2.4}$$

Average deviation is the mean of the absolute values of the distances from the values in a distribution from the mean of the distribution. It gives an alternative variability metric.

Skewness

$$Skewness = \frac{1}{N}\sum_{k=1}^{N}[\frac{x_k - \bar{x}}{\sigma}]^3 \tag{2.5}$$

The Skewness or 'third order moment' is a measure of the asymmetry of a distribution. Figure 2.3 gives examples of positive and negative skewness. Spectral skewness (cf. section 2.5.1.2) can be used to give an idea of the relative pitched-ness of audio, where a positively skewed spectrum could indicate a weak or missing fundamental.

Kurtosis

$$Kurtosis = \{\frac{1}{N}\sum_{k=1}^{N}[\frac{x_k - \bar{x}}{\sigma}]^4\} - 3 \tag{2.6}$$

Kurtosis or 'fourth order moment' is a measure of the relative flatness of a distribution. A distribution with negative kurtosis is characterised by being flatter than the

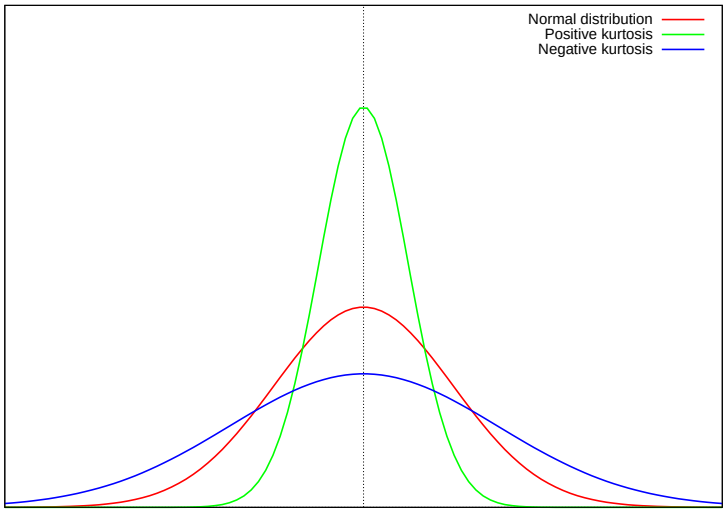


Figure 2.4: Distributions with positive and negative kurtosis

normal[17] distribution, that is, less peaked. A distribution with positive kurtosis has a higher peak than normal with an excess of values near the mean(Rencher, 2002). A graphical representation of various kurtosis 'states' is shown in figure 2.4.

2.5.1.2 Spectral statistical features

The following features are based on standard statistical functions except that the input vector is assumed to be an array containing $N/2$ magnitude/power coefficients and $N/2$ corresponding frequency values. The features are identical to the statistical features in section 2.5.1.1 except that each magnitude component is weighted by its corresponding frequency value. Note also that the first 'bin', which often corresponds to the DC component can optionally be excluded from the input data as it can skew the results.

In the following equations a_k is the magnitude of the kth spectral component, f_k is its frequency and N is the number of 'bins' considered.

Spectral Mean

$$\bar{s} = \frac{1}{\sum_{k=1}^{N} a_k} * \sum_{k=1}^{N} f_k a_k \tag{2.7}$$

Spectral Variance

$$V_s = \frac{1}{\sum_{k=1}^{N} a_k} * \sum_{k=1}^{N} (f_k - \bar{s})^2 * a_k \tag{2.8}$$

[17] The phrase 'normal distribution' is used to refer to a Gaussian distribution

Spectral Standard Deviation

$$\sigma_s = \sqrt{V_s} \tag{2.9}$$

Spectral Skewness

$$SS = \frac{\sum_{k=1}^{N}(f_k - \bar{s})^3 * a_k}{\sigma_s^3} \tag{2.10}$$

Spectral Kurtosis

$$SK = \left(\frac{\sum_{k=1}^{N}(f_k - \bar{s})^4 * a_k}{\sigma_s^4}\right) - 3 \tag{2.11}$$

2.5.1.3 Spectral Centroid

The spectral centroid (Wessel, 1979), sometimes referred to as the 'centre of gravity' of an audio spectrum (Mcadams, 1999) is mathematically equivalent to the Spectral Mean. It is one of the most commonly used features in live electronic music as evidenced by the number of papers that refer to its usage (see (Hoffman and Cook, 2007; Hsu, 2005; Jehan, Machover, and Fabio, 2002)). This popularity is most likely due to its correlation with the 'brightness' percept, its ease of implementation, and the availability of implementations in software commonly used for live electronics. Such implementations include `analyzer~` for Max/MSP as described in section 2.3.5 and MCLD UGens[18] for SuperCollider. The spectral centroid is used in the feature extraction systems described in sections 4.3 and 4.5.

2.5.1.4 Irregularity

The Irregularity feature is usually computed using the magnitude coefficients of an audio spectrum. It gives an indication of the noise content of a spectrum. There exist two methods for computing Irregularity, one by Jensen (Jensen, 1999), and another by Krimphoff (Krimphoff, McAdams, and Winsberg, 1994) which predates it. The relationship between the two functions can be seen clearly in figure 2.5.

Irregularity – Krimphoff(1994)

$$Irregularity = \sum_{k=2}^{N-1} |a_k - \frac{a_{k-1} + a_k + a_{k+1}}{3}| \tag{2.12}$$

Irregularity – Jensen(1999)

$$Irregularity = \frac{\sum_{k=1}^{N}(a_k - a_{k+1})}{\sum_{k=1}^{N} a_k^2} \tag{2.13}$$

[18] `http://mcld.co.uk/supercollider/`

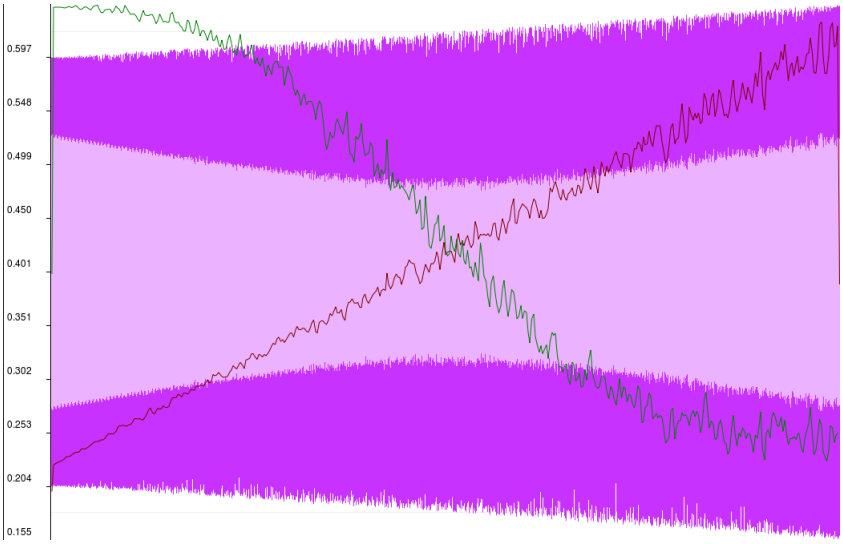


Figure 2.5: Comparison between Irregularity (Krimphoff), Red, and Irregularity (Jensen), Green, for a linear cross-fade between a 440Hz sine wave and white noise

2.5.1.5 Tristimulus

The tristimulus features are based on ratios between bands of harmonics in the spectrum, and the harmonic spectrum as a whole. They were originally proposed in Pollard and Jansson (1982) and are based the concept of tristimulus as it is defined in colour theory where RGB values are represented using a 3-dimensional graph. In the tristimulus functions, the magnitude coefficients of the harmonic spectrum are divided into three bands:

- low order
    1. The fundamental frequency, f_0
    2. The 2nd, 3rd and 4th harmonics
- high order
    3. The 5th harmonic upwards

The relationship between these can be visualised on a triangular graph, where each corner represents an extremum of the tristimulus equations, see figure 2.7. Tristimulus features have been used in a variety of contexts including the analysis of sounds in mother/infant communication (Malloch, Sharp, Campbell, Campbell, and Trevarthen,

Figure 2.6: Korg Kaos pad

1997), MPEG-7 coding (Herrera et al., 1999) and audio synthesis using a Korg Kaos Pad (see figure 2.6) to control the location in two-dimensional space (Riley, 2004).

Tristimulus – low order

$$Tristimulus1 = a_1 / \sum_{k=1}^{N} a_k \tag{2.14}$$

$$Tristimulus2 = (a_2 + a_3 + a_4) / \sum_{k=1}^{N} a_k \tag{2.15}$$

Tristimulus – high order

$$Tristimulus3 = \sum_{k=5}^{N} a_k / \sum_{k=1}^{N} a_k \tag{2.16}$$

2.5.1.6 Spectral Smoothness

Spectral Smoothness (Mcadams, 1999) is equivalent to the Irregularity feature as defined by Krimphoff (cf. section 2.5.1.4) except that log scaling is applied to the magnitude coefficients. According to McAdams:

> Spectral smoothness is related to the degree of amplitude difference between adjacent partials in the spectrum computed over the duration of the tone. A trumpet often has a smooth spectrum and a clarinet a jagged one,

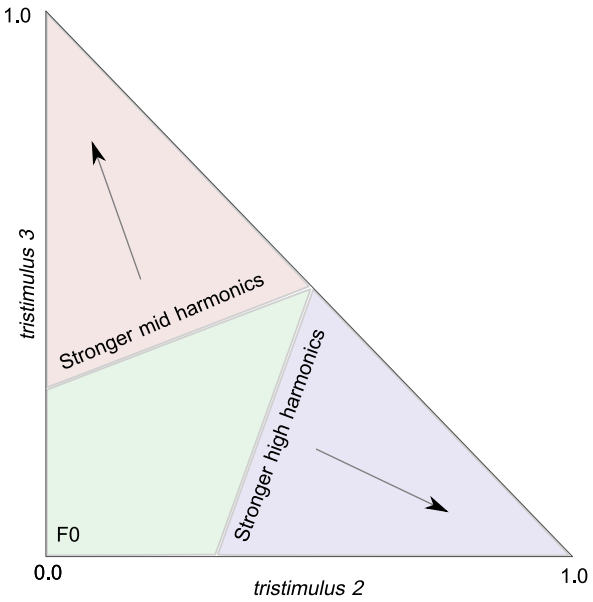


Figure 2.7: Audio tristimulus graph

so the former would have a low value of SS and the latter a higher one. (Mcadams, 1999)

The formula for spectral smoothness is as follows:

$$ss = \sum_{k-1}^{N-1} |20log(a_k) - \frac{20log(a_{k-1}) + 20log(a_k) + 20log(a_{k+1})}{3}| \tag{2.17}$$

The practical implementation of the spectral smoothness function is significantly less efficient than that of Irregularity due to the computationally expensive log function in the inner loop.

2.5.1.7 Spread

The Spectral Spread feature is mathematically equivalent to the spectral variance as described in section 2.5.1.2.

2.5.1.8 Zero Crossing Rate

Zero crossing rate (ZCR) is computed using the time domain representation of a signal, and provides a simple measure of the number of times the signal crosses the zero axis. It gives an approximation of the 'noisiness' of a signal, and has applications in distinguishing sibilants, plosives and fricatives from vowel sounds in speech. According to Herrera-Boyer, Klapuri, and Davy (2006) ZCR has been shown to correlate strongly

with the Spectral Centroid feature. Furthermore, Gouyon, Pachet, and Delerue (2000) demonstrates that the ZCR is effective for the classification of percussive sounds. It is given mathematically as:

$$r = \frac{\sum_{k=1}^{N} I}{N} \qquad \{x_k x_{k-1} < 0\} \tag{2.18}$$

Where x_k is the kth sample in a frame of N samples and I is an indicator whose value is 1 if the conditional expression returns true, and 0 otherwise

2.5.1.9 Spectral Rolloff

The Spectral rolloff point is the frequency that corresponds to the kth percentile of the power spectrum below which k% of the energy resides. It could be considered as a measure of the skewness of the spectral shape, the value being higher for right-skewed distributions (Ong, 2005; Park, 2004). Perceptually, if the signal were considered as being subject to a low-pass filter, the spectral rolloff point would correspond to the cutoff frequency of the filter. According to Peeters, it is correlated to the 'harmonic/noise cutting frequency' (Peeters, 2004). The spectral rolloff is given by the following formula where f_c is the spectral rolloff frequency, $sr/2$ is the Nyquist frequency and k is the percentile. Typically the value of k is 95%, but some authors such as Park (2004) use a lower value such as 85%. In libXtract the value of k can be set arbitrarily. It can be used for differentiating sounds that have more energy in the higher frequencies (such as percussive sounds), by setting a detection threshold (in Hz) following the rolloff function. It can also be used for real-time 'brightness following' as described in section 4.3.2.3.

$$\sum_{0}^{f_c} a^2(f) = k/100 \sum_{0}^{sr/2} a^2(f) \tag{2.19}$$

2.5.1.10 Loudness

The loudness feature in libXtract is based on a Bark band representation of the audio signal (cf. 2.5.2.7), and is based on Peeters' adaptation of total and specific loudness as originally defined in Moore, Glasberg, and Baer (1997).

$$N'(z) = E(z)^{0.23} \tag{2.20}$$

Here $N'(z)$ is the specific loudness of the zth Bark band(Peeters, 2004). The total loudness is given by the sum of the individual loudness coefficients.

2.5.1.11 Spectral Flatness measure

The Spectral Flatness measure (SFM) is defined as the ratio between the geometric mean and the arithmetic mean of an audio spectrum. The SFM gives an indication of the relative noisiness of an audio spectrum. A low value for spectral flatness indicates that the spectrum is more sinusoidal, whereas a high value indicates a flatter, more decorrelated spectrum. It is usually computed using a small number of frequency bands rather than the entire spectrum. Peeters (2004) gives the following band boundaries:

- 250 to 500Hz
- 500 to 1000Hz
- 1000 to 2000Hz
- 2000 to 4000Hz

The SFM can be expressed mathematically as:

$$SFM = \frac{(\prod_{k \in N} a_k)^{1/K}}{\frac{1}{K} \sum_{k \in N} a_k} \tag{2.21}$$

Here a_k is the amplitude in frequency band number k of N bands. The practical implementation of the SFM can be problematic as described in section 2.5.1.12

2.5.1.12 Tonality Factor

The SFM feature defined in section 2.5.1.11 can be used to calculate the tonality factor as described by Johnston (1988). This gives a further indication of the presence or absence of dominant sinusoidal components. The tonality value α can be calculated as shown in equation 2.22. With SFM_{dBMax} set to $-60dB$, the closer α is to 1, the more 'tonal' the spectrum is, and the closer α is to 0, the noisier the spectrum.

$$\alpha = min(\frac{SFM_{dB}}{SFM_{dBmax}}, 1) \tag{2.22}$$

SFM_{dB} is given in equation 2.25, and can be computed by simply using the result from equation 2.21. However this can be problematic due to the nature of the geometric mean computation. The geometric mean of a data set $[a_1, a_2 \ldots a_n]$ is given by:

$$(\prod_{n=1}^{N} a_n)^{1/N} = \sqrt[N]{a_1 \cdot a_2 \cdots a_n} \tag{2.23}$$

This is the method used in equation 2.21, which could be expressed algorithmically as:

```
1    n = 0
2    N = blocksize
3    geometric_mean = 1
4    while n < N:
5        geometric_mean = geometric_mean * input[n]
6    geometric_mean = pow(geometric_mean, 1/float(N))
```

It is apparent from the above listing that since spectral magnitude coefficients are typically in the range (0 <= a <= 1.0), for large values of N (N >= 256), the value of geometric_mean quickly loses precision, eventually reaching 0 or producing denormal[19] values. However using logarithmic identities we can transform the geometric mean into the log-average where the multiplication is converted to a summation (see equation 2.24).

$$(\prod_{n=1}^{N} a_n)^{1/N} = exp[1/N \sum_{n=1}^{N} log(a_n)] \tag{2.24}$$

Peeters therefore suggests a practical method for the direct computation of SFM_{db} based on this relation (cf. equation 2.26), but this is not adopted in libXtract since (due to the use of the log() function in the inner loop) it is computationally expensive compared to an implementation of equation 2.25 using equation 2.21. Tests show that the precision problem caused by the multiplication is largely avoided if only a small number of bands are used.

$$SFM_{db} = 10 \cdot log_{10}(SFM) \tag{2.25}$$

$$SFM_{db} = 10 * \frac{1}{N} \sum_{k} (log_{10} a_k - log_{10} \mu) \tag{2.26}$$

[19] Denormal numbers are values very close to zero in floating point arithmetic. They are significant because on some architectures they need to be implemented in software, thus causing significantly higher CPU loads.

2.5.1.13 Spectral Crest

Another descriptor related to the flatness of the spectrum is the Spectral Crest Factor . It is computed using the ratio of the maximum value within the band to the arithmetic mean of the energy spectrum value(Peeters, 2004).

$$SCM = \frac{max(a(k \in N))}{\frac{1}{K}\sum_{k \in N} a_k} \tag{2.27}$$

Where N is the number of bands considered.

2.5.1.14 Noisiness

There are a number of features that correlate to the 'noisiness' percept, i.e. how close a signal 'sounds' to white noise. These include the Spectral Flatness Measure and its derived features, the Zero Crossing Rate, LPC and Irregularity. However, a simple loudness metric can also be gained by computing the ratio of inharmonic partials' energy to the total number of partials' energy:

$$Noisiness = \frac{\sum_{k=0}^{N} a_k^2 - \sum_{j=0}^{J} h_j^2}{\sum_{k=0}^{N} a_k^2} \tag{2.28}$$

Where a_k is the amplitude of the kth partial and h_j is the amplitude of the jth harmonic.

2.5.1.15 RMS amplitude

The RMS amplitude is an alternative loudness metric which doesn't take into account the effect of frequency on perceived loudness.

$$rms = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} a_k^2} \tag{2.29}$$

2.5.1.16 Inharmonicity

According to Peeters:

> The inharmonicity represents the divergence of the signal spectral components from a purely harmonic signal. It is computed as an energy weighted divergence of the spectral components from the multiple of the fundamental frequency.
>
> (Peeters, 2004)

$$inharmonicity = \frac{2}{f_0}\frac{\sum_k |f_k - k * f_0| * a_k^2}{\sum_k a_k^2} \tag{2.30}$$

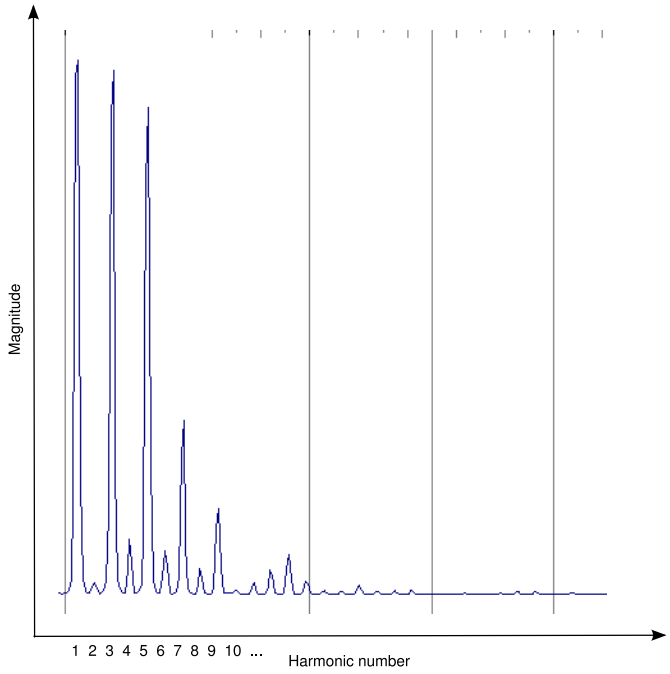


Figure 2.8: A typical B flat clarinet spectrum

The inharmonicity should be computed using the peak spectrum. In equation 2.30 f_k is the frequency of the kth peak and a_k is its amplitude. The inharmonicity coefficient ranges from 0 (purely harmonic signal) to 1.0 (inharmonic signal). The range is [0,1] since $a_k - k * f_0$ is at maximum equal to f_0(Peeters, 2004).

2.5.1.17 Odd to even harmonic ratio

The odd-to-even harmonic ratio simply gives the ratio between the magnitudes (or energy) of the odd and even harmonics. It is given as:

$$OER = \sum_{k=1:2:H} a_k / \sum_{k=2:2:H} a_k \tag{2.31}$$

It is useful for distinguishing sounds that have stronger emphasis on the odd harmonics such as the clarinet, which possesses this quality due to its cylindrical shape. A typical B flat clarinet spectrum is shown in figure 2.8

2.5.1.18 Sharpness

Sharpness is a so-called 'perceptual' feature, based on psychoacoustic studies by Fastl and Zwicker (2006). It gives a perceptually informed measure of the brightness of a sound (similar to spectral centroid), but the centroid is more widely used in live electronic music due to its relative simplicity of implementation and greater computational

efficiency. Sharpness of one acum (unit sharpness) is produced by noise centred on 1 kHz, with a bandwidth of one critical band, having a level of 60 dB. Sharpness is related to the ratio between high and low frequency content. For example, the sharpness of a sound with fixed high frequency content will decrease when additional low frequency components are added. Sharpness is the opposite of sensory 'pleasantness'. It can be defined mathematically as follows:

$$a = 0.11.\frac{\sum_{z=1}^{nband} z.g(z).N'(z)}{N} \tag{2.32}$$

Here z is the index of the band and $g(z)$ is a function defined by:

$$\begin{aligned} g(z) &= 1 \qquad (z < 15) \\ g(z) &= 0.066 * exp(0.171z) \qquad (z >= 15) \end{aligned} \tag{2.33}$$

(Peeters, 2004)

2.5.1.19 Spectral Slope

Spectral slope is a measure of the overall gradient of the data. It is similar to spectral skewness in what it tells us about the spectrum, with negative slope values indicating stronger low frequency components, and positive slope indicating stronger high frequency components. It can be computed using:

$$slope = \frac{1}{\sum_{k=1}^{N} a_k} * \frac{N\sum_{k=1}^{N} f_k * a_k - \sum_{k=1}^{N} f_k * \sum_{k=1}^{N} a_k}{N\sum_{k=0}^{N} f_k^2 - (\sum_{k=0}^{N} f_k)^2} \tag{2.34}$$

(Peeters, 2004)

Spectral Skewness is a preferable metric because of its relative ease of computation and simplicity.

2.5.1.20 Fundamental Frequency

The fundamental frequency of a periodic signal is defined as the inverse of its period.

LibXtract includes two methods for estimating fundamental frequency. xtract_f0 () uses a time domain method based on the average magnitude difference function (AMDF)[20], see also section 2.5.2.5. For a frame of N samples, it attempts to find the value of τ that gives the smallest error where τ is the candidate period in samples. First the minimum sum of errors is calculated:

[20] This method is loosely based on a method originally described by Robert Bristow-Johnson in the comp.dsp newsgroup

$$\tau_{min} = \sum_{n=1}^{N/2} |x_n - x_{n+1}| \tag{2.35}$$

Next the 'error' value is computed for all combinations of n and τ, and f_0 is found when the condition $(\tau < \tau_{min})$:

$$f_0 = sr/(\tau + (\sum_{n=1}^{N/2} |x_n - x_{n+\tau}|)/\tau_{min}) \qquad (\tau < \tau_{min}) \tag{2.36}$$

In cases when the conditional expression never evaluates as true, xtract_f0 returns the constant XTRACT_NO_RESULT, which can be interpreted by the calling function. As can be observed from the help file that comes with the libXtract Pd example, xtract_f0 compares favourably with `fiddle~` by Miller Puckette (Puckette et al., 1998) both in terms of accuracy and tolerance to inharmonicity in the spectrum. However `fiddle~` is significantly more efficient than xtract_f0 in some circumstances [21]

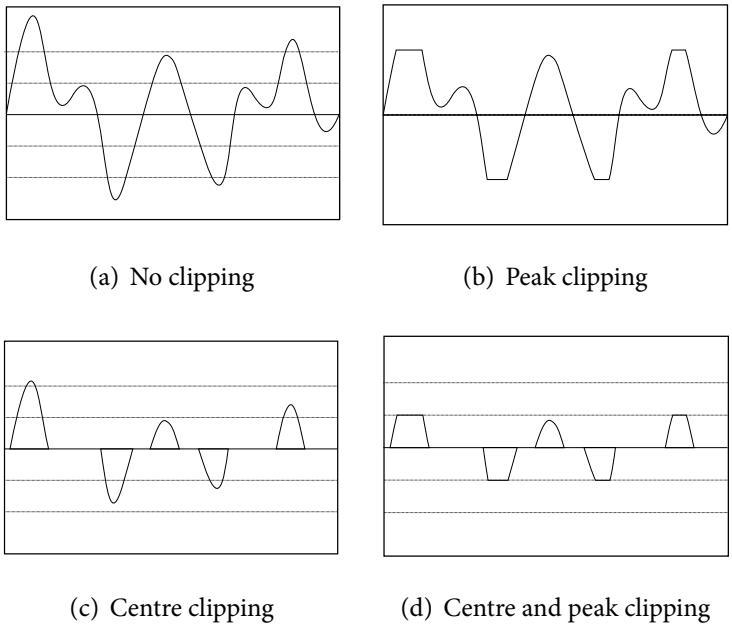
`fiddle~` is also more noise robust than xtract_f0 (). In order to improve this, the libXtract function includes a centre and peak clipping algorithm. The effects of this are shown in figure 2.9. However, this was shown to only marginally increase noise tolerance.

LibXtract also provides an additional function xtract_failsafe_f0 (), which will always find a 'fundamental' in any non silent sound. It does this by first trying xtract_f0 (), and if XTRACT_NO_RESULT is returned, it returns the frequency of the lowest partial in the spectrum.

2.5.2 Vector features

In libXtract a vector feature extraction function is one that returns a result containing more than one value. Common examples are features that consist of a set of coefficients (e.g. MFCC, Bark coefficients, Autocorrelation). The result from any vector feature can be passed in as the input to other feature extraction functions for further processing.

[21] xtract_f0 () can perform well if it returns early in its inner loop, but if it returns late or doesn't find a fundamental, it has an efficiency of $O(c^n)$



(a) No clipping (b) Peak clipping

(c) Centre clipping (d) Centre and peak clipping

Figure 2.9: Operations to reduce the effects of non-fundamental partials on the PDA

2.5.2.1 L-norm

The L-norm, or L_p norm, is a function that can be used to calculate the *length* of a vector in N-space, i.e. its distance from the origin. The value of p indicates the $type$ of distance metric used. For example, where $p = 1$, the L_p norm gives the 'taxicab' norm and where $p = 2$, the L_p norm gives the Euclidean norm (see figure 2.10. The L_p norm for an arbitrary p is generalised in equation 2.37.

$$\| x \|_p \equiv (\sum_{n=1}^{N} |x_n|^p)^{1/p} \tag{2.37}$$

2.5.2.2 Difference vector

The *distance* between two points in N-space is given by the L_p norm of their difference vector, where the difference vector is simply defined as:

$$\Delta\vec{X} \equiv \vec{X} - \vec{X}' \tag{2.38}$$

Where $\vec{X}$ and $\vec{X}'$ are two vectors of the same dimension.

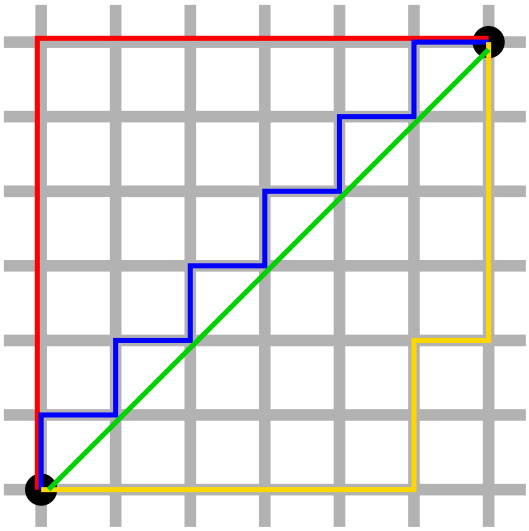


Figure 2.10: Manhattan distance versus Euclidean distance: The red, blue, and yellow lines have the same length (12) in both Euclidean and taxicab geometry. In Euclidean geometry, the green line has length $6x\sqrt{2} \approx 8.48$, and is the unique shortest path. In taxicab geometry, the green line's length is still 12, making it no shorter than any other path shown.

2.5.2.3 Spectral Flux

The Spectral Flux is a measure of the rate of change of the spectrum of an audio signal. It can be computed using the L_p norm of the difference between two successive magnitude (Ong, 2005) or log-magnitude spectra. However the power spectrum is more commonly used[22].

Spectral Flux has applications in onset detection where a high flux value (indicating large amounts of spectral change between frames) has been shown to indicate note onset (or offset) locations(Bello, Daudet, Abdallah, Duxbury, Davies, and Sandler, 2005; Duxbury, Sandler, and Davis, 2002; Jensen and Andersen, 2004). There are several variations on the norm order used in the Spectral Flux computation. For example, in the context of onset detection, Dixon (2006) suggests that L_1 gives better results than the L_2 norm proposed in Duxbury et al. (2002). It is possible to use values of $p < 1.0$ in the L_p norm computation, with lower values giving a steeper gradient (and finer resolution) in the resultant curve (see figure 2.11).

For *onset* rather than *offset* detection tasks it only makes sense to consider positive change in the spectral coefficients. For this reason, most authors use the half-wave recti-

[22] LibXtract allows for all three options

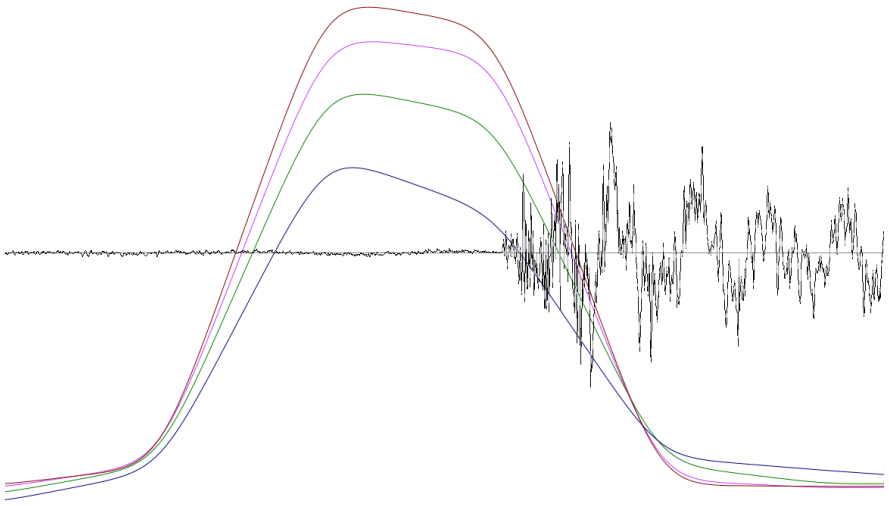


Figure 2.11: The effect of changing p in the L_p-norm computation for Spectral Flux (graph generated using Mazurka Scaled Spectral Flux plugin)

fied difference vector (Bello et al., 2005; Dixon, 2006). Normalisation and/or smoothing of the spectral coefficients is often employed prior to computing the spectral flux value in order to limit the effects of instabilities in the spectrum and improve the detection of soft onsets(Duxbury et al., 2002).

2.5.2.4 Autocorrelation

The autocorrelation function is the cross-correlation of a vector with itself; it measures the similarity of a signal with a delayed version of the signal. It is useful for determining periodicities in a signal, for example detecting the presence of sinusoidal components in an otherwise noisy signal. It is shown mathematically in equation 2.39.

$$r_\tau = 1/N * \sum_{k=0}^{N-\tau-1} x_k x_{k+\tau} \tag{2.39}$$

Where the r_τ gives the autocorrelation value at lag τ in samples. Generally the r_τ is repeated for all values of $0 < \tau < N - 1$ to give the autocorrelation coefficients for one block of samples. The autocorrelation coefficients can be computed efficiently as follows:

- Take the FFT of a signal
- Multiply the FFT by its complex conjugate
- Take the inverse FFT of the result

The autocorrelation function can be used to compute the LPC coefficients (cf. section 2.5.3.4, and also forms part of the pitch detection algorithm used in the Tartini pitch visualisation software (cf. section 3.3.4)(McLeod and Wyvill, 2005).

2.5.2.5 AMDF

The Average Magnitude Difference Function is similar to the Autocorrelation in that it compares a signal with a delayed version of itself. It can be used for pitch detection using the 'valley' locations as an indication of periodicity (cf. 2.5.1.20). It is expressed mathematically as:

$$r_\tau = 1/N * \sum_{k=0}^{N-\tau-1} |x_k - x_{k+\tau}| \tag{2.40}$$

Where the r_τ gives the AMDF value at lag τ in samples. Generally the r_τ is repeated for all values of $0 < \tau < N - 1$ to give the AMDF coefficients for one block of samples.

2.5.2.6 ASDF

The Average Squared Difference Function is identical to the AMDF except that the average of the *squared* differences is used rather than the average of the *absolute value* of the differences:

$$r_\tau = 1/N * \sum_{k=0}^{N-\tau-1} (x_k - x_{k+\tau})^2 \tag{2.41}$$

2.5.2.7 Bark coefficients

The Bark Scale is a psycoacoustically-informed frequency scale, which ranges from 1 to 24 Barks corresponding to the first 24 critical bands of hearing(Zwicker, 1961). The formally defined band-boundaries are given in table 2.3:
Some authors propose an extension of these limits reflecting higher sampling rates:

> Since the Bark scale is defined only up to 15.5 kHz, the highest sampling rate for which the Bark scale is defined up to the Nyquist limit, without requiring extrapolation, is 31 kHz. The 25th Bark band certainly extends above 19 kHz (the sum of the 24th Bark band edge and the 23rd critical bandwidth), so that a sampling rate of 40 kHz is implicitly supported by the data. We have extrapolated the Bark band-edges in our work, appending the values [20500, 27000] so that sampling rates up to 54 kHz are defined.
>
> (Smith and Abel, 1999)

Bark band	Band edge (Hz)	Band centre (Hz)
1	0	50
2	100	150
3	200	250
4	300	350
5	400	450
6	510	570
7	630	700
8	770	840
9	920	1000
10	1080	1170
11	1270	1370
12	1480	1600
13	1720	1850
14	2000	2150
15	2320	2500
16	2700	2900
17	3150	3400
18	3700	4000
19	4400	4800
20	5300	5800
21	6400	7000
22	7700	8500
23	9500	10500
24	12000	13500
25	15500	-

Table 2.3: Bark band limits as defined by Zwicker

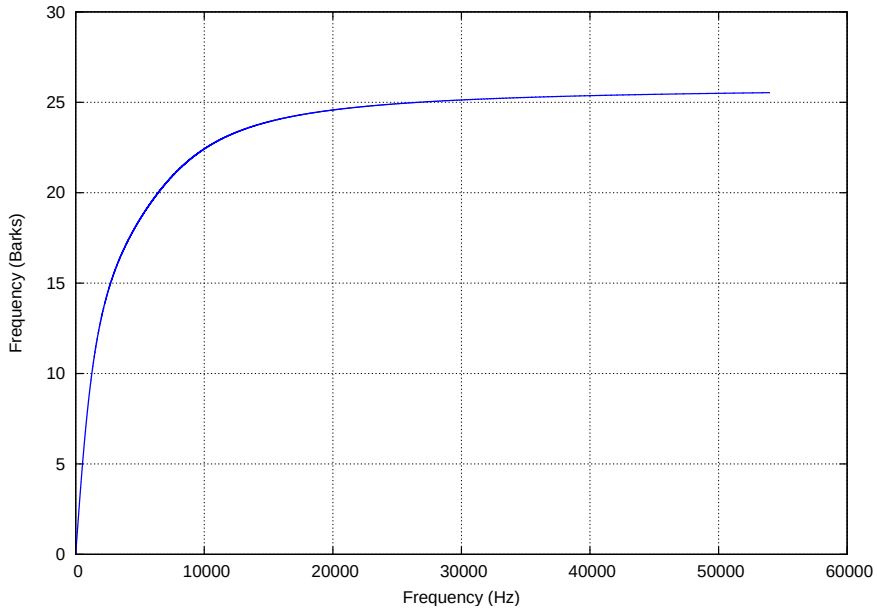


Figure 2.12: Hertz to bark conversion graph

For arbitrary conversions between hertz and bark representations, an hertz-to-bark, bark-to-hertz mapping is given in equation 2.42, and represented graphically in figure 2.12.

$$b = 13tan^{-1}(7.6 * 10^{-4} f) + 3.5tan^{-1}(f/7500)^2 \tag{2.42}$$

Where f is the frequency in Hertz and b is the mapped frequency in Barks.

2.5.2.8 Peak spectrum

The peak spectrum is based on the spectrum[23] as described in section 2.5.2.9. However, rather than including all spectral coefficients, a peak-picking algorithm used in order to determine peaks (local maxima) in the spectrum, and parabolic interpolation is used to give a more accurate estimate of the frequency and amplitude of each peak. The peak picking algorithm is shown diagrammatically in figure 2.13. This is a relatively simple algorithm, and other authors (Park, 2000) have proposed alternative methods that take into account peak prominence and regressively search for missing harmonics. However, the algorithm used in libXtract has been found to be effective at locating peaks,

[23] The magnitude, log-magnitude, power or log-power spectra can be used as candidates for the peak spectrum, although some authors (Smith and Serra, 1987) have have noted that using the log-magnitude spectrum improves the accuracy of the peak frequency estimation

in particular the use of a threshold value *relative* to the maximum coefficient value has been particularly robust in comparison to using an absolute threshold. This technique provides a good compromise between accuracy and computational efficiency.

The peak interpolation technique in libXtract is based on a method described in Smith and Serra (1987) and is shown diagrammatically in figure 2.14. The interpolated peak frequency is given by equation 2.43, with its amplitude in equation 2.44.

$$\alpha = a_{k-1} \tag{2.43a}$$

$$\beta = a_k \tag{2.43b}$$

$$\gamma = a_{k+1} \tag{2.43c}$$

$$p = \frac{1}{2} * \frac{\alpha - \gamma}{\alpha - 2\beta + \gamma}) \tag{2.43d}$$

$$Pfreq_k = q * (k + p) \tag{2.43e}$$

Where a_k is the amplitude of the kth considered bin and q is the sampling rate divided by the window size.

$$Pamp_k = \beta - \frac{1}{4} * (\alpha - \gamma) * p \tag{2.44}$$

2.5.2.9 Spectrum

The libXtract xtract_spectrum () function generates a frequency-domain representation of an audio block from a time domain representation. The resulting data is provided using an internal spectral data type (cf. section 2.5.1), which forms the basis of many other features. The spectrum is computed by taking the Fourier transform of an audio block using the FFT algorithm, and populating the output vector with the values of the magnitude, log-magnitude, power, or log-power coefficients depending on the algorithm chosen by the user.

The magnitude coefficients are calculated using the square-root of the sum-of-the-squares of the real and imaginary components of the FFT, and normalised by dividing by the transform size. This is shown in equation 2.45, where RE_k is the kth real component of the FFT IM_k is the kth imaginary part and N is the transform size.

$$magnitude_k = \frac{\sqrt{RE_k^2 + IM_k^2}}{N} \tag{2.45}$$

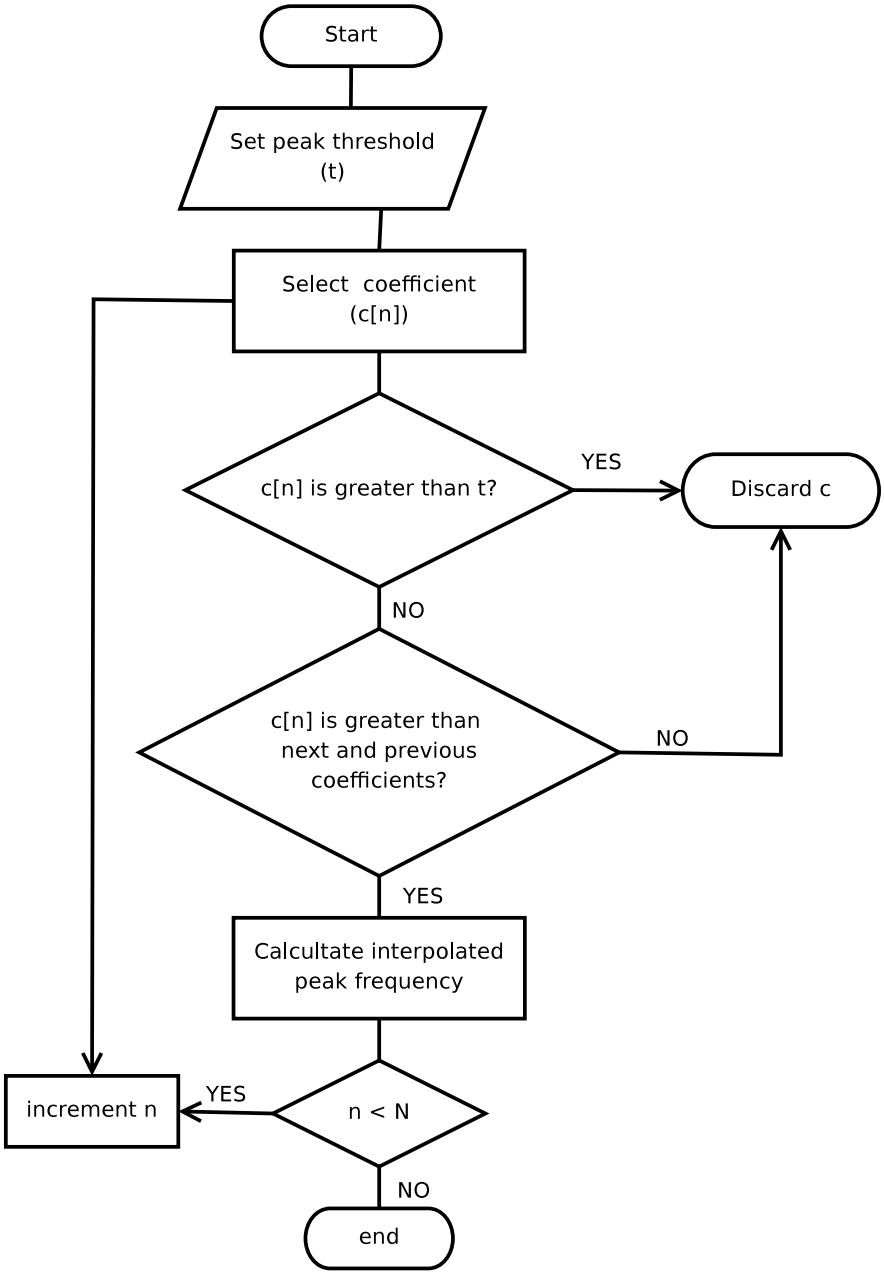


Figure 2.13: LibXtract peak extraction algorithm. The coefficient value $c[n]$ corresponds to the nth magnitude, log-magnitude, power, or log-power coefficient, and the threshold is set by the user as a percentage of the per-frame maximum coefficient value.

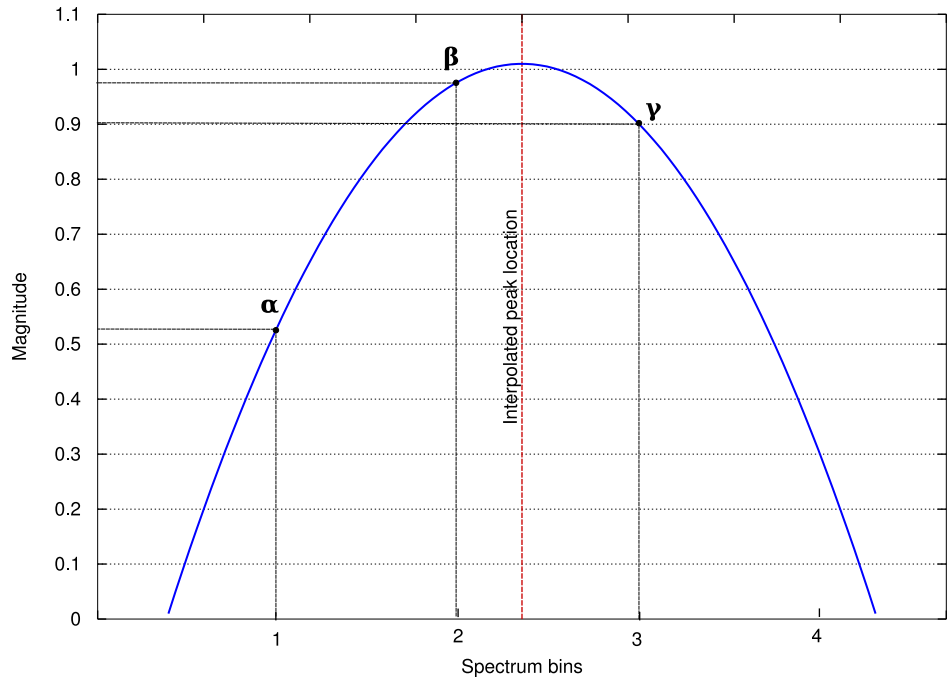


Figure 2.14: Peak interpolation using parabolic fit. Estimated frequency based on bin number for bin 2 = $44100/64 * \beta = 1378Hz$. Interpolated frequency for bin 2 = $Pfreq_k = 44100/64 * (0.52 - 0.9)/(0.52 - 2 * 0.98 + 0.9) = 1863Hz$.

The FFTW library is used to compute the FFT in libXtract, providing a robust and efficient implementation suitable for real-time analysis (cf. Frigo and Johnson (2005)).

2.5.3 Autocorrelation (FFT)

The autocorrelation function is described in section 2.5.2.4, however in practice a more efficient implementation based on the FFT is often used. According to Casagrande:

> There is also an important relationship between the autocorrelation and the Fourier Transform known as the Wiener-Khinchin theorem, which allows us to compute the autocorrelation in time $O(n \log n)$ using a DFT instead of $O(n^2)$.
>
> (Casagrande, 2005)

This is given mathematically as:

$$autocorr(X) = IFFT(|FFT(X)|) \tag{2.46}$$

Where IDFT is the Inverse Discrete Fourier Transformation, and $|FFT(X)|$ is the complex modulus.

2.5.3.1 MFCC

The Mel-scale is an experimentally-determined scale of pitches where each pitch has been judged by listeners to be the same perceptual distance from the previous pitch. This contrasts with the usual linear scale of frequency where a change of 1Hz is perceived differently depending on how high or low the sound is. Conversions between *mels* and *Hertz* can be performed using equations 2.47 and 2.48. The conversion is shown graphically in figure 2.15

$$m = 1127.01048 * log_e(1 + f/700) \tag{2.47}$$

$$f = 700(e^{m/1127.01048} - 1) \tag{2.48}$$

Where f is frequency in Hertz and m is frequency in Mels.

A mel-filterbank is a set of triangular overlapping bandpass filters equally spaced on a mel scale and then mapped onto a linear scale (see figure 2.17). The Mel-frequency Cepstral coefficients (MFCC) are a means of generating a compact perceptually-informed representation of a signal's spectral envelope. The MFCC function is best defined algorithmically as shown below. The number of filters is usually between 8 and 40, and determines the resolution of the final result. Eronen (2001) showed that the optimal

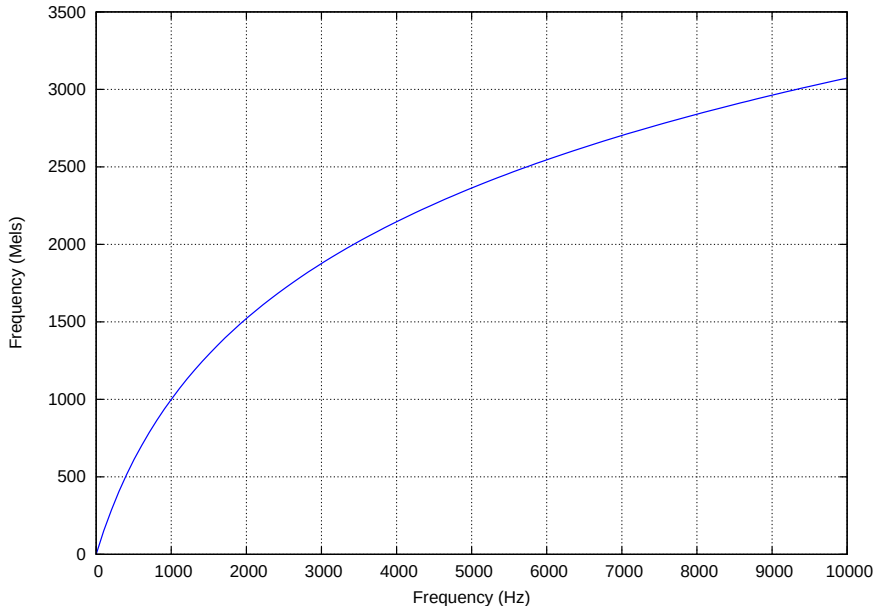


Figure 2.15: Hertz-to-mel unit conversion

number of filters for instrument recognition tasks is 12, although his study was limited to non-percussive orchestral instruments using standard playing techniques. The shape of the filterbank can also be altered, with equal-area-per-filter being used rather than equal gain (see figure 2.16).

MFCC algorithm

1. Take the magnitude spectrum of a signal
2. Convert to a mel-spectrum by applying a series of mel-spaced filters
3. Take the log of the output from each filter
4. Take the discrete cosine transform of the log mel-spectrum to obtain the mel-cepstrum

The MFCC technique was originally developed for speech recognition (Rabiner and Juang, 1993), however it has more recently been shown to be useful for a variety of audio classification tasks including timbre mapping. For example Terasawa, Slaney, and Berger (2005) conducted experiments to measure the difference between perceptual and modelled distances in a 13-dimensional timbre space, concluding that MFCCs were a

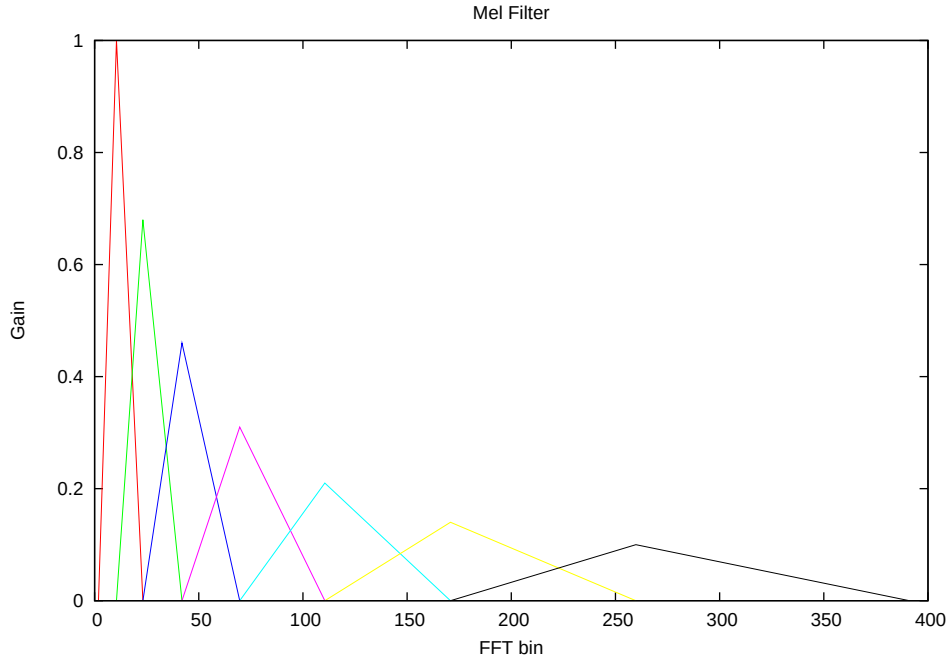


Figure 2.16: Equal area MFCC filterbank

good perceptual model for static sounds. It is also shown in Essid, Richard, and Bertrand (2004); Weng, Lin, and Jang (2004) that MFCC values can be used to effectively characterise a range of instrumental timbres.

2.5.3.2 Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is used in lossy compression algorithms including JPEG image compression and MPEG audio compression. Like the Fourier transform, the DCT expresses a time series in terms of a sum of sinusoids with different frequencies and amplitudes. The Type-II DCT used in libXtract is defined in equation 2.49. In practice it is computed using an FFT-based technique.

$$X_k = \sum_{n=0}^{N-1} x_n cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \tag{2.49}$$

Where X_k is the kth output coefficient and x_n is the n the input value.

A comparison between Discrete Fourier Transform (DFT) and DCT is shown graphically in figure 2.18

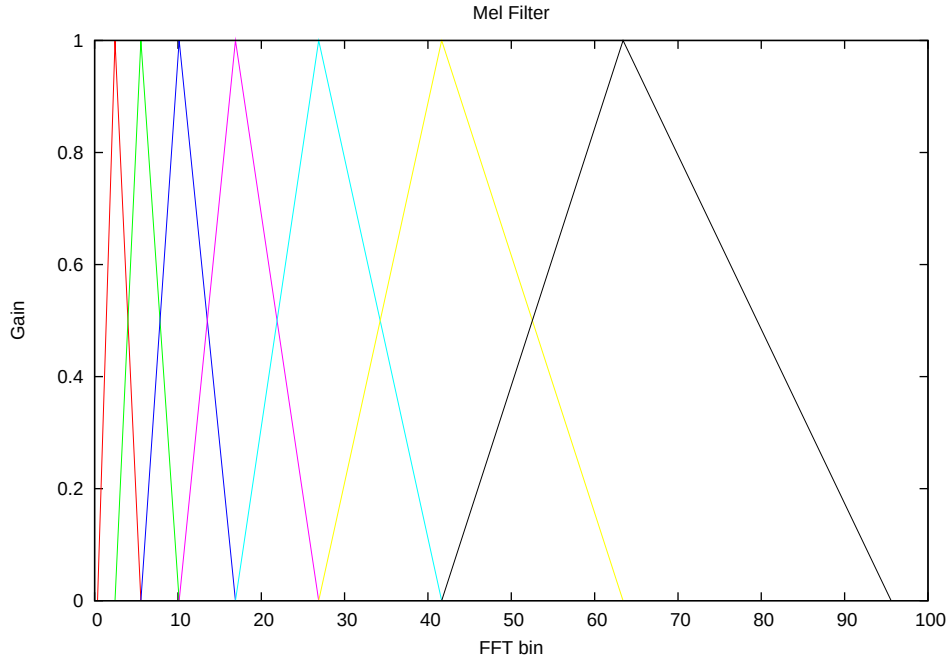


Figure 2.17: Equal gain MFCC filterbank

2.5.3.3 Harmonic Spectrum

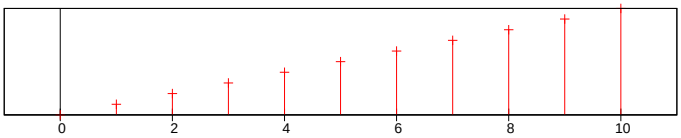
The harmonic spectrum identical to the peak spectrum as described in section 2.5.2.8, except that only *harmonic* peaks are included. Peaks are considered harmonic if they fall within a given threshold of an integer multiple of the fundamental.

2.5.3.4 LPC

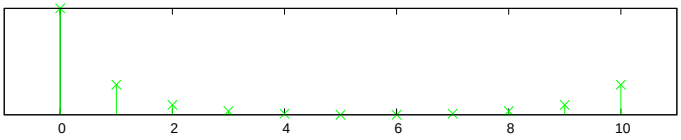
Linear Predictive Coding (LPC) is an audio analysis technique devised for speech analysis and based on models of the human vocal tract(Atal and Hanauer, 1971). It works on the assumption that the sound being analysed is the result of an all-pole (IIR) filter applied to a noise-like source. LPC can be used as an alternative to MFCC for representing the spectral envelope of digital signal in compact form. It is used in compression algorithms such as FLAC[24], and is common in speech analysis and re-synthesis. It has also been used in a musical context for voice synthesis and applying voice-like qualities to other sounds. This was particularly popular in American 'Computer Music', and is evidenced in works by Lansky and Dodge(Roads, 1996). The model for speech synthesis using LPC analysis is shown in figure 2.19.

LibXtract implements the LPC algorithm via the 'autocorrelation method' using Levison-Durbin recursion(Durbin, 1960; Levinson, 1947) as described by Degener(Degener,

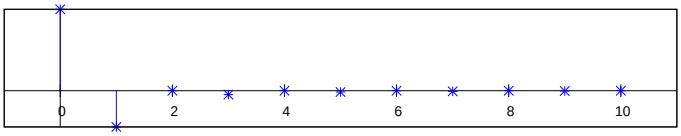
[24] Free Lossless Audio Codec



(a) Time series data



(b) The modulus of its DFT



(c) Its DCT

Figure 2.18: Comparison between DCT and DFT. [Diagram based on MatLab and gnuplot code by Alessio Damato used under the terms of the Creative Commons Attribution ShareAlike license version 2.5].

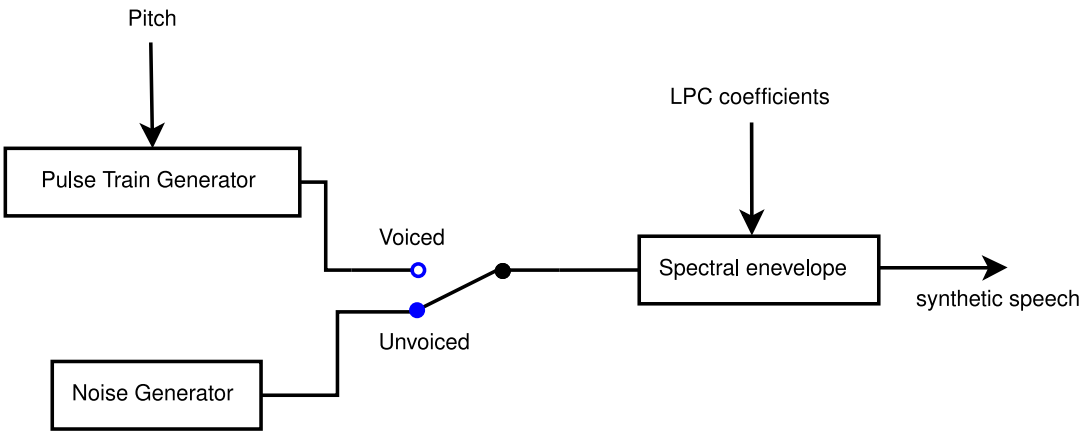


Figure 2.19: Speech synthesis using LPC coefficients to provide spectral envelope

1994).

2.6 Other functionality provided by the library

The libXtract library is under continuing development, and the feature list and other functionality is expanding as new requirements are encountered. However, there are some 'core' non-extraction functions that will be described below for completeness[25]

2.6.1 Windowing functions

In addition to vector and scalar feature extraction functions, LibXtract also provides a number of 'helper' functions that assist with the feature extraction process. One set of such functions are the so-called 'windowing' functions. These functions generate a symmetrical shape which begins and ends at a zero value. The purpose of this is to minimise discontinuities at the endpoints of an analysed data frame, which inevitably occur with waveforms that have non-integral periods. 'Window functions effectively 'link up' the beginning and ending of the analysed segment of the waveform, helping them to behave like single periods of a periodic function even if they are not' (Moore, 1990). A windowed and non-windowed waveform segment are shown in figure 2.20.

LibXtract implements the following windowing functions:

- Gaussian

[25] The full API documentation is provided in appendix C

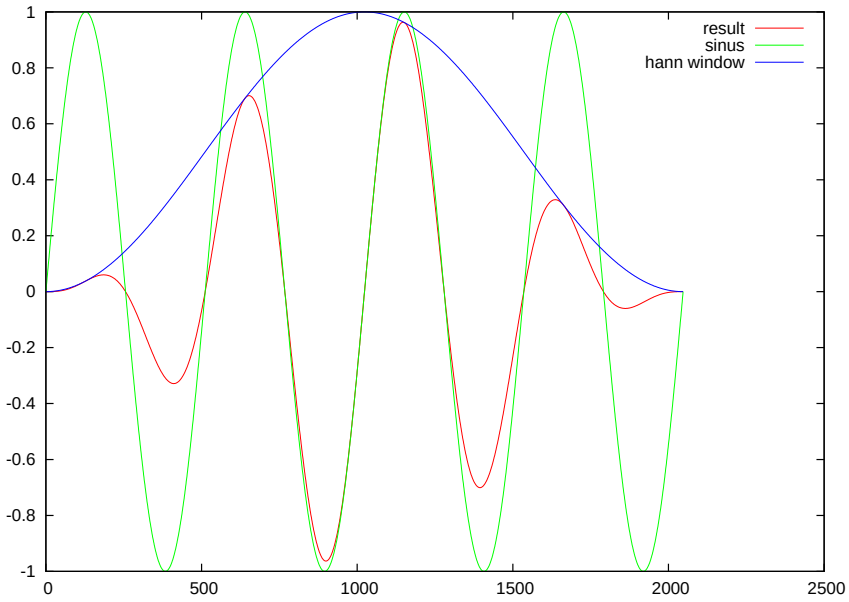


Figure 2.20: The effects of applying a Hann window function to a 2048 point sine wave

- Hamming
- Hann
- Bartlett
- Triangular
- Bartlett Hann
- Blackman
- Kaiser
- Blackman-Harris

The current window list is accessed using the enumeration xtract_window_types_ and the function xtract_windowed()[26]. The implementation uses an on-the-fly window calculation, which is not computationally efficient. A planned future improvement is to use a table-lookup algorithm.

[26] See API documentation in appendix C for more detailed documentation.

2.6.2 Feature descriptors

LibXtract is intended to be self-documenting, that is, data *about* each feature extraction function provided by the library is included in the library in the form of a feature descriptor. Listing C gives a list of the feature descriptor fields.

The data held in the feature descriptor enables an application linking to the libXtract library to determine at run-time, detailed information about each feature by querying the descriptor. The presence of the 'argument donor' field enables the calling application to determine how different feature extraction functions can be 'connected up' by publishing the `ID` of the function whose output should be used to provide the values to the arguments of a given feature.

2.6.3 Combining features

As shown in figure 2.1 feature extraction functions in libXtract can be arbitrarily combined. However, some combinations are more appropriate than others, or represent 'standard' configurations. For example, as noted by Peeters (Peeters, 2004) Centroid, spread, skewness, kurtosis, slope, decrease, roll-off and variation can be computed on spectrum, harmonic peaks and Bark band representations. Some harmonic descriptors such as odd-to-even energy ratio and tristimulus can also computed on a Bark band representation.

LibXtract has been designed to be as flexible as possible, placing only minimal restrictions on the data passed between functions.

2.7 Implementations

Despite the fact that libXtract is a relatively recent library, it has already been incorporated into a number of useful programs.

2.7.1 Vamp libXtract plugin

The vamp-libXtract-plugin acts as a wrapper for the libXtract library, making nearly the entire set of libXtract features available to any Vamp host (cf. section 2.3.7 for a description of Vamp). This is done by providing only the minimum set of feature combinations, the implication of this being that the facility to experiment with different cascaded features is lost, but with the advantage that the interface is simplified. Since the available libXtract features have already been discussed in detail in 2.4, no further details will be given here.

2.7.2 Pd and Max/MSP externals

The libXtract library comes with a Pure Data (Pd) external, which acts as a wrapper to the library's API. This is an ideal use case because it enables feature extraction functions to be cascaded arbitrarily, and for non-validated data to be passed in as arguments through the `xtract~` object's right inlet. The disadvantage of this approach is that it requires the user to learn how the library works, and to understand in a limited way what each function does. A future improvement would be to create a number of wrapper abstractions that simplify the use of `xtract~`.

A Max/MSP external is available, which provides functionality that is analogous to the Pd external. The Pd and Max/MSP bindings for the libXtract library are very important for the library's 'real world' utility in live electronic music. This is because these are two of the most commonly used applications for musicians working in live electronics practice. The use of both the Pd and Max/MSP externals is evidenced in Chapter 4, where detailed case studies involving their use are given.

2.7.3 RPM and ebuild

It is a positive sign of the library's early success that it has already available to the Gentoo GNU/Linux distribution as an ebuild. It is also available on RPMfind as a RedHat RPM.

2.8 Efficiency and real-time use

The library in its current form makes no guarantee about how long it will take for a function to execute. In addition to this, for any given blocksize, there is no defined behaviour determining what will happen if the function does not return in the duration of the block.

Most of the extraction functions have an algorithmic efficiency of O(n) or better, meaning that computation time is usually proportional to the audio blocksize used. However, because of the way in which the library has been designed (flexibility of feature combination has been given priority), certain features end up being computed comparatively inefficiently. For example if only the Kurtosis feature was required in the system shown in figure 2.1, the functions xtract_mean(), xtract_variance (), xtract_standard_deviation () and xtract_kurtosis () must all execute N iterations over their input (where N is the size of the input array). The computational efficiency of xtract_kurtosis () could be improved if the outputs from all the intermediate features

were not exposed to the user or developer, however this would be at the expense of flexibility.

According to Bullock (2007) a set of 17 features including both scalar and vector features can be computed simultaneously with a blocksize of 512 samples, and 20 Mel filter bands with a load of 20-22% on a dual Intel 2.1GHz Macbook Pro Laptop running GNU/Linux with a 2.6 series kernel. Furthermore, in the practical case studies presented in Chapter 4, no problems were encountered with the regard to the efficiency or reliability of the library.

2.9 Future work

There are a number of feature extraction functions planned for inclusion in a future release of libXtract. These include 'chroma' features (cf. Hu, Dannenberg, and Tzanetakis (2003)) and wavelet-based features. Also planned is the inclusion of a feature graph function, that will take a given feature as an argument, and automatically resolve the feature dependency graph required to compute that feature. For example, the function xtract_mean() has no dependencies, so its graph consists of one function. However, the function xtract_variance () requires the mean to have already been computed, so its graph consists of two functions. Future versions of libXtract will include a new function to automatically resolve the dependency graph for a given feature, and compute all required subfeatures.

Other than the current 'delta' features, which take two successive frames as input, LibXtract does not include any 'temporal' features, which model the way in which sound changes over time. Given that at the time of writing, LibXtract has some 60 features, it was considered beyond the scope of this project to incorporate a mechanism for recording changes over time into the library. Instead this is deferred to the calling application (example given in section 4.4.2). However, at some future point features such as 'Temporal Centroid', 'Temporal Increase' and 'Temporal Decrease' (Peeters, 2004) will be included.

2.9.1 Benchmarking

It would also be worthy and useful to the end user and other developers, if thorough benchmarking were conducted including comparisons with other software such as Marsyas (cf. section 2.3).

It has so far been shown through practical application that libXtract is capable of real-time use (cf. section 4), but a thorough benchmarking and testing process would enable further improvements in accuracy and efficiency. LibXtract seeks to become a reference implementation for accurate feature extraction but extensive testing beyond the scope of this project is required to do this.

2.10 Conclusions

In this chapter a new library that can be used for low level audio feature extraction has been discussed. It is capable of being used inside a real-time application, and serves as a useful tool for experimentation with audio features. Use of the library inside a variety of applications has been discussed, along with a description of its role in extracting higher level, more abstract features. It can be concluded that the library is a versatile tool for low-level feature extraction, with a simple and convenient API. It provides advantages over other similar projects in that it is primarily designed with real-time vector extraction in the context of live electronic music, and has been developed alongside practical testing in musical composition and performance situations. In the following chapter audio feature visualisation techniques in the context of live electronic music will be discussed, and in Chapter 4 case studies documenting the implementation of audio feature extraction in four compositions involving electronics will be presented.

CHAPTER 3

Visualisation

Overview first, zoom and filter, then details-on-demand

— from the Visual Information Seeking Mantra (Shneiderman, 1996)

3.1 Introduction

In this chapter I will give an overview of audio feature visualisation techniques and present a new piece of software for feature visualisation developed specifically for use in live electronic music. The libXtract audio feature extraction library presented in chapter 2 is primarily targeted at developers – in itself it is of limited utility to end-users. LibXtract can be incorporated into a live electronics system through its Pd or Max bindings, but in order to create a useful and usable system, it is essential to be able to visualise the data being generated or processed. The Braun software, discussed in section 3.4 has been developed as a practical solution to this problem, but it is by no means the only approach. In the following sections I will outline a range of approaches to visualisation which may be used or adapted to different aspects of live electronics practice. I will then revisit the theme of visualisation in chapter 5, where I propose a novel approach to live electronics performance and composition that brings together feature extraction, visualisation and mapping through a unified interface.

3.2 The importance of visualisation

In his book *Information Visualization: Perception for Design*, Colin Ware suggests that in addition to its traditional meaning 'constructing a visual image in the mind', the term *visualisation* has come to mean something like 'a graphical representation of data or concepts'. He suggests that data visualisation consists of four stages:

- The collection and storage of data itself.
- The preprocessing designed to transform the data into something we can understand.
- The display hardware and the graphics algorithms that produce an image on the screen.
- The human perceptual and cognitive system (the perceiver).

Ware (2004)

This is the definition taken in this dissertation, and this chapter focuses on one specific aspect of information visualisation: the visualisation of quantitative data relating to audio features. More specifically I will present and critique a number of existing software solutions for the visualisation of audio feature data, along with a new application developed as part of this research in order to solve some of the problems identified with existing systems.

3.2.1 Evaluation criteria

In his book, Ware (2004) gives a number of advantages of graphical visualisation of data including 'an ability to comprehend huge amounts of data', 'the perception of emergent properties that were not anticipated', 'understanding of both large-scale and small-scale features of the data'(Ware, 2004). However, whilst this gives us an idea of what visualisation *does* — its effect, it doesn't tell us what visualisation *is* and how we might establish good or bad visualisations in a given context. Tufte goes some way to resolving this by exposing the idea of 'Graphical Excellence'. Writing in general about the graphical display of quantitative data (Tufte's work predates the concept of 'visualisation' presented in Ware (2004), Tufte suggests that 'Excellence in statistical graphics consists of complex ideas communicated with clarity, precision, and efficiency' and goes on to provide

a number of axioms that can be used as a benchmark for establishing the efficacy of audio-related visualisation systems. Tufte writes that 'graphical displays' should:

- show the data
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of the graphic production or something else
- avoid distorting what the data have to say
- present many numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several levels of detail from the broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- be closely integrated with the statistical and verbal descriptions of a data set

(Tufte, 1986)

Here, Tufte is talking about *any* graphical display of data not specifically computer graphics, but with regard to the problems of audio feature visualisation, these criteria are applicable.

3.2.2 Functional visualisation in live electronic music

Audio feature visualisation is useful for the reasons outlined in 3.2.1, but why is this important for composers and performers of live electronic music? According to McLeod 'a graphical interface can give immediate, non-verbal and accurate feedback to a musician'(McLeod and Wyvill, 2005). This feedback can be used in a number of ways including performance analysis, audio feature selection and comparison, and 'artistic visualisation' - that is visualisation for its own sake. Visualisation also has very practical application in troubleshooting, helping to identify feature 'jitter', inaccuracy and erroneous data. It helps to identify events in time in a way in which would be nearly impossible by listening or numerical analysis.

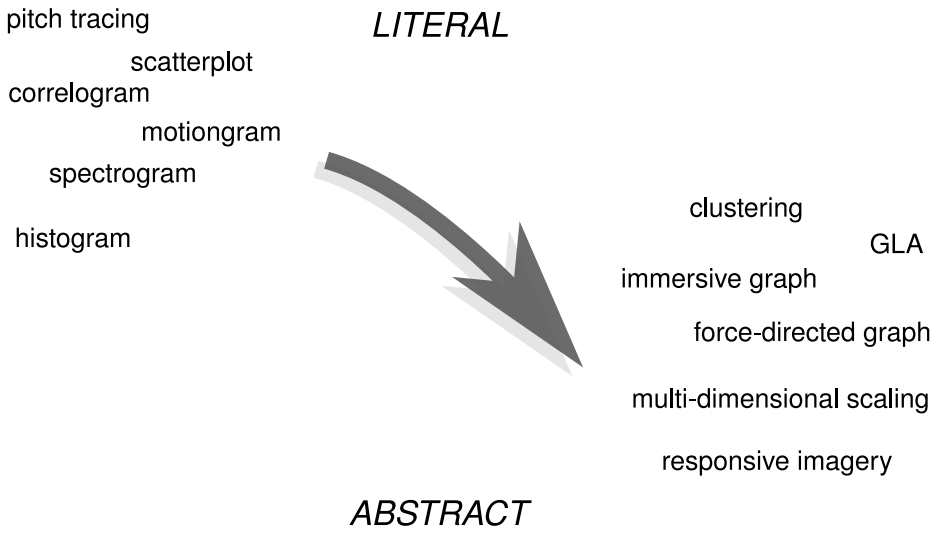


Figure 3.1: Literal/strongly correlated visualisation vs abstract/weakly correlated visualisation

According to Fry (2004) 'the visualisation of information gains its importance for its ability to help us 'see' things not previously understood in abstract data'. Feature visualisation can be thought of as existing on a continuum from the literal to the abstract that mirrors the audio feature extraction process itself (cf. section 1.8.4). Figure 3.1 shows the relationship between literal and abstract visualisation.

If used by a performer in a real-time interactive context, feature visualisation can serve to contradict or reinforce the auditory feedback that the performer experiences. Little research has been conducted into the use of non-traditional visual stimuli for performers of live electronic music, however according to Bregman:

> Perhaps a more profound reason for the similarity of principles of organisation in the visual and auditory modalities is that the two seem to interact to specify the nature of an event in the environment of the perceiver.
> (Bregman, 1990)

It is a creative decision for the software designer, composer and performer what the role of the audio-visual relationship should be in performance, and how the various

multi-modal components should be integrated. The relationship between performer-sound, multi-modal interactive systems and audience will be discussed further in sections 3.3 and 5.2.6. In the following sections a brief survey of relevant applications in the field of audio feature visualisation will be given.

3.2.3 Visualisation methods

Lengler and Eppler (1998) define a visualisation method as follows:

> A visualisation method is a systematic, rule-based, external, permanent, and graphic representation that depicts information in a way that is conducive to acquiring insights, developing an elaborate understanding, or communicating experiences.

A 'periodic table' of visualisation methods by Lengler and Eppler (1998) is shown in figure 3.2[1]. This serves as an excellent reference for current visualisation methods.

[1] The reader is encouraged to refer to the interactive version at `http://www.visual-literacy.org/periodic_table/periodic_table.html`

A PERIODIC TABLE OF VISUALIZATION METHODS

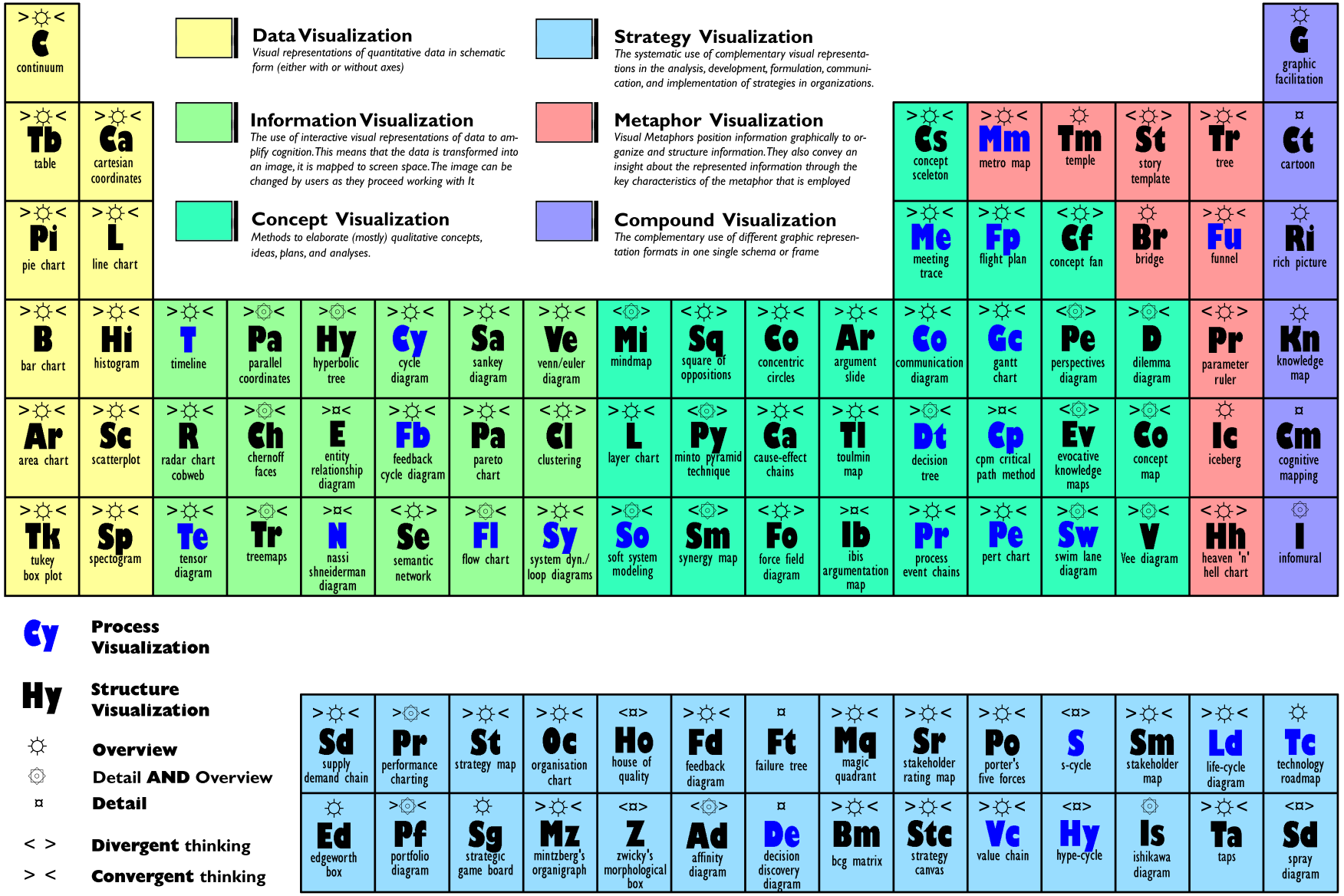


Figure 3.2: Periodic table of visualisation methods(Lengler and Eppler, 1998)

However, in the context of digital arts, specifically visualisation methods for live electronic music performance, visualisation techniques can be more dynamic, exploiting multi-modal interactions between the performer and the computer-based system. The following sections outline some of the visualisation methods currently used in the context of live electronic music.

3.2.3.1 Line tracing

Line tracing is one of the most commonly-used techniques for audio feature visualisation. It is used in waveform visualisation, pitch tracing, spectrogram plots, 'oscilloscope' graphs and arbitrary feature tracing. The procedure of line tracing consists of graphically plotting a time-ordered sequence of values on a 2-dimensional graph for visual display. In general the means of presentation gives little information about the data itself, and there is a direct correlation between the shape of the trace and the shape of the data. For example when plotting frequency values: if the frequency value increases the line goes upwards.

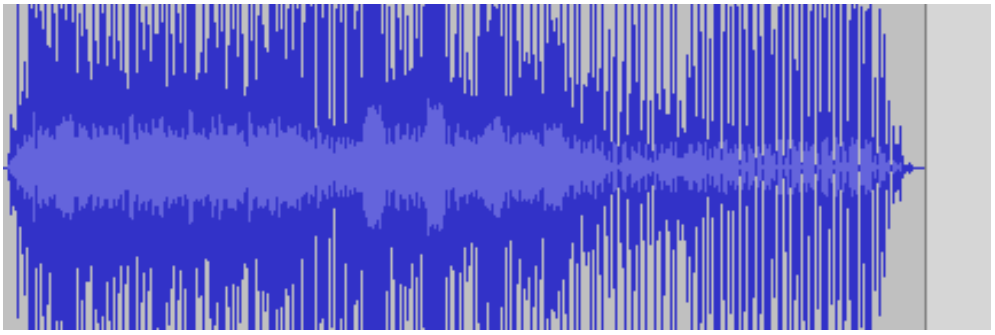
Because of this 'low level' approach line traces can often contain a large amount of information that cannot be easily interpreted by the user. Multi-resolution zoom and pan can be used to partially overcome this problem. For example, in the case of an amplitude trace showing the amplitude of each sample in a waveform, a zoomed-in view can be used to examine discontinuities in the waveform, and a zoomed-out view can be used to gain an idea of the overall 'energy' of the signal at a given point allowing the trained user to visually identify features such as onset locations or 'song' start and finish points. Figure 3.3 shows an audio waveform at various zoom levels.

Extensions of the simple line tracing technique include animated line tracing, scrolling line tracing and pseudo-3D line tracing. These approaches can be freely combined, for example the Sndpeek software, described in section 3.3.3 uses all three to provide an animated spectral plot. Sections 3.3.4, 3.3.3 and 3.4 give further details about software which make use of line tracing techniques.

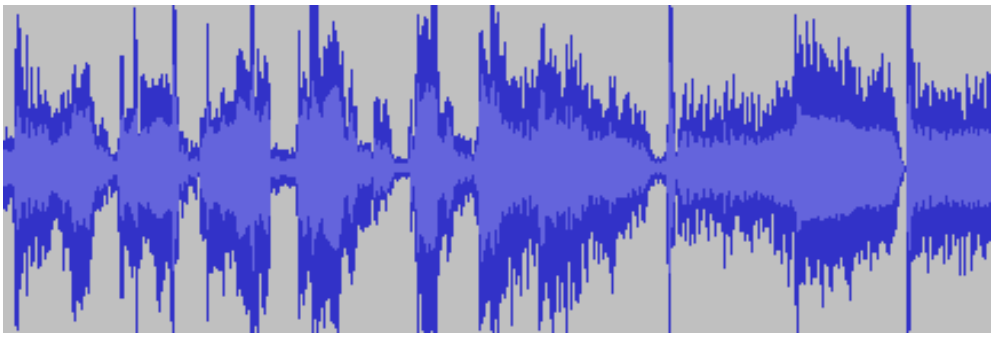
3.2.3.2 Sonogram-style visualisation

Sonograms[2] are a visual means of representing changes in the spectral envelope of a sound over time. Like line tracing, sonogram (or spectrogram) plots are closely correlated representations of numerical data, where the 'value' of the spectral coefficients is represented on a graph. Time is represented proportionally on the x-axis, frequency is

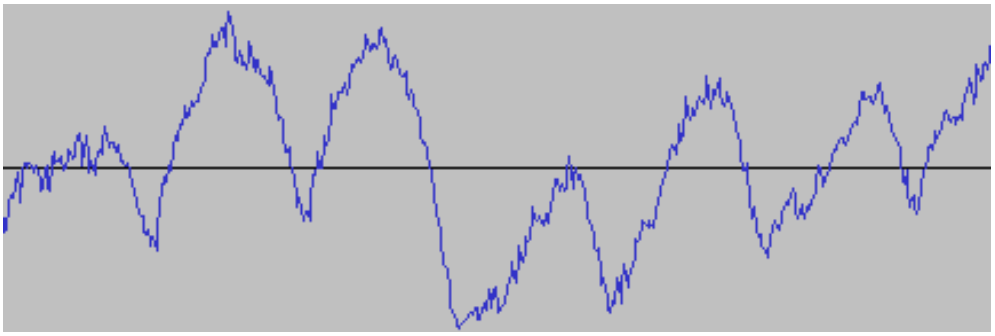
[2] Alternatively called spectrograms



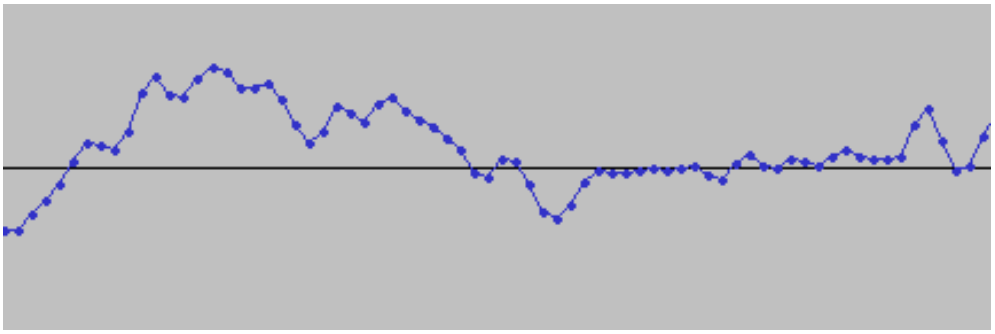
(a) Entire 'song' view



(b) Onsets view



(c) Zero crossings view



(d) Individual samples view

Figure 3.3: Sample amplitude visualisation of an audio waveform with multi-resolution zoom

given on the y-axis and amplitude values are represented using hue. Examples of sonogram software include IRCAM's Audiosculpt(Bogaards, Roebel, and Rodet, 2004), and Stanko Juzbasic's Ceres3[3]. Animated sonogram drawing capabilities are available in FreqTweak[4], which provides a number of visualisation approaches for real-time interaction with spectral processing functions, including a spectral delay.

Sonogram-style analysis is exemplified in Sonic Visualiser, where the techniques commonly used for spectral data are used for visualising the output coefficients of VAMP plugins that output more than one value per analysis input frame. Bark coefficients, MFCC coefficients, Autocorrelation, AMDF and DCT are all examples where a sonogram style analysis can be used to display coefficient values over time.

One useful aspect of sonogram visualisations is that they can help convey timbral and gestural information that is audible in the live electronics (or instrumental part), but not visible or immediately obvious from the score. This is particularly helpful for musicological analysis of live electronic and electro-acoustic music. Sonogram-style visualisations are also widely used in phase-vocoder-based processing for manipulating the analysis coefficients graphically prior to re-synthesis. However, the current author hypothesises that with sonogram-style visualisation the data representation is too low-level to be quickly understood by the performer in a live electronics context.

3.2.4 Metaphor-based visualisation

Metaphor-based visualisation could be defined as a visualisation method that is based on a virtual representation of a physical object. Examples of visual metaphors used in live electronics software include faders, dials, onscreen keyboards, and VU meters (digital and analogue). Figure 3.5 shows a comparison between a photograph of a physical VU meter, and graphical/virtual representations. Metaphor-based visualisations can be useful for the beginner-user because they convey a sense of familiarity, however Hutchison (1997) argues that people do not learn how to use graphical user interfaces by way of metaphor, they learn the meanings of the symbols and the terminology used in order to convey the metaphor. Other case studies corroborate this, showing that the use of visual metaphors do not inherently improve usability (Blackwell and Green, 1999). In the context of interfaces for live electronics the use of metaphor-based visualisation (e.g. analogue VU meters), could be regarded as an unnecessary limitation, restricting

[3] `http://www.music.columbia.edu/~stanko/About_Ceres3.html`
[4] `http://freqtweak.sourceforge.net/`

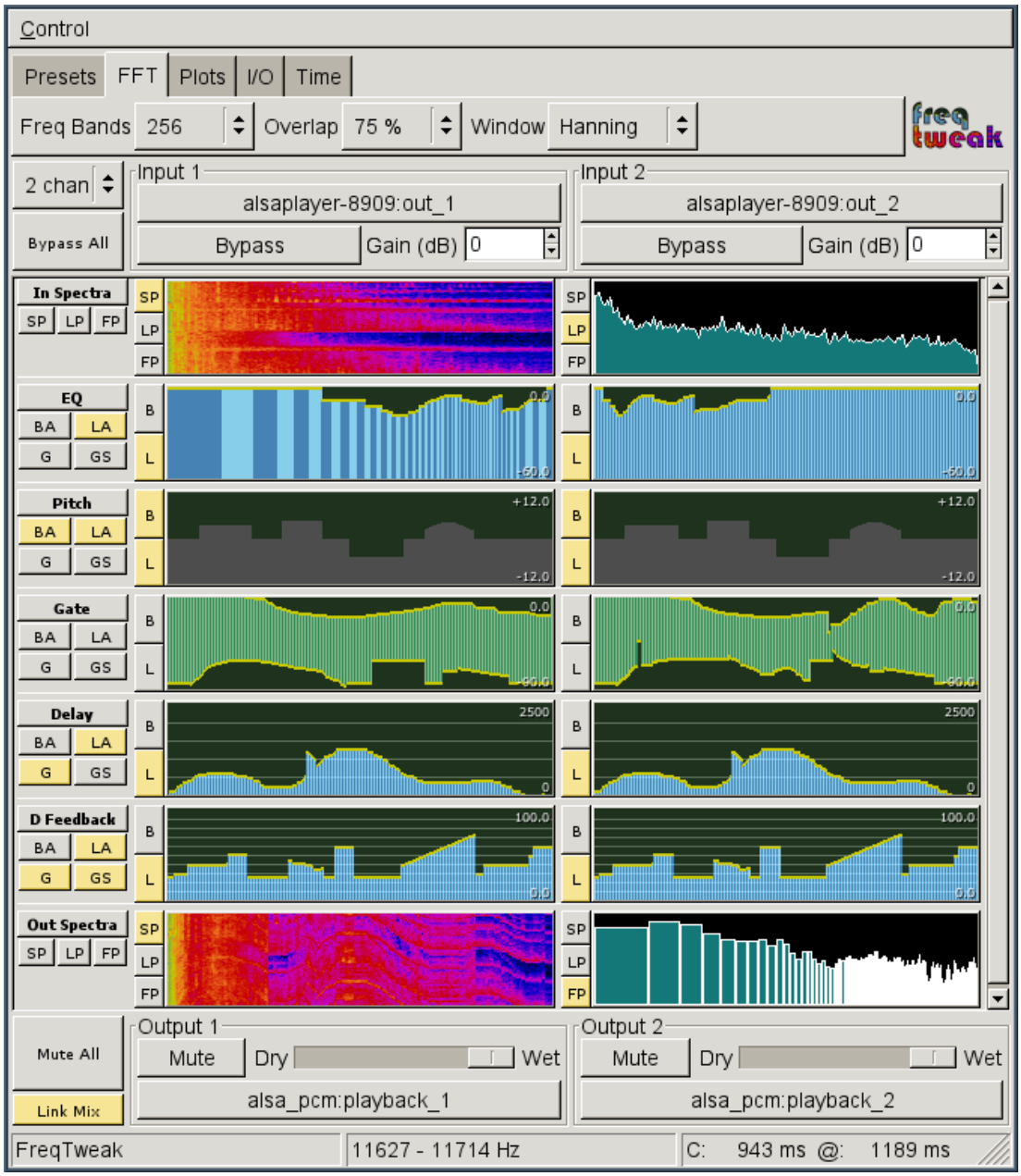


Figure 3.4: The FreqTweak GUI showing various forms of visualisation for live electronics processing including spectral delay (5th row), and real-time scrolling sonogram (top and bottom)

the interface design to existing physical technology, and not taking advantage of new possibilities.

3.2.4.1 Abstract visualisation

Abstract audio feature visualisation could be defined as visualisation where there is a non-obvious or non-literal correlation between the audio itself and the feature-to-graph mapping. Many examples of this type of visualisation can be found in popular MP3 player software such as Apple's I-tunes, WinAmp and Xmms, which support audio visualisation plugins such as Goom[5], MilkDrop[6] and GasLight[7]. These visualisation plugins tend to be based on pitch, amplitude and beat detection along with other spectral data taken from the FFT of the audio signal. The magnitudes of the constituent frequency bins can be mapped onto a wide variety of graphical parameters and obfuscated through the use of effects. For example the Goom plugin often produces a general swirling or movement effects, which bear no obvious relation to the music being heard, but still appears to change in a meaningful way as the music changes.

Another example of abstract visualisation is visualisation that reveals the *structure* of a work rather than the detail of a local audio feature. For example the 'The shape of song'[8] project shows the structure of pitch-based works by analysing repetition in MIDI files. Any repetitions are related to each other by drawing semi-transparent overlapping arcs between the repeated phrases/sections.The result can be a complex and rather beautiful 'map' of the structure of the piece as shown in figure 3.7

3.2.4.2 Object-oriented visualisation

The term 'object-oriented visualisation' is used to refer to a range of visualisation techniques where sonic or musical 'objects' (e.g. individual sounds or songs) are represented graphically as points or shapes in a 2D or pseudo-3D space, with graphical features such as colour, shape and spatial location correspond to audio-musical features. These visualisations often manifest themselves as 'maps' or psuedo-3D 'spaces' where onscreen distance represents some aspect of similarity between the musical objects being represented. That is, 'close' objects are judged to be more similar and 'distant' objects are judged to be less similar. Several examples of this approach are used in the CoMIRVa (Collection of Music Information Retrieval and Visualization Applications)

[5] http://www.ios-software.com
[6] http://www.nullsoft.com/free/milkdrop/
[7] http://www.steelskies.com/gaslight/
[8] http://www.turbulence.org/Works/song/

(a) Typical analogue VU meter

(b) Virtual PPM meter (PPMulator++)

(c) Virtual VU meter (meterbridge)

Figure 3.5: Comparison between physical and virtual VU/PPM meter-based visualisation. [VU meter photograph is a reproduction of an image by user Iainf, released via Wikimedia Commons 05:15, 12 August 2006, used under the terms of the Creative Commons Attribution 2.5 license].

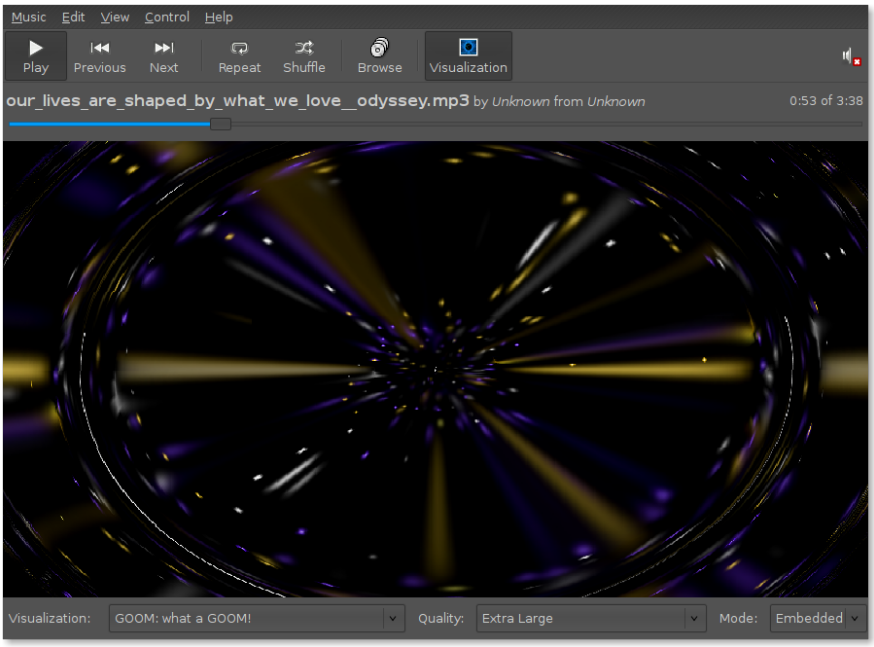


Figure 3.6: The Goom visualisation plugin responds to music changes in real-time using abstract imagery

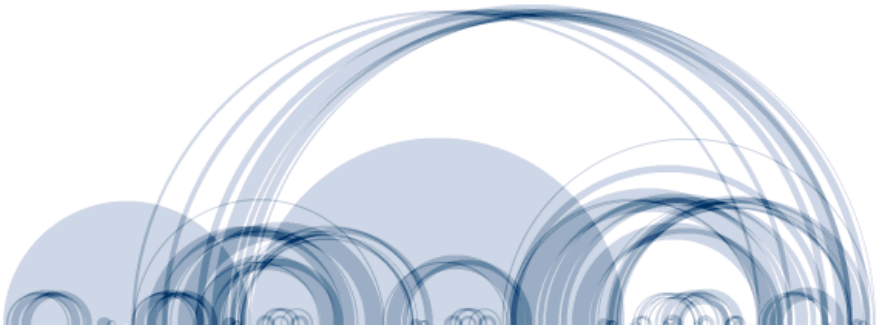


Figure 3.7: Chopin, Mazurka in F sharp Minor. The image illustrates the complex, nested structure of the piece (source: `http://www.turbulence.org/Works/song/`)

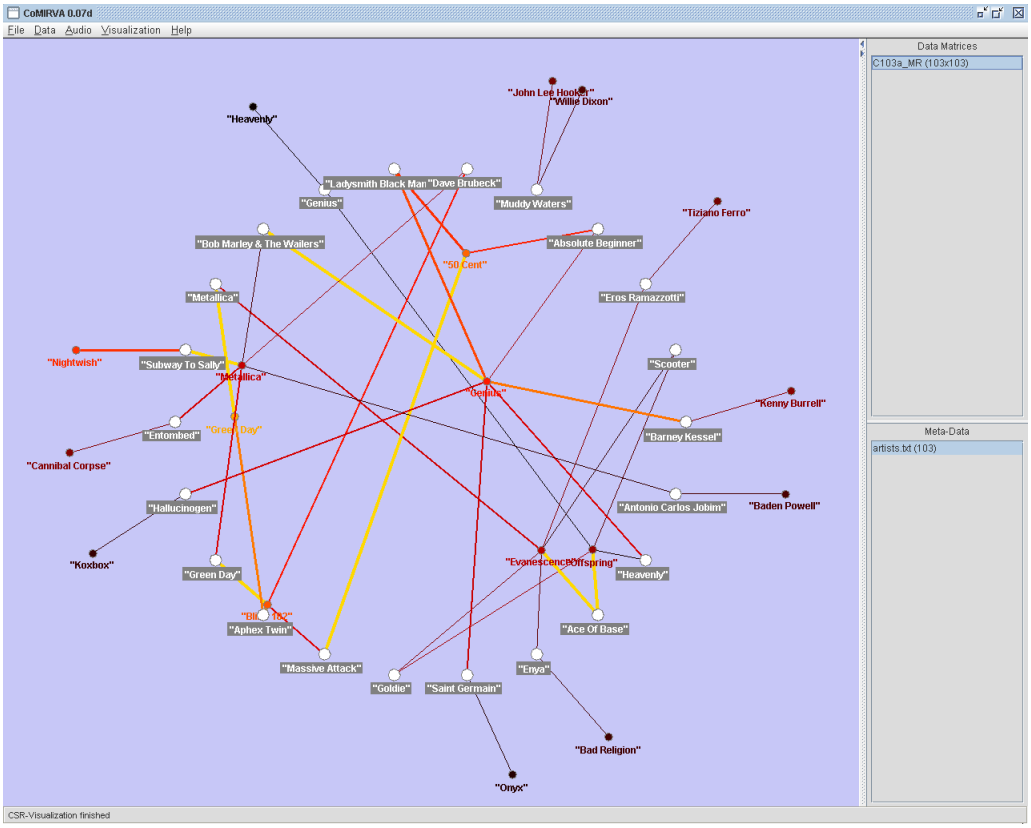


Figure 3.8: CoMIRVa "Continuous Similarity Ring"-visualisation based on prototypical artists of 22 genres (Schedl, 2006)

suite of tools. For example figure 3.8 shows a Continuous Similarity Ring (CSR) visualisation technique where a graph-based model illustrates similarities between entities by using one prototype for each of a number of given classes. In this instance the user can 'browse' musical artists by similarity where prototypical artists serve as reference points for finding similar, but less known artists(Schedl, 2006). An alternative approach to similarity visualisation is given in figure 3.9, where a geographical metaphor is used to show clusters of similar artists as 'islands' that rise from the 'sea', which represents sparse areas of the map(Schedl, 2006).

There are also a growing number of online tools for visualising 'music collections' mostly based on the force-directed graph concept. For example liveplasma uses the Amazon.com API to gather its source data, whilst The World of Music' project uses the Yahoo! Music service(Gleich, Zhukov, Rasmussen, and Lang, 2005). Force-directed

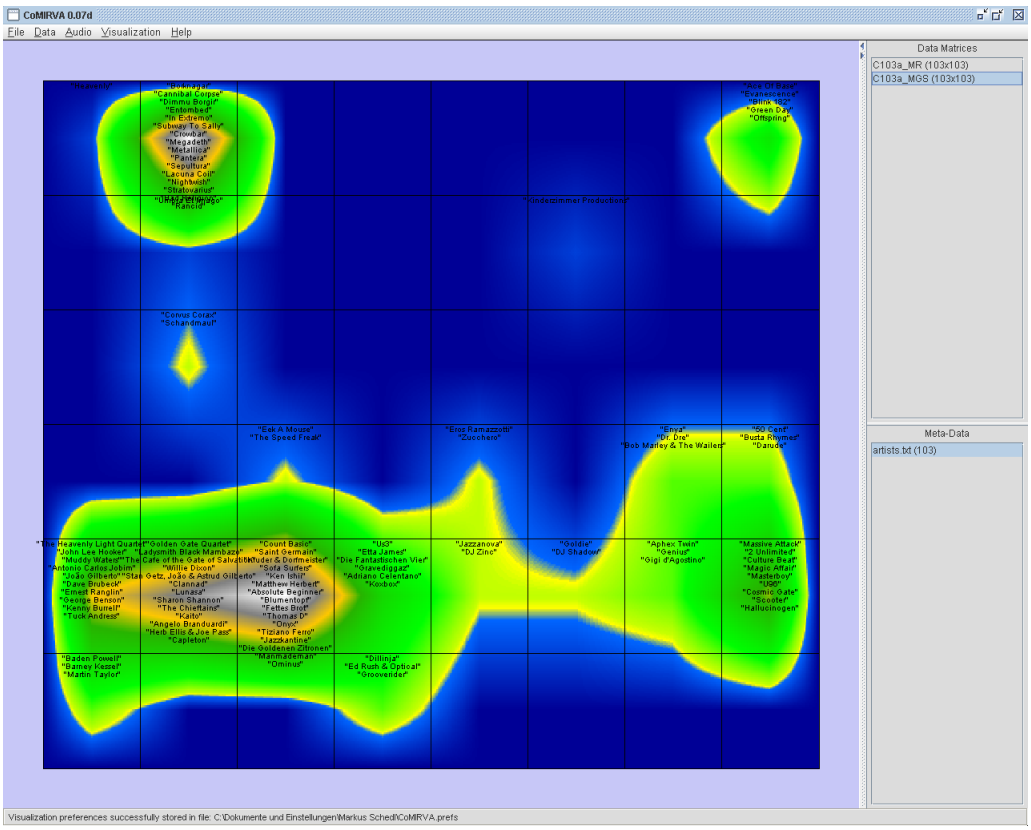


Figure 3.9: CoMIRVa An SDH-visualisation of a SOM trained on a similarity matrix of music artists (Schedl, 2006)

graphs use a class of graph drawing algorithms designed to produce aesthetically-pleasing graphs based on physical modelling principles. Physical rules are used to find the optimum positioning of nodes and vertices in a two-dimensional plane. For example, Eades (1984) proposes an model where each vertex is represented by a steel ring, and each edge is represented by a spring. The vertices are placed in some initial layout and released so that the spring forces on the rings move the system to a minimal energy state (Eades, 1984). Fruchterman and Reingold (1991) propose a alternative to Eades' model where each node behaves as an 'atomic particle' or 'celestial body', with forces of attraction and repulsion operating between adjacent nodes. Their algorithm is fast enough to allow for real-time visualisation of the graph optimisation process as well as enabling the possibility of user interaction with node placement.

One of the advantages of object-oriented feature visualisation over other approaches such as line tracing (cf. section 3.2.3.1) and sonogram-style visualisation (cf. section 3.2.3.2) is that the technique lends itself to an interactive experience for the user where the data is 'explored' rather than simply 'seen'. As such, these techniques are often used for situations where a more abstract dimension-reduced representation of the audio feature space is being visualised. For example, Pope et al. (2004) describe systems where Multidimensional scaling (MDS) analysis and a novel 'Fastmap' algorithm are used to generate a browsable representation of audio object similarity. Tzanetakis and Cook (2001) describe a similar system *TimbreSpace*, where Principal Components Analysis (PCA) is used to map a number of audio features to a low-dimensional representation suitable for object-oriented feature space visualisation. The details of this system are discussed further in section 3.3.5.

3.3 Existing audio feature visualisation software

Due to improvements in computer processing power and the wide applicability of audio feature extraction and Computer Audition to a number of disciplines, a number of bespoke applications for audio feature visualisation are becoming widely used. In addition there are a number of visualisation systems not specifically designed for audio feature visualisation, which can be adapted to the task. In the following sections, several of the most significant developments in the field will be discussed.

3.3.1 'DIY' visualisation

A culture of 'DIY' software development is prevalent in composition and performance of live electronic music. This is encouraged by a proliferation of Domain Specific Languages (DSLs) for audio and multimedia including SuperCollider, Max/MSP and Pd (cf. section 1.5.2). In addition to audio synthesis and control, several environments provide possibilities for the rapid-prototyping of visualisation systems. The following sections describe software packages, which are not designed specifically for audio feature visualisation but nonetheless provide functionality for the user/developer to develop visualisation functionality with relative ease.

3.3.1.1 Jitter

Jitter is an extension of the Max/MSP programming environment, which adds real-time graphics processing capabilities. This enables control data from Max to be mapped onto arbitrary visual parameters for display. Because of its integration with Max/MSP Jitter encourages an experimental approach to visualisation, offering possibilities for multimodal interaction. This is evidenced in the *Sonus* software (Sedes et al., 2004), which provides possibilities for live interaction with spectral visualisations, which in turn effects spectral re-synthesis (see figure 3.10). Bell et al. (2007) describes a different system using Jitter in a multi-modal environment, where simple data visualisation is used to feed information about audio and gesture analysis data back to the performer via an electronic music stand. It is noted by the authors that '[visual] feedback from the system to the performer during rehearsal and performance periods helps to remove problems of missed and false triggers'(Bell et al., 2007). Part of the visualisation is for audio feature extraction data from the `analyzer~` external as described in section 2.3.5. The visual feedback to the performer is shown in figure 3.11.

3.3.1.2 GEM

GEM (Graphics Environment for Multimedia) is a set of externals for the Pd environment, adding functionality for real-time OpenGL graphics rendering. It is supported on the OS X, Windows and Linux platforms, and has also been ported to Max/MSP. OpenGL (Open Graphics Library) is a standardised, widely-used graphics API for defining 2D and 3D graphics. Advantages of OpenGL include portability between platforms and the ability to render graphics using dedicated graphics hardware rather than the computer's CPU, thus improving performance and freeing CPU cycles for other tasks. In addition to these advantages GEM is FLOSS software making it appealing to the 'DIY' media artist wishing to experiment with audio and graphics. For example Johnston, Marks, and Edmonds (2005) describe a system where a system of vertically aligned

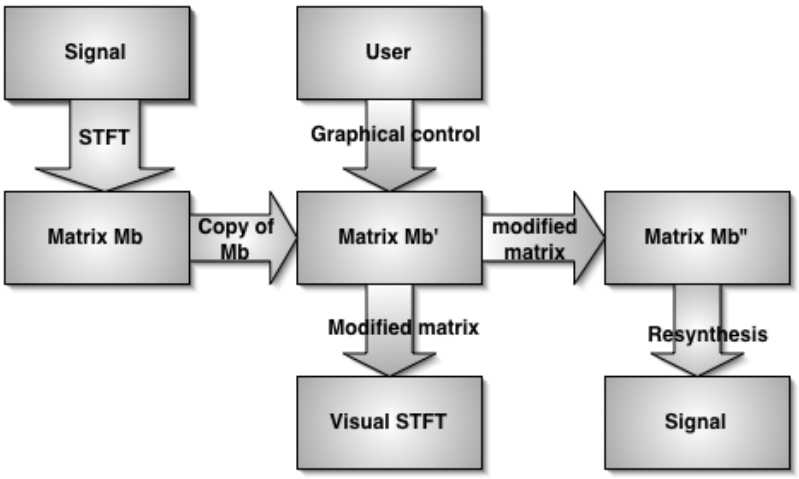


Figure 3.10: Model for multi-modal interaction in Sonus using Jitter, where the user controls the matrix with graphical operations. Transformations are applied on both image and sound (Sedes et al., 2004)

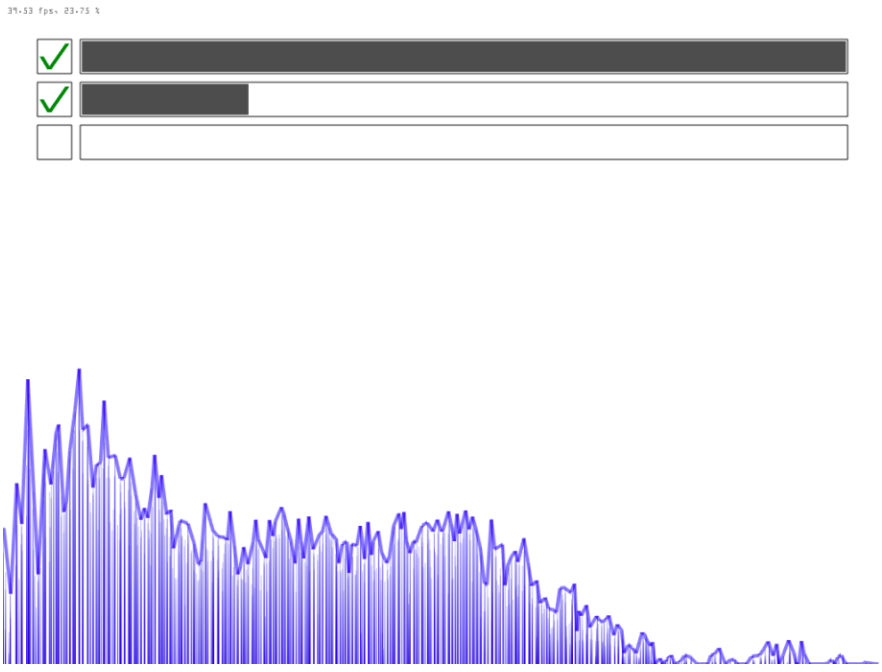


Figure 3.11: Audio feature visualisation for the performer in the MMMS project (Bell et al., 2007)

spheres is used to visualise audio features such as pitch, overall amplitude and the amplitudes of the first three detected partials in real-time. The are spheres distributed using physical modelling principles with an invisible elastic 'string' connecting them so that they move in a plausible way. The mapping of audio features to the visual display is described as follows:

> Each sphere is linked to a particular pitch-class or note, such as A, B♭, C, D, etc., so that whenever a particular note is played, the sphere associated with that pitch-class has force exerted on it. The force is proportional to the volume of the note, so loud notes have a greater impact on the string of spheres than soft notes.[...]
>
> The harmonic structure of the sound has an effect on the appearance of the spheres. [...] In this piece, we identify three partials in the live sound and these affect the transparency and colour of the spheres. For example, if a B♭ is played, the fiddle pitch-analysis object may indicate that in addition to the B♭, the sound contains traces of the 3rd (F) and 5th (D) partial. Thus in our piece the B♭ sphere would become the most visible (least transparent) and would glow with a bluish colour and the F and D spheres would also glow with an intensity proportional to their relative volumes (i.e. less than the B♭ sphere). The end result then is that one of the spheres (the B♭) is floating to one side (dragging the linked spheres with it) and glowing brightly and the D and F spheres glow also, but with less intensity. Once the note has stopped sounding, the visibility and brightness of the spheres gradually fades.(Johnston et al., 2005)

This suggests a highly intuitive interactive experience for the musical performer born out of a genuine collaboration for an actual musical work composed in conjunction with the software development process. The result is a software environment where 'the richness and complexity of the software can provide a stimulating environment for musical experimentation and improvisation'(Johnston et al., 2005).

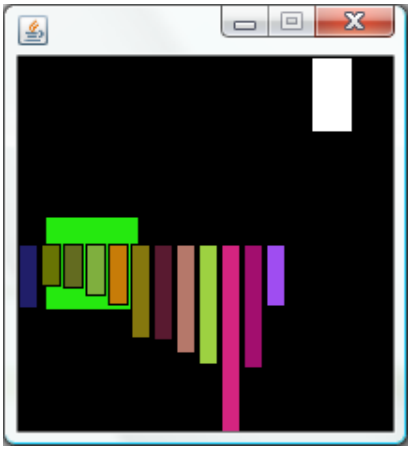


Figure 3.12: Basic audio feature visualisation in Processing used to demonstrate use of Processing with Echo Nest API

3.3.1.3 Processing

Processing is a 'simple programming environment that was created to make it easier to develop visually oriented applications with an emphasis on animation and provide users with instant feedback through interaction'(Fry, 2008). Processing is based on Java, and is freely available under the GNU LGPL license allowing for the use of the Processing libraries in both open and closed-source projects. Processing has a Java-like syntax, and is closer to a 'traditional' textual programming language than graphical languages such as Pd/Gem, Max/MSP/Jitter and vvvv. Superficially, this might make Processing less appealing to the 'DIY' multimedia developer. Due to its procedural syntax and integration with Java, Processing is well-suited to sophisticated and complex visualisations, and the development of self-contained applications. Processing does not contain any built-in audio functionality, but it is possible to interface with Processing using an additional OSC (Open Sound Control) library. This allows for the possibility of visualising audio feature data from other OSC-enabled applications. Processing is used as the basis for the Beatesthesia audio visualiser[9], and has been used to create a basic audio feature visualisation example for the Echo Nest[10] project (see figure 3.12).

[9] `http://www.vimeo.com/993330`
[10] `http://the.echonest.com/`

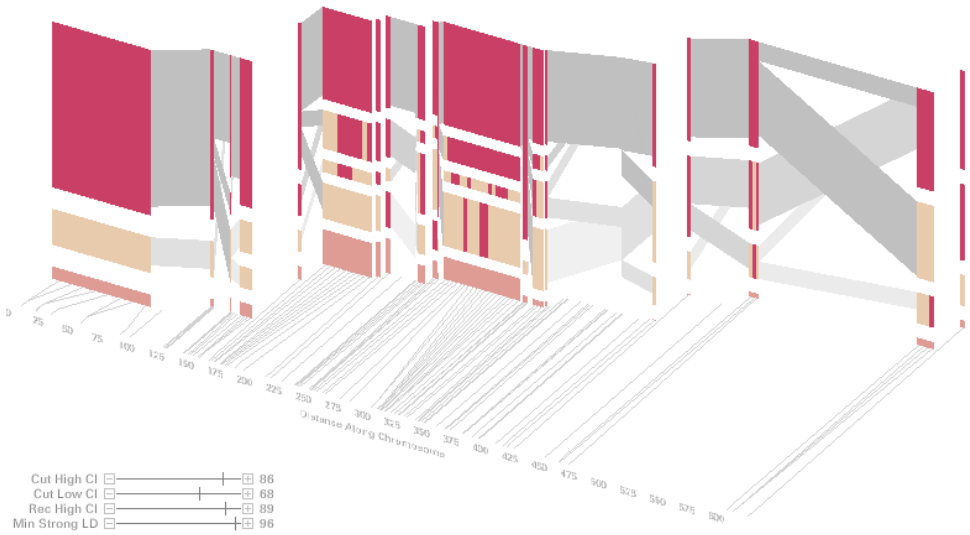


Figure 3.13: Visualisation of complex data using Processing, from Fry (2004)

3.3.1.4 vvvv

According to the vvvv website[11]:

> vvvv is a toolkit for real-time video synthesis. It is designed to facilitate the handling of large media environments with physical interfaces, real-time motion graphics, audio and video that can interact with many users simultaneously.

The software uses a visual programming interface that is superficially similar to that of Max and Pd. It is free for non-commercial use, and (at the time of writing) is only available on the Windows platform. It lends itself to visually-rich high-resolution graphics, and is mainly used in the digital arts for interactive and generative artworks. Examples of audio feature-driven visualisation using vvvv include *Seelenlose Automaten* by Patric Schmidt and Benedikt Groß(see figure 3.14).

3.3.2 Sonic Visualiser

Sonic Visualiser is an application for visualising features extracted from audio files, it was developed at Queen Mary University of London, and has applications in musicology, signal-processing research and performance practice(Cannam et al., 2006). The software is written in C++ and released under the GNU GPL license. It uses the open

[11] `http://vvvv.org`, accessed 20 September 2008

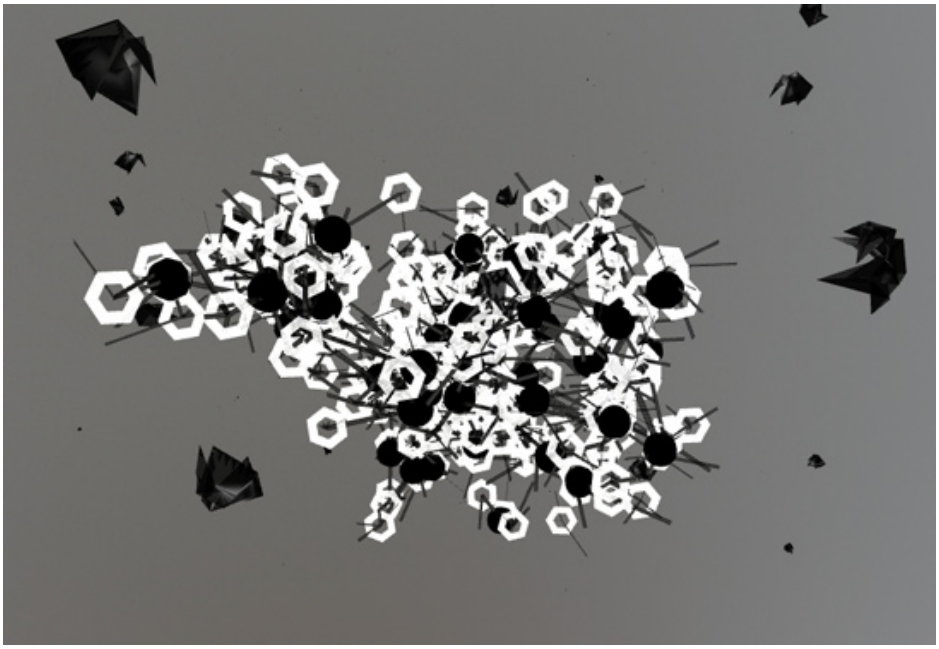


Figure 3.14: Screenshot from *Seelenlose Automaten*, a generative, feature driven audio visualisation made with *vvvv*

source version of Qt4[12] for its user interface, and is available for OS X, Linux and Windows.

The strength of Sonic Visualiser lies in its plugin-based architecture, which means that development effort can be focused on visualisation routines, whilst audio analysis functionality can be supplied by a range of third-party plugins. Sonic Visualiser uses VAMP as its plugin format (cf. section 2.3.7 for further details).

The visualisation functionality consists of line tracing (cf. section 3.2.3.1) for scalar time-series data, and sonogram-style visualisation for multi-dimensional time-series data. A selection of other features such as textual annotations (that can be exported via separate XML format), rulers, 'piano roll'-style pitch visualisation and 'time instant' markers are also provided to aid the visualisation process. A number of these facilities are shown in figure 3.15.

Sonic Visualiser allows features to be compared vertically using 'pane' widgets, or juxtaposed in layers. A fine degree of control over the colours and drawing style in each layer is provided. For example scalar time series values can be shown as 'points', 'stems',

[12] `http://www.trolltech.com`

'connected points', 'lines', 'curves', or a 'segmentation' plot. Linear, logarithmic and auto-aligned scales are provided in order to make certain data more clearly readable, and zoom and pan are provided on a per-pane basis.

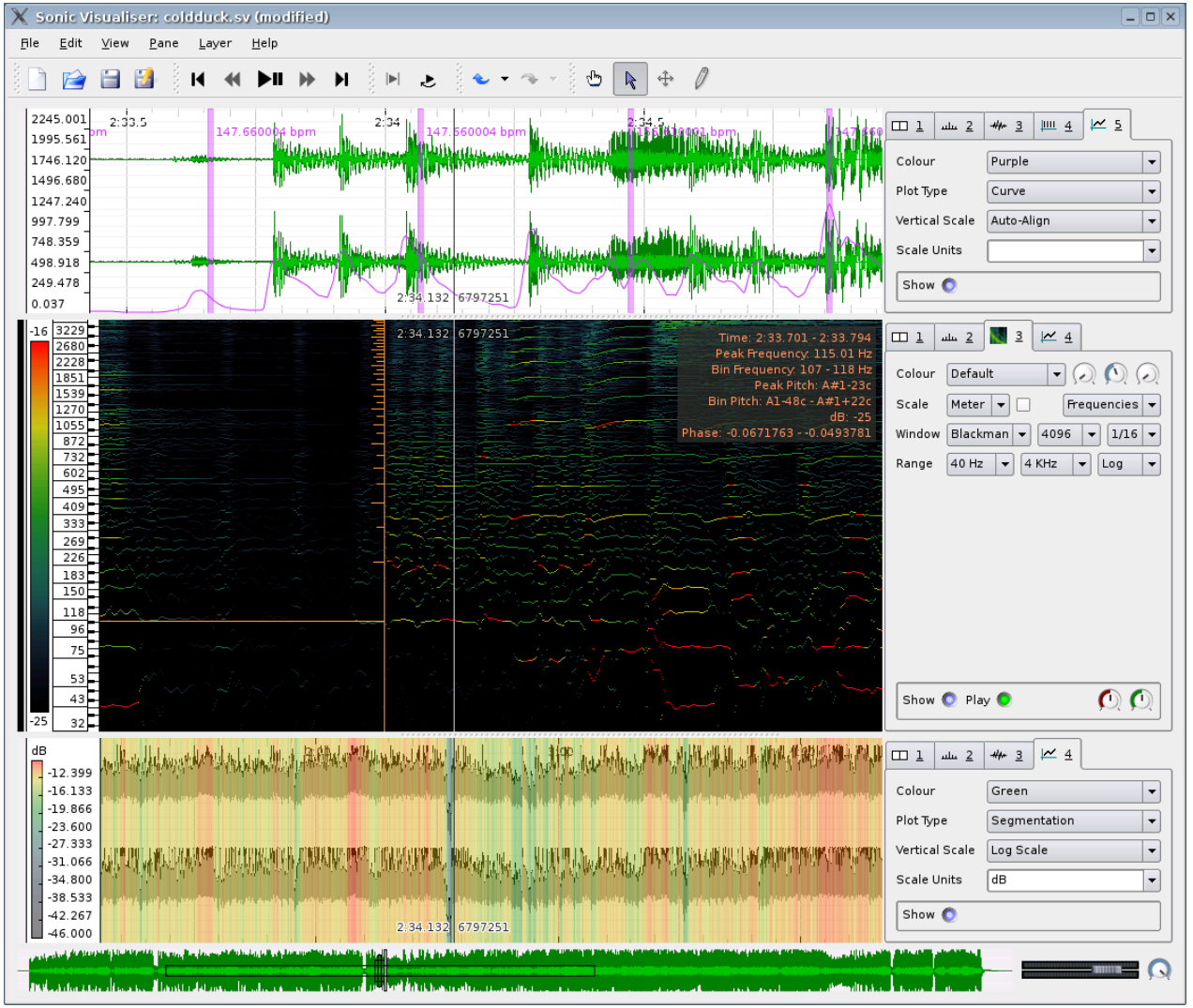


Figure 3.15: Sonic Visualiser 0.9 showing a waveform, beat locations detected by a Vamp plugin, an onset likelihood curve, a spectrogram with instantaneous frequency estimates and a "harmonic cursor" showing the relative locations of higher harmonics of a frequency, a waveform mapped to dB scale, and an amplitude measure shown using a colour shading. [Image and caption from `http://www.sonicvisualiser.org/screenshots.html`.]

All of these features combined with the large number of available VAMP plugins makes Sonic Visualiser an excellent tool for studying audio data, particularly musical recordings(Cannam et al., 2006), however it is explicitly non-real-time making it unsuitable for live performance analysis. It presents a powerful set of tools for exploring recorded audio, which could prove useful in helping composers to anticipate what analysis values will be emitted in a live feature extraction context.

3.3.3 Sndtools

Sndtools is a set of audio feature visualisation applications developed at Princeton University for the purpose of 'simultaneously displaying related audio and visual information in real-time'(Misra, Wang, and Cook, 2005). The software is written in C++ and available under the GNU GPL version 2. It takes advantage of the OpenGL graphics standard to provide hardware-accelerated rendering, making available a range of advanced visualisation processes such as rotation, scaling and lighting in real-time. Sndtools currently provides three tools for audio feature visualisation and interaction: rt_lpc, rt_pvc, sndpeek.

3.3.3.1 sndpeek

sndpeek (shown in figure 3.16) is a tool for the visualisation of the following audio features:

- Time-domain waveform
- FFT magnitude spectrum
- Correlation between left and right channels
- Spectral features: centroid, rms amplitude, rolloff, flux

The spectral features are extracted using the MARSYAS framework (cf. section 2.3.4) and displayed numerically. The time-domain waveform is displayed as an animated line trace, and the magnitude spectrum can be visualised either as an animated line trace, or an animated pseudo-3D waterfall plot where more recent frames are in the foreground and previous frames fade into the background. The left/right channel correlation is displayed as an animated Lissajous curve.

The visualisation can be controlled in real-time using a number of keyboard commands as shown in table 3.1. Elements of sndpeek are also used in the Audicle environment, which takes many novel and interesting approaches to the visualisation of live coding processes (cf. Wang and Cook (2004)).

Command	Action
s	toggle fullscreen
f	freeze frame! (can still rotate/scale)
d	toggle dB plot for spectrum
r	toggle rainbow waterfall
b	toggle waterfall moving backwards/forwards
j	move spectrum + z
k	move spectrum - z
L, R mouse button	rotate left or right
[,]	rotate up or down
-	scale DOWN the spectrum
=	scale UP the spectrum
shift–	scale DOWN the waveform
shift-=	scale UP the waveform
c	LOWER log factor of spectrum display
v	RAISE log factor of spectrum display
shift-c	LOWER amount of viewable waveform
shift-v	RAISE amount of viewable waveform
l	scale DOWN the Lissajous!
L	scale UP the Lissajous!
y	decrease delay for Lissajous plot
Y	increase delay for Lissajous plot

Table 3.1: Sndpeek keyboard commands

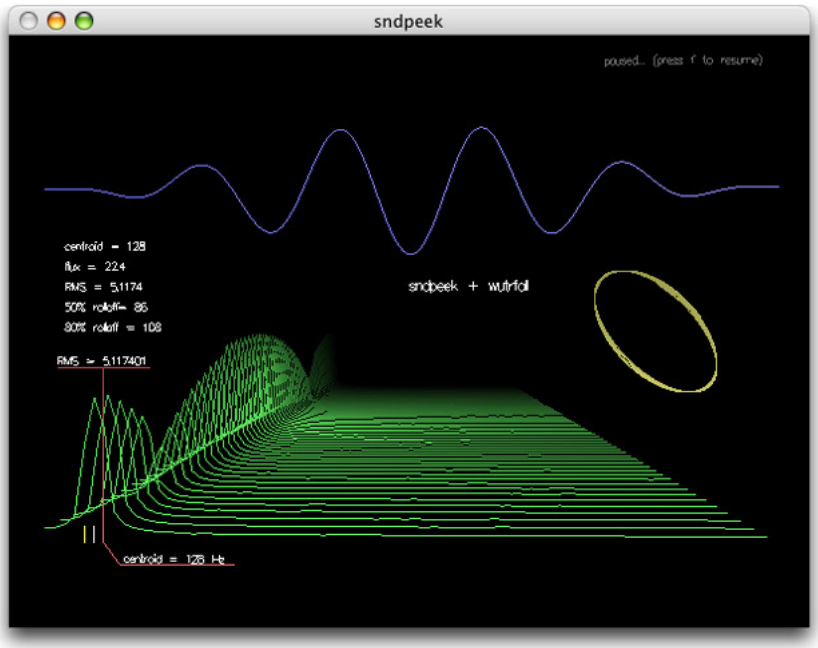


Figure 3.16: Sndpeek UI showing waterfall, Lissajous and waveform plot

The visualisations provided by sndpeek are aesthetically pleasing, and 'fun'. The ability to visualise the spectral features (centroid and RMS) at their relative position in the magnitude spectrum in real-time is particularly informative and serves to clarify the meaning of the centroid value. However in some cases the visual appeal of the animations detracts from their ability to clarify and explain the data being displayed. For example the maximum 'width' of the waveform plot is set to 1024 samples, meaning that for audio sampled at 44100Hz, it is only possible to view 23 milliseconds-worth of waveform data at a given moment. Effectively, the waveform's instantaneous amplitude is being visualised, and so another visualisation technique such as a 'level meter' may have been a more effective way to show the data. Likewise, a longer 'history' for the waterfall plot would enable the visualisation of medium-term spectral changes. The practical utility of the numerical spectral feature display is questionable since the numbers change so rapidly they only give a rough idea of the current value. Another visualisation technique such as a continuous line trace may have been more applicable. Some of these issues are addressed in the current author's Braun software discussed in section 3.4.

3.3.3.2 rt_lpc

The rt_lpc tool performs real-time LPC analysis (cf. section 2.5.3.4) , visualisation and re-synthesis. It provides a multi-modal interface for the transformation of sound. The system's architecture is shown in figure 3.17. The visual components in the display include:

- Original waveform
- Predicted waveform
- Error waveform
- Vocal tract shape visualisation
- Pitched/Unpitched decision
- FFT magnitude spectrum and waterfall plot
- Information on the current adjustable parameters including:
  - LPC order
  - Pitch shift factor

3.3.3.3 rt_pvc

The rt_pvc tool performs real-time PVC (Phase Vocoder) analysis and re-synthesis, providing time-stretching and pitch-shifting functionality. The visual components in the display include:

- Original waveform
- FFT magnitude spectrum and waterfall plot
- Information about the current parameter values including:
  - Time-expansion factor
  - Time-expansion factor
  - Analysis window size
  - Hop size
  - Phase adjustment

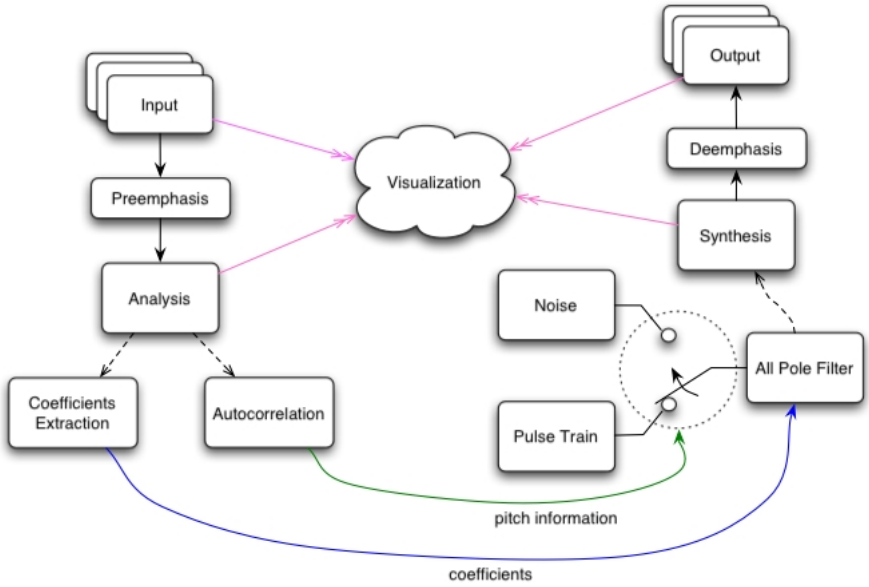


Figure 3.17: rt_lpc system architecture. [Diagram taken from a JPEG provided on the Princeton University Computer Science Wiki. Used by kind permission of Prof. Perry R. Cook].

– Buffer delay

For practical visualisation purposes, for example in live performance, rt_lpc and rt_pvc present similar difficulties to sndpeek in terms of the short time frame over which data is recorded for display, and the sheer density of information displayed at a given time.

3.3.4 Tartini

Tartini is a piece of software for the visualisation of audio features extracted from a monophonic performance in real-time. It is written in C++ using the Qt library for its user interface, and is available under the GNU GPL license. It provides extraction and visualisation of pitch, RMS amplitude, Peak amplitude, Peak correlation, and 'purity'. According to McLeod and Wyvill (2005), the purpose of Tartini is to:

> provide useful, practical feedback to musicians, both amateur and professional [...] using visual feedback from a computer similarly to provide useful information to help musicians improve their art. [The] system can help beginners to learn to hear musical intervals and professionals to understand some of the subtle choices they need to make in expressive intonation.

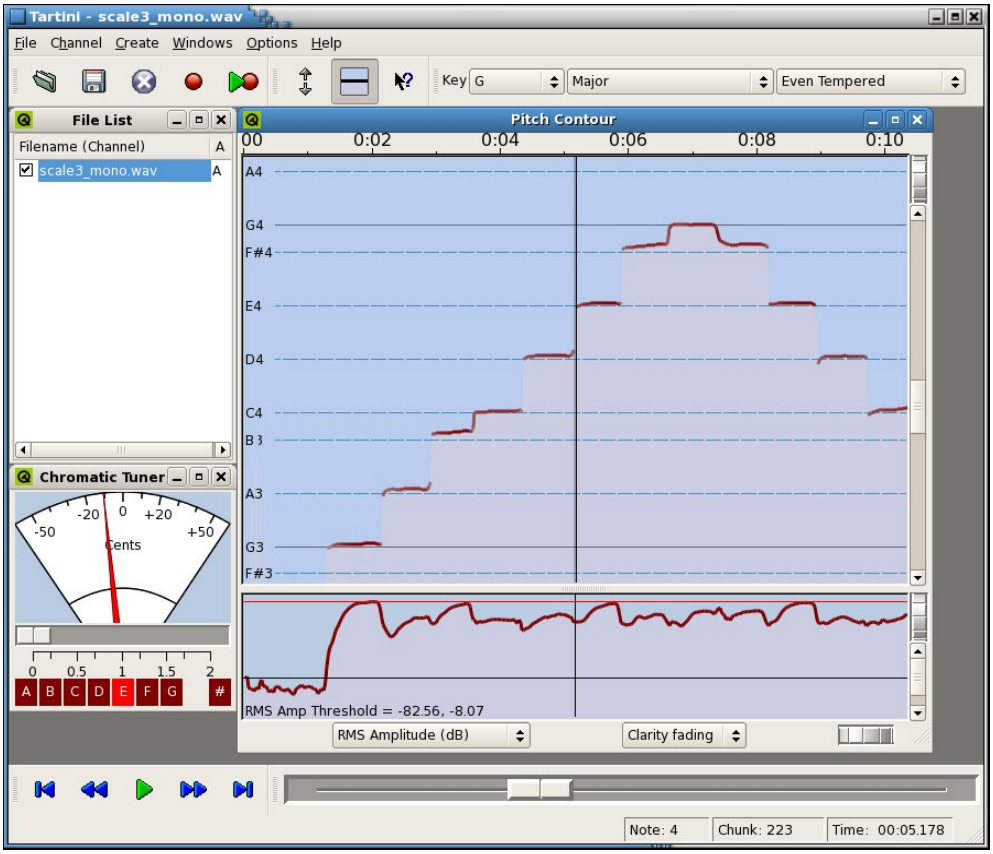


Figure 3.18: Tartini main window showing pitch contour and 'tuner'

(McLeod and Wyvill, 2005)

In order to achieve these goals, the Tartini system provides a number of visualisation techniques, which can be used individually and 'maximised' on the screen, or juxtaposed in the Tartini 'workspace'. Figure 3.18 shows the 'pitch contour' and 'chromatic tuner' windows in the same view. The 'pitch contour' is the primary method for viewing changes in pitch over time. It is presented to the user as a scrolling line trace, where the line height represents pitch, and the line's translucency reflects the 'clarity' of the pitch (i.e. the regularity of the waveform). According to McLeod and Wyvill (2005):

> We use the clarity measure in combination with the RMS power to weight the alpha value (translucency) of the pitch contour at a given point in time. This maximises the on-screen contrast, displaying the pitch information most relevant to the musician. The clearer the sound the larger the weight

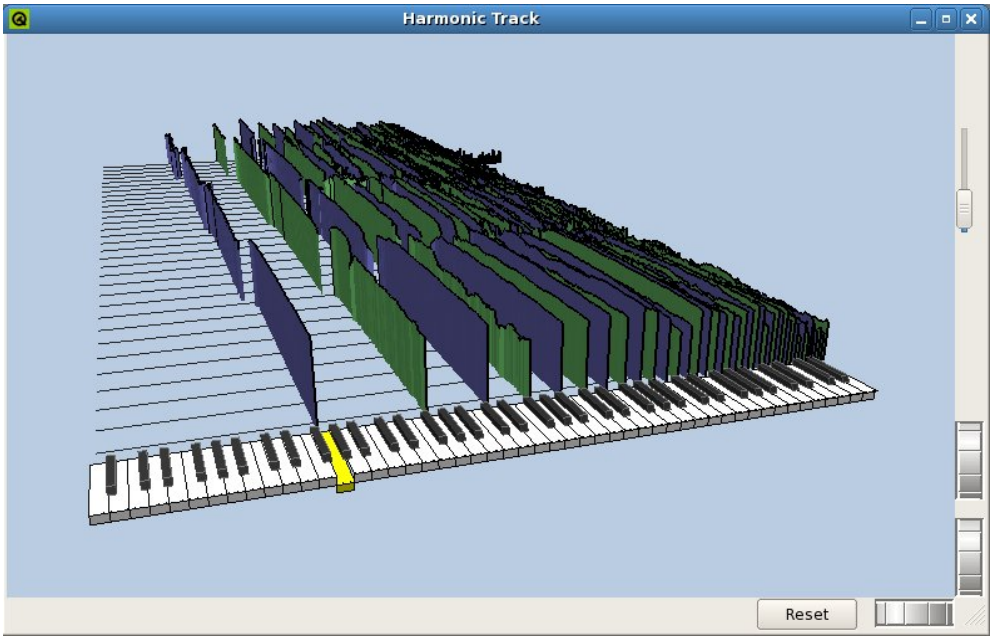


Figure 3.19: Tartini harmonic track visualisation

> and the louder the sound the larger the weight. This ensures that background sounds and non-musical sounds are not cluttering the display, but are faded into the background. Sounds below the noise threshold level are completely ignored.
>
> (*ibid.*)

In addition to the pitch contour, several other visualisation methods are given for visualising pitch. For example the 'Harmonic Track' analysis provides a visualisation of the current note and its harmonics in alignment with a 'piano keyboard' image, so that the absolute pitch and relative amplitude of each harmonic can be clearly seen. The harmonic tracks are animated in real-time in a manner similar to the sndpeek waterfall plot, so that as time progresses, they move further from the keyboard. Tartini also provides a 'chromatic tuner' view, which uses an analogue auto-tuner metaphor that would be familiar to musicians.

More recent versions of Tartini include a vibrato analysis tool that allows detailed analysis of vibrato, including visualisations of vibrato speed, vibrato depth, and an overall view that shows the evenness of the vibrato (see figure 3.20.

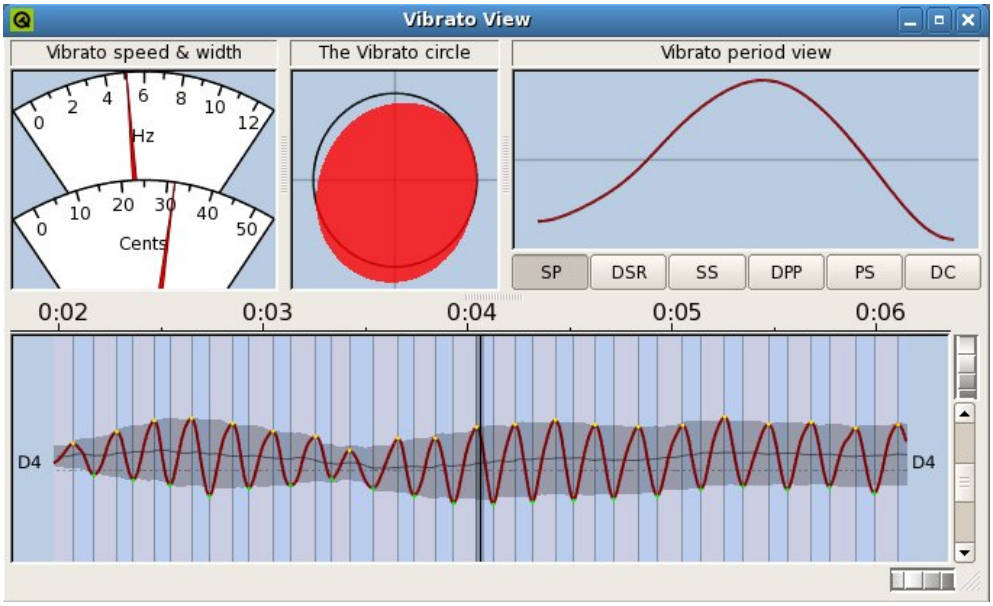


Figure 3.20: Tartini vibrato visualisation

3.3.5 Marsyas3D

Marsyas3D is the collective name of a group of tools developed by George Tzanetakis for visualising audio feature data extracted using his Marsyas library (cf. section 2.3.4). The software is currently unreleased, but is described in detail in Tzanetakis and Cook (2001). Functionality is provided for browsing (and editing) large[13] collections of audio files, and as such, the primary target user groups are the MIR and Computer Audition communities. However, some of the visualisation techniques presented in the software have application beyond these communities, and potential for use in live electronic music. The 'immersive' approach to user interface design, incorporating physical interfaces such as the Princeton Scalable Display Wall – an 8 x 18 foot rear-projection screen with 16-channel sound – are particularly promising.

The overall architecture of Marsyas3D is shown in figure 3.21, with the Marsyas3D system operating on a client-server basis with the underlying data representation (provided by Marsyas) allowing multiple concurrent views on the same data set. Several of the graphic displays are capable of real-time visualisation whilst a sound is being played back, but the system is not designed for real-time audio input (although this is technically feasible). For example, at the heart of the Marsyas3D system is the TimbreSpace browser, a system for mapping audio files into a virtual 2D or 3D space(Tzanetakis,

[13] 'large' is defined by the authors as being collections of more than 100 audio files

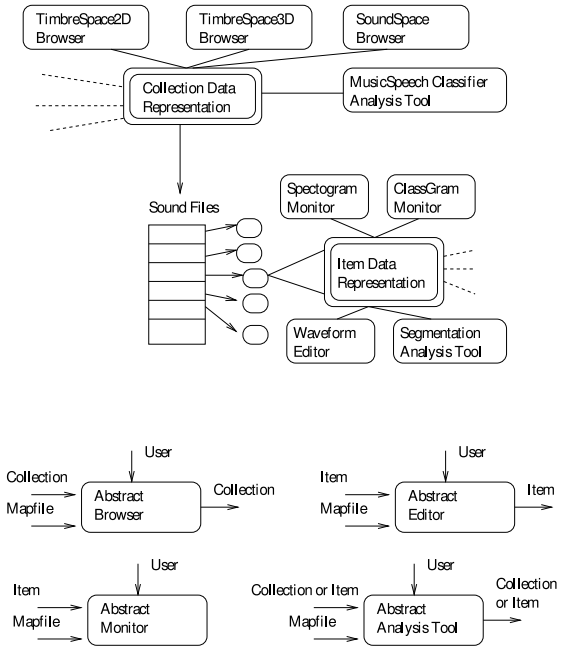


Figure 3.21: Marsyas3D User interface Component Architecture(Tzanetakis and Cook, 2001)

2002). TimbreSpaces are spatial visualisations of the kind described in section 3.2.4.2, and represent arbitrary mappings of audio features to co-ordinates in the graphical space. For example, features such as 'song tempo', 'song length' and 'auditory brightness' could be mapped to the x, y, and z dimensions respectively. Alternatively, a dimension-reduction technique such as Principle Components Analysis (PCA) or Multidimensional Scaling Analysis (MDS) can be used to reduce the dimensionality of a feature vector to a more abstract 'higher level' feature set (cf. section 1.8.4) prior to visualisation. TimbreSpace visualisations support zooming, panning and rotation as well as semantic zooming and constraint-based selection. For example the user can select all the files that have positive x values, have triangular shape and have red color(Tzanetakis, 2002).

Figure 3.22 shows a TimbreSpace visualisation where a number of soundfiles containing orchestral music are represented as cube 'objects' in a pseudo-3D space. The positioning of the objects is determined by their musical similarity as determined by a given clustering algorithm, and their colouring is based on TimbreGrams. According

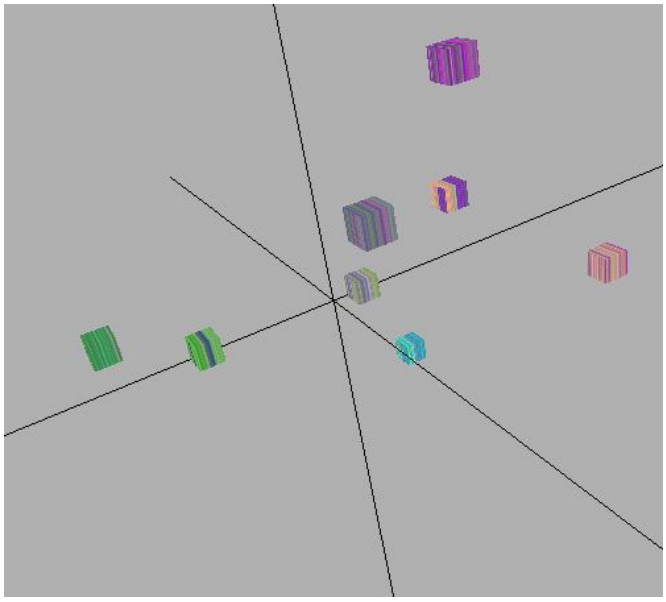


Figure 3.22: 3D TimbreSpace of orchestral music with TimbreGram texture mapping(Tzanetakis, 2002)

to Tzanetakis (2002):

> The basic idea behind TimbreGrams is to map audio files to sequences of vertical colorstripes where each stripe corresponds to a short slice of sound (typically sizes 20 milliseconds - 0.5 seconds). Time is mapped from left to right. The similarity of different files(context) is shown as overall color similarity while the similarity within a file (content)is shown by color similarity within the Timbregram.
>
> (Tzanetakis, 2002)

The TimbreGram concept combined with dimension reduction processes and 3D visualisation techniques make Marsyas3D/Marsyas a powerful system for MIR and Computer Audition. It is the goal of the current author to develop similar systems, specifically designed for use in the composition and performance of live electronic music.

3.3.6 Conclusions

In the previous sections a number of software-based systems for audio feature visualisations have been described. Of these, only the 'DIY'-based systems in Gem and Jitter (cf. sections 3.3.1.1 and 3.3.1.2), are designed specifically for use in live electronic music, with the work in Johnston, Marks, and Candy (2007) being of particular relevance. sndtools presents a real-time visualisation system, but there is a sense that this is visualisation for its own sake, rather than to serve a particular need in clarifying live audio. Perhaps the sndtools systems make more sense in the context of *live coding* where the visualisations could form part of the visual experience for the audience alongside displays of the code itself, and other more abstract visualisations.

The disadvantage of using software such as Sonic Visualiser for visualisation of audio features in live electronics is that the visualisation entails a three-stage process:

$$record \rightarrow analyse \rightarrow visualise$$

What is required for composition and performance with live electronics is 'immediate' real-time visualisation of features whilst the performer is playing as is provided in some of the 'DIY' systems. Tartini is promising in this respect, and serves as an excellent tool for pitch and amplitude analysis, however it currently lacks flexibility for visualising a broad range of features. In future research it would be good to see software developed that incorporates the domain-specificity and applicability of current 'DIY' approaches, with the substance, sustainability and professionalism of a generalised application like Sonic Visualiser. This would mean that rather than spending time developing a bespoke piece of software for every musical composition, composers and performers would be able to focus on purely artistic matters. To develop such a system would require a much closer collaboration between composers and performers, and the research/development communities as well as a well-funded and tightly managed development processes.

In the following section, a new piece of software, which I have developed in order to address some of the visualisation issues outlined in this section will be discussed.

3.4 Braun

Braun is a simple but powerful visualisation tool for the purposes of observing the relationships between features in real-time. It is named after Karl Ferdinand Braun, inventor of the first Cathode Ray Tube Oscilloscope (figure 3.23). It has been developed

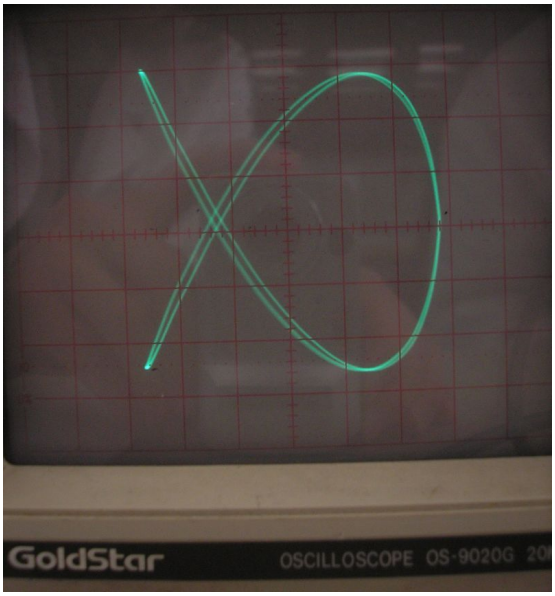


Figure 3.23: Screen of analog oscilloscope Goldstar in XY mode

so that a performer or composer can experiment with different analysis techniques in order to establish which feature might be suitable for a given application. It can also be used to visually discover how a feature extraction algorithm responds to a given input.

Braun has been implemented as a tool for visualising Open Sound Control (OSC) data on a scrolling graph, and as such it can be used to visualise arbitrary numerical data and is not limited to any particular audio features. For example, it could be used to simultaneously visualise an audio feature, and another continuous control parameter such as sensor data, or foot controller data. A typical use-case is shown in figure 3.24. It was developed for the purpose of comparing various features graphically in real-time in order that various observations can be made about those features.

3.4.1 Design

Braun is a 'building block' application following the Unix philosophy as defined by Mike Gancarz:

1. Small is beautiful.
2. Make each program do one thing well.
3. Build a prototype as soon as possible.
4. Choose portability over efficiency.

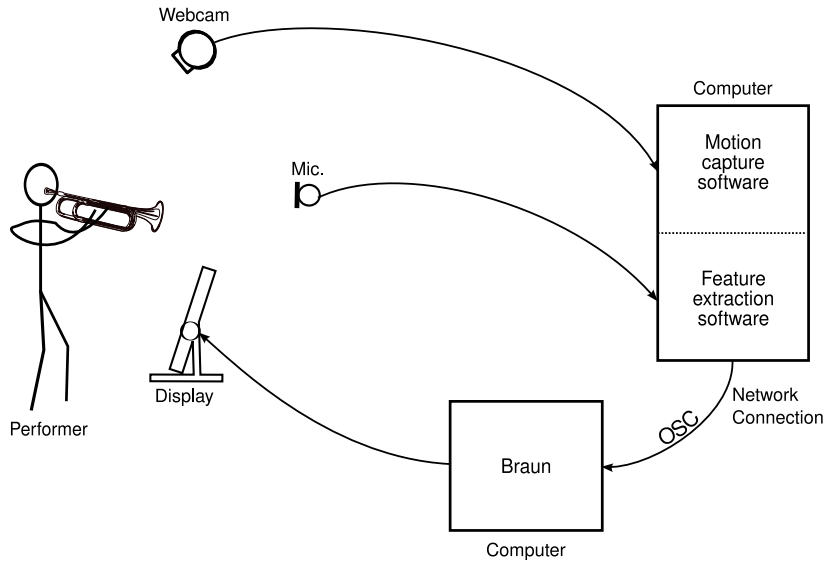


Figure 3.24: Multimodal live analysis system employing Braun for simultaneous visualisation of audio features and gesture data via motion capture

5. Store data in flat text files.
6. Use software leverage to your advantage.
7. Use shell scripts to increase leverage and portability.
8. Avoid captive user interfaces.
9. Make every program a filter.

(Gancarz, 1995)

The design of Braun follows every one of these points in its design and deployment, in addition it seeks to only provide the 'best' or 'right' configuration for use rather than providing a plethora of configurable options through preference panes.

Braun operates as an Open Sound Control (OSC) server, and requires other applications to send it OSC data on a given OSC port. This makes it possible for Braun to be very 'lightweight' since it doesn't actually generate any of the data being visualised. It enables Braun to focus on producing readable data visualisation, whilst taking advantage of ('leveraging') the power of other task-specific applications for processing and analysing the audio data. This process is shown diagrammatically in figure 3.24 where

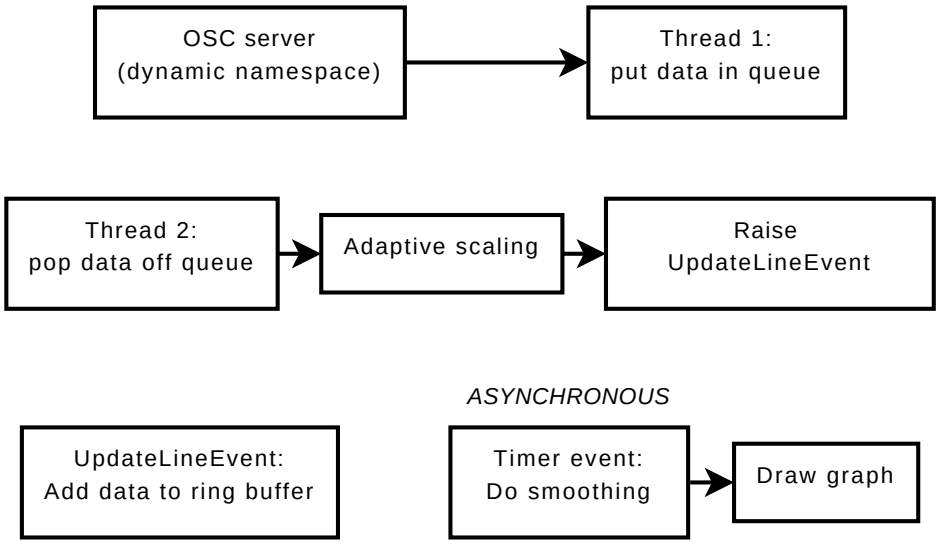


Figure 3.25: Braun control flow graph

the motion capture, and audio analysis software generates an OSC data stream, which is sent to Braun in real-time. A simplified representation of the application's control flow is shown in figure 3.25.

The design of Braun seeks to meet Tufte's axioms of 'graphical excellence' as discussed in section 3.2.1. It does this by providing the following features:

- Colour scheme chosen to maximise contrast between individual lines
- Black background to maximise contrast between lines and background
- Multi-resolution zoom and pan to provide 'detail-on-demand'(Shneiderman, 1996)
- Line smoothing to remove unnecessary data when 'zoomed out'
- Anti-aliased lines to improve clarity and visual appeal
- Current value shown in status bar on mouse hover
- Data automatically scaled to range of incoming data
- Avoid clutter by limiting the graph to five lines

In addition to these features, Braun also allows the visualiser to be paused arbitrarily to gain 'frozen snapshots' of the graph, with the optional capability for the current snapshot to be saved as a PNG image file.

Braun is written in Python using the wxPython, FloatCanvas and liblo packages, and is available on the GNU/Linux, OS X and Windows platforms. The overall structure of the application is shown in figure 3.26.

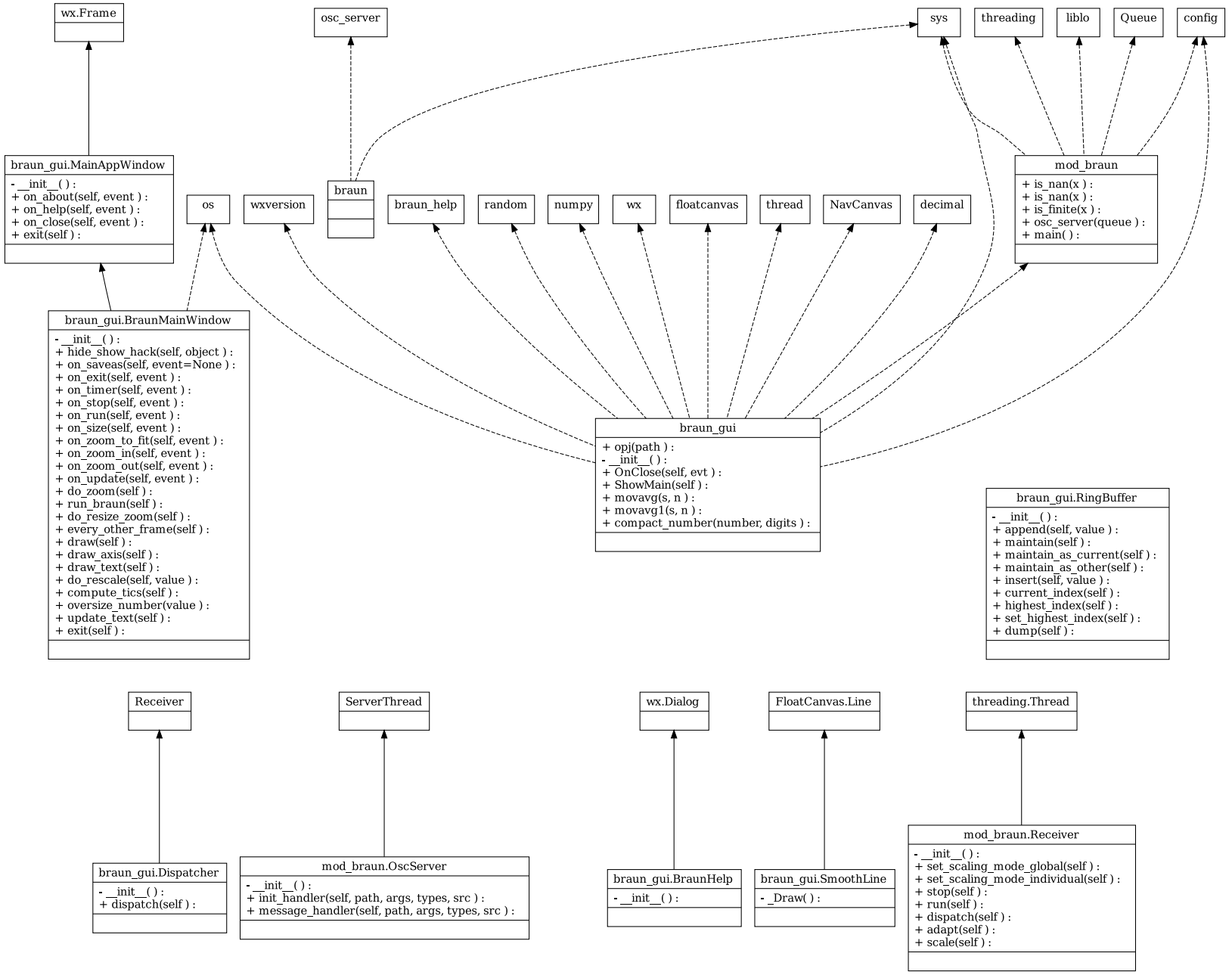


Figure 3.26: Braun class diagram, generated from code with autodia/graphviz

3.4.2 Scaling and Smoothing

In order to ensure that any data received by Braun can always be shown on the graph, Braun employs an *adaptive scaling* algorithm, which continually adjusts the y-axis scale to reflect the range of incoming data. The algorithm can be approximately summarised as follows:

1. Adapt
    - if current > maximum: maximum = current
    - if current < minimum: minimum = current
2. Scale
    - value = ((value − minimum) / range) * global_range

This is useful for showing data with comparable ranges on the same graph without having to worry about pre-scaling the data. However if the ranges have orders of magnitude difference, the data with the greatest range will obtain the greater visual significance. In such circumstances, pre-scaling is recommended prior to sending the data to Braun.

In addition to scaling, Braun provides smoothing functionality to help clarify the data. Care has been taken to provide enough smoothing to enhance the visualisation, without obscuring too much detail. In addition, smoothing is progressively removed as the user zooms into the canvas, and detail in the data is revealed to the level of individual values (see figure 3.27).

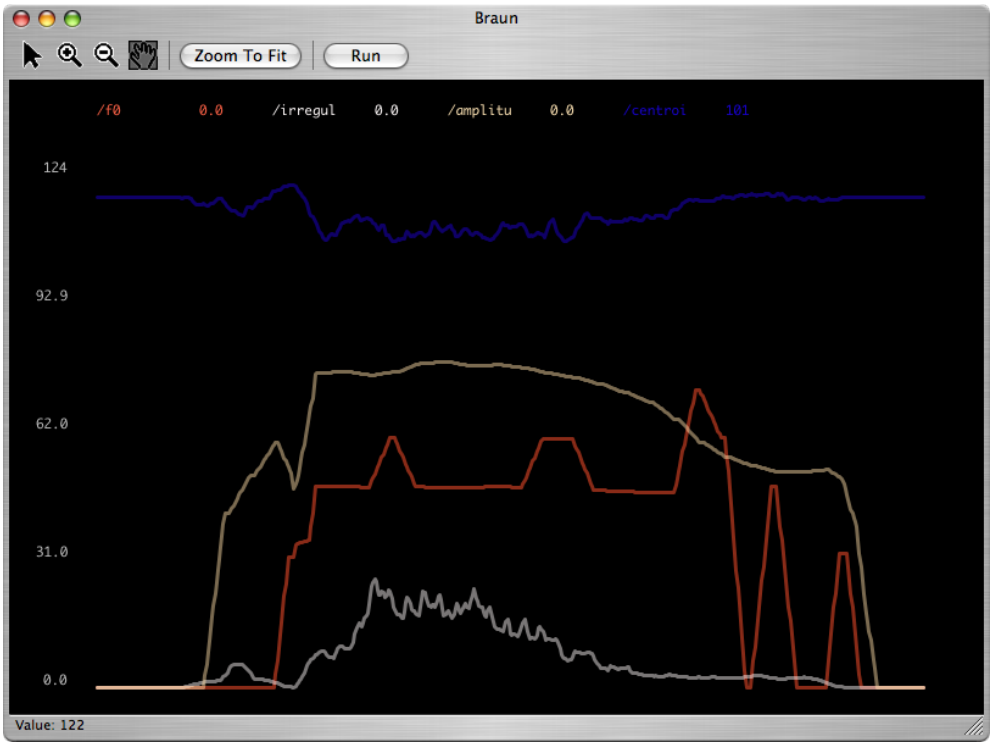
The actual smoothing algorithm consists of a simple moving average taken over one buffer's worth of data, with a period determined by the current resolution. This is expressed mathematically as shown in equation 3.1, and shown graphically in figure 3.28.

$$SMA_k = \frac{\sum_{n=1}^{n=p} x_n}{p} \tag{3.1}$$

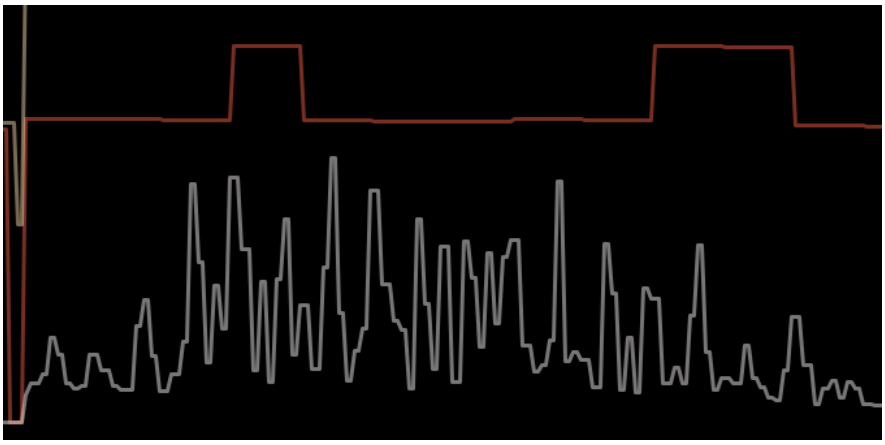
Where SMA_k is the kth moving average value, p is the period of the average and x_n is the nth value in the current group.

3.5 Conclusions

In this chapter a range of techniques for audio feature visualisation have been discussed, and observations about specific pieces of software have been made, with conclusions drawn in section 3.3.6. A new piece of software written specifically for visualising features in the context of live electronic music has been presented. The software works as an OSC visualiser, allowing a diverse range of features to be visualised through a consistent interface. This also allows non-audio control data to be presented on the same graph as audio feature data. However, Braun is not presented as a conclusive 'solution' to audio feature visualisation issues in live electronic music, but rather as a more targeted alternative to existing systems. Examination of existing systems (including Braun) shows that there is much work still to be done in providing usable, stable and domain-specific visualisation systems for live electronics data. In particular, lessons should be learned from the many 'DIY' systems that have been produced for bespoke works (installations, performances, interactive systems), and incorporated into more permanent and generalised software.

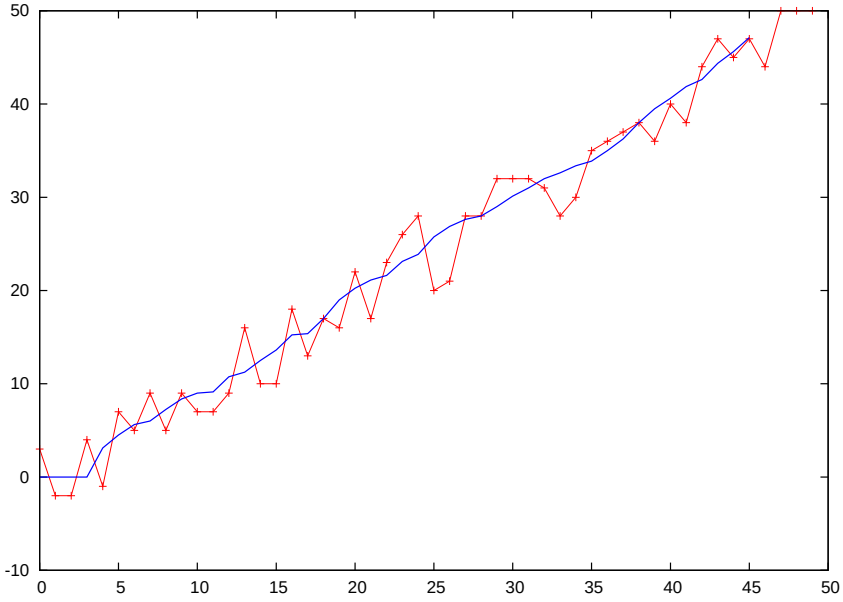


(a) Main window showing features for spoken word 'hello'

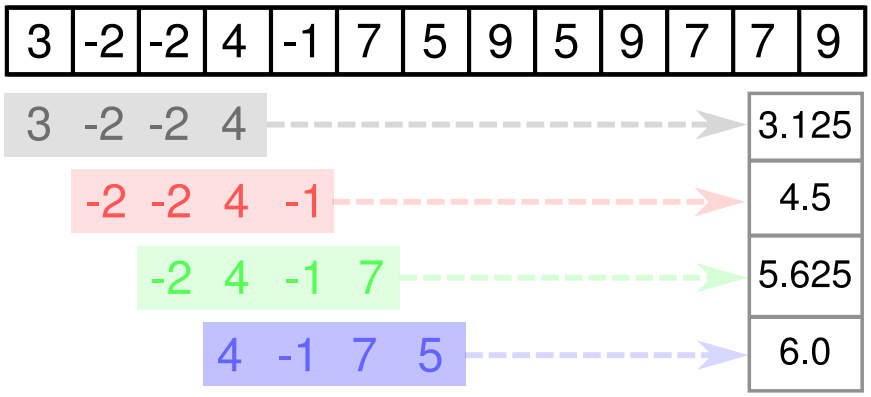


(b) Close up section showing extra detail

Figure 3.27: Braun screen shots showing multi-resolution zoom and smoothing



(a) Moving average expressed graphically



(b) Moving average algorithm

Figure 3.28: Moving average illustration

CHAPTER 4

Case Studies

Nothing resolves design issues like an implementation.

— J. D. Horton

4.1 Introduction

In this chapter a number of musical works that explore various aspects of audio feature extraction, will be discussed. The works were composed and performed during the course of this research. In some of the compositions, the feature extraction aspect is central to the overall conception of the piece, in others, it is used in specific sections as and when required. In both cases, the musical idea has always taken precedence over the technical idea.

The pieces are exposed here as case studies presented chronologically in order of composition, thus showing the technical and artistic progression of the research.

4.2 Fission

Fission for two violas and live electronics was written at the request of Rivka Golani for the 2005 Music Extra festival in Birmingham. It was composed for two of Golani's viola

Students David Matheu and Larissa Brown who performed the piece at Birmingham Conservatoire's Recital Hall in June 2005.

4.2.1 The piece

The piece divides into three movements: a dance-like opening movement; an energetic central movement; and a slow epilogue. In the second movement the process of 'fission' is characterised musically through use of the electronics to split the sound (harsh, staccato chords played by the violas) into sinusoidal spectral components (the atomic building blocks of sound) and play them back into the performance space as 'resonance'.

Fission was the first work to be composed as part of this thesis, and draws upon the research in Fujinaga and MacMillan (2000), Fraser and Fujinaga (1999) and Fujinaga (1998), which explores the use of a small set of audio features and a k-NN classifier to perform musical instrument classification. In the first movement of the work, pitch detection is used in a very simple manner to initiate a 'reverberation' or 'comb filtering' effect on selected notes. No feature extraction techniques are used in the third movement, so this will not be discussed in detail.

4.2.2 The electronics

The following sections give a technical explanation of the processes used in the live electronic aspect of *Fission*, and provide critical discussion of their artistic contribution to the work.

4.2.2.1 First movement

A graph showing the basic DSP and logical flow for the first movement of *Fission* is shown in figure 4.1. The basic principle of this system is that when a specific pitch is detected by the pitch detector, the pitch selector tests it against a set of predefined pitches. If there is a match between the incoming pitch and the predefined pitch, then the 'gate' is opened, and the viola signal is sent to the reverberator. If a non-matching pitch is received, or a given time period has elapsed, the 'gate' is closed, no reverberation is applied to the signal. This system was implemented as a Pd patch, and duplicated for each viola, with each viola having its own independent pitch set for the pitch selection. The samplewise delay was applied to the signal before the 'gate' to compensate for the time taken for the pitch to be extracted.

In the Pd patch for the first movement, pitch detection is achieved using the `fiddle~` Pd object by Puckette as outlined in section 1.7.3. `fiddle~` uses a 'Maximum Liklihood' pitch detection algorithm, (Puckette et al., 1998). Maximum-likelihood f0 estimation

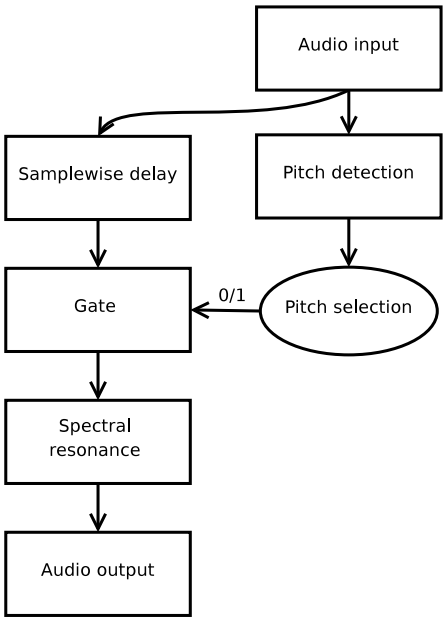


Figure 4.1: Logical and DSP flow in *Fission I*

was originally proposed by Noll (1969) for parameter estimation in human speech and musical works using the principle of minimising the difference between a given input spectrum and a range of likely harmonic spectra. In Puckette et al. (1998), the "likelihood function" is given by:

$$\mathcal{L}(f) = \sum_{i=0}^{k} a_i t_i n_i \tag{4.1}$$

Where f is frequency, a_i is a factor depending on the amplitude of the ith peak, t_i depends on how closely the ith peak is tuned to a multiple of f, and n_i depends on whether the peak is closest to a low or high multiple of f(Puckette et al., 1998). The value of f whose likelihood is highest is given as output, but no output is given if there are less than four spectral peaks detected or there is a fundamental present and the total power of the contributing peaks is not at least one hundredth of the signal power. This causes `fiddle~` to give no 'cooked' output if the fundamental is weak. A block diagram showing the `fiddle~` algorithm is shown in figure 4.2.

Puckette et al. (1998) don't give an evaluation of the performance of `fiddle~`'s pitch detection capabilities, however de la Cuadra et al. (2001) suggests that 'ML works well when the true pitch is centred on a reference signal in the matrix multiply, but [...]

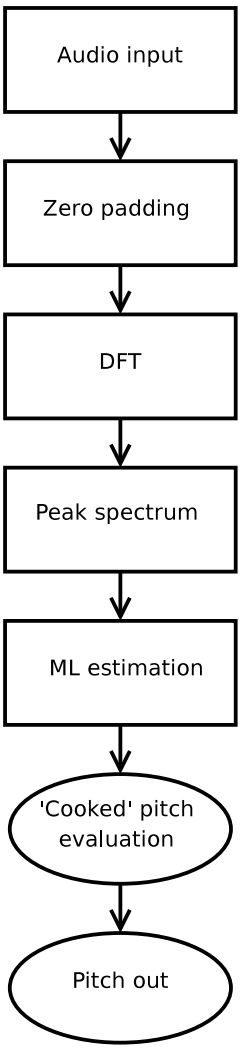


Figure 4.2: Fiddle pitch detection DSP/data graph

gives erratic answers when the input signal is halfway between two cases'. That is, because maximum-likelihood chooses the 'most likely' pitch match from a fixed set such as 'all chromatic pitches within an eight octave range', it is more likely to give inaccurate estimates when a pitch falls between two semitones. de la Cuadra et al. (2001) therefore surmise that 'Keyboard and woodwind instruments are more appropriate for ML than strings or voice since the latter instruments can easily produce non-discrete pitches, particularly in vibrato'. However, such an assertion requires further empirical evidence to establish its truth or otherwise. In *Fission* the pitch detection results of `fiddle~` were found to be very reliable, whereas in *Variations* (discussed in section 4.5), it was found to work less well. This could be attributed to the lack of vibrato in the opening section of *Fission*, and the use of polyphony in *Variations*. It was also found during the course of this research that differences in tuning between keyboard instruments (e.g. due to room temperature differences, or lack of regular maintenance) can also cause pitch detection problems.

In *Fission* `fiddle~` was used with the following settings:

- Window size: 1024
- Number of pitches to report: 1
- Number of peaks used to determine pitch: 20
- Number of raw peaks to output: 20

The window size of 1024 points gave a good compromise between pitch detection accuracy and audio latency. The additional raw peaks were used for timbre detection in the second movement of *Fission* and will be discussed in section 4.2.2.2.

4.2.2.2 Second movement

The second movement of *Fission* uses `fiddle~` not for pitch detection but for onset detection and (in conjunction with other objects) timbre detection. Together these features are used to perform a pseudo-score following task.

The musical material in the opening section of the second movement of *Fission* (bb. 1-26), comprises a series of four-note chords split between the two violas and played staccato. As in the first movement, the idea was that certain 'events' (in this case chords) would receive live electronic treatment according to the feature extracted from the live

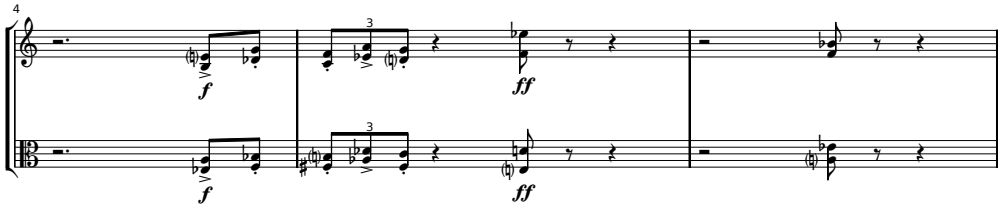


Figure 4.3: *Fission* second movement bars 4-7

sound. For example in figure 4.3 a spectral resonance technique is required on the *fortissimo* chord in bar 5, but not on the chords preceding it.

One solution to this would have been to use a manually controlled cue list, where a separate live electronics performer triggers the precession through the piece by iterating over the cues with a physical controller such as a computer keyboard. However, these techniques tend to provide a poor sense of interaction for the instrumental performers, and are prone to timing errors or omissions on the part of the electronics performer. It was also clear however, that the pitch detection technique used in the first movement would not suffice as a basis for 'score following', since the chords are highly-dissonant, giving unpredictable pitch detection results.

The chosen solution was to use onset detection to determine whether or not a note had been played, and root mean squared (RMS) amplitude to determine when a note ended. The onsets were used to step through a list of cues using a bespoke cue-list manager made with Pd. In addition to this, timbre detection was used to automatically distinguish between pizzicato and arco in bars 30-44 (see figure 4.4). The generalised flow diagram for this system is shown in figure 4.5.

Onset detection

For the onset detection the `bonk~` object(Puckette et al., 1998) was initially tried, but it was found not to be suitable for the imprecise attacks of string instruments. This finding is in keeping with the algorithm employed by `bonk~`, which is intended for the onsets of 'percussion instruments' (Ibid.). Collins (2005a) conducts an extensive comparison of onset detection techniques, comparing various methods for non-pitched percussive (NPP) and pitched non-percussive (PNP) onsets. This survey compares 16 detection functions on 3094 NPP onsets and 446 PNP onsets. The report states that:

'Performance on the PNP task was markedly worse for all detection functions assessed. High rates of false positives were an inseparable side effect of matching onsets.'

(Collins, 2005a)

In a more recent study, Collins (2005b) propose an onset detection technique based on the 'constant-Q' pitch detection method proposed by Brown and Puckette (1992), and notes improvements over previous methods, although vibrato and a lack of timbral cues are cited as reasons for possible inaccuracy. Likewise (Zhou and Reiss, 2007) propose a hybrid detection technique using a pitch-based approach or an energy-based approach depending on a 'soft' or 'hard' decision from a pre-onset detection classification stage. Stowell and Plumbley (2007) also proposes an 'adaptive whitening' scheme, which introduces a further pre-processing stage that has been shown to improve classification results across a range of onset detection methods.

Much of this research has been conducted after the composition of *Fission* and so recent implementations such as Stowell's OnsetDS library, could not be used. However, an empirical comparison was conducted between the onset detection functionality provided by the `bonk~` and `fiddle~` objects as outlined in Puckette et al. (1998). `bonk~` provides an onset detection method based on a 'bounded-Q' analysis, which is somewhat similar to that proposed in Collins (2005b).

There is little documentation about the 'attack' detection used by `fiddle~`, but it appears to use a relatively simple method based on heuristics relating to the RMS amplitude and the total spectral energy per analysis frame in decibels[1]. Tests using the opening section of the second movement of *Fission* showed that `fiddle~` outperformed `bonk~` for the onset detection task, with `bonk~` giving no output at all for many of the chords. This was unsurprising given that 'The **bonk** object was written for dealing with sound sources for which sinusoidal decomposition breaks down; the first application [being] drums and percussion'(Puckette et al., 1998). In *Fission*, `fiddle~` was used with the following messages sent as initialisation parameters:

- reattack 100 10: This sets the period in milliseconds over which a re-attack is reported if the amplitude rises more than 1dB to 100 ms
- amp-range 40 60: This sets the low amplitude threshold to 40 and the high amplitude threshold to 60. If the signal is below the low threshold, no pitches or peaks are output. If the signal is below the high threshold, no 'cooked' values are output

[1]Noted from source code available at `http://crca.ucsd.edu/~msp/Software/`

Only the sound from viola I was used as a feed to the onset detector in order to avoid the detection of multiple attacks if the diads making up the chords were not played exactly together. Likewise, highly directional miniature microphones were used to avoid acoustic spill between the instrumental feeds to the audio interface, and the distance between the performers was optimised to gain the best compromise between minimum spill, and maximum sense of ensemble between the players.

As can be heard in the recording of the live performance (cf. E — CD track 2) of *Fission*, using the 'attack' decision from `fiddle~` worked well as a means of automating the cue list for the live electronics. However, one onset was 'missed' between bars 23-26, meaning that the cuelist was one step behind at bar 30. The resonance effect heard on the *pizzicato* section is therefore incorrect. This was the only error in this performance, and was 'manually' corrected by the live electronics performer for the following cue.

In the following section I will outline the timbre detection technique used to differentiate between pizzicato playing and arco in bars 30-44 of the second movement of *Fission*.

Timbre detection

The second requirement of the pseudo-score-follower in *Fission* was a means to distinguish between pizzicato and arco playing. This was used between bars 30 and 44 to apply algorithmically controlled spatialisation to the pizzicato sections, making the sound appear to 'spread' into the auditorium from 'front-centre' to 'rear split-stereo'. In order to achieve this, an instrumental timbre recognition technique based on the system described by Fujinaga and MacMillan (2000) was used to classify the playing of viola I as either pizzicato or arco. The decision of the classifier was then used accordingly to switch the algorithmic spatialiser on or off. The schematic diagram for this system is shown in 4.5. An example section where such a transition might occur is shown in figure 4.4.

In Fujinaga and MacMillan (2000), a system is described whereby a k-nearest neighbour (k-NN) classifier is used in an exemplar-based learning system to classify previously unknown audio examples based on previous learning. An initial training database of (known) sounds is gathered, and classified by a human operator. Various audio features are then extracted from these sounds and stored as 'feature vectors' in the training database. These feature vectors are then sequentially presented to the k-NN classifier in

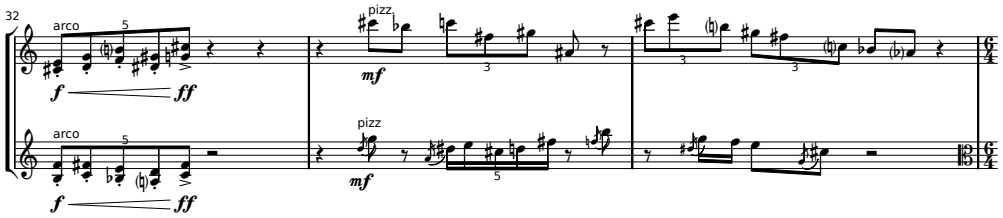


Figure 4.4: *Fission* second movement bars 30-32

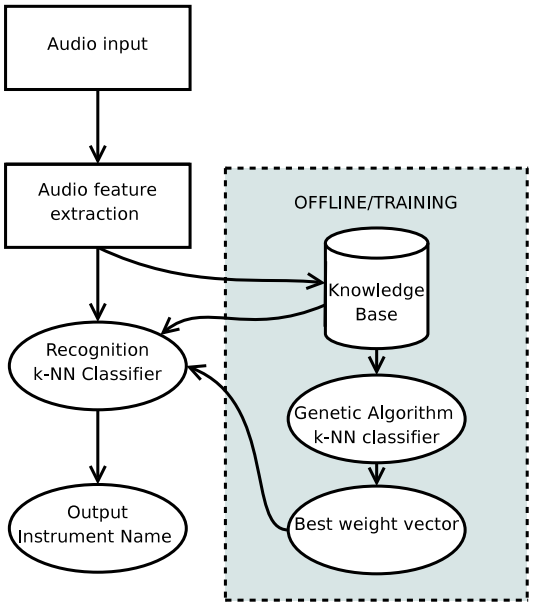


Figure 4.5: Overall architecture of k-NN classifier as implemented in Fujinaga and MacMillan (2000)

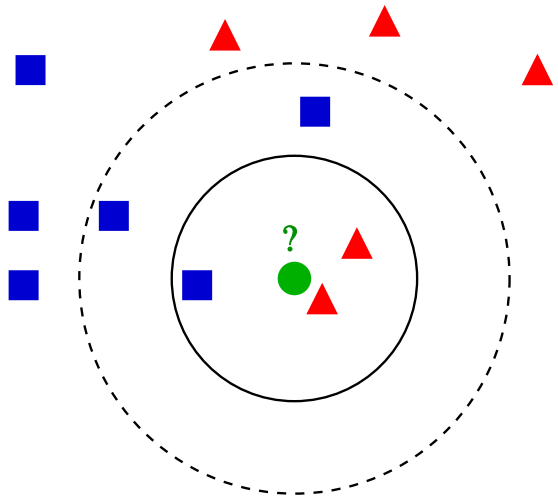


Figure 4.6: Example of k-NN classification. The test sample (green circle) should be classified as being of the same class as the blue squares or as the red triangles. If $k = 3$ it is classified as a belonging to the triangle class, if $k = 5$ it is classified as belonging to the square class. [Diagram based on KnnClassification.svg by user Antti Ajanki, released on Wikimedia Commons on 18:27, 28 May 2007. Used under the terms of the Creative Commons Attribution ShareAlike license version 2.5].

a non-real-time 'training phase' whereby the classifier is provided with each vector in turn, along with the expected classification decision.

The accuracy of k-NN (and other classifiers) can be improved significantly by using weighted feature vectors, whereby a given feature vector is multiplied by another (fixed) vector of the same dimension to give greater significance to certain features. In Fujinaga and MacMillan (2000), a Genetic Algorithm is used to find the optimal set of weights for the given training set.

k-NN

k-NN classification is a relatively simple and effective classification algorithm. It works on the principle of measuring the 'distance' between an unknown point, and a number of pre-classified points in an n-dimensional space, and assigning the class of the k nearest neighbours to the unknown point. The value of k is usually a positive integer (<= 10). A k-NN classifier is trained simply by providing a database of feature vectors, pre-labelled with their corresponding class. Once such a database has been established, classification is initiated by presenting the classifier with a new, unknown vector. The distance between the vector and all known vectors in the feature space is computed using

id	class	Centroid	Flatness	Kurtosis
1	tuba	211.723	8.80794	5.956
2	tuba	114.112	7.6041	6.932
3	tuba	213.292	5.4721	4.162
4	tuba	211.407	6.26487	4.142
5	tuba	115.153	7.25021	4.321
6	tuba	113.034	6.9553	5.989
7	tuba	13.034	6.9553	5.989
8	bazuki	714.45	3.7818	1.212
9	bazuki	414.872	3.88178	0.023
10	bazuki	616.629	7.72178	3.76
11	bazuki	315.197	7.64036	3.184
12	bazuki	714.163	7.5501	0.156
13	bazuki	813.468	7.70989	2.952
14	bazuki	715.162	7.77762	3.578
15	flügelhorn	510.445	3.4255	9.915
16	flügelhorn	159.823	2.6651	7.423
17	flügelhorn	414.45	3.7818	7.567
18	flügelhorn	314.872	1.88178	8.456
19	flügelhorn	216.629	0.72178	9.342
20	flügelhorn	115.197	0.64036	9.434
21	flügelhorn	620.433	2.345	4.234

Table 4.1: Fictitious feature data for three instruments. The k-nearest neighbours are shown in table 4.2, where $k = 5$.

a standard distance metric such as Euclidean or Manhattan distance (see figure 2.10). An implementation of the algorithm in Python is provided in Appendix A, and was used to produce the results shown in table 4.2 from the data shown in table 4.1 and figure 4.7.

Unlike MLP Neural Networks (cf. section 4.5.1.2), k-NN classifiers are relatively noise intolerant(Wilson and Martinez, 2000) and sensitive to the relative ranges in the feature vectors presented to them. It is therefore important to carefully select the most appropriate features for use in training, and to normalise and weight the features accordingly. Examples of this process will be evidenced in the following sections.

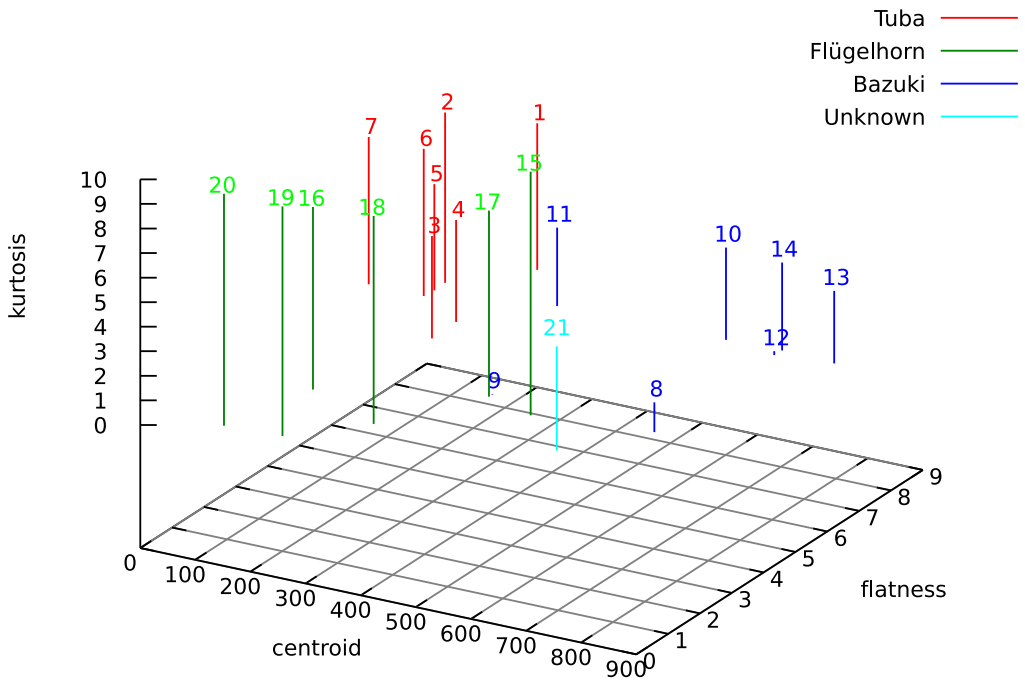


Figure 4.7: Data in table 4.1 shown on a 3-D graph. Points 8, 10, 12, 14 and 15 are the k nearest neighbours.

index	class
8	bazuki
10	bazuki
12	bazuki
14	bazuki
15	flügelhorn

Table 4.2: The majority of the nearest neighbours have class 'bazuki', the unknown vector is therefore classified as 'bazuki'

4.2.2.3 Experiment 1

An attempt was made to replicate the Fujinaga/MacMillan system outlined in Fujinaga and MacMillan (2000). In order to do this, a Pd patch was created where the `fiddle~` object was used to i) detect a fundamental pitch, ii) detect attacks, and iii) output spectral peaks as frequency/amplitude pairs. An 'AND' logic gate was added to the output of fiddle, which determined an onset as being any point at which `fiddle~` reported an onset *and* (within a given period τ) a new pitch. This improved the `fiddle~` onset reporting accuracy by a factor of 200%. A window size of 1024 samples was used, as this was considered to give the best compromise between frequency and time resolution. 20 peaks were collected for analysis. Due to the harmonic nature of the sources, many of the higher partials were reported as 0 (not meeting the fiddle internal peak detection threshold), so including more peaks was considered superfluous. The attack thresholds were set at 0 for the minimum amplitude to be reported, and 20 for the minimum amplitude at which an attack will be detected. This low threshold enabled many quieter sounds to be registered as attacks, but because of the 'AND' post processing, the occurrence of false or multiple onsets was minimal.

In the first experiment, viola samples from the University of Iowa[2] sample library were used. This consisted of 31 arco soundfiles and 28 pizzicato soundfiles, and within these there were a total of 276 arco notes and 236 pizzicato notes. Snapshots of the spectral peaks were taken at 0ms, 25ms and 50ms from onset detection. These boundaries were chosen to give a balanced representation of the attack portion of each note whilst taking account of FFT window boundaries.

A number of Pd abstractions were created to assemble a feature vector for classification. The choice of components was based on previous research in Eronen (2001), McKinney and Breebaart (2003) and Fujinaga and MacMillan (2000). This consisted of the amplitudes of the first 5 peaks, the mean of the first 20 peaks, deviation, skewness, kurtosis, spectral centroid, irregularity and tristimulus extracted using libXtract.

Three feature vectors per note were then added to a database using Karl MacMillan's k-NN object, giving a total database size of 1569 entries. This database was then saved, and reloaded (with normalisation), and tested with data from exactly the same sample set. No cross-validation was used in this initial test, in order to provide a measure for benchmarking the success of further tests. Out of 276 arco notes, 97% were successfully identified, and out of 236, 76% of the pizzicato notes were successfully identified.

[2] http://theremin.music.uiowa.edu/MIS.html

The system was then tested with samples from the Philharmonia sample collection[3]. Out of 48 pizzicato samples, 92% were identified correctly. Out of 23 arco-Non-Vib 45% were recognised correctly.

4.2.2.4 Experiment 2

In the second experiment, all viola samples from the IOWA and Philharmonia collections were used, but only one snapshot (from 0ms) was used for each note. When tested using samples from the same set the recognition rate was 96% when testing with arco samples, with 2.5% being incorrectly identified as pizzicato, and 1.5% as col legno. When testing with pizzicato samples, the recognition rate was lower, with 62% successfully identified as pizzicato and 32% incorrectly identified as arco.

4.2.2.5 Experiment 3

A Genetic Algorithm object (GAPop by Georg Holzmann[4]) was then added to the Pd patch, in order to determine a set of weights for the feature vector. According to research in Fujinaga and MacMillan (2000) this should increase the recognition rate. All weights were initially set to an equal (arbitrary) value, and the fitness function for the GA was determined empirically. A process was then set up whereby the patch would present new samples to the classifier, and if the classification failed, the GA would iterate until it succeeded. On success, the sample queue would start from the beginning again, and the process would continue until all samples had been successfully classified. After millions of iterations this point was never reached, so it is likely for the data set being used there is no set of weights that will allow 100% recognition rate. It was found that a mutation range of 0.1 and a mutation probability of 0.1 yielded good results, with a 10% increase in recognition for pizzicato. However, an empirically determined set of weights yielded a 19% increase.

4.2.2.6 Experiment 4

One of the problems encountered with experiments 1-3, is that whilst the `fiddle~` plus 'AND' logic, works well for arco onset detection, a number of false attacks are produced for pizzicato, with a significant amount of pizzicato notes not being recognised at all. This leads to spurious and incomplete training data. This could partially account for the lower recognition rates for pizzicato, that is – poor onset detection for pizzicato meant that a non-representative feature vector was provided to the k-NN classifier in both training and performance. This lead to only slightly better results than could be expected from random decision.

[3] `http://www.philharmonia.co.uk/thesoundexchange/sound_samples/`
[4] `http://grh.mur.at/software/gapop.html`

In the fourth experiment, only half or the available soundfiles were used to train the k-NN. The remainder of the soundfiles were then used to test. This yielded a 97% success rate for arco and a 53% success rate for pizzicato, increasing to 54% with a weighted vector.

This experiment was then repeated with the feature vector being 'sampled' at delay times of 0ms, 25ms and 50ms from the point of onset detection. This was done in order to take account of variations in detected onset location. This provided a far larger training set and yielded an increase in recognition rate of 15%; 16% with weights.

4.2.2.7 Experiment 5

In the final experiment, the second 50% of the soundfile database was used as a training set, and the first 50% used to test. Delay times of 0ms, 25ms, 50ms were again added to the pizzicato training yielding three feature vectors for each pizzicato input soundfile taken at 25ms intervals from the detected onset location. No delay from the onest was used for the arco soundfiles and gave a total training database size of 1668 vectors. This gave the best results for pizzicato 71% (unweighted), 73% (weighted).

4.2.2.8 Live Testing

A new database was made using a smaller range of samples in order that pizzicato and arco should be more evenly represented. Examples were selected at random from the IOWA and Philharmonia collections, although more esoteric arco techniques such as Sul Pont, and harmonics were removed since detection of these would not be required for *Fission*. It was anticipated that this might improve pizzicato recognition rates, by decreasing the similarity between the two sets of data.

For the purposes of this research, the results achieved with a live acoustic instrument in a concert environment are of primary interest. In order to test the musical usefulness of this classification technique in an actual musical work, the k-NN classifier trained in the above experiments was used throughout the rehearsal and performance of *Fission* in the Recital Hall at Birmingham Conservatoire.

Despite recognition rates of 70-80% with studio recordings, a lower rate of 60-65% was achieved in a live environment. This could be attributed to the differences in acoustic qualities between the recorded sound and the live sound as illustrated in the following list:

- The studio recordings lack ambient environmental sound that can be picked up in a live situation

- Microphone characteristics and techniques vary between the concert hall and the studio
- Performers alter their playing style and play more 'deliberately' when recording for a sample library
- The studio recordings consist of edited isolated notes, but this rarely occurs in a musical context due to legato playing, polyphony, or room reverberation

However, despite the relatively low timbre recognition rates (~60%), this was enough to provide a sense of interaction for the performers. It was observed that the performers would actually adapt their playing style to produce results that were more like the studio recordings in order to evoke a response from the live electronic system. For example at *Fission* bars 29-43, the performers tended to adopt a more detached and clearly articulated playing style than they would have done had they not be working with the interactive system.

4.2.3 Conclusions

In order to produce a reliable classification system, the means of establishing note onsets and extracting spectral peaks must be consistent and reliable. In this case, `fiddle~` has not proved to be reliable enough. This is either because the optimal settings have not been found, or there are no generic settings that will work well with a diverse range of articulations. One possibility for rectifying this could be the use of a classifier to determine onsets as well as timbre, rather than using a threshold based system, albeit a sophisticated one.

In order to improve classification results, the training set must be chosen carefully, preferably reflecting in some way the 'unknown' data that the classifier will eventually be presented with. In these experiments, the training data has been chosen arbitrarily, and some sets have demonstrated good recognition rates whilst others have not. It has been shown that in the context of a simple 2-state classifier as shown above, studio soundfiles can provide a reasonable training set for live audio. It would also be useful to conduct the experiments again with a training set with a mixture of studio recordings, and live audio from a variety of acoustic settings. However, as noted by Tetko, Livingstone, and Luik (1995), it is equally important not to 'overfit' the training data to the test data, as this can result in an inflexible classifier.

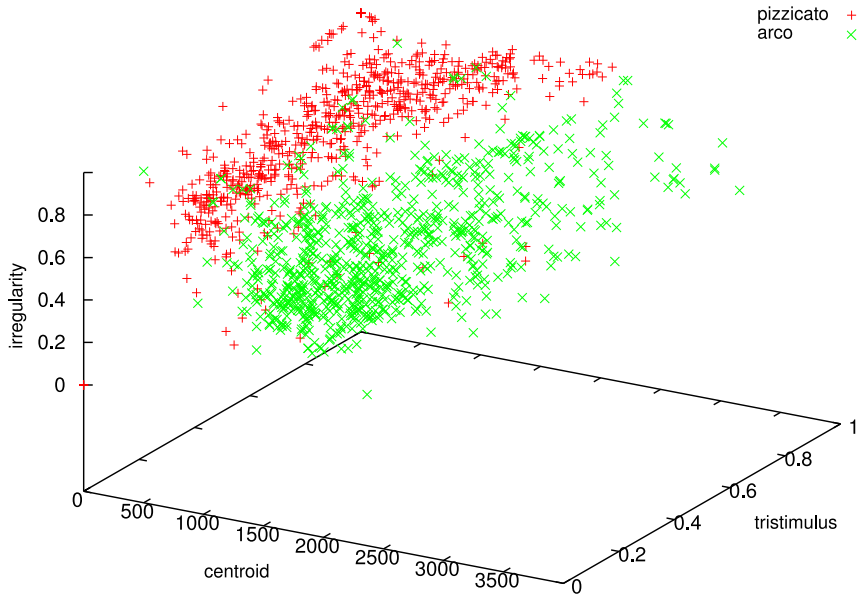


Figure 4.8: Plot of values in 3 dimensions for 735 pizzicato soundfiles and 833 arco soundfiles. Separability can be seen in the centroid and irregularity dimensions, but not in the tristimulus dimension.

Finally, graphical analysis of the training vectors using GNUPlot, gives further information about the success of the system. For example in figure 4.8 it can be seen clearly that pizzicato and arco are linearly separable in some dimensions, but not in others. Spectral Centroid is the most clear distinguishing characteristic. Further analysis of these graphs could give insights into the most appropriate feature weighting and feature selection. Cursory analysis of the graphical data also reveals some spurious data in all dimensions (although not necessarily from the same vector), which lies well outside the cluster boundaries of other data within the same dimension. This 'dirty' data comprises a small proportion of the total data, and a decision needs to made whether to keep it, or introduce a data cleaning algorithm that purges it from the training set.

4.3 Undercurrent

Undercurrent for Bass trombone and live electronics was the second composition to be written as part of this research. It uses feature extraction differently from *Fission*: rather than providing a feature vector to a classifier for 'recognition' a much smaller feature vector is used 'directly' as a control source in a many-to-one mapping. This 'aggregative'

mapping process is described in detail in section 4.3.2.3.

4.3.1 The piece

The piece lasts around 12 minutes in performance, and divides into three sections, each of which has a live electronics component. It was composed as part of Ensemble Interakt's 2006 concert programme, and performed in the Recital Hall at Birmingham Conservatoire by Dr. Simon Hall in June 2006 as part of the 'Music Extra' festival. For the electronics part, Pd was used incorporating the current author's own DSSI (Disposable Soft Synth Interface) host, which was used to host the free GVerb plugin[5], a high quality software-based reverb. The piece explores the idea of the Bass trombone as an extension of the male voice, using many extended techniques such as breathing, singing and articulating sibilants and plosives into the instrument. The live electronics part extends this concept further by extending and expanding the gestures of the performer.

4.3.2 The electronics

The following sections describe the live electronics system devised for *Undercurrent*.

4.3.2.1 Section I

The title *Undercurrent* refers to the algorithmic re-synthesis of the opening gesture, which is present throughout the piece as an accompaniment to the Bass trombone part and forms a sonic 'background' to the work. Figure 4.9 shows the logical structure of the Pd patch that controls this aspect of the piece. As changes are made between sections using the performance interface, new rules are provided to the stochastic event generator, therefore affecting a change in the live electronic 'undercurrent'. The events themselves consist of a 'tuple' of key-value pairs taken from those shown in table 4.3.

The constraints for these values are expressed on a per-section basis and are also expressed as a tuple of key/value pairs as shown in table 4.4.

The stochastic event generator contains a Markov chain (random walk), for each re-synthesis parameter, and an iterator which generates a new set of values at pseudo-random intervals. In addition certain parameters have a 'meta-constraint' forcing them to converge on a certain value. For example in part I of the piece pitch 'evolves' towards 1, i.e. the original pitch of the sound.

The overall effect is a constantly changing texture based on the opening gesture, which appears to change in a specific 'direction' throughout the work.

[5] `http://plugin.org.uk`

key	description
play	start a new event
dur	event duration
pitch	event pitch
amp	event amplitude
specstart	spectral stretch start value
specmid	spectral stretch mid value
specend	spectral stretch end value
spec_attacktime	spectral stretch rise time
spec_decaytime	spectral stretch fall time
xstart	x position start
xmid	x position mid point
xend	x position end point
x_attacktime	x position rise time
x_decaytime	x position fall time
ystart	y position start
ymid	y position mid point
yend	y position end point
y_attacktime	y position rise time
y_decaytime	y position fall time

Table 4.3: Keys for control tuples in *Undercurrent*

key	description
init	initial value of variable
inc	amount the value will be incremented/decremented by on each iteration
rise_chance	the likelihood that a value will rise expressed as a percentage
min	the minimum possible value of the variable
max	the maximum possible value of the variable

Table 4.4: Keys for constraint tuples in *Undercurrent*

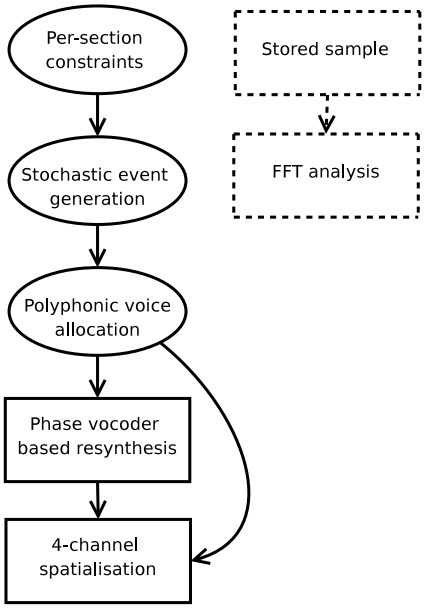


Figure 4.9: Schematic showing the logical structure for the algorithmic synthesis in *Undercurrent*

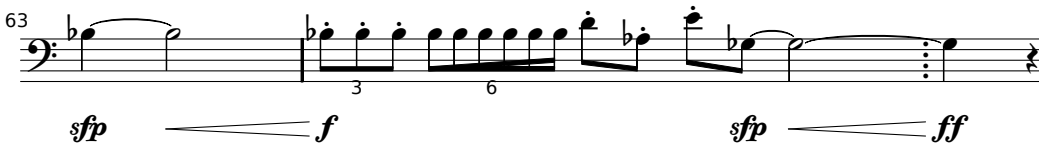


Figure 4.10: *Undercurrent* accelerando motif bb. 63'1-65'1

4.3.2.2 Section II

In the second section of undercurrent, the 'background' re-synthesis continues as described in 4.3.2.1, but with modified parameters for the stochastic event generation. However a new musical 'layer' is also introduced in the form of a heterophonic part in the live electronics, which elaborates on the gestures of the Bass trombone.

The 'protagonist' in this section is an accelerando gesture, which evolves out of a transformation of the motif introduced in bar 53 (see fig 4.10). Between bars 53 and 90, the live electronic part extends the trombone gesture by imitating the gestural shape of the accelerando motifs. This is achieved through the use of amplitude envelope following, onset detection, and pitch detection to determine the starting pitch for the gesture in the live electronic part as illustrated in fig 4.10.

The key components of the live electronics in section two are the feature extraction

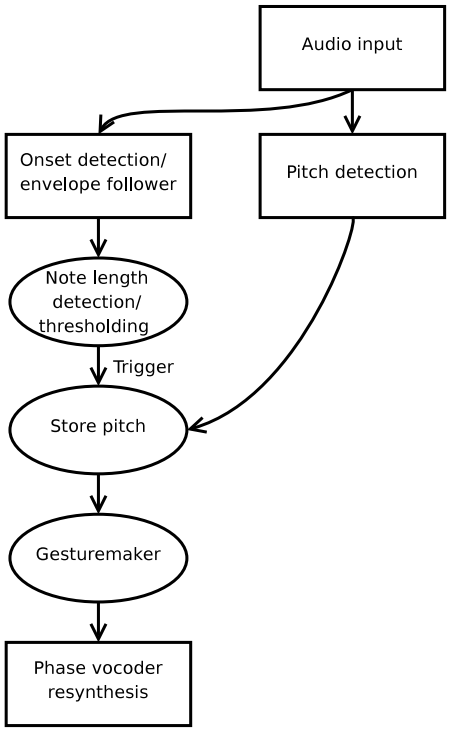


Figure 4.11: Schematic showing the logical structure of the interactive gesture generation in *Undercurrent* section 2

and the 'gesture-maker' algorithm. The basic principle is that when the trombone plays a note between 100 and 1000ms in duration, the pitch of the note will be sent to the gesture-maker to spawn a new gesture. The gesture-maker 'subpatch' itself has three voices of polyphony, and sends control data to the phase vocoder re-synthesis, which plays back re-synthesised fragments of the trombone sound (individual notes) according to the parameters generated by the gesture-maker. The musical effect of this is one of self-similar heterophony, where the electronics part appears to be imitating the trombonist, but 'branching off' from the trombone line in a fractal-like manner rather than simply imitating it.

4.3.2.3 Section III

In the final section of *Undercurrent*, two audio features from libXtract – spectral centroid and amplitude are combined using a many-to-one mapping algorithm. This enables the trombone's change in timbre as affected by the 'wah' mute, to generate proportional control data change. The algorithm for combining the two parameters is as follows:

$$x = c/K_1 + a/K_2 + C \qquad (x < 0.2, 0, x)$$

Where c is the spectral centroid, a is the signal amplitude, K_1 and K_2 are scaling constants and C is the offset. In *Undercurrent* the values used were: $K_1 = 300$, $K_2 = 20$, $C = -3$. The resultant value was sent to the signal processing unit using the following logic:

```
if ((x > 95 and x < 105) or (x > 70 and x < 90)) or
    ((x > 190 and x < 210) or (x > 140 and x < 180)):
    return x
else:
    pass
```

In addition to this, smoothing and LBYL ('look before you leap') are used to provide a smoother and more consistent data stream, and to even out any abrupt changes in value. The 'look before you leap' algorithm is an algorithm devised by Matt Wright for the `lbyl` Max/MSP object. The basic principle is that disproportionate changes in value

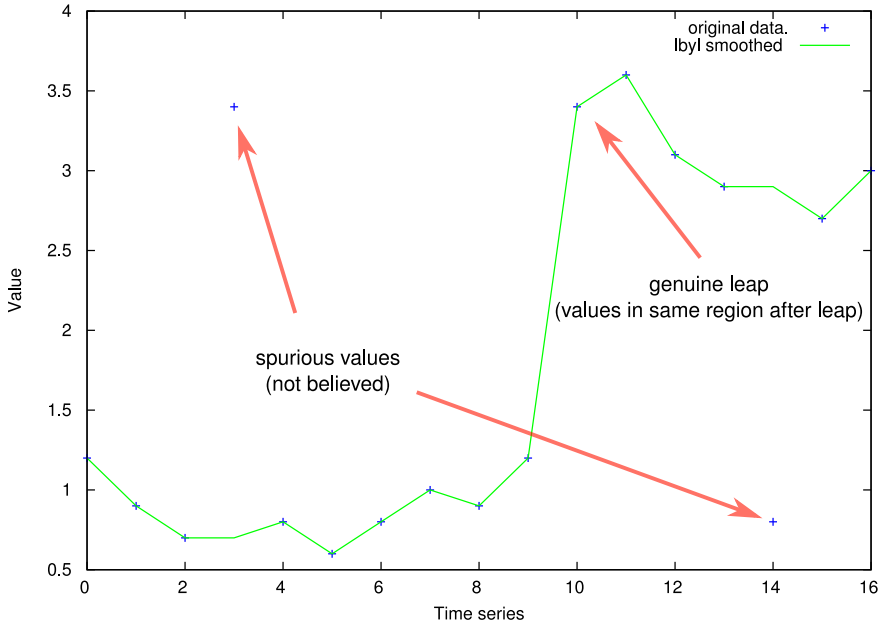


Figure 4.12: Look before you leap algorithm shown graphically

are removed from the data stream, unless the change is followed by other values in the same region without further jumps. This is illustrated in figure 4.12.

The smoothed and scaled data is used to directly control the amount of spectral shift in the live electronics part in real-time. This gives the performer an immediate sense of interaction, with a direct sense of cause and effect between performed gesture, and resultant processing in the electronics. Likewise, this causality is also obvious to the audience, with an obvious and immediate correlation between performance gesture – both musical and physical and the change in the electronics part. This was found to be most effective, but it was felt that the technique could become musically uninteresting if used too often throughout the course of a work, or used merely for effect without a sense of integration with the musical material used in the rest of the work. In the context of *Undercurrent* a sense of integration was achieved by relating the gestural shape of the wah-controlled timbral processing to the musical contour of the gesture shown in figure 4.10.

4.3.3 Conclusions

Other than *Sparkling Box*, which was not composed by the current author and is described in section 4.4, *Undercurrent* is the only piece submitted as part of this research, which includes an element of improvisation. There was a strong sense of collaboration between the performer and the current author when the work was composed, and both the score and the live electronics part evolved out of a reciprocal process of experimentation with instrumental technique and live electronic processing. However, in terms of feature extraction, it would have been useful to experiment with different features for control data in section three of the piece. For example, the 'spectral rolloff' feature had not been included in the libXtract library when *Undercurrent* was composed, but it might have provided a more suitable feature for following the wah mute timbre than amplitude combined with centroid. As a result of this process 'spectral rolloff' has now been added to libXtract.

4.4 Sparkling Box

As part of the artistic aspect of the Integra project (cf. section 1.1), composer Malin Bång was commissioned to write a piece of music for instruments and live electronics. The live electronics element was developed in collaboration with the performance technology research centre, Integra:Lab at Birmingham Conservatoire.

4.4.1 The piece

On November 5th 2006, Bång came to the studios at Birmingham Conservatoire to discuss the composition of a new piece for small ensemble and live electronics. The ensemble comprised of: contrabass, trombone, bass clarinet, piano and 'cello.

Bång presented an idea about creating 'dialogues' among the musicians in the ensemble and between the musicians and the electronics (which could also contain recordings of the instruments). She had composed a few phrases for each instrument, each having a starting point in a movement chart that was constructed from symbolic metaphor prototypes of different kinds of movement or motion. This chart contained different ways of increasing and decreasing speed or intensity, as encapsulated by the five symbols: wind, bike, squirrel, machine, and a café-bird. Bång had made recordings of the instrumentalists performing these phrases, which had 'space' for the musicians to improvise according to the symbolic metaphors she had devised. She also presented record-

ings of the performers speaking about their 'wildest experiences' in both Danish and English.

It was proposed that one of the key components of her piece would be the use of special dialogue moments, when a player chooses what instrumental material (from the electronics) she/he would like to dialogue with. Most of the discussion during meetings about the piece concerned the nature of these dialogues, and how they might be realised with the electronics. Possibilities discussed were:

- The electronics are controlled by a dedicated performer, who follows the score, and triggers changes at specific points
- The performers have some kind of physical interface (e.g. foot switches) through which they can trigger changes in the electronics
- Some kind of 'machine listening' is employed whereby the live electronics triggers its own changes via some audio cue (e.g. a change in pitch, dynamic or timbre)

In addition to this interactive element, Bång requested that the piece should have another 'layer' that comprised of a series of pre-composed background 'montages' that were triggered at set points in the piece. The material for composing these montages was taken from the instrumental and vocal recordings of the performers.

4.4.2 The electronics

According to the composer, the aim of the electronics was to 'create dialogues in different ways, using the electronics as a platform and a dialogue partner to the musicians'[6]. It was therefore decided that the system needed to 'listen' to the performers as they played, and 'respond' by playing back short pre-recorded audio excerpts. After initial discussions it was clear that the 'response' of the live electronics needed to appear as organic, and that it should be neither too predictable nor too random sounding.

In order to achieve this, it was decided that an audio feature extraction system should be implemented, which analysed the sound as the musicians were playing, and returned values representing various sonic characteristics. These characteristics were:

- Tempo
- Loudness

[6] Malin Bång extract from Integra blog

- Pitch
- Brightness
- Noisiness

It was decided that to use these parameters directly would not provide a musically-interesting correlation between the performers' gestures and the resulting sound, so instead the first (and second) derivatives of the features were taken. In effect we devised a system that was measuring the rate of change of the following musical qualities:

- Accelerando/Decelerando
- Crescendo/Diminuendo
- Rising pitch/Falling pitch
- Brighter sound/Duller sound
- Noisier sound/Harmonic sound

This meant that we could define 'responses' from the computer based on the performers' gestures rather than their 'current state'.

A system was setup whereby the composer could set a 'range' for each of the above parameters, and then assign a probability to each sound in a given 'pool', which determined the likelihood that the sound would be triggered when the performer's action caused the value of a given parameter to enter the defined range. In effect the composer could define a rule such as: 'when the bass clarinet causes an accelerando value of 0.7, there will be a 30% chance that soundfile X will be triggered'. The soundfile 'pool' available to each performer was also determined by the composer. The schematic for the live electronic system is shown in figure 4.13.

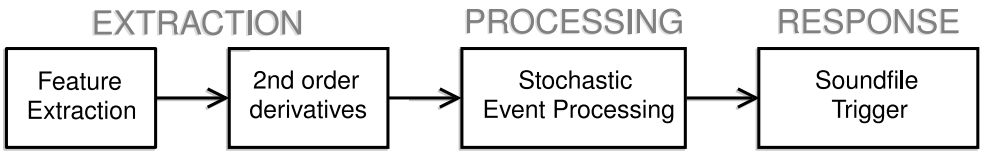


Figure 4.13: *Sparkling Box* live electronics flow diagram

4.4.2.1 Max/MSP implementation

The live electronics system for *Sparkling Box* was implemented in Max/MSP, which was chosen because the composer already had a degree of familiarity with it. In addition to meeting the musical requirements of the work, it was important that as an Integra project, the design and construction met with the 'Integra philosophy'. This meant that the patch should have clear, consistent user interface, which made operation as easy-as-possible for the user without compromising the underlying functionality. The most important design-goal when constructing the patch was that it should 'just work', and that minimal interaction should be required from the technical staff in concerts and rehearsals. Despite some of the limitations posed by Max/MSP[7], we were able to achieve these goals with a reasonable degree of success.

The bottom 'OUTPUT' section (cf. figure 4.14) is for reference only, and displays the current status of the various audio features being extracted from the performer's sound. An 'Input Source' can be selected either by using the drop-down menu, or using a cuelist. Once an input source has been selected, the patch is immediately active and will begin responding to the sound being presented to the given input.

The patch has two modes 'Event Driven' and 'Continuous'. 'Event Driven' only takes feature measurements after a note onset has been detected, whereas 'Continuous' constantly takes measurements at a given interval. 'Event Driven' was found to be the most useful.

The window shown in figure 4.15 is opened when the user selects 'open pools' from the main window. This enables the composer to add soundfiles to the 'pool' for each microphone, whereby each microphone corresponds to an instrumental input. In future versions, more sophisticated functionality will be added, to add and remove soundfiles arbitrarily from the pools, but for the purposes of this piece, adding the soundfiles was a 'once only' operation.

[7] By 'limitations', we mean the limitations of the basic Max widget set. Javascript GUI programming was not an option due to time constraints

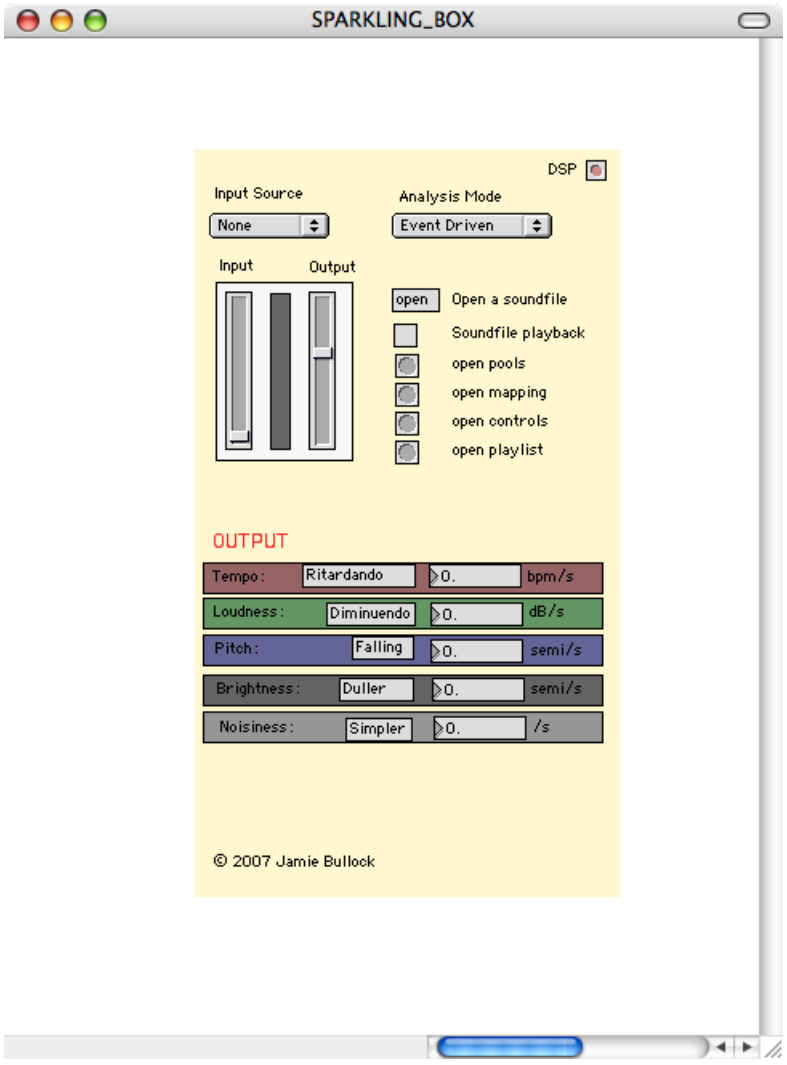


Figure 4.14: *Sparkling* Box main screen

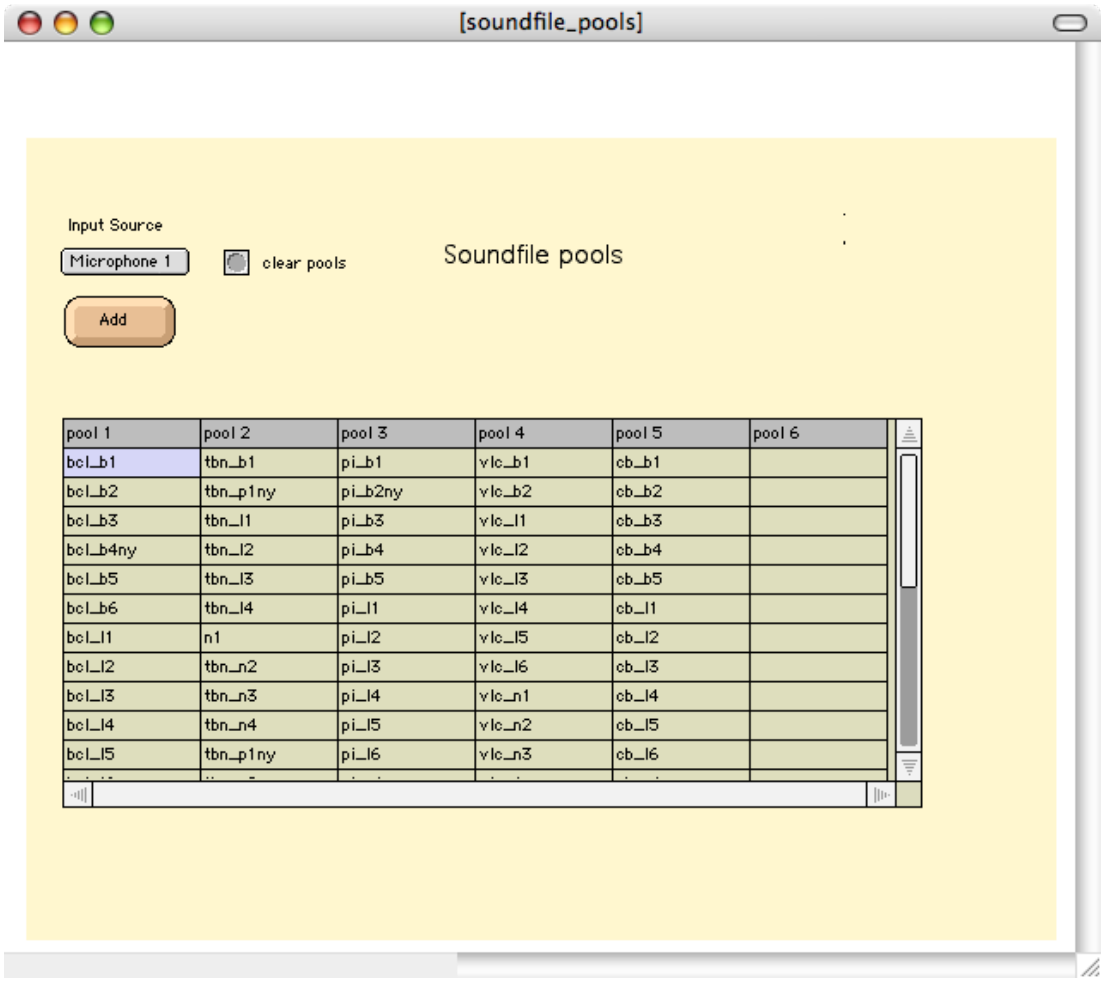


Figure 4.15: *Sparkling Box* soundfile pools

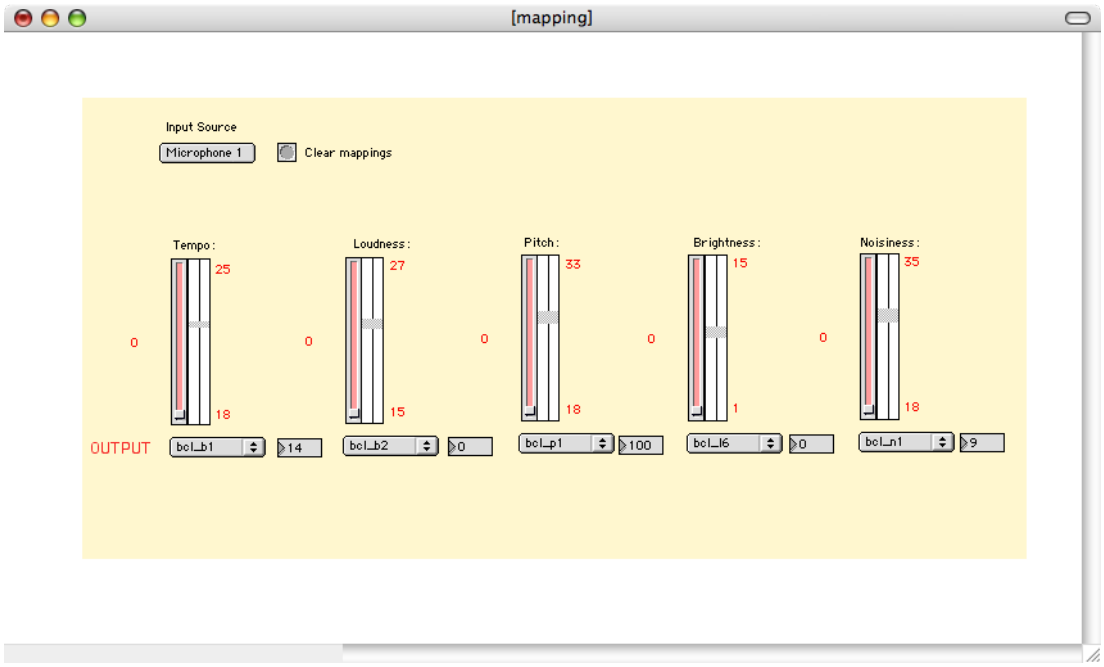


Figure 4.16: *Sparkling Box* mapping screen

When clicking 'open mapping' from the main (top-level) patch window, the window shown in figure 4.16 is displayed.

This window enables the composer to set a 'trigger threshold' for each parameter for each microphone, and assign likelihoods to each soundfile for each parameter. This example shows that for Microphone 1, when the accelerando rate is between 18 and 25, the soundfile 'bcl_b1' has a 14% chance of being triggered, and when the loudness 'crescendo' rate is between 15 and 27, the soundfile 'bcl_b2' has a 0% chance of being triggered etc.

Figure 4.17 shows the controls for the measurement of each audio feature. The number beneath each control indicates the number of 'events' over which a given feature derivative is measured. The system takes a moving average over a given number of events whereby in 'Event driven' mode, an event corresponds to a note onset, and in 'Continuous' mode an event corresponds to a new 'raw' feature value. A value of 8 was found to be satisfactory for all of the features, and this was therefore set as the default.

When clicking 'open playlist' from the main (top-level) patch window, the window shown in figure 4.18 is displayed. This window shows the master cuelist for the piece. This is the most important interface during performance. In *Sparkling Box*, the cues

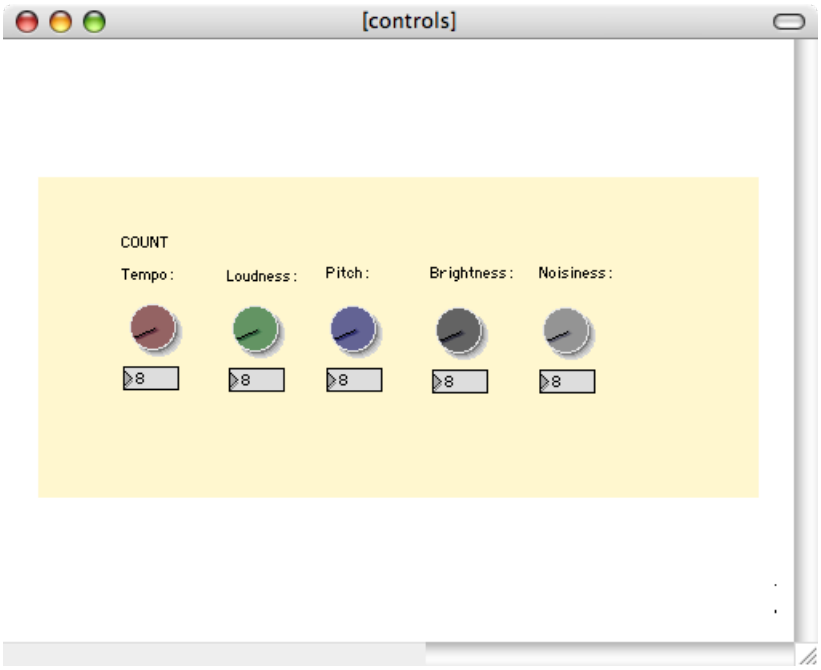


Figure 4.17: *Sparkling Box* controls

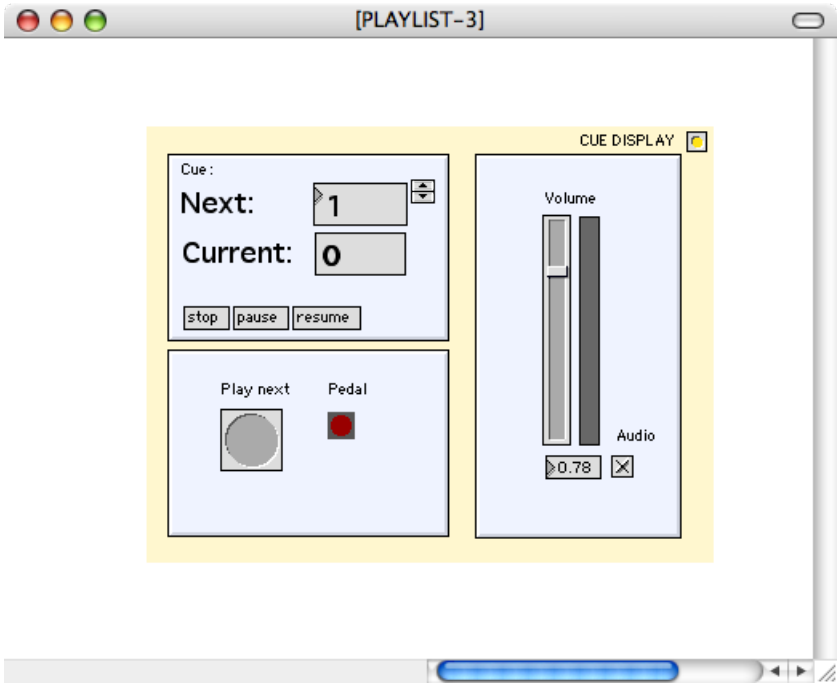


Figure 4.18: *Sparkling Box* playlist screen

were advanced using a MIDI footpedal, controlled by the Bass Clarinetist. When the footpedal is pressed, the 'Pedal' indicator lights, and the next cue is played. Each cue corresponds to either the playback of a background 'montage' soundfile, or the activation of one of the microphones for the 'dialogue' sections of the work. In future versions of the piece it might be worthwhile to explore automatic advancement through the cuelist as used in the current author's own work, *Fission* (cf. section 4.2).

4.4.2.2 The Live electronics setup

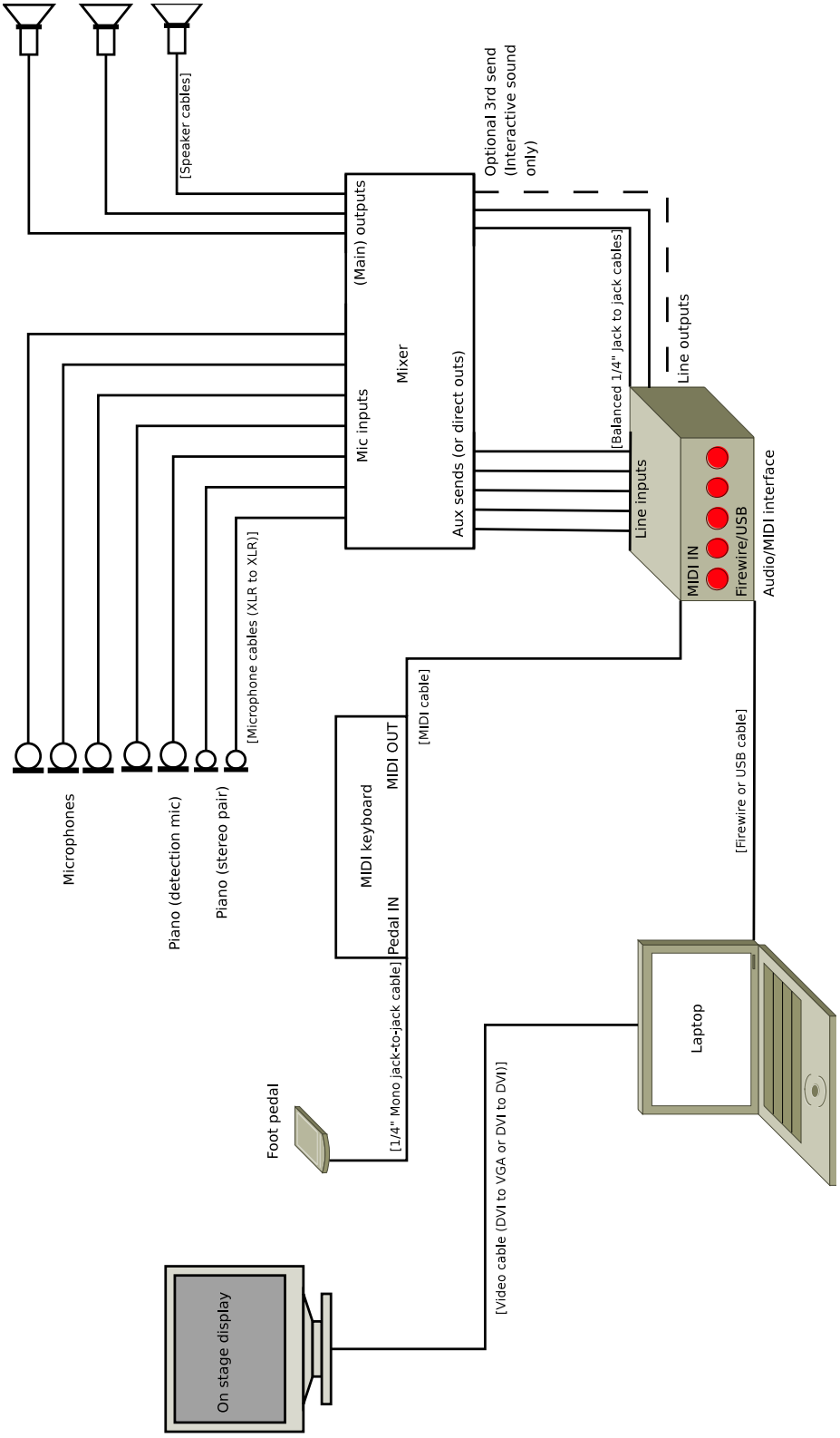


Figure 4.19: *Sparkling Box* schematic diagram

The schematic diagram in figure 4.19 shows how the live electronics for *Sparkling Box* should be set up in performance and rehearsal. In this setup, the optional on-stage display can be used to give a visual indication to the performer controlling the cues. The piano's 'stereo pair' microphones are only used for amplification, and are not fed to the computer - a separate 'detection mic' is used instead. The detection microphones should be gated and compressed to give the best possible signal to the feature extraction software, however care should be taken not to lose, or smear the note attacks

4.4.3 Performance

The piece, *Sparkling Box* was performed by the Athelas Sinfonietta on Friday 31st August 2007 at Norrköpings Konstmuseum (Kristinaplatsen) in Norrköping, Sweden as part of the "Nordic Music Days" Festival. The piece was played twice on the same day, with a full auditorium on both occasions. The live electronics was directed by Lamberto Coccioli[8], and was reported as providing a good sense of interaction for the performers. An audio recording is provided on track five of the CD accompanying this document.

4.4.4 Future Work

The system developed for this piece provides a robust platform for further development including increased generalisation so that the interactive element might be used in works by different composers. The system also suggests possibilities for a more semantically meaningful layer of mapping, with the software responding to the 'mood', 'style' or gestural shape of the players' improvisations.

4.4.5 Conclusions

In the previous sections, a live electronics system developed for a work composed by Malin Bång as part of the Integra project has been described. The piece was performed at an international music festival, and because the live electronics software was designed with usability in mind, the system was operated without the presence or input of the current author. Using audio feature extraction as a means of generating control data for stochastic processes proved to be a successful way to create a sense of interaction for the performers of the work. In terms of the 'instrument-player continuum' discussed in section 1.5.4, the *Sparkling Box* electronics tend towards Rowe's 'player paradigm' in imbuing the electronics with a degree of autonomy through the use of stochastic processes. However, even with a fairly sophisticated probabilistic approach the sense of musical responsiveness and correlation between instrumental sound and *recorded* instrumental sound was sometimes too 'obvious' or at times too 'random', and in future

[8] `http://www.lambertococcioli.com`

versions of the system it would be interesting to explore the possibility of a 'machine learning' system where the software actually *adapts* to the performers' gestures rather than simply responding to them.

Anecdotal evidence gained from working with members of the Athelas Sinfonietta in testing and developing the electronics suggests that the opportunity to work and experiment with similar systems would be welcomed. The collaboration between the composer, performers, and the current author was one of the key elements in the success of the live electronics, but the sense of interaction could have been stronger had the performers been empowered (through more usable and readily accessible technology) to develop their own modes of interaction with the technology.

4.5 Variations

Variations for piano and Live Electronics is the most substantial piece composed as part of this research, and its composition has been an ongoing project since the beginning of the research process. It was performed in 2007 at the Birmingham Conservatoire Recital Hall as part of the 'New Generation Arts' festival.

The 'theme' of *Variations* is the opening gesture shown in figure 4.20. The variations expand and develop this gesture, with the electronics providing musical resources for variety beyond the scope of what would otherwise be possible with the piano alone. In most of the variations, the electronics are used to expand the timbral or gestural possibilities of the piano, through the use of resonance, delays, harmonisation and granular synthesis, but in the fifth movement the notion of 'the instrument with live electronics' is subverted, and the piano becomes the 'processor', acting as a resonator for the live electronics part, which plays synthesised chords triggered by the electronics performer.

In addition to the theme exposed in the first variation, there is a secondary theme, which the work draws upon, from Webern's piano *Variations* Op. 27, Variation I. This work serves as an 'inspiration' for *Variations* particularly in its use of symmetry as a compositional determinant. The gestural shape that serves as the basis for Webern's first Variation is also 'referenced' in the final movement of *Variations*, as can be observed by comparing figures 4.21(a) and 4.21(b).

In the following sections the use of live electronics in each variation will be discussed, giving emphasis to the use of feature extraction and related techniques.

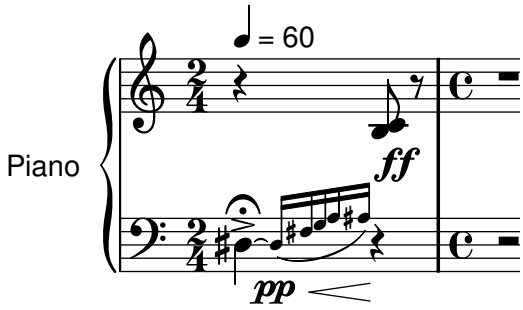
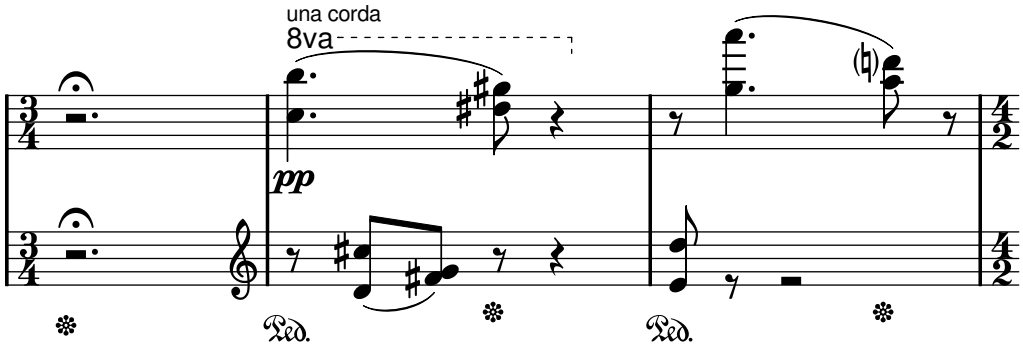
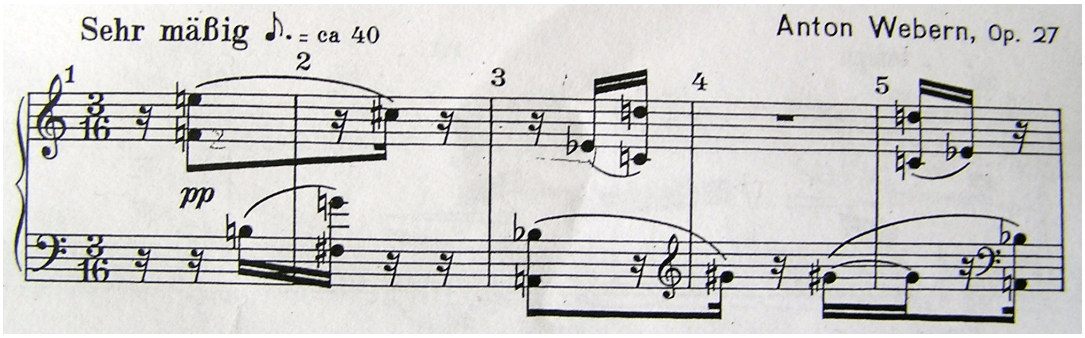


Figure 4.20: The opening bar of *Variations*



(a) *Variations* final movement excerpt



(b) Webern *Variations* Op. 27 opening bars

Figure 4.21: Comparison between the 'theme' in Webern's *Variations* Op. 27, and the 'hidden theme' in *Variations*

4.5.1 Variation I

The basic premise of the electronic part was to attempt to apply electronic processing so as to add additional colour and spatialisation to selected notes. For a monophonic instrument this is often easily achieved using pitch detection, however polyphonic pitch detection is a non-trivial task and the technology is in its infancy – often providing unreliable results (de Cheveigné and Baskind, 2003). Instead it was decided that a simple monophonic pitch detector combined with a 'chord detector' based on an Artificial Neural Network (ANN) could be adapted for the required task. The pitch detector was used to detect whether one of two notes was being played (A4 or C#5), and the chord detector was used to determine whether or not the 'ostinato' diad(B3 and C4) was being played. The logical graph for this setup is shown in fig 4.22, For the individual notes, reverberation, with a long 'reverb time' setting is applied in series with a harmoniser with a 'transposition' setting of one octave. This gives a resonant effect, and emphasises the first harmonic of the piano spectrum. When the diad is detected, a reverberation with a short reverb time is applied, this gives the effect of the chord being 'projected' into a larger space, but also has the effect of making the development section at bar 14 sound much drier and more intimate. Since pitch detection techniques have been discussed in some detail in sections 2.5.1.20 and 4.2.2.1, the focus will now be on the techniques used to detect the B3/C4 diad.

4.5.1.1 Chord Detection

In order to achieve recognition of the B3/C4 diad shown in the opening gesture (figure 4.20) and repeated as an ostinato in the first variation, the chord recognition problem was approached as standard audio classification problem. That is, rather than create a system capable of recognising when the *pitches* B3 and C4 were being played simultaneously, the chord was just treated as a *sound*, with the classification system giving a decision as to whether it was being played or not. This was considered to be the simplest approach and most likely to succeed. The technique (if not the implementation) should also generalise well to other works.

It was anticipated that there would be a high likelihood of 'false positives' and other classification problems in such a system since:

- The interval of a minor second occurs frequently at other points in the piece, for example D4/E♭b4 in bar 17, or F#3/G3 in bar 23
- The inverted interval of the major 7th (e.g C4/B4) occurs frequently and has a very similar spectral content to B3/C4

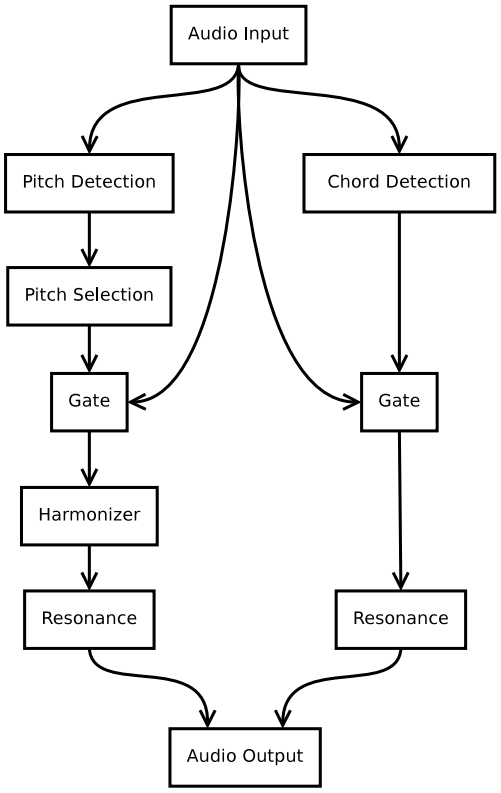


Figure 4.22: Flow diagram for the live electronics in *Variations I*

- The chosen interval is played at a variety of dynamic levels from *pp* to *ff*, and dynamic level affects spectral content (cf. section 1.8.1)

It was therefore decided that a more robust classifier than the k-NN system discussed in section 4.2.2.2 would be needed, and so an ANN based on the multi-layer perceptron (MLP) architecture was chosen since it has been shown to be generally noise robust (Lee and Oh, 1994). The specific implementation used was the Fast Artificial Neural Network (FANN) library, the MLP functionality from which is available as a Pd external through a library binding by Davide Morelli[9].

The MLP was also chosen for its generalisation capabilities since it would need to be able to generalise under the following circumstances:

- Different pianos with different timbral qualities (bright, dull, metallic etc)

[9] `http://www.davidemorelli.it/`

- Different physical environments with a range of acoustics
- Different piano tunings
- Different microphones and microphone placements

In an ideal performance situation the instrument should be tuned to concert pitch, and the microphone and microphone placement should be as specified in the performance notes. However, this cannot always be guaranteed since in practice performance venues vary greatly. It is also not acceptable to retrain the system for every performance, therefore the system needs to generalise well within the limited context of recognising 'one chord within a piece'. According to Wasserman (1989), one of the key properties of Neural Networks is their ability to generalise between different input data[10], and this along with the ready availability of an MLP implementation in Pd, suggested that this would be the most appropriate solution for this particular work. In the following section the basic operation of the MLP will be described, and then I will proceed to discuss the Pd implementation.

4.5.1.2 Neural Networks

A neural network can be defined as a model of reasoning based on the human brain(Negnevitsky, 2001). According to Wasserman (1989):

> Artificial neural networks are biologically inspired; that is, researchers are usually thinking about the organisation of the brain when considering network configurations and algorithms. At this point the correspondence may end.
>
> (Wasserman, 1989)

In their simplest form, Artificial Neural Networks (ANNs) are modelled on basic structures that have been observed in brain cells, more specifically biological neurons as shown in figure 4.23.

Biological neural networks are formed by interconnections between neurons made by the extension of dendrites to a connection point called a synapse. Synapses carry chemical signals from one neuron to another. If the combined 'level' of these signals passes a given threshold, it will cause the neuron to 'fire' sending out a signal to other

[10] Under the correct training circumstances

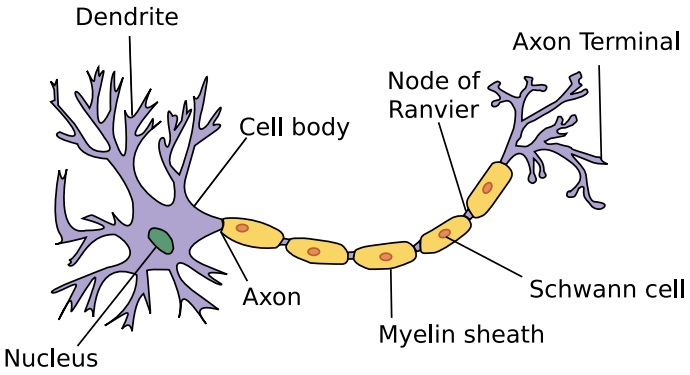


Figure 4.23: The structure of a biological Neuron. [Image used courtesy of user 'dhp1080' on Wikimedia Commons, under the terms of the Creative Commons 'Attribution-Share Alike 2.5 Generic' license].

neurons connected to it. Some cell inputs will stimulate this firing activity and others will inhibit it. The human body is estimated to contain in the order of 10^{11} neurons with an order of 10^{15} interconnections. These neural networks form the basis of human thought and movement.

4.5.1.3 Simple Perceptron

The biological neuron described in section 4.5.1.2, can be modelled computationally by using the weighted sum of a set of inputs post-processed by an 'activation function'. The activation function is a scaling function defining the 'gain' of the neuron. This is shown mathematically below, and diagrammatically in figure 4.24.

$$O = f(\sum_{i=1}^{N} w_i x_i + \theta) \tag{4.2}$$

Where O is the output of the neuron, $f()$ is the activation function, w_i is the ith weight for the ith input x_i and θ is the neuron's 'bias'. The bias is a constant term defining the 'offset' of the neuron - in geometric terms, this could be thought of as defining the spatial offset of the decision boundary.

For very basic systems, the activation function $f()$ can be a simple thresholding function:

$$f(x) = \left\{ \begin{array}{ll} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{array} \right\} \tag{4.3}$$

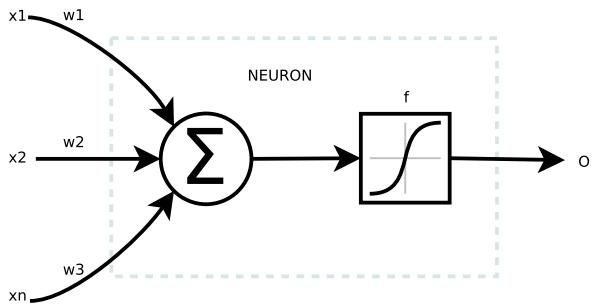


Figure 4.24: An artificial neuron

This provides the possibility to create logical functions such as *AND*, *OR* and *NAND* just by changing the weights applied to the input values. A short 'C' program, which demonstrates this is provided in appendix B. A simple neuron, which can be 'trained' to solve different problems by modifying its input weights and/or the thresholding function is known as a *perceptron*. The perceptron can be represented computationally as shown in the function neuron() (line 86) in the code listing.

Perceptrons can also be used as more general classifiers, and can 'learn' through changes in the weights' values how to 'recognise' a variety of patterns. For example the logical functions *AND*, *OR* and *NAND* are shown graphically in figure 4.25, with the decision boundary shown as a straight line. This decision boundary is known as a linear discriminant function, and can be shown mathematically by rearranging equation 4.2 as:

$$x_1 w_1 + x_2 w_2 + \ldots + x_n w_n + \theta = T \tag{4.4}$$

Here all input patterns where the weighted sum of the inputs $x_1 \ldots x_n$ is greater than T belong to one class and all other inputs belong to another class. If we only have two inputs x_1 and x_2, the value of n is 2, and $x_1 w_1$ and $x_2 w_2$ can be thought of as defining two points representing a straight line in two dimensions. This straight line divides two classes of data, and represents the decision boundary of the perceptron. Examples of decision boundaries for several logical functions are given in figure 4.25. It can be seen that all of the logical functions except *XOR* can be represented as a linearly separable graph with a single decision boundary, and can therefore be solved using a perceptron. The value of n gives the dimensionality of the input data, so where $n = 3$, a plane in

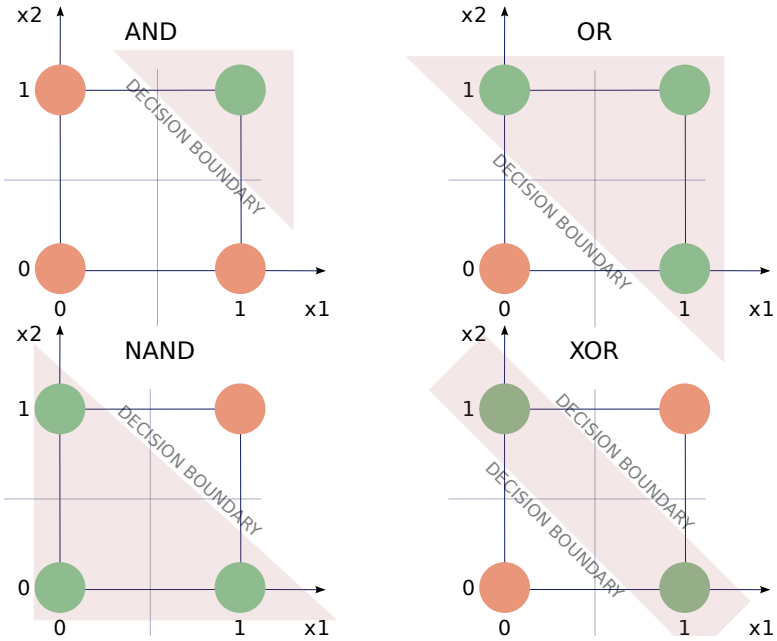


Figure 4.25: A diagram showing the separability of various logical operators on a 2-D graph

3-dimensional space represents the separation between two classes of data, and for n-dimensional data, an n-dimensional hyperplane represents the decision boundary. Geometrically, the bias value θ can be thought of as altering the position, but not the angle of this boundary.

It can therefore be concluded that the perceptron, regardless of the number of inputs (the dimensionality), or the number of weights is incapable of classifying data through which no linear discriminant boundary can be drawn. This can be confirmed using the simple perceptron in listing B, where the geometric space for all of the logical functions is linearly separable, except for the XOR() function which requires three perceptrons combined in order to construct a working classifier [11]. This combination of perceptrons forms a 'neural network' (figure 4.26), which provides the ability to solve more complex problems.

[11]The weight values given in this example are only one possible set of weight values that would give the desired behaviour in the perceptron.

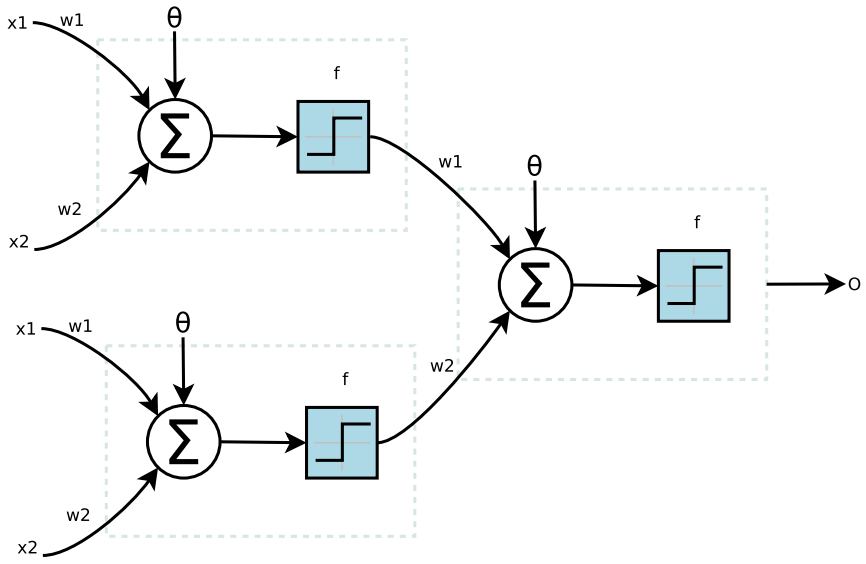


Figure 4.26: Neural network for solving the XOR problem

4.5.1.4 Multilayer Perceptron

A perceptron is the simplest form of feed-forward ANN. The multilayer perceptron is a more complex feed-forward network capable of working with more sophisticated classification problems. A two-layer perceptron can make classification decisions between patterns that are not linearly separable, and a three-layer network can partition the decision space into arbitrary shapes as shown in figure 4.27.

Figure 4.28 shows the basic architecture of a multilayer perceptron. In use the MLP is usually trained with a pre-classified data set in order that the most appropriate set of weights can be established for the given classification task. It is suitable for a broad range of classification problems including general pattern recognition (Bishop, 1995), timbre recognition (Herrera, Amatriain, Batlle, and Serra, 2000) and gesture recognition (Watson, 1993). The MLP differs from the k-NN classifier described in section 4.2.2.2, in that it is generally thought to be more noise robust – that is, it is less sensitive to extraneous variations in the input data. The MLP has good generalisation capabilities(Montana and Davis, 1988), and so when trained and configured correctly it can recognise new data of the same class as given training data even if there are significant differences in several dimensions of the data vector.

In the following section, the implementation of a system based on a multi-layer perceptron for performing isolated chord recognition in the context of *Variations* for

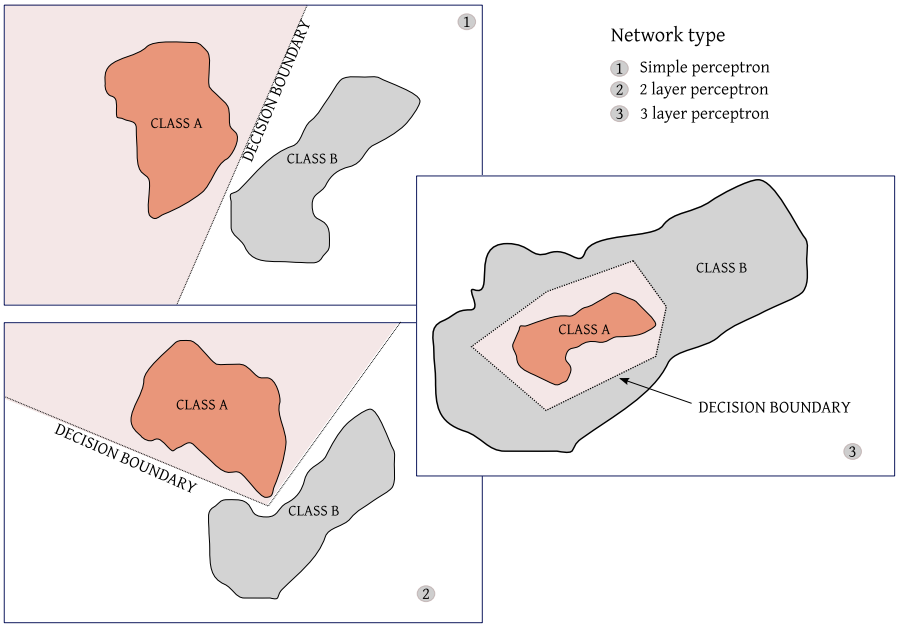


Figure 4.27: Difference in decision boundaries for networks with different numbers of layers

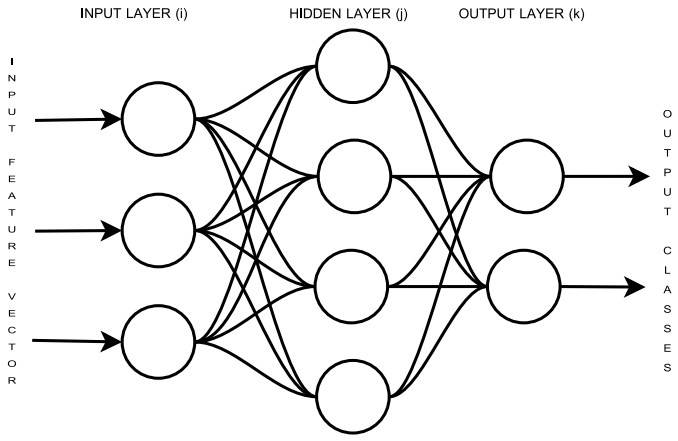


Figure 4.28: The basic architecture of the multilayer perceptron

piano and live electronics will be discussed.

4.5.2 Implementation in Pd

In order to experiment with the use of the MLP in a musical context, the readily available bindings to the Fast Artificial Neural Network (FANN) library for Pure Data were used. The Pd object ann_mlp by Davide Morelli doesn't provide a complete set of bindings to the library, but it does make available the MLP functionality from a Pd context.

In order to train the MLP, a small library of soundfiles was gathered. This consisted of a selection of chords and notes from a studio recording of the first movement of *Variations* made using a Yamaha grand piano. From these soundfiles four soundfiles containing only the 'correct' chord, and nine soundfiles containing 'incorrect' chords were chosen. A small Pd abstraction was then made to iterate over the soundfile set, playing each soundfile twenty times. This was done in order to account for variations in detected onset location, which can in turn create variations in the detected features. This gave a total of 260 discrete audio segments for the training system.

The Pd aubio bindings in the form of the `aubioonset~` external (Brossier, 2006) were used to detect onsets in the audio soundfiles as they were played back. If and when an onset was detected, a feature 'snapshot' was taken using the Pd libXtract bindings in the form of the `xtract~` external. Each feature snapshot (consisting of a space-delimited list) was added to a text file, with each alternate line containing the 'class' of the input ('0' meaning 'incorrect' and '1' meaning 'correct'). This file was then used to train the ANN using the 'train on file' functionality. This wraps the train_on_file () function, which steps through the lines in the file taking each pair of lines as an input/expected-output tuple. A 3-layer MLP was used throughout the experiments, since the classification problem was considered to be in the order of of 'type 3' as shown in figure 4.27. A schematic diagram of the system used in the following experiments is shown in figure 4.29.

4.5.2.1 Experiment 1

In the first experiment the following features were extracted using the libXtract bindings in combination with `fiddle~`:

- Fundamental frequency (`fiddle~`)
- Amplitudes of first 4 harmonics (`fiddle~`)
- Spectral centroid (libXtract)

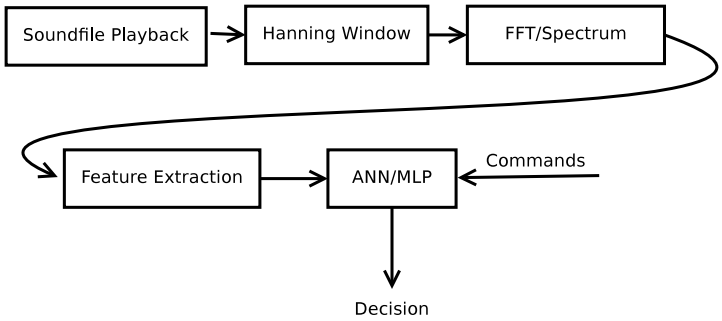


Figure 4.29: Schematic diagram showing the system used for training the multi-layer perceptron in Pd

- Odd to even harmonic ratio (libXtract)
- Spectral inharmonicity (libXtract)

This gave a total feature vector size of 8 elements. It was thought that these features would characterise the audio segments well, whilst providing minimal 'overlap' between features. The MLP network was trained with a 'desired error' of 0.02, and a 'maximum iterations' of 10^5. This limitation on the number of training iterations helps to prevent the network from becoming too 'specialised' and 'overfitting' the data (Tetko et al., 1995).

4.5.2.2 Experiment 2

In the second experiment the size of the feature set was reduced, and the second and third order moments were used along with a reduced feature set as follows:

- Fundamental frequency (`fiddle~`)
- Spectral centroid (libXtract)
- Spectral variance (libXtract)
- Spectral standard deviation (libXtract)
- Spectral inharmonicity (libXtract)

In addition to the revised feature set, the output from the feature extraction functions was normalised and scaled to the range -1.0 to 1.0. This was anticipated to give superior classifier performance.

	Concert pitch	+ 1/4 tone	- 1/4 tone
false	236/249	265/279	228/242
true	73/86	92/104	72/82

Table 4.5: Results for MLP-based chord detection.

4.5.3 Results

In both experiments the trained neural network was tested by presenting it with new data of 'unknown' classification with the expectation that it would correctly identify the class of the audio sample. Again a small library of audio soundfiles was created for this purpose, but this time the audio files were duplicated with transpositions of +1/4 tone and -1/4 tone in order to establish the sensitivity of the classifier to changes in pitch.

The system used in the *second* experiment was found to give the best results as shown in table 4.5. This gives the number of MLP classifications corresponding to a *known* output value as a proportion of the total number of onsets detected. For example '236/249' means that for a given number of input samples known to be 'false' (i.e. not the desired chord), 249 onsets were detected, and out of those 236 were classified as 'false'. Overall this gave a success rate of 87% and a false-positive rate of 5%.

4.5.4 Conclusions

In the *second* experiment, several improvements were made to the method, which led to significantly improved recognition rate, namely:

1. The input vector was normalised and scaled
2. Fewer features were used
3. More 'stable' features were chosen

During this research, the stability of the selected features has been found to be particularly significant in audio classification tasks. It was also found that certain features in the libXtract library display instability under given criteria, or are prone to irregular 'leaps' in value. It has been noted during these experiments that a smaller, more stable and carefully selected feature set is particularly effective in giving improved results.

The results also show that the MLP-based classifier is invariant to pitch shifting within 1/4 tone from the desired chord, with no significant difference between the transposed and non-transposed tests. The false-positive rate was also very low. This is especially important in the context of the given use-case, where an individual chord needs

to be detected, and when that chord is detected, an 'effect' applied. These results suggest that the likelihood of the effect being applied to a *different* chord is approximately 5% under studio conditions. In the context of *Variations*, as the composer of the work, the current author considers this to be an 'acceptable' false-positive level, but hypothetically for other composers and other works, only 100% successful recognition rate with 0% false positives might be acceptable. Given current recognition technology, if this were the *requirement* in a live electronics context, other means for selectively processing sounds should be sought, for example a 'manual' trigger controlled by the instrumentalist or live electronics performer could be used.

4.5.5 Variations II-VII

Variations II to VII make no use of audio feature extraction, and therefore warrant no further discussion.

4.6 Conclusions

In this chapter, I have described four musical works, each of which exemplifies a unique approach to the use of audio feature extraction in live electronic music. *Fission* uses pitch detection, onset detection and k-NN based timbre detection to perform pseudo score following, *Undercurrent* uses raw feature data to enable the performer to use changes in instrumental timbre as a live electronic 'control source', *Sparkling Box* uses raw feature data to trigger pre-recorded samples based on a probability matrix, and *Variations I* uses chord recognition to selectively apply resonance to a specific chord only. Out of the systems used in these works, the only one that provided an entirely reliable and deterministic performance environment was that used in *Undercurrent*. The other systems, although effective, had elements of unreliability or uncertainty due to insufficiently accurate classification and 'detection' techniques. However, in some instances (*Sparkling Box* being a case in point), an element of unpredictability was requested by the composer, and learning the idiosyncrasies and foibles of the system was embraced by the performers who effectively learnt to 'play' the system through their instrumental gestures.

The works composed by the current author in this submission, have been written as part of a reciprocal experimental process in order to establish the viability (or otherwise), of certain feature extraction techniques in live electronic music. In conducting

this research it was found that the systems requiring the highest degree of accuracy and predictability were the least effective, whilst more flexible approaches to feature usage created a more musically-meaningful experience for performers and composers. In the concluding chapter I will expand on the notions of flexibility, fluidity and exploration in a feature extraction context, and suggest possibilities for future work.

CHAPTER 5

Conclusions

Computers are useless. They can only give you answers.

— Pablo Picasso

In this thesis a new library for audio feature extraction has been developed, and audio feature visualisation in the context of live electronic music has been discussed. A number of case studies have been presented, which explore some of the possibilities offered by audio feature extraction in musical composition and performance. In this chapter, I will return to some of the ideas exposed in chapter 1 in order to establish the efficacy of the implementations presented so far. I will also present possibilities for future work and new ideas arising from the research findings of the project.

5.1 Research findings

In chapter 1, the central research question of the thesis was presented as follows:

> "what are the possibilities for the implementation of audio feature extraction techniques in live electronic music?"

The research process has resulted in the following outcomes, which explicitly address this question. It has been shown that it is possible to:

1. develop a robust and extensive feature extraction library that can be used to generate real time control data for live electronics systems (chapter 2);

2. develop visualisation systems in order for composers and performers to interact with extracted feature data in a meaningful way (chapter 3);

3. use a range of extracted features in a musically meaningful manner in the context of concert music for instruments and live electronics (chapter 4).

However, not all features included in the libXtract feature extraction library were found to be musically useful or meaningful. In addition, the manner in which the feature data was used and post-processed was found to be at least as significant to the musical result (if not more so) than the nature of the chosen feature. For example, excellent results were achieved in *Sparkling Box* (section 4.4), where relatively common features such as 'pitch', 'loudness' and 'beat' were used but converted to their second order derivatives via post-processing. By contrast, in *Undercurrent* (section 4.3), a more direct mapping of low-level timbral feature data onto control parameters was used and the musical result was less effective.

Several conclusions can be drawn from the findings of chapter 4:

1. Significant further research is needed in order to establish the musical usefulness of a wide range of low-level features in live electronic music

2. Further research is needed in order to develop higher level features suited to the musical requirements of live electronic music

Other conclusions of this thesis relate to the nature of 'implementation' within the project, and the potential mismatch between musical requirements and feature extraction research. The primary research question of the thesis contains two implicit notions of 'implementation', which warrant further discussion. The first meaning, as applied in chapters 2–3 concerns the *development* of an implementation in software of a number of algorithms for audio feature extraction and visualisation with a view to their use in live electronic music. The second meaning of 'implementation', as applied in chapter 4 concerns the *usage* of this software in the context of actual works and the development of new systems for those works. This relationship is shown in figure 5.1.

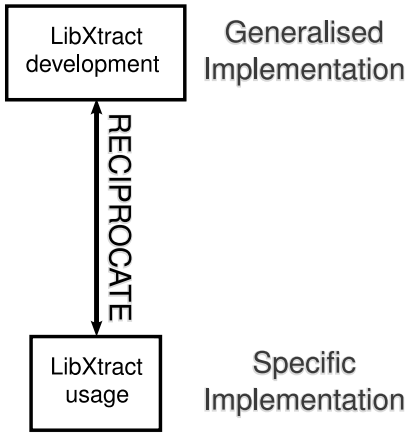


Figure 5.1: General and work-specific implementations of audio feature extraction

In the following sections, research findings resulting from the relationship between these two notions of 'implementation' will be presented, and their significance will be examined in relation to the composition and performance process.

5.1.1 Technology and the composer

The relationship between the two levels of implementation often worked well during the research process, with new features being implemented in the libXtract library in response to requirements for specific works. Likewise, new features added to the library independently from any specific work, prompted exploration of new possibilities and experimentation in work-specific contexts. However, during the course of this research, a methodological dilemma became apparent in that work-specific requirements were instigated from a *compositional* perspective, and musical needs were given priority over technical ones. Whilst this isn't a problem in itself, it does become a problem if the composer also has the dual role of software engineer (cf. section 1.1.3).

To facilitate our discussion of this matter, we revisit the flow diagram initially shown in section 1.1.1 (reproduced in figure 5.2). If one person is performing all of the tasks in figure 5.2, then a number of problematic issues can arise. It seems desirable for a composer to exert a direct influence on the live electronic components of his or her work through experimentation and development of work-specific aspects. However, this research has shown that the development of more generalised solutions (shown in red in figure 5.2) is too far removed from the central composition process to integrate successfully into the working method of a composer-developer. It requires the individual to

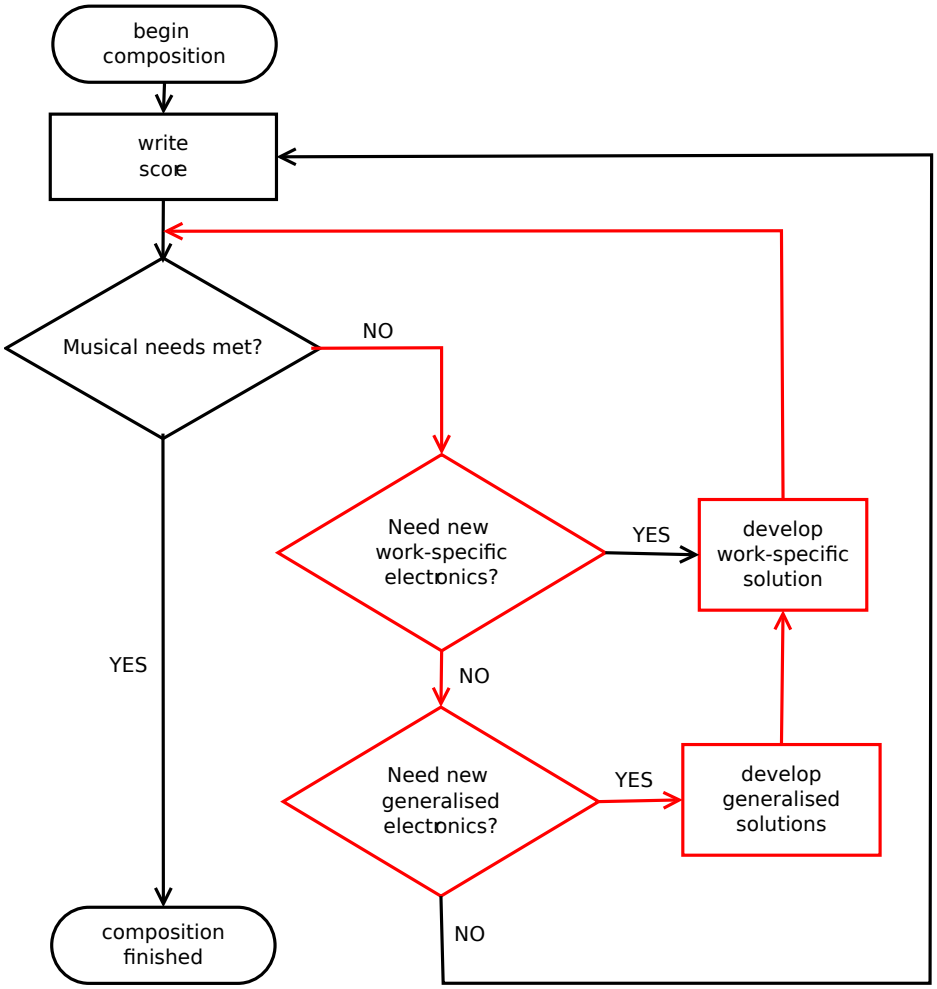


Figure 5.2: The methodological dilemma: developing generalised solutions is one stage removed from the central process

become too distanced from the musical work, and to *think* in a different, more generalised and rational way about the electronics. The development of *generalised* systems is a less overtly creative, more controlled process than either musical composition or the 'creative coding' of ad-hoc solutions for specific works. In short, I have found that rather than generating synergy as suggested in section 1.1.1, having one individual operate as both composer and software engineer in the same compositional process can create tensions between general and work-specific requirements. As shown in the diagram, if one always works with the goal of producing general and sustainable software in mind, this quickly starts to dominate the creative process.

By contrast, the development of *work-specific* solutions by a composer with no concerns for sustainable or generalised development can be a rewarding and effective process (Greenberg, 2007, xxii-xxiii). This 'creative coding' approach, requires the composer to have a degree of expertise and specialist knowledge, but once this is gained, software such as Max/MSP and Pd offer a vast range of possibilities and liberate the composer from the need for a technical assistant. However, this approach is not without its own problems. Having the composers develop their own work-specific solutions can lead to the kind of 'ad-hoc' systems discussed in section 3.3.1. These systems are generally innovative, but because of their bespoke nature, don't tend to directly benefit the broader community and are not sustainable in the long term. This problem is neatly summarised by Pierre Boulez in the conclusion to his now-famous article, 'Technology and the composer':

> Research/Invention, individual/collective, the multiple resources of this double dialectic are capable of engendering infinite possibilities. That invention is marked more particularly by the imprint of an individual goes without saying; we must still prevent this involving us in humdrum, particular solutions which somehow remain the composer's personal property. What is absolutely necessary is that we should move towards global, generalisable solutions. In material as in method, a constant flow must be established between modes of thought and types of action, a continual exchange between giving and receiving. Future experiments, in all probability, will be set up in accordance with this permanent dialogue. Will there be many of us to undertake it?

(Boulez, 1977)

There are several possible solutions to this dilemma, each with their own requirements and drawbacks:

1. The composer could have at his/her disposal a team of technical assistants and/or programmers to conduct the development of their electronics

2. Sufficiently usable and generalised solutions could be developed so that the composer's needs are entirely satisfied by existing software

3. Composers could place greater emphasis on the technological aspects of the work, forming new musical aesthetics and concepts that challenge existing notions of musical composition

Item 1 is to a certain extent the 'solution' that arose from the founding of the Institut de Recherche et Coordination Acoustique/Musique (IRCAM) by Boulez in 1977 (the same year as the 'Technology and the Composer' article). IRCAM, perhaps by nature of its structure helped to establish in the 1980's a 'musical assistant' culture, where composers would be the 'grand architects' of the live electronics for a work, and the musical assistants had responsibility for the 'realisation' of the composers' ideas. This contrasted with earlier approaches by Stockhausen, Berio, Risset and a large number of other composers who started out by constructing their own electronics and software-based systems. In a sense the 'musical assistant' culture helped composers to work more effectively by removing the requirement for expert knowledge in electronics technology, but had the side-effect of distancing the technology from the creative process. It ultimately created a culture of dependence – perhaps even subservience.

Item 2 pertains to a composer-centred approach to software development – a traditional approach to software for live electronics from the perspective of the development process. Here consultation with practitioners, user interface research, 'professional quality' programming and extensive testing could result in a system that is specifically designed to meet the needs of composers and performers working with live electronics[1]. This is the approach taken by projects like Integra (cf. section 1.1), Jade [2]

[1]This approach contrasts starkly with existing systems such as Max/MSP, SuperCollider, and Pure Data, which are essentially programming languages, arguably offering an overwhelming amount of flexibility for the composer/performer.

[2]`http://www.electrotap.com/jade/`

and Jamoma [3], which provide collections of 'higher level' modules as an alternative to Max/MSP's low-level primitives.

Item 3 included perhaps provocatively, refers to new approaches arising in response to the my own work as outlined in this thesis. On a personal level, I have found an increasing level of frustration in trying to reconcile a traditional score-oriented approach to composition, with the new possibilities offered by live electronics processing. Presented here is the anecdotal observation that the relationship between composer, performer, score and electronics is often awkward and uneasy. Achieving a homogeneous and idiomatic discourse between all of these elements often proves to be a difficult task, exacerbated by 'unreliability' in the live electronics software. I firmly believe that the 'solution' to the composer/developer dilemma lies in a dualistic approach where new compositional forms and models need to be developed alongside the development of new, as yet unforeseen software.

5.2 Future work

In section 1.5.6 the idea of 'sound as controller' was introduced. It was proposed that data derived from acoustic sound could form 'an alternative (or perhaps addition) to physical controllers as a source of control data'. On this subject Simon Emmerson writes:

> The need for human-computer interfaces more sensitive to the needs of performers is emerging as the most important new field of research. Two approaches have emerged in the final years of the century: devices which track and measure physical action (sometimes known as 'controllers') and those which analyse the sound produced in performance through processes such as pitch and envelope tracking, real-time spectral analysis and measurement of noise components; the results of this analysis are converted into a suitable format for the control of sound production or processing.
>
> (Emmerson, 2000)

It has been shown in sections 4.2 and 4.3 that when used as Emmerson describes — for controlling processing parameters, sound can make an effective and musically meaningful control source. However, such a direct and 'obvious' mapping between change in

[3] `http://jamoma.org/`

instrumental sound and change in processing, can often become perceptually tiring if overused (as noted in section 4.3.2.3). In the following sections conclusions are made about other uses of the control data derived from audio feature extraction, leading to the presentation of a new concept of the musical score, which draws together the various threads of this thesis.

5.2.1 Score following

One approach to the 'sound as controller' paradigm is to make the performed sound a translator between the logical time of the score, and the absolute time of the concert hall. The sound 'triggers' precise events as determined by the score, at their correct relative locations in time as determined by the composer. If the 'triggering' doesn't work in a predictable way, the electronics is deemed to have failed, since it doesn't facilitate correct 'realisation' of the score.

In an interview with Andrew Ford, Pierre Boulez talks about the inflexibility of works for instrument(s) and tape as a form of imprisonment: 'as a performer, you are a prisoner of the tape'(Ford, 1997). Here, Boulez is referring to the subservient role an instrumental performer plays in controlling the translation of logical time into absolute time in works involving tape – or other 'fixed media'.

McNutt expands on this, referring to the 'prison of perfection' in works involving feature-driven score following:

> [...] while pitch tracking and score following can give the performer more freedom to shape time, they build another kind of 'prison': the prison of perfection. Pitch trackers can be negatively affected by variations in acoustics, microphones and mic placement, instruments, performers, and specific performances. Any of these can produce unexpected data, which will jeopardise the working of the score follower as it tries to correlate input data with expected events.
>
> (McNutt, 2003)

One solution to this problem is to develop increasingly accurate forms of score following and associated technologies (pitch detection, feature extraction, beat tracking), until there is minimal likelihood of score following error (cf. Orio et al. (2003); Pardo and Birmingham (2002)). This seems like the 'brute force' approach, which anthropomorphises the machine, gauging proximity to human 'accuracy' as the ultimate goal.

However, as has been shown in sections 4.4.2 and 4.4.5 it is often the case that the composer and/or performer will want to subvert an apparent inadequacy in the electronics, and use the sense of indeterminacy as an artistic tool. As noted by Moon and Lawter (1998):

> Previous research in the area of open form score following was motivated by problems and limitations inherent in following fixed form scores. Many of these problems arose because of the indeterminate nature of both pitch tracking and human performance.
>
> The indeterminacy of pitch tracking is due to many variables, not only in the algorithm itself, but in the input signal strength and quality. The indeterminacy of human performance, on the other hand, is often criticised, because score following algorithms demand consistency. This is a fundamental contradiction of a so-called "interactive" music, and is the main reason for developing an environment in which indeterminateness of human input becomes a positive, rather than a negative, aspect of score following.
> (Moon and Lawter, 1998)

McNutt comments on this as follows:

> A significant opportunity afforded by open form score following is the use of extended techniques. These sounds (particularly multiphonics) are often unstable and difficult to identify as specific tones; hence they are rarely used in pieces which rely on pitch- based score following. Moon's strategy allows such sounds to be fully integrated into the work. The extension of flute timbres creates a middle ground between the flautist's sound and the computer's, in which the performer can subtly and flexibly adjust her playing to the computer's processing and synthesis. By welcoming the full range of the performer's expressive abilities, Moon does much to heal the schism of electroacoustic performance. The interactive loop is complete: computer and flautist listen to each other and make significant decisions based on one another's sounds.
>
> (McNutt, 2003)

McNutt's observations are pertinent: the more 'perfect' a score follower becomes, in a sense, the less interactive it becomes. A very accurate score follower is not 'equivalent' to a human accompanist, because the score follower is purely *reactive*, it has no 'understanding' of the score it follows. That is not to say that score following is not useful – on the contrary, it is very useful. It should just be taken in the wider view of interactivity that McNutt proposes.

5.2.2 Gesture recognition

Hsu (2006) describes a system for extracting audio features from a live performers and storing a set of 'timbral and musical contours' for 'recent gestures or phrases played by the human improviser'. Each gesture curve is stored in a database and then accessed at some later time as a source of audio-derived control data for mapping onto given performance parameters in the live electronics. This approach is similar to the system used for the author's own work *Undercurrent* (cf. section 4.3), except that in *Undercurrent* the features are 'accumulated' in a many-to-one mapping, providing a resultant 'higher level' feature, and in Hsu (2006), the feature data is stored for time delayed recall. The point is that both of these systems (indeed all of the systems described in chapter 4) treat the audio feature data in an instantaneous manner, and ignore the relationship *between* feature data resulting from successive analysis frames. Feature gestures are treated as series of discrete points rather than as unified symbols. However, by *storing* the feature trajectories as individual gesture profiles, Hsu (2006) invites the further possibility of further 'higher level' analysis of the audio feature data, namely dimension reduction over time.

In this way not only can sonic gestures be *stored*, they can also be *recognised*, allowing for symbolic and/or semantic labels to be associated with given gestures. This opens up musical possibilities to move beyond the conventional notion of score and score follower, and towards a more organic approach to musical time.

5.2.3 Timbral score following

An alternative to traditional score following approaches, which are mainly based on pitch and amplitude information, is to perform timbral score following, which is based on a wide range of extracted audio features. Such an approach is implemented in the *Comparser* software developed by Pieter Suurmond for the Schreck Ensemble[4]. *Comparser* avoids the 'symbolic approach' taken by followers, which use the traditional mu-

[4] `http://kmt.hku.nl/~pieter/SOFT/CMP/doc/cmp.html`

sical score as their basis, and instead uses a recorded soundfile of the musical work as its 'score'. *Comparser* takes an audio file, and constructs an analysis using various feature extraction techniques such as ZCR, Autocorrelation, LPC coefficients and Spectral Peaks to form a feature vector. Vectors for each analysis frame are stored sequentially over the course of the soundfile, and used in a supervised learning process based on Avalanche networks (a form of Neural Network introduced in Grossberg (1982)). The user specifies specific locations in the soundfile, which when recognised in live performance will trigger specific user defined cues. A *Comparser* 'avalanche' visualisation is shown in figure 5.3.

What is interesting about *Comparser* is that it is relatively tolerant to changes in metre and tempo, and although it is very bad at detecting an *exact* location in the 'score' according to the live performance input, it is very good at identifying the *approximate* location. This suggests possibilities for a pseudo-improvisatory approach in the performance – possibly adapted to the Schreck ensemble's specific requirements.

5.2.4 IanniX: a new approach to the musical score

IanniX[5] is a piece of software developed by La Kitchen in Paris, and inspired by the UPIC system devised by the late Iannis Xenakis.

The software proposes a 'graphical representation of a multidimensional and multiformal score, a kind of poly-temporal meta-sequencer'(Coduys and Ferry, 2004). IanniX is not directly related to audio feature extraction or score following, but is relevant in that it presents a new way of thinking about the structure of a musical score and relation to absolute (performed) time. In IanniX the visualisation in the user interface *is* the score, and indeed the approach it takes is entirely idiomatic to and enabled by computer technology – a multidimensional score (in the IanniX sense) would be impossible for a human conductor to follow.

However, although the IanniX score is designed to be 'interpreted' by computer rather than performer, it suggests new possibilities for performer-computer interaction, in particular audio feature-driven processing. Time is represented spatially using a pseudo-3D graphical environment, with vector graphics such as circles and lines representing control data trajectories. An example is shown in figure 5.4.

[5] `http://iannix.la-kitchen.fr/`

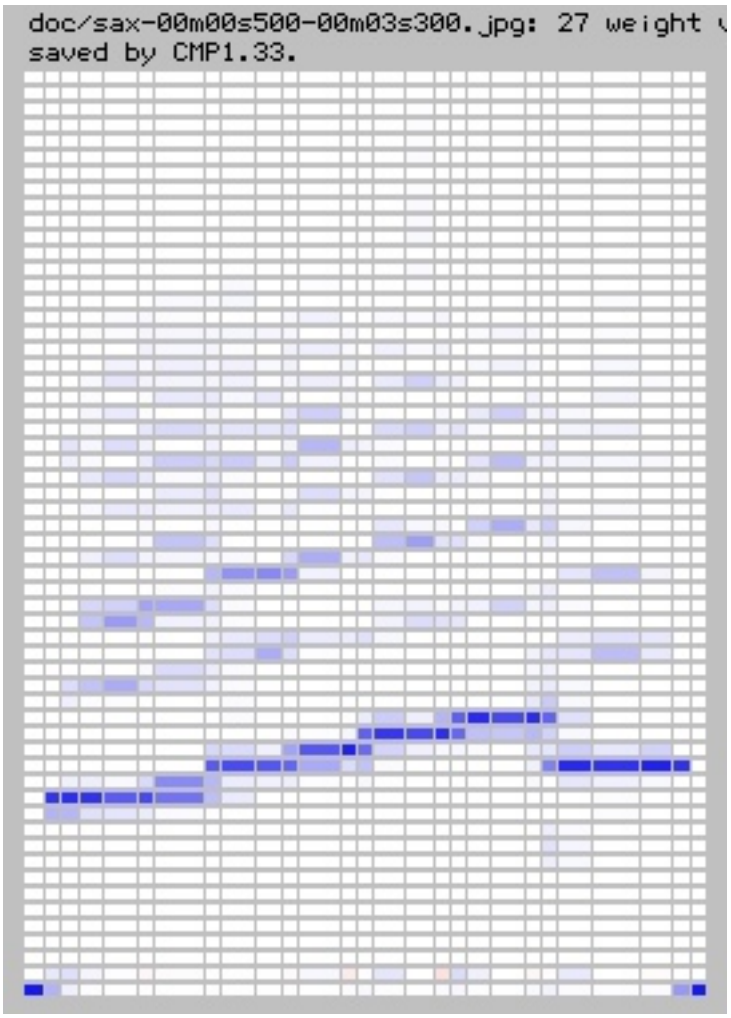


Figure 5.3: The bottom row represents silence, the darker the blue, the more quiet the sound. The second row (from bottom) represents the RMS difference and is the only bipolar scalar in the vector. The third row (from bottom) represents the energy in the lowest frequency band, the top row shows the energy in the highest frequency band. Time progresses from left to right. Red is used for negative values, blue for positive values. [label and image take from `http://kmt.hku.nl/~pieter/SOFT/CMP/doc/cmp.html`]

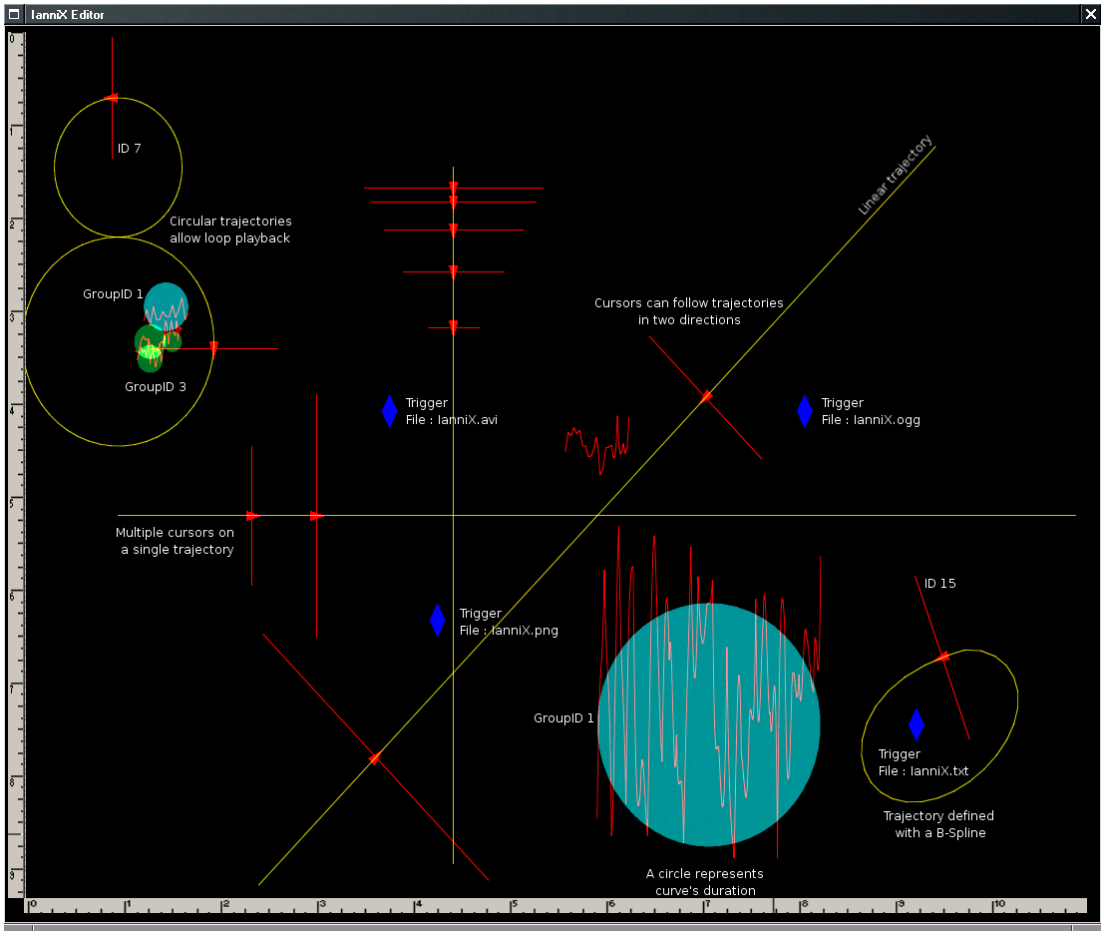


Figure 5.4: IanniX score(Coduys and Ferry, 2004)

5.2.5 Towards the asynchronous score

Despite its novel concept, the representation of time in IanniX is sequential, albeit multi-dimensional with multiple parallel timelines. That is, the musical work is essentially composed of a series of 'events' that must take place in the same order in relation to each other (logical time) whenever the piece is performed.

In the following sections, a new concept – the asynchronous score – is presented. Here, rather than comprising a series of (simple, multidimensional or parallel) time-ordered events, the score is presented as a map by which the performer may *navigate* freely in the live electronics 'system space', by using their instrumental sound as a navigational control. This follows the same motivation as the 'open score' concept that initially became popular in the mid-1950s. Open scores (or 'mobile form'), as exemplified by Stockhausen's *Piano Piece XI* (1956) and Boulez' *Third Piano Sonata* (1955-7), often

comprise a number of notated fragments of music the ordering of which can be chosen by the performer. Later works such as Berio's *Circles* (1960) and Stockhausen's *Zyklus für einen Schlagzeuger* also leave musical elements such as dynamics, pitch or timbre (within a given range) to the performer.

With graphical user interfaces, this idea can be extended, so that the electronic processing becomes the composed element of the work, with the composer 'mapping out' a series of environments for the performer's sound to inhabit. This is a representation of the musical work made possible through graphical user interfaces (GUIs), data visualisation and live electronics processing. It is idiomatic with regard to the technology. That is, such an approach would be impossible with traditional hardware-based approaches, because representing the components of the system visually, with a representation of the performer's location within the system is not possible.

However, the asynchronous score concept doesn't imply a static landscape for the performer where each zone corresponds to a fixed action-reaction system, which essentially responds in the same way every time the performer visits that region. Rather, like a geographical landscape, the asynchronous score can change in response to interaction or interference, and become self modifying – that is geographical elements can modify each others' states. With asynchronous scoring, these events aren't 'sequenced' in 'timelines' but are a consequence of a network of interaction – with the 'triggers' presented in virtual space rather than in time. The space-time connection in asynchronous scoring is implicit rather than explicit (as in IanniX).

5.2.6 Sonar 2D: a work in progress

Sonar 2D is a software application being developed by the current author for the purposes of visualising and interacting with extracted audio feature data. Sonar 2D is a simple but powerful graphical user interface (GUI) application for creating a 2-dimensional 'sonic map' which can be navigated through changes in instrumental sound. Various 'zones' on the map can be associated with given actions, which are encapsulated by various processing modules. In addition the state of the zones themselves can be modified through user or machine interaction allowing for complex possibilities through a simple interface.

Sonar 2D provides an example of a feature-driven asynchronous score. Figure 5.5 shows a mock-up of the user interface. The numerals in the square boxes are not part of

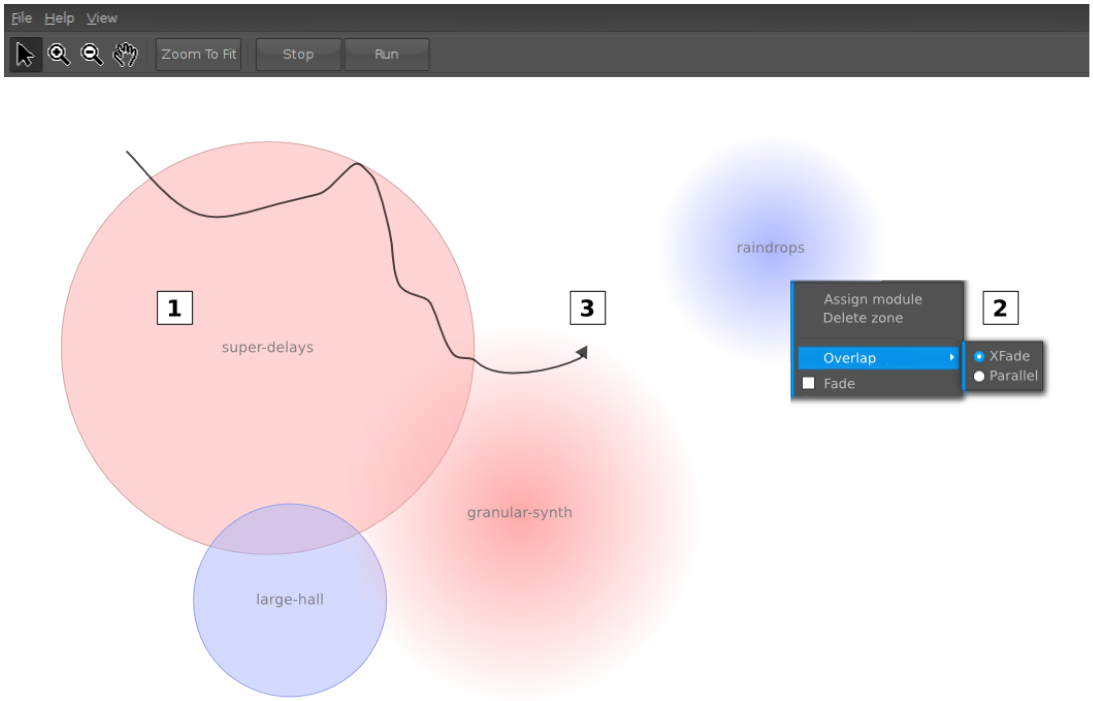


Figure 5.5: Sonar 2D mock-up with numeric workflow annotations

the mock-up, and have been added to illustrate the three aspects of the core workflow as described below.

The basic principle is that the sound of an instrumental performer is represented on screen by a moving line, and that as the line intercepts circles, various transformations get applied to the instrumental sound. The graphical interface could be considered as a 'sonic map', where the performer can navigate through various 'zones' by changing their instrumental sound (timbre, pitch, dynamics). The screen space in the Sonar 2D GUI represents an area of 'sonic possibility' for the performer, and the circles (zones) represent subsets of this. When the leading end of the line enters a given circle, the processing of a selected module is applied to the performer's sound. For example in 5.5, when the triangle enters the 'granular-synth' zone, granular synthesis is applied to the instrumental sound. The processing can be simple or complex, and modules can be standalone or connected up.

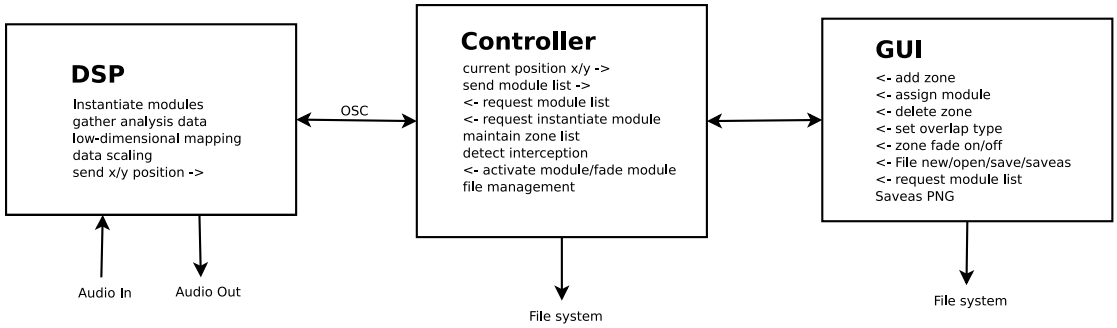


Figure 5.6: Sonar 2D system architecture

5.2.6.1 Sonar 2D workflow

The user begins by playing their instrument and watching the line (3) move around the screen. In this way the performer can get a feeling for which parts of the screen correspond to certain aspects of their instrumental tone.

Once the dimensions of the sonic space have been explored, the performer (or composer) can start placing circles on the canvas (1), where they want a given form of processing to occur. Zones are drawn on the canvas by left-clicking the canvas and dragging the mouse over a given area. Once a circle has been drawn, the user will be prompted for the name of the zone, and this subsequently appears at the centre of each zone. There is no limit to the size or number of zones, and zones may overlap in which case the processing provided by the modules is accumulated. The relationship between overlapping zones is given by each zone's 'Overlap' setting (2). An overlap value of 'XFade' will mean that the modules are connected in parallel, but as the line moves between the overlapping zones, a proportional crossfade will be performed between the modules. A value of 'parallel' means that the modules will be used simultaneously for the period that the intersection of the two zones is occupied.

Settings for each zone can be changed by right-clicking on a given zone (apple-click on OS X). When each zone is initially created it will be drawn blue, however when a processing module has been assigned to the zone using the 'Assign module' option (2), its colour will change to pink, indicating that it is active. Selecting 'Fade' will cause the module to have a radial blur indicating that the feed signal's level will fade in as the line approaches the centre of the zone. For example in figure 5.5, granular synthesis will gradually be applied to the instrumental sound as the line enters the zone labelled 'granular-synth'.

5.2.6.2 Application design

The overall architecture of Sonar 2D is shown in figure 5.6. The separation of the system into 3 primary components (DSP, Controller, GUI) enables the decoupling of the user-interface from the DSP 'engine'. The implication of this is that so long as the same OSC interface is supported, the DSP engine can be changed or updated without making changes to the GUI or controller code. In order to simplify the process of creating this interface, and instantiating or controlling modules from the *Controller*, the Integra library is used. One of the core objectives of the Integra project is to greatly simplify the task of sharing data between various components of a live electronics system. Sonar 2D takes advantage of this simplicity by using libIntegra as an 'abstraction layer' between the DSP and the controller.

5.2.7 Visualisation as score

In the author's work *Starling Motion* for flute and live electronics, the Sonar 2D zone map is used as both the live electronics configuration and a graphical score. That is, the performer uses the visual feedback given by the GUI, along with the aural feedback from the DSP modules to 'navigate' through the piece. The work has no set order or duration, neither is the performer expected to perform in any particular style. Instead, the process of 'learning' the piece is a process of spending time exploring the possibilities offered by the map – the asynchronous score.

The work-in-progress version of Sonar 2D can be found on the author's own website at `http://www.jamiebullock.com`. It is hoped that others will contribute to the development of this project, and that it will act as a starting point for developing new concepts of the score in live electronic music, and eventually incorporate other novel visualisation techniques.

5.3 Postlude

The aspects of musical composition and performance presented in this thesis that have been most successful have been made so through human collaboration. Perhaps the most successful system developed was made for *Sparkling Box* (cf. section 4.4), because the current author was able to focus entirely on devising the electronics in response to the musical ideas of composer Malin Bång. This resulted in a system that was reliable, flexible, musically interesting, and sufficiently general to be used in other works and by other composers. Does this reinforce the requirement for a delineation of roles –

Composer/Musical Assistant in the composition process? The evidence presented in this thesis would suggest this. Perhaps though, the way to move beyond this apparent need for collaboration in the creation of the musical work, is to challenge our existing notion of what constitutes 'composition'. Perhaps our perception of composing a musical work is so inextricably linked to the production of conventionally notated scores, that the potential of technology to 'liberate' the composer and performer from fixed notation is forgotten, or at least not sought out. However, it has hopefully been argued convincingly in this chapter that if composers are willing to move beyond traditional musical models, such possibilities do exist – albeit in an embryonic state. What has been demonstrated though is that through audio feature extraction, instrumental sound can be mapped through multi-layered and multi-modal processing to make audio rather than physical control the primary means of interaction with a live electronics system. In this way interaction through audio feature extraction can become not just a means of controlling processing parameters, or advancing cues through 'recognition', but a means of navigating the live electronics space using the medium with which performers are most familiar: the medium of sound.

Appendices

APPENDIX A

k-NN classifier implementation

```
'''Simple k-NN classifier'''

import math

class Vector(list):
    "Defines an N dimensional vector with label and index"
    index = 0
    def __init__(self, arg):
        super(Vector, self).__init__(arg)
        self.label = None
        self.index = self.inc()

    @classmethod
    def inc(self):
        "Increment the globally unique index"
        self.index += 1
        return self.index

class NNException(Exception):
    "NN exception handler"

class NN:
```

```
23      def __init__(self, dim, k = 5):
24          # List of known vectors (Vector class)
25          self.known = []
26          # Unknown vector (Vector class)
27          self.unknown = None
28          # The ids of the k nearest neighbours
29          self.nn = []
30          # Dictionary calculated distances (id:distance) pairs
31          self.db = {}
32          self.k = k
33          self.dim = dim
34          self.winner = None
35
36      def check(self):
37          if self.dim is None:
38              raise NNException("Dimension not set")
39          if self.unknown is None:
40              raise NNException("Unknown vector not set")
41          if not self.known:
42              raise NNException("Known vectors not set")
43
44      def distances(self):
45          "Calculate distances between unknown feature and known features"
46          try:
47              self.check()
48          except NNException:
49              raise
50
51          for vector in self.known:
52              dist = self.euclidean(vector, self.unknown, self.dim)
53              self.db[vector.index] = dist
54
55      def knn(self):
56          "Get the k nearest neighbours"
57          if not self.k:
58              raise NNException("Distance list empty")
```

```
59
60          # Get the distances and sort them
61          distances = self.db.values()
62          distances.sort()
63
64          # Reverse the db items to get values as keys
65          db_as_list = [list(item) for item in self.db.items()]
66          for n in range(len(db_as_list)):
67              db_as_list[n].reverse()
68          db = dict(db_as_list)
69
70          for k in range(self.k):
71              distance = distances[k]
72              id = db[distance]
73              self.nn.append(id)
74
75      def majority(self):
76          "Get the class of the majority of the k nearest neighbours"
77          k = self.k
78          classes = {}
79
80          for vector in self.known:
81              if not k:
82                  break
83              if vector.index in self.nn:
84                  try:
85                      classes[vector.label] += 1
86                  except KeyError:
87                      classes[vector.label] = 1
88                  k -= 1
89          most = 0
90
91          for item in classes.items():
92              val = item[1]
93              if val > most:
94                  most = val
```

```
95                  self.winner = item[0]
96
97      def nn_as_vectors(self):
98          "Return the k nearest neighbours as a list of vectors"
99          if not self.nn:
100             raise NNException("self.nn not set yet")
101         vectors = []
102
103         for vector in self.known:
104             for index in self.nn:
105                 if vector.index == index:
106                     vectors.append(vector)
107         return vectors
108
109     @staticmethod
110     def euclidean(a, b, dim):
111         "Calculate the euclidean distance between point a and point b"
112
113         dist = 0
114         exponent = 1 / float(dim)
115
116         for n in range(dim):
117             dist += math.pow(a[n] - b[n], 2)
118
119         return math.sqrt(dist)
120
121     def add(self, obj, label = None):
122         "Add an object to the vector list"
123         v = Vector(obj)
124         if label is not None:
125             v.label = label
126             self.known.append(v)
127         else:
128             self.unknown = v
129
130 if __name__ == "__main__":
```

```
131
132     import os, sys
133
134     # Read in files
135     try:
136         dir = sys.argv[1]
137         cwd = False
138     except IndexError:
139         dir = os.getcwd()
140         cwd = True
141
142     dir.rstrip("/")
143
144     files = os.listdir(dir)
145
146     if not cwd:
147         files = [dir + "/" + fname for fname in files]
148
149     files = map(os.path.abspath, files)
150     datafiles = []
151     DIMENSIONS = 3
152
153     for fname in files:
154         if fname.endswith(".data"):
155             datafiles.append(fname)
156
157     knn = NN(DIMENSIONS)
158
159     data = {}
160
161     for filename in datafiles:
162         data[filename] = file(filename, "r")
163
164     errormsg = "Error parsing file: %s, line %d"
165
166     # Add known vectors
```

```
167     for inst in data.values():
168         i = 0
169         for vector in inst.readlines():
170             i += 1
171             v = vector.rstrip()
172             v = v.split(" ")
173             try:
174                 v = map(float, v)
175             except:
176                 raise NNException(errormsg % (inst.name, i))
177             label = os.path.basename(inst.name)
178             label = os.path.splitext(label)[0]
179             knn.add(v, label)
180 
181     # Add unknown
182     v = [620.433, 2.345, 4.234]
183     knn.add(v)
184 
185     knn.distances()
186 
187     knn.knn()
188     knn.majority()
189     print knn.winner
190     print knn.nn_as_vectors()
```

APPENDIX B

Simple perceptron code listing

```
#include <stdio.h>

/* Code for basic neuron */

#define C 1
int neuron(int x, int y, float w0, float w1, float w2, float T);
int AND(int x, int y);
int OR(int x, int y);
int NAND(int x, int y);
int XOR(int x, int y);

main()
{
  printf("\n AND function: \n");
  int x,y,z;
     x = 0;
     y = 1;
     printf("x = %d, y = %d, z = %d\n", x, y, AND(x, y));

  printf("\n OR function: \n");
     printf("x = %d, y = %d, z = %d\n", x, y, OR(x, y));

```

```
23    printf("\n NAND function: \n");
24       x = 1;
25       y = 1;
26       printf("x = %d, y = %d, z = %d\n", x, y, NAND(x, y));
27
28    printf("\n XOR function: \n");
29       x = 1;
30       y = 1;
31       printf("x = %d, y = %d, z = %d\n", x, y, XOR(x, y));
32
33       printf ("\n");
34
35       return 0;
36
37 }
38
39 int XOR(int x, int y)
40 {
41       return (AND ( (OR(x,y)), (NAND(x,y)) ) );
42 }
43
44 int AND(int x, int y)
45 {
46         float w0,w1,w2,T;
47
48       /* weights for AND function */
49
50       w0 = 0.0;
51       w1 = .75;
52       w2 = .75;
53       T = 1.0;
54
55       return (neuron (x, y, w0, w1, w2, T));
56 }
57
58 int OR(int x, int y)
```

```
59 {
60         float w0,w1,w2,T;
61
62       /* weights for OR function */
63
64       w0 = 0.0;
65       w1 = 1.5;
66       w2 = 1.5;
67       T = 1.0;
68
69       return (neuron (x, y, w0, w1, w2, T));
70 }
71
72 int NAND(int x, int y)
73 {
74         float w0,w1,w2,T;
75
76       /* weights for NAND function */
77
78       w0 = 2.0;
79       w1 = -.75;
80       w2 = -.75;
81       T = 1.0;
82
83       return (neuron (x, y, w0, w1, w2, T));
84 }
85
86 int neuron(int x, int y, float w0, float w1, float w2, float T)
87 {
88       if ((w0 * C + w1 * x + w2 * y) >= T)
89         return 1;
90       else
91         return 0;
92 }
```

APPENDIX C

LibXtract main header and public API

```
1  #define XTRACT_FEATURES 60
2
3  /** \brief Enumeration of features, elements are used as indixes to
       an array of pointers to feature extracton functions */
4  enum xtract_features_ {
5      XTRACT_MEAN,
6      XTRACT_VARIANCE,
7      XTRACT_STANDARD_DEVIATION,
8      XTRACT_AVERAGE_DEVIATION,
9      XTRACT_SKEWNESS,
10     XTRACT_KURTOSIS,
11     XTRACT_SPECTRAL_MEAN,
12     XTRACT_SPECTRAL_VARIANCE,
13     XTRACT_SPECTRAL_STANDARD_DEVIATION,
14     XTRACT_SPECTRAL_AVERAGE_DEVIATION,
15     XTRACT_SPECTRAL_SKEWNESS,
16     XTRACT_SPECTRAL_KURTOSIS,
17     XTRACT_SPECTRAL_CENTROID,
18     XTRACT_IRREGULARITY_K,
19     XTRACT_IRREGULARITY_J,
20     XTRACT_TRISTIMULUS_1,
21     XTRACT_TRISTIMULUS_2,
```

```
    XTRACT_TRISTIMULUS_3,
    XTRACT_SMOOTHNESS,
    XTRACT_SPREAD,
    XTRACT_ZCR,
    XTRACT_ROLLOFF,
    XTRACT_LOUDNESS,
    XTRACT_FLATNESS,
    XTRACT_FLATNESS_DB,
    XTRACT_TONALITY,
    XTRACT_CREST,
    XTRACT_NOISINESS,
    XTRACT_RMS_AMPLITUDE,
    XTRACT_SPECTRAL_INHARMONICITY,
    XTRACT_POWER,
    XTRACT_ODD_EVEN_RATIO,
    XTRACT_SHARPNESS,
    XTRACT_SPECTRAL_SLOPE,
    XTRACT_LOWEST_VALUE,
    XTRACT_HIGHEST_VALUE,
    XTRACT_SUM,
    XTRACT_NONZERO_COUNT,
    XTRACT_HPS,
    XTRACT_F0,
    XTRACT_FAILSAFE_F0,
    XTRACT_LNORM,
    XTRACT_FLUX,
    XTRACT_ATTACK_TIME,
    XTRACT_DECAY_TIME,
    XTRACT_DIFFERENCE_VECTOR,
    XTRACT_AUTOCORRELATION,
    XTRACT_AMDF,
    XTRACT_ASDF,
    XTRACT_BARK_COEFFICIENTS,
    XTRACT_PEAK_SPECTRUM,
    XTRACT_SPECTRUM,
    XTRACT_AUTOCORRELATION_FFT,
```

```
58      XTRACT_MFCC,
59      XTRACT_DCT,
60      XTRACT_HARMONIC_SPECTRUM,
61      XTRACT_LPC,
62      XTRACT_LPCC,
63      XTRACT_SUBBANDS,
64      /* Helper functions */
65      XTRACT_WINDOWED
66  };
67
68  /** \brief Enumeration of feature initialisation functions */
69  enum xtract_feature_init_ {
70      XTRACT_INIT_MFCC = 100,
71      XTRACT_INIT_BARK,
72      XTRACT_INIT_WINDOWED
73  };
74
75  /** \brief Enumeration of feature types */
76  enum xtract_feature_types_ {
77      XTRACT_SCALAR,
78      XTRACT_VECTOR,
79      XTRACT_DELTA
80  };
81
82  /** \brief Enumeration of mfcc types */
83  enum xtract_mfcc_types_ {
84      XTRACT_EQUAL_GAIN,
85      XTRACT_EQUAL_AREA
86  };
87
88  enum xtract_lnorm_filter_types_ {
89      XTRACT_NO_LNORM_FILTER,
90      XTRACT_POSITIVE_SLOPE,
91      XTRACT_NEGATIVE_SLOPE
92  };
93
```

```
94  /** \brief Enumeration of return codes */
95  enum xtract_return_codes_ {
96      XTRACT_SUCCESS,
97      XTRACT_MALLOC_FAILED,
98      XTRACT_BAD_ARGV,
99      XTRACT_BAD_VECTOR_SIZE,
100     XTRACT_DENORMAL_FOUND,
101     XTRACT_NO_RESULT, /* This usually occurs when the correct
            calculation cannot take place because required data is missing
             or would result in a NaN or infinity/-infinity. Under these
            curcumstances 0.f is usually given by *result */
102     XTRACT_FEATURE_NOT_IMPLEMENTED
103 };
104
105 /** \brief Enumeration of spectrum types */
106 enum xtract_spectrum_ {
107     XTRACT_MAGNITUDE_SPECTRUM,
108     XTRACT_LOG_MAGNITUDE_SPECTRUM,
109     XTRACT_POWER_SPECTRUM,
110     XTRACT_LOG_POWER_SPECTRUM
111 };
112
113 /** \brief Subband scales */
114 enum xtract_subband_scales_ {
115     XTRACT_OCTAVE_SUBBANDS,
116     XTRACT_LINEAR_SUBBANDS
117 };
118
119 /** \brief Enumeration of data types*/
120 typedef enum type_ {
121     XTRACT_FLOAT,
122     XTRACT_FLOATARRAY,
123     XTRACT_INT,
124     XTRACT_MEL_FILTER
125 } xtract_type_t;
126
```

```
127 /** \brief Enumeration of units */
128 typedef enum unit_ {
129     /* NONE, ANY */
130     XTRACT_HERTZ = 2,
131     XTRACT_ANY_AMPLITUDE_HERTZ,
132     XTRACT_DBFS,
133     XTRACT_DBFS_HERTZ,
134     XTRACT_PERCENT,
135     XTRACT_BINS,
136     XTRACT_SONE
137 } xtract_unit_t;
138
139 /** \brief Boolean */
140 typedef enum {
141     XTRACT_FALSE,
142     XTRACT_TRUE
143 } xtract_bool_t;
144
145 /** \brief Window types */
146 enum xtract_window_types_ {
147     XTRACT_GAUSS,
148     XTRACT_HAMMING,
149     XTRACT_HANN,
150     XTRACT_BARTLETT,
151     XTRACT_TRIANGULAR,
152     XTRACT_BARTLETT_HANN,
153     XTRACT_BLACKMAN,
154     XTRACT_KAISER,
155     XTRACT_BLACKMAN_HARRIS
156 };
157
158 /** \brief Enumeration of vector format types */
159 typedef enum xtract_vector_ {
160     /* N/2 magnitude/log-magnitude/power/log-power coeffs and N/2
        frequencies */
161     XTRACT_SPECTRAL,
```

```
162     /* N spectral amplitudes */
163     XTRACT_SPECTRAL_MAGNITUDES,
164     /* N/2 magnitude/log-magnitude/power/log-power peak coeffs and N
            /2
165      * frequencies */
166     XTRACT_SPECTRAL_PEAKS,
167     /* N spectral peak amplitudes */
168     XTRACT_SPECTRAL_PEAKS_MAGNITUDES,
169     /* N spectral peak frequencies */
170     XTRACT_SPECTRAL_PEAKS_FREQUENCIES,
171     /* N/2 magnitude/log-magnitude/power/log-power harmonic peak
            coeffs and N/2
172      * frequencies */
173     XTRACT_SPECTRAL_HARMONICS,
174     /* N spectral harmonic amplitudes */
175     XTRACT_SPECTRAL_HARMONICS_MAGNITUDES,
176     /* N spectral harmonic frequencies */
177     XTRACT_SPECTRAL_HARMONICS_FREQUENCIES,
178     XTRACT_AUTOCORRELATION_COEFFS,
179     XTRACT_ARBITRARY_SERIES,
180     XTRACT_AUDIO_SAMPLES,
181     XTRACT_MEL_COEFFS,
182     XTRACT_LPC_COEFFS,
183     XTRACT_LPCC_COEFFS,
184     XTRACT_BARK_COEFFS,
185     XTRACT_SUBFRAMES,
186     XTRACT_NO_DATA
187 } xtract_vector_t;
188
189 /** \brief Data structure containing useful information about
        functions provided by LibXtract. */
190 typedef struct _xtract_function_descriptor {
191
192     int id;
193
194     struct {
```

```
195         char name[XTRACT_MAX_NAME_LENGTH];
196         char p_name[XTRACT_MAX_NAME_LENGTH]; /* pretty name */
197         char desc[XTRACT_MAX_DESC_LENGTH];
198         char p_desc[XTRACT_MAX_DESC_LENGTH]; /* pretty description */
199         char author[XTRACT_MAX_AUTHOR_LENGTH];
200         int year;
201     } algo;
202
203     struct {
204         xtract_vector_t format;
205         xtract_unit_t unit;
206     } data;
207
208     int argc;
209
210     struct {
211         xtract_type_t type; /* type of the array/value pointed to by
                argv */
212         float min[XTRACT_MAXARGS];
213         float max[XTRACT_MAXARGS];
214         float def[XTRACT_MAXARGS]; /* defaults */
215         xtract_unit_t unit[XTRACT_MAXARGS];
216         int donor[XTRACT_MAXARGS]; /* suggested donor functions for
                argv */
217     } argv;
218
219     xtract_bool_t is_scalar;
220     xtract_bool_t is_delta; /* features in xtract_delta.h can be
            scalar or vector */
221
222     /* The result.<> entries in descriptors.c need to be checked */
223     union {
224
225         struct {
226             float min;
227             float max;
```

```
228             xtract_unit_t unit;
229         } scalar;
230 
231         struct {
232             xtract_vector_t format;
233             xtract_unit_t unit;
234         } vector;
235 
236     } result;
237 
238 } xtract_function_descriptor_t;
239 
240 /**
241  *
242  * \brief An array of pointers to functions that perform the
        extraction
243  *
244  * \param *data: a pointer to the start of the input data (usually
        the first element in an array)
245  *
246  * \param N: the number of elements to be processed
247  *
248  * \param *argv: an abitrary number of additional arguments, used to
        pass additional parameters to the function being called. All
        arguments are compulsary!
249  *
250  * \param *result: a pointer to the first element in the result
251  *
252  * Each function will iterate over N array elements, the first of
        which is
253  * pointed to by *data. It is up to the calling function to ensure
        that the array is in the format expected by the function being
        called.
254  *
255  * For scalar and delta features, *result will point to a single
        value.
```

```
256  *
257  * For vector features it will point to the first element in an array
         .
258  *
259  * Memory for this array must be allocated and freed by the calling
260  * function.
261  *
262  * All functions return an integer error code as descibed in the
         enumeration
263  * return_codes_
264  *
265  * The preprocessor macro: XTRACT must be defined before this can
         be used
266  *
267  * example:<br>
268  * \verbatim
269 #include <stdio.h>
270 #define XTRACT
271 #include "libxtract.h"
272
273 main () {
274 float values[] = {1.0, 2.0, 3.0, 4.0, 5.0};
275 int N = 5;
276 float mean;
277
278 xtract[MEAN]((void *)values, N, NULL, &mean);
279
280 printf("Mean = %.2f\n", mean);
281 }
282 \endverbatim
283  * The calling function may additionally make some tests against the
         value returned by xtract
284  *
285  */
286 #ifdef XTRACT_H
```

```
287 extern int(*xtract[XTRACT_FEATURES])(const float *data, const int N,
        const void *argv, float *result);
288
289 #endif
290
291 /** \brief A structure to store a set of n_filters Mel filters */
292 typedef struct xtract_mel_filter_ {
293     int n_filters;
294     float **filters;
295 } xtract_mel_filter;
296
297 /** \brief A function to initialise a mel filter bank
298  *
299  * It is up to the caller to pass in a pointer to memory allocated
        for freq_bands arrays of length N. This function populates these
        arrays with magnitude coefficients representing the mel
        filterbank on a linear scale
300  */
301 int xtract_init_mfcc(int N, float nyquist, int style, float freq_min,
        float freq_max, int freq_bands, float **fft_tables);
302
303 /** \brief A function to initialise bark filter bounds
304  *
305  * A pointer to an array of BARK_BANDS ints most be passed in, and is
         populated with BARK_BANDS fft bin numbers representing the
        limits of each band
306  *
307  * \param N: the audio block size
308  * \param sr: The sample audio sample rate
309  * \param *band_limits: a pointer to an array of BARK_BANDS ints
310  */
311 int xtract_init_bark(int N, float sr, int *band_limits);
312
313 /** \brief An initialisation function for functions using FFT
314  *
```

```
315  * This function initialises global data structures used by functions
         requiring FFT functionality. It can be called multiple times
     with different feature names. Calling it more than once with the
     same feature name is not a valid operation and will result in a
     memory leak.
316  *
317  * \param N: the size of the FFT
318  * \param feature_name: the name of the feature the FFT is being used
         for,
319  * e.g. XTRACT_DCT
320  *
321  */
322 int xtract_init_fft(int N, int feature_name);
323
324 /** \brief Free memory used for fft plans
325  *
326  * This function should be used to explicitly free memory allocated
     for ffts by xtract_init_fft(). It is primarily intended for use
     if a new FFT needs to be taken with a different blocksize. If
     only one fft size is required then there is no need to call this
     function since it will be called when the program exits.
327  * */
328 void xtract_free_fft(void);
329
330 /** \brief Make a window of a given type and return a pointer to it
331  *
332  * \param N: the size of the window
333  * \param type: the type of the window as given in the enumeration
     window_types_
334  *
335  */
336 float *xtract_init_window(const int N, const int type);
337
338 /** \brief Free a window as allocated by xtract_make_window()
339  *
```

```
340  * \param *window: a pointer to an array of floats as allocated by
         xtract_make_window()
341  *
342  */
343 void xtract_free_window(float *window);
344
345 /* \brief A function to build an array of function descriptors */
346 xtract_function_descriptor_t *xtract_make_descriptors();
347
348 /* \brief A function to free an array of function descriptors */
349 int xtract_free_descriptors(xtract_function_descriptor_t *fd);
350 /* Free functions */
```

APPENDIX D

The Open Source Definition

The following text was taken from the Open Source Initiative website[1].

D.1 Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

   The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

   The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in

[1] `http://www.opensource.org/docs/osd`

which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

   The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

   The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

   The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

   The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

   The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

   The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

   The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

    No provision of the license may be predicated on any individual technology or style of interface.

APPENDIX E

Audio Recordings

A number of audio recordings have been submitted on a compact disc accompanying this thesis. The purpose of these recordings is to provide an illustration of the live electronics techniques alluded to in the texts (dissertation and scores). The recordings also provide evidence of live performance.

E.1 Track listing

Tracks 1 – 3: **Fission for two violas and live electronics**

Larissa Brown and David Matheu, violas
Jamie Bullock, live electronics
recorded live at the Music Extra festival
Recital Hall, Birmingham Conservatoire, 23 June 2005

Track 4: **Undercurrent for bass trombone and live electronics (excerpt bb. 53 – 103)**

Dr. Simon Hall, bass trombone
Jamie Bullock, live electronics
Recorded in Birmingham Conservatoire live room one in preparation for New Generation Arts festival
Birmingham Conservatoire live room one, 2007

Track 5: **Sparkling Box for contrabass, trombone, bass clarinet, piano, 'cello and live electronics**

Malin Bång, composer
Lamberto Coccioli, electronics (software by Jamie Bullock)
Performed by Athelas Sinfonietta
Rei Munakata, conductor
Recorded live at Nordic Music Days festival
Norrköping Art Museum, 30 August 2007

Tracks 6 – 12: **Variations for piano and live electronics**

Katherine Lam, piano
Jamie Bullock, live electronics
Recorded in the studio following performance at New Generation Arts festival
Birmingham Conservatoire live room one, 2007

References

Amatriain, Xavier. 2004. An object-oriented metamodel for digital signal processing with a focus on audio and music. Ph.D. thesis, Departament de Tecnologica, Universitat Pompeu Fabra, Barcelona, Spain. 20, 21

Atal, Bishnu S., and Suzanne L. Hanauer. 1971. Speech analysis and synthesis by linear prediction of the speech wave. *The Journal of the Acoustical Society of America* 50(2B): 637–655. 83

Aucouturier, Jean-Julien, and Francois Pachet. 2004. Improving timbre similarity: how high's the sky? *Journal of Negative Results in Speech and Audio Science* 1(1). 47

Baumann, Stephan, Tim Pohle, and Vembu Shankar. 2004. Towards a socio-cultural compatibility of mir systems. In *In proceedings of the 5th international conference on music information retrieval*, 460–465. Barcelona, Spain: ISMIR. 42

Bell, Bo, Jim Kleban, Dan Overholt, Lance Putnam, John Thompson, and JoAnn Kuchera-Morin. 2007. The multimodal music stand. In *Proceedings of the 7th international conference on new interfaces for musical expression*, 62–65. New York, USA: ACM. xii, 49, 107, 108

Bello, Juan Pablo, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B. Sandler. 2005. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing* 13(5):1035–1047. 72, 73

Beyls, Peter. 1988. Introducing oscar. In *Proceedings of the international computer music conference*. Cologne: ICMA. 16

Biggs, Michael A. R. 2004. Learning from experience: approaches to the experiential component of practice-based research. *Forskning, Reflektion, Utveckling* 6(21). 9

Bishop, Christopher M. 1995. *Neural networks for pattern recognition*. Oxford, UK: Oxford University Press. 177

Blackwell, Alan F., and T. R. G. Green. 1999. Does metaphor increase visual language usability? In *Proceedings 1999 ieee symposium on visual languages*, 246–253. Tokyo, Japan: IEEE. 99

Bogaards, Niels, Axel Roebel, and Xavier Rodet. 2004. Sound analysis and processing with audiosculpt 2. In *Proceedings of the international computer music conference*. Miami, USA: ICMA. 99

Borchers, Jan, and Max Muhlhauser. 1998. Design patterns for interactive musical systems. *IEEE MultiMedia* 05(3):36–46. 49

Boulez, Pierre. 1977. Technology and the composer. The Times Literary Supplement. 189

Bregman, Albert S. 1990. *Auditory scene analysis*. Cambridge, MA, USA: MIT Press. 94

Brossier, Paul. 2006. Automatic annotation of musical audio for interactive applications. Ph.D. thesis, Queen Mary University of London, UK. 47, 179

Brown, Judith C., and Miller S. Puckette. 1992. An efficient algorithm for the calculation of a constant q transform. *Journal of the Acoustical Society of America* 92(5):2698–2701. 141

———. 1993. A high resolution fundamental frequency determination based on phase changes of the fourier transform. *Journal of the Acoustical Society of America* 94(2): 662–667. 27

Bullock, Jamie. 2007. Libxtract: A lightweight library for audio feature extraction. In *Proceedings of the international computer music conference*. Copenhagen, Denmark: ICMA. iii, 89

Bullock, Jamie, and Lamberto Coccioli. 2006. Modernising musical works involving yamaha dx-based synthesis: a case study. *Organised Sound* 11(3):221–227. 27

Cannam, Chris. 2008. The vamp audio analysis plugin api: A programmer's guide. `http://www.vamp-plugins.org/`. Revision 1.0, covering the Vamp plugin SDK version 1.2. 51

Cannam, Chris, Christian Landone, Mark Sandler, and Juan Pablo Bello. 2006. The sonic visualiser: A visualisaton platform for semantic descriptors from musical signals. In *Proceedings of the 7th international conference on music information retrieval*. Victoria, Canada: ICMA. 50, 111, 115

Casagrande, Norman. 2005. Automatic music classification using boosting algorithms and auditory features. Ph.D. thesis, University of Montreal. 80

Chadabe, Joel. 1997. *Electric sound. the past and promise of electronic music*. New Jersey: Prentice-Hall. 14

de Cheveigné, Alain, and Alexis Baskind. 2003. F0 estimation of one or several voices. In *Proceedings of the 8th european conference on speech communication and technology*, 833–836. Geneva, Switzerland. 171

Chion, Michel, Claudia Gorbman, and Walter Murch. 1994. *Audio-vision*. New York, USA: Columbia University Press. 36

Ciufo, Thomas. 2002. Three meditations for prepared piano and computer. Accessed November 12 2008. 27

Clark, Melville, and Paul Milner. 1964. Dependence of timbre on the tonal loudness produced by musical instruments. *Journal of the Audio Engineering Society* 12(1):28–31. 35, 36

Coduys, Thierry, and Guillaume Ferry. 2004. Iannix: aesthetical/symbolic visualisations for hypermedia composition. In *Proceedings international conference sound and music computing*. The Hague, Netherlands: IEEE. xiv, 194, 196

Collins, Nick. 2005a. A comparison of sound onset detection algorithms with emphasis on psychoacoustically motivated detection functions. In *Proceedings of the audio engineering society convention 118*, 28–31. New York, USA: AES. 30, 31, 140, 141

———. 2005b. Using a pitch detector for onset detection. In *Proceedings of the 6th international conference on music information retrieval*, 100–106. London, UK: ISMIR. 141

———. 2006. Towards autonomous agents for live computer music: Realtime machine listening and interactive music systems. Ph.D. thesis, University of Cambridge. 9, 16, 28, 32

de la Cuadra, Patricio, Aaron Master, and Craig Sapp. 2001. Efficient pitch detection techniques for interactive music. In *Proceedings of the international computer music conference*. Havana, Cuba: ICMA. 32, 137, 139

Degener, Jutta. 1994. Digital speech compression: putting the gsm 6.10 rpe-ltp algorithm to work. *Dr. Dobbs Journal* (December). 83

Dixon, Simon. 2006. Onset detection revisited. In *Proceedings of the international conference on digital audio effects*, 133–137. Montreal, Quebec, Canada: DAfx. 72, 73

Durbin, James. 1960. The fitting of time-series models. *Review of the International Statistical Institute* 28(3). 83

Duxbury, Chris, Mark Sandler, and Mike Davis. 2002. A hybrid approach to musical note onset detection. In *Proceedings of the 5th international conference on digital audio effects*. Hamburg, Germany: DAFx. 72, 73

Eades, Peter. 1984. A heuristic for graph drawing. In *Congressus numerantium*, vol. 42, 149–160. 106

Emmerson, Simon. 2000. *Music, electronic media and culture*, chap. 'Losing touch?': the human performer and electronics, 194–216. Aldershot, Hampshire: Ashgate. 190

Emmerson, Simon, and Dennis Smalley. 2001. *The new grove dictionary of music and musicians*, vol. 8, chap. Electro-acoustic Music, 59–65. 2nd ed. London: MacMillan. 12, 13

Eronen, Antti. 2001. Comparison of features for musical instrument recognition. In *Proceedings of the ieee workshop on the applications of signal processing to audio and acoustics*, 19–22. IEEE. 80, 147

Essid, Slim, Gaël Richard, and David Bertrand. 2004. Efficient musical instrument recognition on solo performance music using basic features. In *Proceedings of the 25th aes international conference*. London, UK: AES. 82

Fastl, Hugo, and Eberhard Zwicker. 2006. *Psychoacoustics: Facts and models*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. 33, 68

Fletcher, Harvey, and W. A. Munson. 1933. Loudness, its definition, measurement and calculation. *The Journal of the Acoustical Society of America* 5(2):82–108. 33

Ford, Andrew. 1997. *Composer to composer : conversations about contemporary music / andrew ford ; photographs by malcolm crowthers & belinda webster*. Hale & Iremonger, Sydney :. BibTex item created from National Library of Australia Catalogue record nla.cat-vn1552975. 191

Fraser, Angela, and Ichiro Fujinaga. 1999. Toward real-time recognition of acoustic musical instruments. In *Proceedings of the international computer music conference*, 175–177. Beijing, China: ICMA. 136

Frigo, Matteo, and Steven G. Johnson. 2005. The design and implementation of fftw3. *Proceedings of the IEEE* 93(2):216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation". 80

Fruchterman, Thomas M. J., and Edward M. Reingold. 1991. Graph drawing by force-directed placement. *Software - Practice and Experience* 21(11):1129–1164. 106

Fry, Ben. 2004. Computational information design. Ph.D. thesis, School of Architecture and Planning, Massachusetts Institute of Technology. xii, 94, 111

———. 2008. *Visualizing data*. Sebastopol: O'Reilly Media, Inc. 110

Fujinaga, Ichiro. 1998. Machine recognition of timbre using steady-state tone of acoustic musical instruments. In *Proceedings of the international computer music conference*, 207–210. Ann Arbor, USA: ICMA. 136

Fujinaga, Ichiro, and Karl MacMillan. 2000. Realtime recognition of orchestral instruments. In *Proceedings of the. international computer music conference*, 141–3. Berlin, Germany: ICMA. xiii, 136, 142, 143, 144, 147, 148

Gancarz, Mike. 1995. *The unix philosophy*. Newton, MA, USA: Digital Press. 127

Gerhard, David. 2003. Pitch extraction and fundamental frequency: History and current techniques. University, Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada. 36

Gleich, David, Leonid Zhukov, Matthew Rasmussen, and Kevin Lang. 2005. The world of music: Sdp embedding of high dimensional data. In *Ieee symposium on information visualization*. Minneapolis, USA: IEEE. Interactive Poster. 104

Gouyon, Fabien, François Pachet, and Olivier Delerue. 2000. On the use of zero-crossing rate for an application of classification of percussive sounds. In *Proceedings ot the cost g-6 conference on digital audio effects*, 147–152. Verona, Italy: DAFx. 64

Greenberg, Ira. 2007. *Processing: Creative coding and computational art*. 1st ed. friends of ED. 188

Grey, John M. 1975. An exploration of musical timbre. Tech. Rep., Stanford Univ. Dept. of Music, Palo Alto, California. 23

Grossberg, Stephen. 1982. *Studies of mind and brain*, chap. Learning by Neural Networks, 65–156. Boston, MA: D. Reidel Publishing. 194

Guercio, Mariella, Jerome Barthelemy, and Alain Bonardi. 2007. Authenticity issue in performing arts using live electronics. In *4th sound and music computing conference (smc'07)*, vol. 1, 226–229. Lefkada, Grèce. 4

Haseman, Brad. 2006. A manifesto for performative research. *Media International Australia* (118). 9

Herrera, Perfecto, Xavier Amatriain, Eloi Batlle, and Xavier Serra. 2000. Towards instrument segmentation for music content description: a critical review of instrument classification techniques. In *Proceedings of the international symposium on music information retrieval*. Massachusetts, USA: ISMIR. 177

Herrera, Perfecto, Juan Bello, Gerhard Widmer, Mark Sandler, Òscar Celma, Fabio Vignoli, Elias Pampalk, Pedro Cano, Steffen Pauws, and Xavier Serra. 2005. Simac: Semantic interaction with music audio contents. In *Journal of intelligent information systems (accepted*. 20

Herrera, Perfecto, Xavier Serra, and Geoffroy Peeters. 1999. Audio descriptors and descriptor schemes in the context of mpeg-7. In *Proceedings of the international computer music conference*. Beijing, China: ICMA. 22, 42, 62

Herrera-Boyer, Perfecto, Anssi Klapuri, and Manuel Davy. 2006. Automatic classification of pitched musical instrument sounds. In *Signal processing methods for music transcription*, ed. Anssi Klapuri and Manuel Davy, 163–200. New York: Springer. 63

Hoffman, Matt, and Perry R. Cook. 2007. Real-time feature-based synthesis for live musical performance. In *Nime '07: Proceedings of the 7th international conference on new interfaces for musical expression*, 309–312. New York, NY, USA: ACM. 60

Hsu, William. 2005. Using timbre in a computer-based improvisation system. In *Proceedings of the international computer music conference*. Barcelona, Spain: ICMA. 29, 49, 60

———. 2006. Managing gesture and timbre for analysis and instrument control in an interactive environment. In *Nime '06: Proceedings of the 2006 conference on new interfaces for musical expression*, 376–379. Paris, France, France: IRCAM –- Centre Pompidou. 193

Hu, Ning, Roger B. Dannenberg, and George Tzanetakis. 2003. Polyphonic audio matching and alignment for music retrieval. In *Ieee workshop on applications of signal processing to audio and acoustics*, 185–188. 89

Hutchison, Andrew. 1997. Empty icons in the metaphor trap. In *Proceedings of the australian society for computers in learning in tertiary education*. Perth, Australia. 99

Jehan, Tristan. 2001. Perceptual synthesis engine: An audio-driven timbre generator. Master's thesis, Massachesetts Institute of Technology. 49

———. 2005. Creating music by listening. Ph.D. thesis, M.S. Media Arts and Sciences, Massachusetts Institute of Technology, Cambridge, MA. ix, 28

Jehan, Tristan, Tod Machover, and Mike Fabio. 2002. Sparkler: An audio-driven interactive live computer performance for symphony orchestra. In *Proceedings of the international computer music conference*. Göteborg, Sweden: ICMA. 60

Jensen, Kristoffer. 1999. Timbre models of musical sounds. Ph.D. thesis, Datalogisk Institut. DIKU Report 99/7. 60

Jensen, Kristoffer, and Tue Haste Andersen. 2004. *Computer music modeling and retrieval*, vol. 2771/2004 of *Lecture Notes in Computer Science*, chap. Real-Time Beat Estimation Using Feature Extraction, 155–178. Berlin, Germany: Springer. 72

Jensenius, Alexander Refsum. 2002. How do we recognize a song in one second? the importance of salience and sound in music perception. Master's thesis, University of Oslo, Oslo, Norway. 47

———. 2007. Action — sound. developing methods and tools to study music-related body movement. Ph.D. thesis, Department of Musicology, University of Oslo, Oslo, Norway. 19

Johnston, Andrew, Benjamin Marks, and Linda Candy. 2007. Sound controlled musical instruments based on physical models. In *Proceedings of the 2007 international computer music conference*, 232–239. Copenhagen, Denmark. 125

Johnston, Andrew, Benjamin Marks, and Ernest Edmonds. 2005. 'spheres of influence': an interactive musical work. In *Ie2005: Proceedings of the second australasian conference on interactive entertainment*, 97–103. Sydney, Australia, Australia: Creativity & Cognition Studios Press. 107, 109

Johnston, J. D. 1988. Transform coding of audio signals using perceptual noise criteria. *IEEE Journal on Selected Areas in Communications* 6(2):314–323. 65

Klapuri, Anssi. 1999. Sound onset detection by applying psychoacoustic knowledge. In *Proceedings of the acoustics, speech, and signal processing*, 3089–3092. Washington, DC, USA: IEEE Computer Society. 30

Krimphoff, Jochen, Stephen McAdams, and Suzanne Winsberg. 1994. Caractérisation du timbre des sons complexes. ii : Analyses acoustiques et quantification psychophysique. *Journal de Physique* 4:625–628. 60

Lee, Youngjik, and Sang-Hoon Oh. 1994. Input noise immunity of multilayer perceptrons. *ETRI Journal* 16(1):35–43. 172

Leman, Marc, Valery Vermeulen, Liesbeth De Voogdt, Johannes Taelman, Dirk Moelants, and Micheline Lesaffre. 2003. Correlation of gestural musical audio cues and perceived expressive qualities. In *Gesture workshop*, ed. Antonio Camurri and Gualtiero Volpe, vol. 2915 of *Lecture Notes in Computer Science*, 40–54. Springer. 42, 44

Lengler, Ralph, and Martin Eppler. 1998. Towards a periodic table of visualization methods for management. In *Proceedings of the conference on graphics and visualization in engineering*, 6. Clearwater, Florida, USA: GVE. xi, 95, 96

Lerch, Alexander, Gunnar Eisenberg, and Koen Tanghe. 2005. Feapi: A low level feature extraction plugin api. In *Proceedings of the 8th international conference on digital audio effects*. Madrid, Spain: DAFx. 50

Lesaffre, Micheline, Marc Leman, Liesbet Devoogdt, Bernard De Baets, Hans De Meyer, and Jean-Pierre Martens. 2006. A user-dependent approach to the perception of high-level semantics of music. In *proceedings icmpc*, 1003–1008. Bologna. 42

Lesaffre, Micheline, Marc Leman, Koen Tanghe, Bernard De Baets, Hans De Meyer, and Pierre Martens. 2003. User dependent taxonomy of musical features as a conceptual framework for musical audio-mining technology. In *Proceedings of the stockholm music acoustics conference*, ed. R. Bresin, 635–638. Stockholm, Sweden. 42

Levinson, N. 1947. The wiener rms error criterion in filter design and prediction. *Journal of Mathematics and Physics* 25(4):261–278. 83

Lewis, George. 2000. Too many notes: Computers, complexity and culture in voyager. *Leonardo Music Journal* 10. 29

Lindsay, T., I. Burnett, S. Quackenbush, and M. Jackson. 2003. Fundamentals of audio descriptors. In *Introduction to mpeg-7 multimedia content description interface*, ed. B.S. Manjunath, P. Salembier, and T. Sikora, 283–298. West Sussex: John Wiley and Sons Ltd. 21

Lippe, Cort. 1993. A composition for clarinet and real-time signal processing: Using max on the ircam signal processing workstation. In *Proceedings of the 10th italian colloquium on computer music*, 428–432. Milan. 20, 32

Malloch, S. N., B. S. Sharp, A. M. Campbell, D. M. Campbell, and C. Trevarthen. 1997. Measuring the human voice: analysing pitch, timing, loudness and voice quality in mother/infant communication. In *Proceedings of the institute of acoustics*, vol. 19, 494–500. Institute of Acoustics. 61

Manning, Peter. 2004. *Electronic and computer music*. Oxford, UK: Oxford University Press. 14

Marozeau, Jeremy, Alain de Cheveigné, Stephen McAdams, and Suzanne Winsberg. 2003. The dependency of timbre on fundamental frequency. *Journal of the Acoustical Society of America* 114:2946–2957. 35

Martin, Keith D., Eric D. Scheirer, and Barry L. Vercoe. 1998. Musical content analysis through models of audition. In *Proceedings of the acm multimedia workshop on content-based processing of music*. New York, USA: ACM. 16

Mcadams, Stephen. 1999. Perspectives on the contribution of timbre to musical structure. *Computer Music Journal* 23(3):85–102. 60, 62, 63

McCartney, James. 1996. Supercollider: a new real time synthesis language. In *Proceedings of the 1996 international computer music conference:*. Hong Kong: ICMA. 26

McEnnis, Daniel, Cory McKay, Ichiro Fujinaga, and Philippe Depalle. 2005. Jaudio:a feature extraction library. *Proceedings of the 2005 International Conference on Music Information Retrieval*. 48

McFarlane, W. Matthew. 2001. The development of acousmatics in montréal. *eContact! Journal of the Canadian Electroacoustic Community* 6(2). 36

McKinney, Martin F., and Jeroen Breebaart. 2003. Features for audio and music classification. In *Proceedings of the 4th international conference on music information retrieval*. Baltimore, MD: Johns Hopkins University. 147

McLeod, Philip, and Geoff Wyvill. 2005. A smarter way to find pitch. In *Proceedings of the international computer music conference*, 138–41. Barcelona, Spain: ICMA. 74, 93, 119, 120

McNutt, Elizabeth. 2003. Performing electroacoustic music: a wider view of interactivity. *Organised Sound* 8(3):297–304. 17, 191, 192

Melara, R. D., and L.E. Marks. 1990. Interaction among auditory dimensions: Timbre pitch and loudness. *Perception and Psychophysics* (482):169–178. 35

Métois, Eric. 1991. Musical sound information. Ph.D. thesis, Ecole Nationale Supérieure des Télécommunications de Paris, France. 25, 26

Misra, Ananya, Ge Wang, and Perry Cook. 2005. Sndtools: Real-time audio dsp and 3d visualization. In *Proceedings of the international computer music conference*. Barcelona, Spain. 115

Montana, David J., and Lawrence Davis. 1988. Training feedforward neural networks using genetic algorithms. Tech. Rep., BBN Systems and Technologies Corp., Cambridge, MA. 177

Moon, Barry, and John Lawter. 1998. Score following in open form compositions. In *Proceedings of the international computer music conference*, 21–24. Ann Arbor, USA: ICMA. 192

Moore, Brian C. J., Brian R. Glasberg, and Thomas Baer. 1997. A model for the prediction of thresholds, loudness, and partial loudness. *Journal of the Audio Engineering Society* 45(4):224–240. 64

Moore, F. Richard. 1990. *Elements of computer music*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. 85

Negnevitsky, Michael. 2001. *Artificial intelligence: A guide to intelligent systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. 173

Noll, A. M. 1969. Pitch determination of human speech by the har-monic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate. In *Proceedings of the symposium on computer processing communications*, 779–797. IEEE. 137

Ong, Bee Suan. 2005. Towards automatic music structural analysis identifying characteristic within-song excerpts in popular music. Master's thesis, Music Technology Group, UPF, Barcelona, Spain. 64, 72

Orio, Nicola, Serge Lemouton, and Diemo Schwarz. 2003. Score following: state of the art and new developments. In *Nime '03: Proceedings of the 2003 conference on new interfaces for musical expression*, 36–41. Singapore, Singapore: National University of Singapore. 18, 191

Paradiso, Joseph A. 2003. Dual-use technologies for electronic music controllers: A personal perspective. In *Proceedings of the 2003 conference on new interfaces for musical expression*, ed. François Thibault, 228–234. Montreal, Quebec, Canada: Faculty of Music, McGill University. 19

Pardo, Bryan, and William Birmingham. 2002. Improved score following for acoustic performances. In *Proceedings of the international computer music conference*. Gothenburg, Sweden: ICMA. 191

Park, Tae Hong. 2000. Salient feature extraction of musical instrument signals. Master's thesis, Dartmouth College, Hanover, New Hampshire. 76

———. 2004. Towards automatic musical instrument timbre recognition. Ph.D. thesis, Princeton University, NJ, USA. 35, 64

Peeters, G. 2004. A large set of audio features for sound description (similarity and classification) in the cuidado project. Tech. Rep., IRCAM, Paris, France. 52, 64, 65, 67, 68, 69, 87, 89

Penttinen, Henri. 2006. Loudness and timbre issues in plucked stringed instruments – analysis, synthesis, and design. Ph.D. thesis, Laboratory of Acoustics and Audio Signal Processing, Helsinki University of Technology, Helsinki, Finland. 36

Pollard, H., and E. Jansson. 1982. A tristimulus method for the specification of musical timbre. *Acustica* 51. 61

Pope, Stephen Travis, Frode Holm, and Alexandre Kouznetsov. 2004. Feature extraction and database design for music software. In *Proceedings of the international computer music conference*. Miami, USA: ICMA. 52, 106

Puckette, Miller. 1988. The patcher. In *Proc. of the 14th international computer music conference*, 420–429. Köln. 26

———. 1996. Pure data: another integrated computer music environment. In *in proceedings, international computer music conference*, 37–41. 26

Puckette, Miller, and Cort Lippe. 1994. Getting the acoustic parameters from a live performance. In *Proceedings of the international conference for musical perception and cognition*, 63–65. Liege, Belgium. 19

Puckette, Miller, and Zack Settl. 1993. Nonobvious roles for electronics in performance enhancement. In *Proceedings of the international computer music conference*, 134–137. San Francisco, USA: ICMA. xv

Puckette, Miller S., Ted Apel, and David Zicarelli. 1998. Real-time audio analysis tools for pd and msp. In *Proceedings, international computer music conference*, 109–112. San Francisco, USA: ICMA. 27, 70, 136, 137, 140, 141

Rabiner, Lawrence, and Biing-Hwang Juang. 1993. *Fundamentals of speech recognition*. New Jersey, USA: Prentice Hall PTR. 81

Rencher, A. C. 2002. *Methods of multivariate analysis*. New York: John Wiley & Sons. 56, 59

Riley, Alexander. 2004. A real-time tristimulus synthesizer. Tech. Rep., University of York. 62

Roads, Curtis. 1996. *The computer music tutorial*. Cambridge, MA, USA: MIT Press. 83

Rowe, Robert. 1992. *Interactive music systems: machine listening and composing*. Cambridge, MA, USA: MIT Press. ix, 16

———. 2004. *Machine musicianship*. Cambridge, MA, USA: MIT Press. 16, 29

Salembier, Phillipe, and Thomas Sikora. 2002. *Introduction to mpeg-7: Multimedia content description interface*. New York, NY, USA: John Wiley & Sons, Inc. 21

Schaeffer, P. 1966. *Traité des objets musicaux*. Paris: Seuil. ix, 36, 37

Schedl, Markus. 2006. The comirva toolkit for visualizing music-related data. Tech. Rep., Department of Computational Perception, Johannes Kepler University, Linz, Austria. xi, 104, 105

Schiesser, Sébastien, and Caroline Traube. 2006. On making and playing an electronically-augmented saxophone. In *Nime '06: Proceedings of the 2006 conference on new interfaces for musical expression*, 308–313. Paris, France, France: IRCAM –- Centre Pompidou. 27

Schubert, Emery, and Joe Wolfe. September/October 2006. Does timbral brightness scale with frequency and spectral centroid? *Acta Acustica united with Acustica* 92: 820–825. 35

Schwarz, Diemo, Arshia Cont, and Norbert Schnell. 2005. From boulez to ballads: Training ircam's score follower. In *Proceedings of international computer music conference (icmc)*. Barcelona, Spain: ICMA. 20

Sedes, Anne, Benoît Courribet, and Jean-Baptiste Thiébaut. 2004. Visualization of sound as a control interface. In *Proceedings of the 7th international conference on digital audio effects*. Naples, Italy: DAFX. xi, 107, 108

Shneiderman, Ben. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In *Ieee visual languages*, 336–343. College Park, Maryland 20742, U.S.A.: IEEE. 91, 128

Slaney, Malcolm, and Richard F. Lyon. 1990. A perceptual pitch detector. In *Proceedings of the international conference on acoustics speech and signal processing*, 357–360. New Mexico, USA: ICASSP. 30, 32

Smalley, Dennis. 1986. *Spectro-morphology and structuring processes*, chap. 4, 61–93. 1st ed. London: MacMillan. ix, 40, 41, 42

Smith, Julius O., and Jonathan S. Abel. 1999. Bark and erb bilinear transforms. *IEEE Transactions on Speech and Audio Processing* 7(6):697 – 708. 74

Smith, Julius O., and Xavier Serra. 1987. Parshl: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. In *Proceedings of the icmc*. Chmpaign-Urbana, USA: ICMA. 76, 77

Stowell, Dan, and Mark Plumbley. 2007. Adaptive whitening for improved real-time audio onset detection. In *Proceedings of the icmc*. Copenhagen, Denmark. 141

Terasawa, Hiroko, Malcolm Slaney, and Jonathan Berger. 2005. The thirteen colors of timbre. In *Proceedings of the ieee workshop on applications of signal processing to audio and acoustics*, 323–326. New York, USA: IEEE. 81

Tetko, Igor V., David J. Livingstone, and Alexander I. Luik. 1995. Neural network studies, 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences* 35(5):826–833. 150, 180

Thoresen, Lasse. 2002. Spectromorphologic analysis of sound objects. an adaption of pierre schaeffer's typomorphology. Paper presented at GRM, Paris. ix, 38, 39

Trueman, D. 1999. Reinventing the violin. Ph.D. thesis, Princeton University, New Jersey. 27

Tufte, Edward R. 1986. *The visual display of quantitative information*. Cheshire, CT: Graphics Press. 93

Tzanetakis, George. 2002. Manipulation, analysis and retrieval systems for audio signals. Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, USA. Adviser: Perry Cook. xii, 48, 122, 123, 124

Tzanetakis, George, and Perry Cook. 2000. Marsyas: A framework for audio analysis. *Organised Sound* 4(3):169 – 175. 48

———. 2001. Marsyas3d: A prototype audio browser-editor using a large scale immersive visual and audio display. In *Proceedings of the international conference on auditory display*. Helsinki, Finland: ICAD. xii, 49, 106, 122, 123

Vercoe, B. 1984. Synthetic performer in the context of live performance. In *Proceedings of the international computer music conference*, 199–200. Paris: ICMA. 27

Wang, Ge, and Perry R. Cook. 2004. The audicle: a context-sensitive, on-the-fly audio programming environ/mentality. In *Proceedings of the international computer music conference*. Miami, USA: ICMA. 115

Ware, Colin. 2004. *Information visualization: Perception for design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 92

Wasserman, Philip D. 1989. *Neural computing: theory and practice*. New York, NY, USA: Van Nostrand Reinhold Co. 173

Watson, Richard. 1993. A survey of gesture recognition techniques. Computer Science Technical Report, Trinity College Dublin, Dublin, Ireland. 177

Weng, Chih-Wen, Cheng-Yuan Lin, and Jyh-Shing Roger Jang. 2004. Music instrument identification using mfcc: Erhu as an example. In *Proceedings of the 9th international conference of the asia pacific society for ethnomusicology(apse)*. Phnom Penh, Cambodia. 82

Wessel, D. L. 1979. Timbre space as a musical control structure. *Computer Music Journal* 3(2):45–52. ix, 23, 24, 60

Wilson, D. Randall, and Tony R. Martinez. 2000. Reduction techniques for instance-based learning algorithms. *Machine Learning* 38(3):257–286. 145

Zhang, Cha, and Tsuhan Chen. 2003. *Handbook of video database design and applications*, chap. From Low Level Features to High Level Semantics, 613 – 627. London, UK: CRC Press. 42

Zhou, Ruohua, and Joshua D Reiss. 2007. Music onset detection combining energy-based and pitch-based approaches. In *Proceedings of the third music information retrieval evaluation exchange (mirex)*. Vienna, Austria: MIREX. 141

Zwicker, E. 1961. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *Journal of the Acoustical Society of America* 33:248–249. 74