An Interactive System for Visualizing and Manipulating Spectral Data

Musical Instrument Design through Experimentation

Richard Thomas Eakin

MASTER THESIS UPF / 2009

Master in Sound and Music Computing

Master thesis supervisor:

Xavier Serra

Department of Information and Communication Technologies

Universitat Pompeu Fabra, Barcelona

UNIVERSITAT POMPEU FABRA

An Interactive System for Visualizing and Manipulating Spectral Data

Musical Instrument Design through Experimentation

Richard Thomas Eakin
Universitat Pompeu Fabra
Barcelona, Spain
Master's Thesis in Sound and Music Computing

Abstract

Current techniques for sound analysis and synthesis make powerful tools for designing new music instruments. Yet, the tools for investigating the possibility of creating new sounds from existing ones are rarely interactive by design, commonly geared towards either scientific or commercial applications. As music instruments need to be interactive in order to allow for expression, it is difficult to expand the usability of these powerful tools in new directions.
A unique and exciting method is introduced within this thesis for exploring spectra data created through the well-known and powerful analysis/synthesis methods known as *SMS (Spectral Modeling Synthesis).* The system is designed to be as interactive as can be, exposing many low level parameters within an environment that can be manipulated in real-time. The main reason why this system is unique is that arbitrary time/frame specification is indefinitaly allowed. Modifications are then performed on some or all of the data in any number of ways, while not requiring a large analysis file to create realistic and interesting sounds. In order to make this type of sound navigation intuitive, a real-time graphical interface is designed as a method of visualizing the sound's features. Within this system, one can use a variety of types of sounds as material for a new music instrument, whether the chosen sample is short, long, simple, or complex.
This thesis suggests a new system for researching music instrument design, based on spectral data through real-time, interactive experimentation. A prototype instrument and experimental variations are designed as proof of concept of the exploratory nature of this system. Preliminary modifications are implemented and tested, lending insight into which methods seem valuable within a musical context.

Acknowledgments

To my thesis supervisors Xavier Serra and Günter Geiger, I am most thankful for their availabilty and help before and during my thesis work at the MTG. Without Xavier's expert advice in the field I would have been without direction, unable to navigate the many possibities open for exploration within sound and music computing. Due to Günter's guidance in technical issues that seemed to never have an end, I am now a better engineer and craftsman.

I also must thank Xavier for starting my work at the MTG by directing the work I did with libsms, picking up from his own source code for SMS analysis and synthesis. I am grateful that I was given the opportunity to partake in the design of such useful tools.

To everyone else at MTG who listened to my ideas and shared theirs with me, thank you for the opportunity to take part in such a lively research group.

As I never had an opportunity to thank my old mentors from UCSD, well, this place seems appropriate! Thank you Shlomo Dubnov for introducing me to the world of audio analysis, Tom Erbe for training me in almost every thing I know related to computer music, and Miller Puckette for training me in every thing else. You were a great inspiration to me while I was an undergraduate who was still deciding whether or not I could survive in this rewarding but difficult field of research.

To my family, especially my uncle James Eakin, thank you for believing in me.

To Rici, thank you for inspiring me with your kindness. I know it was too large of a burden for you, but it is what made me succeed.

Contents

‘

Chapter 1

Introduction

The ear is the musician's most powerful tool. While science offers tools that aid in analysis, processing and transformation of audio data, a trained ear is supreme in detecting sound 'features', or characteristic details. One of the most promising and diverse synthesis techniques today is based on *spectral modeling*, wherein a sound is decomposed into time-varying spectral data such as with the *SMS* [Serra, 1989] system. During re-synthesis of this spectral data, one can notice distinguishable sound features with their ears alone, somehow existing within a set of low-level data. These features have been given various names depending on the field of study (i.e., *fundamental frequency* versus *pitch*, *loudness* versus *dynamics*) and some (namely *timbre*) are nearly impossible to localize by their popular definitions alone. A common understanding is that most musical sounds contain time-varying features, and that these features are both a large contribution to the liveliness of acoustic instruments and shortcoming in many electronic counterparts. Yet, further investigation is necessary to capture and control these time-varying qualities in a synthetic application that strives to become as lively as acoustic instruments. Scientific advancements in acoustics and sound modeling most definitely open new avenues for investigation, but in the end it is necessary for a musical instrument to be created through trial and error, creating and listening, and using the instrument to make music.

This thesis focuses on interactive manipulations of spectral data that would lead to the design of interesting musical instruments and compositional ideas. The importance of the system remains in exploring interesting sounds with human senses, searching for musically potent manipulations. However, ears are not the only useful tools in discovering meaningful sound manipulations. Visualizing the data representation and how it varies in time is also key, allowing one to tie sound visualization to gestures in an interactive environment. Reacting to the creation of sound, taking part in the sound as an interactive instrument, allows one to turn interesting features of otherwise commonly recorded sound into music.

SMS techniques are now widely used in sound synthesis and transformations.

One main goal in these techniques has always been to find new and interesting sounds not offered by natural instruments, yet derived from naturally rich sounds. I approach this goal by exploring arbitrary spectral data within an open-ended, interactive system that can be used to investigate a sound just as well as it can be an instrument for live performance. There exists the possibility of either creating new and interesting sounds, or the same possibility to create discordant, unpleasant sounds. This concept is familiar among anyone who has tried to play any variety of musical instruments such as horns or stringed instruments. At first, because of the complexity of the instrument, it is difficult to discover how to simultaneously manipulate the many parameters with musical control. With time, one can explore musical phrases and ideas in a variety of ways, building control over their instrument, and possibly striving to create their 'personal sound' - expressive playing that makes the sound created by their instrument special. Through practice and experimentation, expression is attainable.

Within the last one hundred years of music, most traditional musical instruments have received new usage, commonly known as the 'extended techniques', wherein performers use unorthodox methods to create new sounds from their instrument. In this movement, the focus is not on traditionally important musical qualities such as melody, harmony, or rhythm, but rather on timbrel changes to the instrument. For example, by putting objects like screws or glass between the strings of a piano key, composers were able to augment this standard instrument in ways that made certain notes 'brighter' or 'rougher'. The process of discovering these new sounds was through experimentation; some things worked while others did not. The exploration described in this paper has followed a similar approach by experimenting within a real-time synthesis environment. A spectral data representation of sound reveals features that lend themselves to musical manipulation, although it is up to the instrument designer to create an environment for using the data to create interesting, new, sounds, thereby augmenting what is already available.

1.1 Evolving Forms of Recorded Sound Manipulation

In music playback systems, gestural control along with the dynamic nature and complexity of (possibly many) acoustic instruments is forgotten. The system is static, able to reproduce a sound only as good, although sometimes with less fidelity, as the original source. Then, in the nature of music evolution, someone tries to see what *else* this recorded audio can be made to do. It is true that natural sounds tend to sound more 'pleasant' to the ear than those created by electronic techniques, and are certainly both easier and safer to produce, but with electronics it is possible to

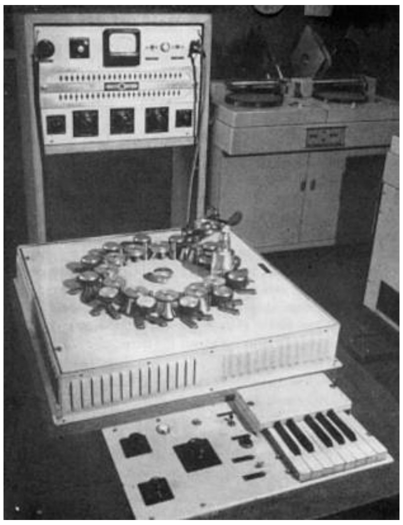


Figure 1.1: Pierre Schaeffer's *chromatic phonogéne*, a novel device for turning a sample into a melodic instrument, playable with a one-octave keyboard.

do new things that acoustic instruments cannot accomplish on their own. Once the system's purpose changes from 'exact playback' to 'modified playback', the recorded sound is once again apart of a music instrument, now with the help of electronics.

In 1951, one of the first electro-acoustic studios was built, *Radiodiffusion* [Palombini, 1993], led by composer Pierre Schaeffer and the *Musique Concréte* movement. The studio built devices for playing back modified, recorded audio, such as the *chromatic phonogéne*, allowing for discrete pitch control of a sample based on an one-octave keyboard (see fig 1.1), the *continuous phonogéne*, allowing continuous pitch control of a sample, and even an early device for time-stretching called the *universal phonogéne*. The results were rough and, of course, experimental, but the instruments allowed for the creation of new music material in a time when music for acoustic instruments had come to be known as 'classical'.

Still it wasn't until the 1990's, more than seventy years after the creation of the phonograph, that 'turntablists' began to perform with recorded audio in live performances. These musicians created new sounds by quickly varying the speed of the record and switching between two records playing simultaneously (see figure 1.2). Through experimentation, a new musical vocabulary [Hansen, 2002] was created that established a turntable as a new musical instrument, using existing audio as material.

As recorded audio merged towards discrete, digital representations, both gains and drawbacks arose. For one, effects that could be achieved with scratching a vinyl record sounded much worse when imitated with digital signal processing. Still, the gains of a virtual library (hard-drive based audio collections)) and the flexibility of interface selection led to novel instruments for improving the usability of DJ techniques Andersen [2005], Alonso and Hansen [2008]. Likewise, *digital audio workstations*, sample-based software for organizing large amounts of audio, saw a similar

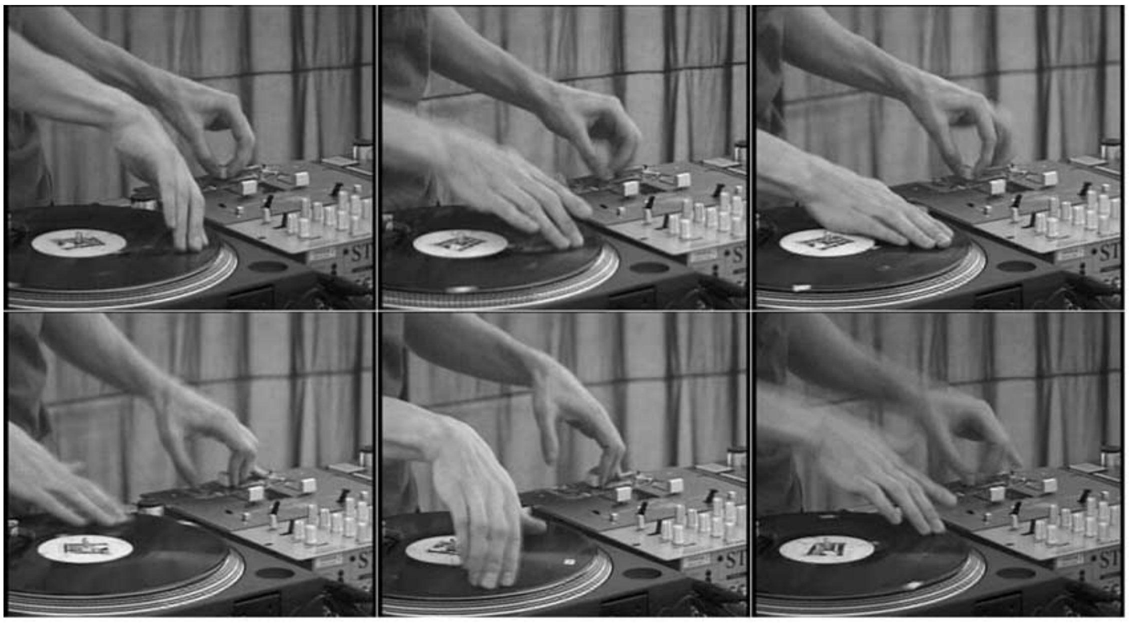


Figure 1.2: *Images of a live turntable performance by DJ 1210 Jazz, taken from the article Hansen [2002]*

phenomenon: the sound organization schemes found in programs like ProTools[1] or Ableton Live[2] became the standard, but the sound representation is still based on time-domain waveforms. The available manipulations within the programs are also similar, though far better established than those that Pierre Schaeffer and company had created more than fifty years past.

It is not until the advancement of spectral audio representation and processing that a plethora of new sound manipulations again surfaced, arguably beginning with the *phase vocoder* [Gordon and Strawn, 1987], a device capable of decoupling frequency and time. Representations such as the sinusoids plus residual model (see Serra [1989] and section 2.1.2) further parameterize a sound recording, producing a number of possible manipulations based on spectral features. Applications for combining, or *morphing*, multiple sounds in the spectral domain were made possible, which allowed for new, rich timbres impossible to create with acoustic instruments [Tellman et al., 1995, Polansky and Erbe, 1996].

Following the progression of time-domain audio manipulation, the current step is how to turn spectral data, and the parameters of manipulation it provides, into a *new* and *expressive* instrument. It is clear that spectral domain manipulations are capable of producing plenty of rich sounds, yet the extent to which spectral data

[1]Digidesign's Protools: `http://www.digidesign.com/`

[2]Ableton Live: `http://www.ableton.com/`

can expand music instrumentation is far from complete.

1.2 Motivation

One can pinpoint two main motivations behind the work presented in this thesis. The first is a desire for creative exploration of sound using spectral techniques, in the hope of improving an understanding of the musical features in audio. Current sound modeling techniques reach high grounds in terms of understanding how humans perceive sound, yet there is still much more to be discovered within the 'timbre wastebasket'. Julius Smith writes

> What is the right "psychospectral model" for sound? We know that the cochlea of the inner ear is a kind of real-time spectrum analyzer. The question becomes how is the "ear's spectrogram" processed and represented at higher levels of audition, and how do we devise efficient algorithms for achieving comparable results? [Smith, 2008]:[3]

There are many approaches to the question posed here, stemming from varying areas of research. I believe that an exciting approach is to create real-time music instruments that expose various levels of abstraction and allow one to experiment with a wide range of manipulations.

The second motivation is to create a uniquely sounding instrument for music performance and composition. To this aim, this thesis is premised on the believe that one need not start not start from scratch, but the exact opposite; start from something that sounds interesting and dissect it into usable substances for music. Again, many have approached this dissection, from various standpoints. I choose to use all scientific means possible, yet still maintain that achievement lays in what sounds most interesting to the ear.

Lastly, as an advocate of improvisation, I find it valuable to allow for quick experimentation with various algorithms and mapping schemes. Often, musician/programmers end up designing intricate computer music instruments based on theories developed in computer science applications. This is not the traditional manner in which music instruments have been designed, and although a tradition is not the rule, it warrants evaluation. Long-lasting music instruments start with an innovation, then adapt of the course of usage in order to allow for more expression. I consider the SMS techniques used in this paper the original innovation, laid out over twenty years ago. It appears that my job is to find how to adapt those techniques to allow for musical expression. I believe that first creating an environment which allows for interactive exploration of the data is key in discovering new possibilities.

[3] http://ccrma.stanford.edu/~jos/sasp/Future_Prospects.html

1.3 Thesis Goals

This thesis attempts to increase the potential for interactive manipulations of spectral data. In order for this to happen, a major goal is to make a closer connection between the processing and perception of sound data. For this reason, manipulations are performed in real-time, with immediate audible results.

Visualizing sound and parameters during performance is becoming necessary in systems with large sound databases. This thesis also contributes towards better data visualization by extracted spectral features, in an attempt to display a representation closer to how the human ear condenses sound information while listening to music.

It should also be mentioned that making an easy-to-play instrument is not a goal. While some believe that computers should do all the hard work and make playing music easy, I find that expression becomes difficult and the instrument becomes boring to play quite fast in such an environment. Instead, I aim to make an instrument possible of creating new expressions from existing audio through gestures that are learned through practice, similar to the learning curve associated in many traditional instruments.

Chapter 2

Background

This chapter provides an overview of the fundamentals in spectral processing, focusing on the main tools used later for creating sound data and visual representations. It is by no means exhaustive, but aims to suffice for understanding the process of data extraction and abstraction using spectra, as well as providing reference to original publications of the mentioned concepts.

The practical usage of spectral analysis differs greatly from spectral synthesis. While one comes from and necessarily depends on the other, in most cases the processes must take place with their own independent time constraints. Furthermore, the focus of this thesis resides in how to use the data to create interesting sounds, ideally different from the original. Hence, a close look is given to the strengths and weaknesses of synthesis options concerning how to best support real-time manipulations. On the other hand, the detailed parameters of frequency analysis are not crucial to understanding the following chapters in this thesis, so only the basic concepts are provided here.

2.1 Concepts of Spectral Analysis and Synthesis

Digital sound is first recorded as a time-varying waveform, but much can be gained in terms of manipulation by transforming the signal to a spectral representation. In this section, I will review the various levels of sound representations discovered through frequency analysis of an audio signal. Each abstraction level brings the sound representation closer to our perceptual understanding of the sound, which is explained thoroughly along with Matlab examples in [Amatriain et al., 2002]. In order to fully understand the complicated nature of spectral sound representations and how to choose parameters that will lead to good results, it is helpful to visualize the analysis process. *SMSTools* by Jordi Bonada is shown in Figure 2.1, which contains a variety of analysis visualizations, each giving insight into different aspects of the overall process[1].

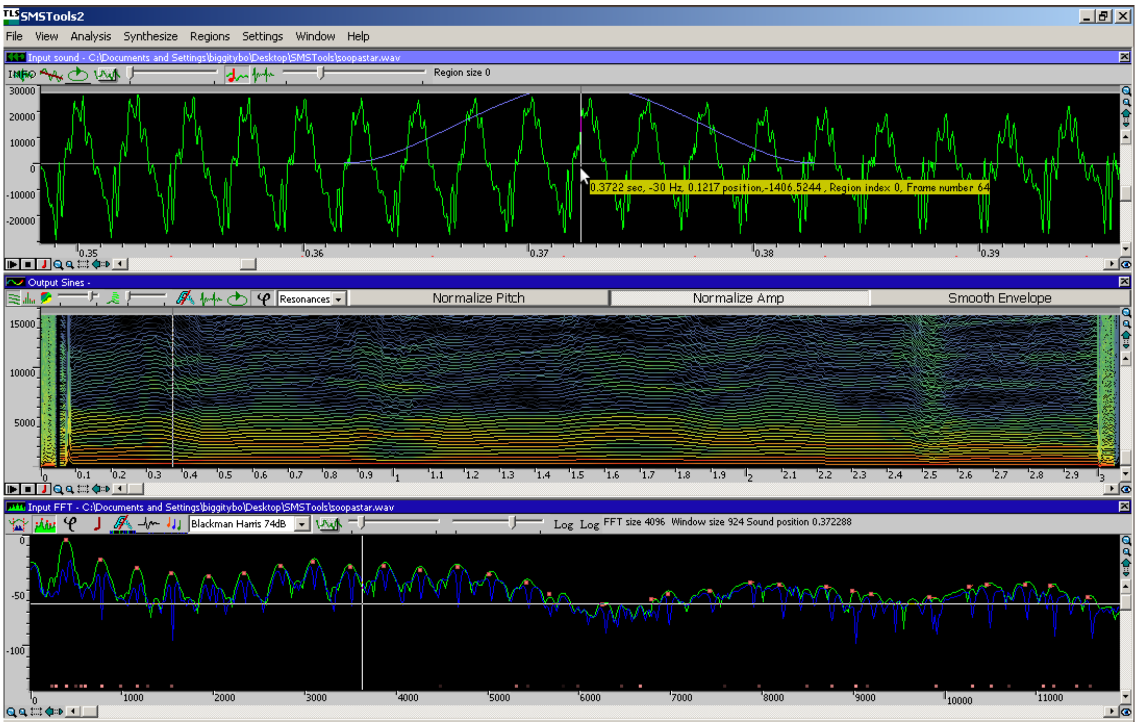

SMSTools2
File View Analysis Synthesize Regions Settings Window Help
Input sound - C:\Documents and Settings\biggitybo\Desktop\SMSTools\soopastar.wav
Region size 0
0.3722 sec, -30 Hz, 0.1217 position,-1406.5244 , Region index 0, Frame number 64
Output Sines -
Resonances
Normalize Pitch
Normalize Amp
Smooth Envelope
Input FFT - C:\Documents and Settings\biggitybo\Desktop\SMSTools\soopastar.wav
Blackman Harris 74dB
Log Log
FFT size 4096 Window size 924 Sound position 0.372288

Figure 2.1: *SMSTools2* is a tool for visualizing the analysis process in sinusoids plus residual modeling. The top displays the current windowed frame of sound samples, while the bottom displays the magnitude spectrum (green), detected sinusoidal peaks (red), and subtracted residual (blue). The middle display shows time-varying sinusoidal tracks up to 15khz. The sample used here is of a singing voice.

2.1.1 Fourier Analysis

The conversion from a time-domain waveform into a time-frequency is achieved with *Fourier Analysis*, now a common too for almost every type of signal processing, as it breaks down a complex waveform into a summation of simple sinusoids with a frequency range. As we are nowadays only dealing with discrete signals, the Discrete Fourier Transform (and its optimized version, the Fast Fourier Transform or FFT) becomes the basis for spectral processing. The standard mathematical definition, derived in [Smith, 2007] is[2]

[1] *SMSTools2* can be downloaded here: `http://mtg.upf.edu/technologies/sms/`

[2] Full definition of the DFT: `http://ccrma-www.stanford.edu/~jos/mdft/DFT_Definition.html`

$$X(\omega_k) = \sum_{n=0}^{N-1} x(t_n) e^{-j\omega_k t_n}, \qquad k = 0, 1, 2, ..., N-1 \tag{2.1}$$

where $x(t_n)$ is the input time-domain signal, N is the number of samples in the signal, and $X(\omega_k)$ is the output frequency domain spectrum, a set of magnitudes and phases at discrete pitch intervals dependent on the sampling rate. When analyzing time-varying sounds, the signal is sectioned into consecutive windows with a specific function that will accentuate spectral peaks, known as Short-Time Fourier Transform (STFT) [Allen, 1977]. The top portion of Figure 2.1 shows this windowing, herein using one of several Blackman-Harris windows, and the resulting spectral magnitudes on the bottom. The method in entirety is described in [Smith et al., 1987].

In audio applications, it is most common to convert the resulting complex sine/cosine pairs into polar form: magnitude and phase pairs. As sound is interpreted on a logarithmic scale, it is then useful to convert the magnitudes into decibels.

2.1.2 Spectral Modeling

Spectral modeling consists of various methods for decomposing short time spectra into data more perceptually meaningful [Serra, 1997, Amatriain et al., 2002]. *Spectral Modeling Synthesis (SMS)*, presented by Xavier Serra in [Serra, 1989], is one of the most successful of these methods because the resulting data representation lends nicely to transformation of a wide variety of musical signals. The first step is to detect peaks in the magnitude spectrum and organize them into sinusoids represented as {frequency, magnitude, phase} sets and then track the sinusoids from frame to frame. To overcome the limitation of frequency resolution in Fourier Analysis, parabolic interpolation is computed from each peak and its neighbors. It was shown in [McAulay and Quatieri, 1986] that reconstructing a time-domain waveform from only these *sinusoidal tracks* could produce a synthetic sound nearly identical in perception to the original sound, yet offered more control over frequency components of the sound.

If a sound is known to be harmonic, which suggests that sinusoidal peaks are all multiples of a fundamental frequency, one can further abstract the sinusoidal tracks into time-varying harmonics. There exist many approaches for determining the fundamental frequency (f0) of a set of harmonics[Beauchamp, 1993, de Cheveigne and Kawahara, 2002]. Once the fundamental frequency is determined, sinusoidal tracking becomes much easier and more accurate if spurious peaks in the spectrum are discouraged during the peak-picking process. The resulting time-varying sinusoidal tracks are visualized in the middle portion of Figure 2.1, a display scheme similar to the popular *sonogram*.

While sinusoidal tracking goes a long way in modeling acoustic instruments, many sounds contain a noise-like component that is not easily reproduced with sinusoids. The *sinusoids plus residual* model [Serra, 1989][3] takes care of this lacking sound component by subtracting the re-synthesized deterministic component and subtracting the resulting waveform from the original, thereby creating a 'residual' waveform. As spectral phases in the residuum tend to be stochastic, provided that sinusoidal analysis and subtraction has been successful, one can generalize the model by disregarding phases and using a random number generator to produce a new stochastic phase spectrum during re-synthesis. This is essentially filtered white-noise, where the filter is derived from the magnitude spectrum of the residual. Figure 2.2 shows a block diagram of this analysis process.

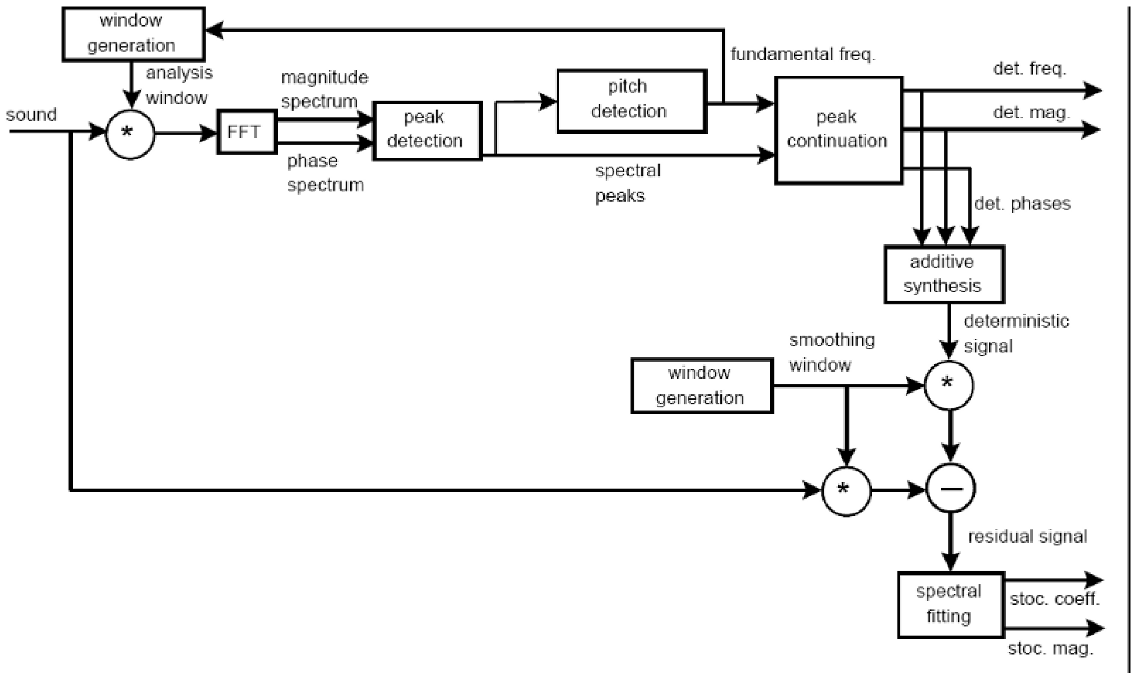


Figure 2.2: Diagram of the analysis process in the sinusoids plus noise residual [Serra, 1997].

Lastly, many methods exist for improving the data produced in SMS analysis, which can greatly improve the fidelity of synthesis. These methods exist because real world sounds can not be modeled perfectly with these techniques, yet in many cases

[3]Also called *sinusoids plus noise*, *deterministic plus residual*, or *deterministic plus stochastic*, with little difference in the underlying model.

analysis flaws can be fixed or greatly improved ‘by hand’. One common improvement for reducing high-pitched artifacts is known as ‘track cleaning’; combining short sinusoidal tracks into longer ones, or removing those which vary too rapidly. Another useful method is to ‘meld’ or ‘fuse’ the sinusoidal and residual components processing the residual with a time-domain comb filter [Amatriain et al., 2002]. This is especially useful when modifying one of the components separately from the other, as the filtering helps to ‘fill in’ gaps in the residual magnitude spectrum. Furthermore, if sinusoidal analysis does not detect all deterministic components, the stochastic residual tends to sound much louder than it should. Many intuitive techniques are used to fix these problems, such as high pass filtering or matching the amplitude of the synthetic sound to the original [Serra, 1997]. These post-processes are mentioned here because they all greatly increase the fidelity of manipulations presented later in this thesis (Chapter 3)

2.1.3 Spectral Envelope

The spectral envelope, or spectral shape, is a curve that describes the amplitude variation over a frequency spectrum. The spectral shape of a sound contributes immensely to its perceived timbre, especially in voice and other formant-based sounds. A stochastic residual is in itself a spectral envelope, while creating an envelope of the sinusoidal data takes a little more work.

There are many ways to compute and represent spectral envelopes. One of the first methods for computing the spectral envelope was through the use of *linear prediction analysis* [Rodet and Ph, 1986], although it has been shown to neglect sinusoidal peaks in the spectrum (see [Schwarz and Rodet, 1999] for a comparison of this and other enveloping techniques). For the purposes of this work, I will only present here the *Discrete Cepstrum Spectral Envelop* (DCSE), introduced in [Galas and Rodet, 1991] and implemented according to [Cappe and Moulines, 1996]. A newer, recursive technique is presented in [Robel and Rodet, 2005], is left for future investigation within the context of this thesis.

DCSE is especially useful when trying to envelope sinusoidal peaks, as it almost guarantees that the envelope will intersect every peak. Figure 2.3 displays one frame of magnitude spectra, extracted peaks, and the deterministic envelope created using the DCSE.

DCSE requires as input a frame of sinusoidal peak frequencies and amplitudes, conveniently acquired through sinusoidal modeling analysis 2.1. The frequencies are scaled from 0 to 0.5, where 0.5 corresponds to half the maximum frequency desired to be covered by the resulting spectral envelope (and is also the *Nyquist freqyency* expressed in radians). Computing the discrete cepstrum can be seen as fitting a curve to the data set by using cosines as the basis functions. This is achieved by computing a least squares solution in the ’log-spectral domain’, expressed in the

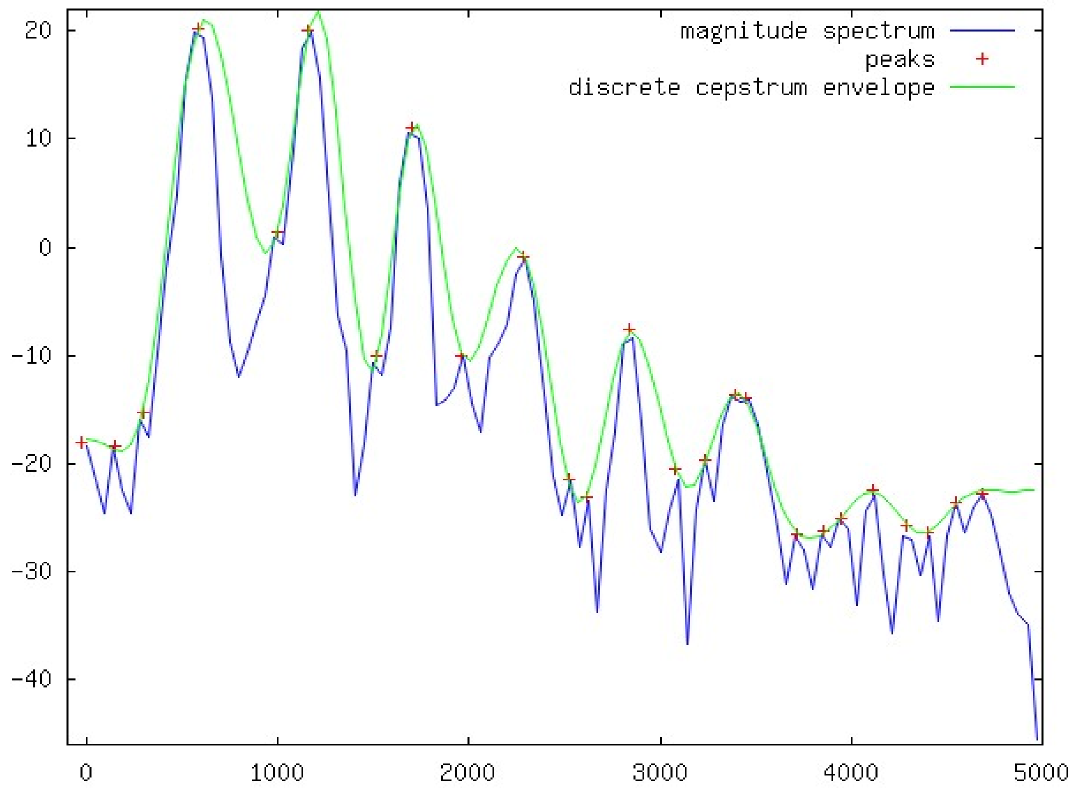


Figure 2.3: *A plot of the magnitude spectrum, detected peaks, and the generalized spectral envelope that can be used to maintain or modify the spectral shape.*

following squared error:

$$\epsilon = \sum_{k=1}^{L} \| \log a_k - log|S(f_k; c)| \|^2 + \lambda R[S(f; c)] \tag{2.2}$$

where a_k and f_k are linear amplitudes and normalized frequencies, respectively, at peak k within the magnitude spectrum S. The second term is the regularizing function

$$R[S(f; c)] = \int_{-1/2}^{1/2} [\log |S(f; c)|]^2 \delta f \tag{2.3}$$

that penalizes rapid variations according to the parameter λ (higher values will create a smoother curve). In the end, the cepstrum coefficients c describe the changes in magnitude over the frequency spectrum. The number of coefficients also effects the smoothness of the curve; less creates a more generalized curve, while more ensures a more accurate envelope. Methods for numerically computing c are provided by Cappe and Moulines [1996].

Lastly, the cepstrum coefficients need to be converted back to frequency in order to create the envelope. This is efficiently computed with the *FFT* and taking the exponential of the result, thereby giving a curve in the spectral domain.

It may seem at first unnecessary to compute the spectral envelope through many additional steps when the magnitude spectrum, from which sinusoidal peaks are derived, is in itself an envelope. Yet, this envelope tends to be more erratic due to fluctuations in noise and transients and therefor does not very well define the shape specifically related to deterministic data of the sound frame. As will be shown in chapter 3, computing the spectral envelope opens up the possibility for many higher-level manipulations while retaining high fidelity sound synthesis.

2.1.4 Synthesis of Spectral Data

Depending on the underlying model, synthesis of spectral data varies in complexity. A nice aspect of the sinusoids plus residual model is that synthesis is performed on a frame by frame basis, only requiring one frame of history to smoothly connect the frames together. Components are linearly interpolated between frames to create the illusion of evolving partial tracks and noise envelopes (Wright and III [2005] implements other, more sophisticated methods of interpolation). Figure 2.4 shows a block diagram that conceptually explains the synthesis of one frame.

A reconstruction of the deterministic data is known as *additive synthesis* [Moorer, 1977], traditionally implemented with an oscillator bank that constructs an array of sinusoidal data into waveforms by looking up phase information in a table. The resulting waveforms are successively summed into one complex waveform. A more efficient method is to sum the partials in the frequency domain and use the inverse-FFT algorithm to convert all partials to a complex waveform at once [Freed et al., 1993].

When utilizing spectral envelopes, sinusoidal track magnitudes can be obtained by using the corresponding frequency as an index into the envelope. If no modifications are performed before resynthesis, and provided the sinusoidal data and spectral envelope is correctly computed, this will still create a perceptually identical sound reconstruction. At the same time, manipulations using spectral envelope information rather than discrete sinusoidal magnitudes are much easier to perform, with more control over various aspects of the timbre (see chapter 3).

As already mentioned, reconstruction of the stochastic component can be thought of as “performing a time-varying filtering of white noise” [Serra, 1997], shown as *subtractive synthesis* in the figure. As with the deterministic component however, a more efficient approach is to generate a waveform with the inverse-FFT of the spectral magnitude coefficients and randomly generated phases. The two time-domain waveforms are then summed together to construct the final signal.

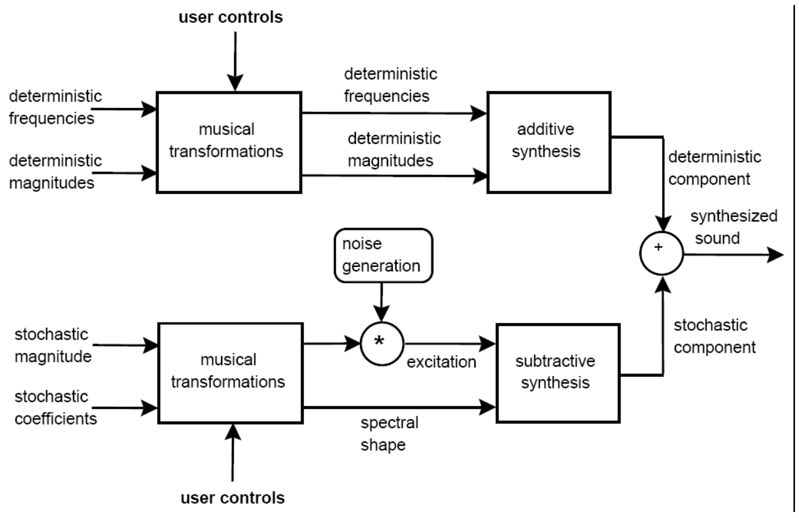


Figure 2.4: Diagram of the synthesis process in the sinusoids plus noise model [Serra, 1997].

2.1.5 Other Real-Time Concerns

Some additional concepts of organizing the spectral data in a manner that simplifies real-time synthesis and manipulation is worth mentioning, as the available manipulations depend greatly on the ease of sound recreation from the data. Many advancements to improve the accuracy of spectral analysis and resynthesis exist yet either quickly become too computationally expensive for real-time usage or compromise the ability to modify data before synthesis.

A large problem in frame based analysis is that transients do not fit well within one single frame. This important and distinct sound component is difficult to model as sinusoids because they frequency and magnitude evolution is too erratic for the framerate, yet not stochastic enough to be modeled as a stochastic residual. One approach to fitting transients into a sinusoidal model suggests to resample the frames where frequencies are varying rapidly to capture the evolution of the transients [Fitz et al., 2000]. The problem then is how to efficiently re-synthesize frames that need to be accessed at an irregular rate, as the most efficient known method for additive synthesis, inverse-FFT (see 2.1.4), depends on a steady frame rate. Another method extracts the transient events into a third component [Verma and Meng, 2000] before sinusoidal decomposition, and then during resynthesis tries to make the phases of the transients naturally lead to into the phases of the sinusoidal tracks. Yet, then the exact phases of the sinusoids are required before the transient is reconstructed, thereby blocking the possibility of magnitude or frequency adjustments during this short time. Without a much more complicated system than the one presented in this work, both of these techniques would make all of the manipulations presented in chapter 3 unfeasible in real-time. While SMS is in itself highly manipulable in real-time, applying some basic restrictions to the data further opens doors to many new manipulations. The sacrifice is that more complicated sounds cannot be accurately reconstructed, even if most sounds will offer interesting results in this system.

Many types of sounds, especially those with smooth evolution, do not benefit from using original phase values in re-synthesis [Serra, 1989]. If original phases are disregarded in the sinusoidal model, new ones can be calculated from a given frequency/amplitude pair. The first frame synthesized starts with randomly generated phases, then at the end of each frame the phase values of each partial track are stored so the next frame picks up where the last one left off. It is a similar case in the stochastic residual model, where phases are disregarded and randomly regenerated when needed.

Modifications in time are trivial if phases are generated at the time of synthesis. A constant interpolation time from frame to frame is first provided (in the inverse-FFT resynthesis method, this is the hop size) and then any frame can occur before the next with a smooth result. Furthermore, all or individual frequency and magnitude values can be modified without creating clicks or pops from phase discontinuity.

2.2 Software for Spectral Processing

This section describes the software used for spectral processing in the rest of this thesis, specifically using the techniques reviewed in Section 2.1. There is a short review of the history of SMS applications, followed by a description of *libsms*, a general purpose library in C with extensions for scripting and real-time processing.

2.2.1 The Original SMS and Child Applications

SMS was originally used in terminal-based applications that were controlled and tested with bash scripts. In the original SMS C package, MusicKit Jaffe and Boynton [1989] was used to create modifications over time with envelopes.[4] A score file was read that directed functions for time stretching, amplitude, frequency and hybrid modifications.

[4]sound examples from the SMS website:
http://mtg.upf.edu/technologies/sms?p=Sound%20examples

Eduard Resina later created the Windows application *SMS-Composer* Resina [1998], introducing a graphical user interface and sequencer for the available modifications. Many useful elements were incorporated, such as a clef for the pitch range and an envelope visualizer, that made composing with an SMS model more natural than what a script file can offer. More recently, the *CLAM* framework Amatriain and Arumi [2005], containing the original *SMSTools*, has been developed to allow for a collection of effects combined with SMS processing. Data transformations can be configured through 'transformation scores'; an XML script or with a graphical XY grid that allows one to set breakpoints for various parameters.

Many applications have been designed to take advantage of sophisticated audio processing techniques alongside SMS in order to achieve certain application's necessities. One example is presented in Loscos et al. [2000], where a system for singing karaoke impersonation is implemented by using parameters of an input voice to control the spectral models created from a professional singer. The singers' voice is analyzed for pitch, spectral envelope, and formant structure. This information is used to align the real-time signal with the stored analysis files, with the additional help of a score with melodic and lyrical information.

2.2.2 Additive Synthesis Within Max/MSP and Pure Data

The main tools for synthesis and modifications within this thesis were created with Puckette's second patcher *Pure Data* [Puckette, 1996][5], an open-source and generally more open-to-anything child of Max/MSP [Puckette, 1991]. I provide here an introduction to this type of programming environment and list some approaches to additive synthesis within Max and Pure Data.

Introduction to Graphical Programming

The Max paradigm [Puckette, 1988] is a real-time, graphical programming environment geared towards, although not limited to, sound and music applications. The system is based around connecting various processing and control objects together with wires that portray dataflow, hence the common name 'dataflow programming'. This type of application began in the 1980's with Miller Puckette's *Patcher*, which has since been commercialized and made famous as *Max/MSP*. Since then, Puckette has created an open-source version of Max/MSP called *Pure Data* (hereafter simply referred to by its community name, Pd) that aimed to fix some of the original application's flaws and remain within an experimental community[6]. Puckette [2007] provides an extensive tutorial in programming audio software with this environment

[5]Pure Data website: http://crca.ucsd.edu/~msp/software.html
[6]Pure Data community website: http://puredata.info

with examples. Within this document, $\boxed{\text{framed}}$ words represent Pd objects, compiled methods written in C, usable in real-time via the graphical 'patching' interface (see Figure 2.6, an example Pd patch that will be explained in Section 2.2.3). Those with a '∼' represent objects that process audio streams.

Max/MSP Objects

In 1999, the members of the CNMAT research group designed Max/MSP objects for interpreting and synthesizing sinusoidal tracks stored within *SDIF* [Wright et al., 1999a,b] files. The system reads and buffers analysis data processed with an external application, thereafter assessable by time index. This will send out sinusoidal data in a list, which is then routed to an oscillator bank called $\boxed{\text{sinusoids}\sim}$. While this was one of the early real-time additive synthesizers that aimed at high-fidelity sound re-construction, CNMAT has since extended their tools for Max/MSP in many different ways. They provide tutorials in spectral synthesis on their website[7] for learning how to use these tools [Wright et al., 2007].

The IRCAM research institute also provides a suite of analysis/synthesis software for Max/MSP within the *FTM*[8] library, called *Gabor* Schnell and Schwarz [2005]. This system processes audio among a group of high-level objects that handle data vectors and matrices specifically for the library, but also closely related to SDIF data. Many advanced processing techniques are available within the library, such as sinusoidal and spectral envelope extraction (see Section 2.1.3).

Pd Objects

As such tools for additive synthesis were missing from Pd when I began my research, I wrote an external called $\boxed{\text{sdiflists}}$ for importing *SDIF* 1TRC frames to be used as raw additive data, and $\boxed{\text{oscbank}\sim}$ for reconstructing this data into a waveform. Though based on the CNMAT method of additive synthesis described above, I took a more flexible approach than the implementation found in Max/MSP by not restricting the order of frames, thereby loosing some sound fidelity (various problems arose during the notorious 'births' and 'deaths' of the sinusoidal tracks), and also allowed for the wavetable to exist directly in Pd. The data was buffered using Pd internal and as such could be manipulated quite easily. I released the overall system as *Trax*, which later (Section 5.1) will be analyzed in in further detail.

[7]CNMAT *Spectral Tutorials*: `http://cnmat.berkeley.edu/patch/2741`
[8]FTM: `http://ftm.ircam.fr/`

2.2.3 `libsms` and Resulting Software

`libsms`[9] is an open-source, cross-platform C library for spectral analysis, synthesis and modifications. The code originates from the SMS C package (Section 2.2.1) that Serra wrote in support of his PhD thesis [Serra, 1989] twenty years ago. The quality of analysis and synthesis was quite high using this software with only a few alterations to make it usable on a modern-day Linux computer, even without newer analysis advancements. A few other applications had stemmed from this same original SMS code, although they were either heavily entrenched in a programming API (CLAM) or closed source (SMSTools). Thus, `libsms` was born out of a need for a more general purpose, open to the public tool set for SMS processing.

After removing the code for NeXT and switching to Erik de Castro Lopo's `libsndfile`[10] library for sound file input/output, I was able to achieve analysis and re-synthesis of many monophonic samples of musical instruments. The next step was turning the code into a general purpose library: changing to floating-point computation, revamping memory allocation, and prefixing namespaces. Documentation was improved with manpages, and Doxygen[11], making it easy for a new user to navigate through the code.[12].

While there have been many advancements to SMS analysis since 1990, the synthesis algorithms are virtually the same. The additive synthesizer uses the Inverse-FFT method (see Section 2.1.4), which uses very little CPU power on modern day computers. Synthesis routines are completely frame based, only needing one frame in history in order to compute linear interpolation of the data. On the other hand, the analysis routines in `libsms` require many frames of data due of a circular buffer that aids in finding the correct analysis window size. If a good window size is found quickly (possibly by manually setting it) then analysis is feasible in real-time. However, this isn't always the case and it would require a sophisticated, less general analysis system that would be a project in itself to build.

Many additional features have been incorporated into `libsms`. Discrete Cepstrum Spectral Enveloping (see Section 2.1.3) has been incorporated into the analysis system, which opens a world of modifications and feature extraction. Other simpler things were necessary, such decoupling the timescale and samplerate (it was most likely the case that only 44.1k and lower samplerates were used in 1990). Ongoing work is underway in order to modularize analysis components hoping to increase the integrity of the analysis through unit testing as well as pedagogical value. Other features, such as the ability to read multi-channel sound files, allow for many

[9]`libsms`: `http://mtg.upf.edu/static/libsms/`

[10]`libsndfile` homepage: `http://www.mega-nerd.com/libsndfile/`

[11]Doxygen: `http://www.stack.nl/~dimitri/doxygen/`

[12]A more complete list of changes is packaged along with the library, in the folder ./doc/changes.txt.

different applications to take advantage of SMS techniques. For the specific purposes of this, thesis, a substantial set of experimental envelope modifications have been added to the source. `libsms` depends on a few other open-source libraries, such as random number generation, matrix operations, or FFT's. These details can be found within the library's included documentation.

There are currently 3 types of programs that use `libsms`: command-line tools, a SWIG-wrapped python module (with examples), and Pd externals. The following sections briefly describe these programs, while there is more documentation included within the packages.

Command-line Tools

The command line tools are refactored and enhanced versions of the tools Serra original included in his SMS package.

`smsAnal` is a program for sinusoids plus residual analysis that originally contained twenty-four possible command-line arguments. More have since been added related to sound file reading (choosing which channel) and spectral envelope estimation. Consequently, it is most convenient to use the program in bash scripts, such as the examples included within the library package.

`smsSynth` is a program for synthesizing analysis data. There were original version contained arguments for choosing the output samplerate (although limited to 44,100 or 22,050) and which components to synthesize. Additional options have since been included for choosing between synthesis type (oscillator bank or Inverse-FFT), hop size, time scaling, stochastic gain, frequency transpose, and output file types.

`smsPrint` is used to print the analysis data. The format now prints using YAML[13], an easy to read markup specification, with an option to save to file. This was originally useful for importing in other applications, although it has been superseded by the python module explained in the next section. It is still a nice way to look at the data in text format.

`smsResample` adjusts the framerate of an analysis file. It has only been changed to stay consistent with the library.

`smsClean` tries to combine short tracks into longer, smoother tracks. It also has only been changed to keep up to date with the library.

[13]YAML: http://www.yaml.org/

pysms

`pysms` is a promising python [14] module for spectral analysis and synthesis techniques. Its quick and easy syntax, object orientation, and third-party modules make it a powerful tool for both prototype applications and script-based programs.

The module is created via a *SWIG* wrapper [Beazley, 1996] and `NumpY`[15] bindings for fast array processing. A common complaint about digital signal processing in python is that it is slow at processing arrays in comparison to C or other compiled programming languages. Yet, `NumPy` aims to help this in every way possibly, especially by providing a SWIG interface file that effectively *typemaps* C array pointers so it is possible to pass `NumPy` arrays into C functions and vice versa. `pysms` takes full advantage of all that SWIG provides in order to make `libsms` functionality both scriptable and usable in the python interpreter. SWIG actually makes programming with `libsms` similar to C++. Exception handling is automatic; if an error in `libsms` is detected, an exception will be thrown that can be 'caught' within python, else the program terminates, without any additional python code. The main structures for analysis and synthesis automatically call an initializing method to set sane parameters and functions can be overloaded to allow different numbers of arguments. It usually takes less than a tenth of the programming lines and time in order to create both simple and complex applications, with or without graphical interfaces.

Within the context of this thesis, pysms is mostly used to visualize the data and its features created through SMS analysis. Figure 2.5 shows two programs that I have created for visualizing spectral data. On top is an image of the traditional sonogram representation of a voice sample and on bottom is the same sample of the SMS decomposition into sinusoids and noise. As later explained in more detail (see chapter 4), sophisticated visual interfaces can use `pysms` along with two more powerful graphical API's: the combination of *PyGame*[16] and *PyOpenGL*[17] streamline high resolution prototyping, creating vivid interactive visualizations of SMS data with reasonably short scripts.

New code is normally prototyped in python, then ported to C and immediately wrapped back into python. Tests are included within the main library package, some of which compare the same methods in pure python and `libsms`. There are also examples of how to use some modifications, enveloping and viewing the data with *matplotlib*[18], now a common and powerful scientific plotting module.

[14]Python: `http://www.python.org/`
[15]NumpY: `http://numpy.scipy.org/`
[16]PyGame: `http://www.pygame.org/`
[17]PyOpenGL: `http://pyopengl.sourceforge.net/`
[18]matplotlib: `http://matplotlib.sourceforge.net/`

smspd

There are 4 externals that all operate on a common smsbuf buffer, similar to how Pd's delread~ and dealwrite~: a unique symbol is given as the object's first argument, which is then stored in a global list of symbols. This symbol, along with the globally declared smsbuf class, is then used to pass around a pointer with a structure of SMS data to other objects that contain the same symbol as a first argument. An example Pd patch illustrating this system is shown in figure 2.6. Here is a brief description of the current objects:

smsbuf is a buffer for storing, recalling or viewing SMS data. It can read and save data to file, print analysis information, and send out data to Pd in lists.

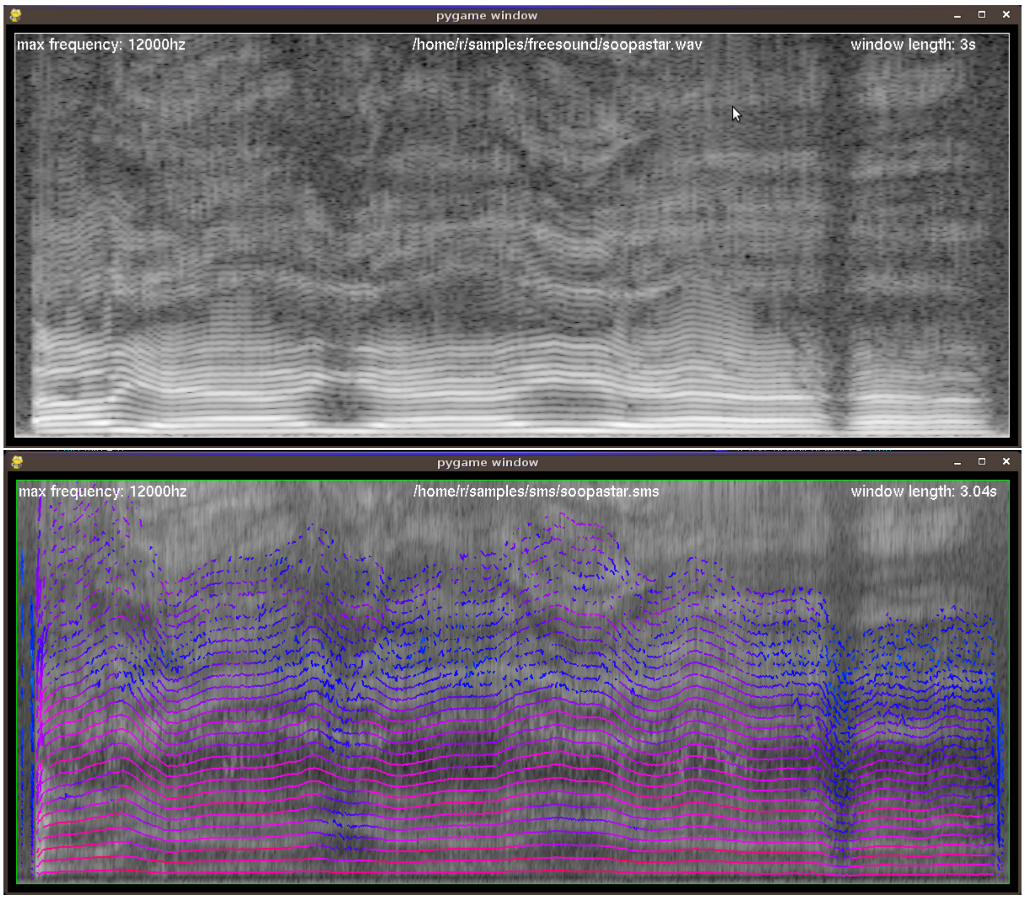


Figure 2.5: (top) 3 second long sample of singing voice, represented as a time-varying magnitude spectrum. (bottom) The same voice sample, represented as sinusoids (colored lines) and residual magnitude spectrum (gray-scale background), extracted using `libsms` and visualized using *PyGame/OpenGL*.

smsanal can analyze either a file or Pd array with audio data in a separate thread, storing the analysis data in an smsbuf.

smssynth~ synthesizes SMS data within an smsbuf into a time-domain waveform. The object reads a floating point index at the beginning of every block (the IFFT hop size, independent of Pd's block-rate) and interpolates between two concurrent data frames.

smsedit permanently modifies the data in an smsbuf. The edits are frame-based[19].

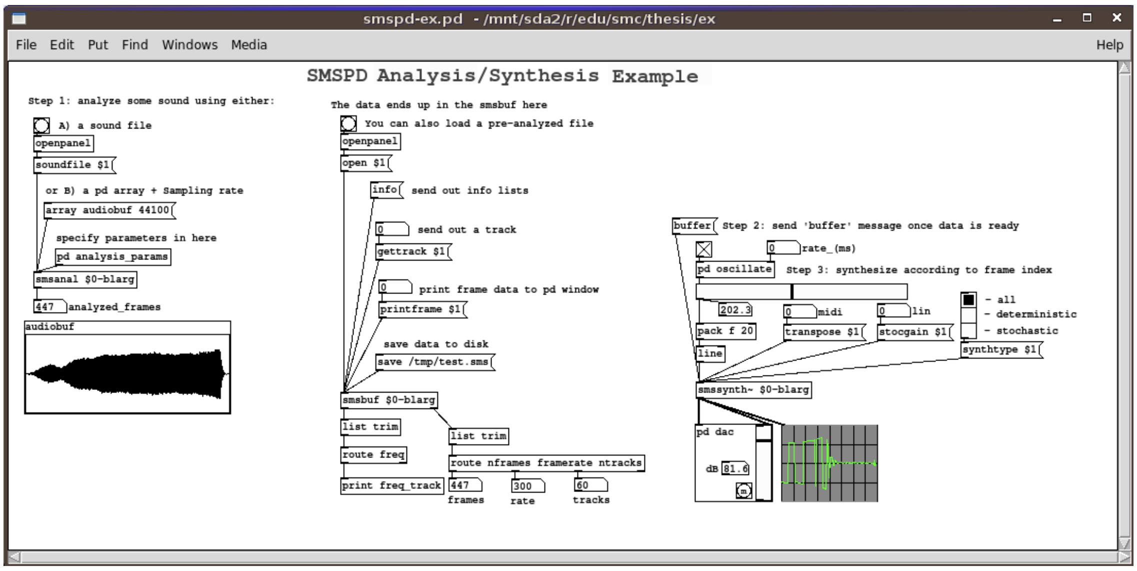


Figure 2.6: Example Pd patch for using smspd for analysis/synthesis. First, either load a sound file or put audio in an array and analysis that with smsanal. The data ends up in smsbuf, which can also load and save analysis data. Then initialize the smssynth~ and synthesize the data via frame index. Notice that all smspd objects use a common buffer name as first argument.

The choice to make smsbuf an object within Pd, and therefore globally accessible, was made hoping that it would simplify further advancements in data modifications. While smsedit is currently a fairly basic external for low level modifications,

[19] smsedit constitutes the first attempt at modifying the SMS data within Pd, but as it soon became clear that a system based on editing frames would be difficult to make interactive, data modifications shifted to smssynth~. The prior method of frame editing still shows promise in the realm of composition, although outside the scope of this thesis.

it requires very little code to access SMS data. Furthermore, there are certain types of manipulations where multiple data files are necessary. For example, $\boxed{\text{smssynth}\sim}$ can create 'hybrids' of two data sets by imposing a spectral envelope of one sound onto the partials of another [Serra, 1994], requiring access to two separate data buffers. On the other hand, some effects, such as polyphony and harmonization, are easily created with multiple $\boxed{\text{smssynth}\sim}$ objects that all have access to a single $\boxed{\text{smsbuf}}$.

Chapter 3

Interactive Manipulations

While the flexibility of spectral modeling continuously yields new and exciting transformation possibilities, the task of meaningfully manipulating the analysis data is not simple. It may be due to how easy one can ruin a good sound by either buggy programming or nonsense transformations that few successes are heard based on spectral modeling synthesis or similar techniques. Moreover, most commercial successes that use SMS aim to replicate the acoustic abilities of natural music instruments. There is much ongoing research in this area, yet surely the full potential has not yet been reached concerning how to manipulate spectral data in ways that *extend* the abilities to manipulate sound.

This chapter introduces some basic concepts for exploring new possibilities in sound transformations through interactive manipulation. Many research projects and resulting applications have led to advancements in non-interactive sound transformations that use scripts, possibly that contain graphical front-ends ([Amatriain and Arumi, 2005, Park et al., 2007] are just two of many examples). Yet, it is difficult to discover new manipulations with these attempts because the amount of time spent creating sound files, analyzing output statistics, etc., along with the vast number of transformational dimensions ends up overwhelming the possibilty of creative transformations.

As the approach taken here aims to be interactive and with a feeling similar to the control of acoustic instruments, I will lastly discuss mapping strategies that lead towards a responsive and expressively capable electronic instrument. The ultimate goal is to create a system that allows one to quickly experiment and 'hear' interesting results.

3.1 Manipulation Levels

While the classification of manipulation levels presented here may be debatable, it is clear that some are quite simple while others require a level of complexity delving into psycho-acoustic models. I will try to provide a simple taxonomy for different types of spectral manipulations in use today , all of which propose real-time capabilities, in order to clarify the approach of manipulation taken within this thesis.

3.1.1 Low-Level

By far, the most common modifications to spectral data deal with time and frequency scaling, which were taken advantage of even at the birth of sinusoidal modeling [McAulay and Quatieri, 1986, Smith et al., 1987]. While time and frequency are probably the most important axes of control within any music instrument, they are considered low level manipulations within the case of spectral data because there is no further parametrization necessary.

Multiplying the frequency of every partial track by a constant will transpose the sound while retaining harmonicity, if it exists. To manipulate this as pitch with a 'western' sense of melody, that is based on the equal tempered scale (see 4.3.2), it is useful to specify the frequency scalar in semitones:

$$y = \sqrt[12]{x} \tag{3.1}$$

so that scaling the frequencies by a factor of $x = 1$ will transpose the data 100 cents, the equivalent of one key on a piano. This type of manipulation has no knowledge of spectral characteristics such as envelope or tilt, or even fundamental frequency.

Time stretching or compressing is easily accomplished by re-sampling the synthesis frame rate. In a real-time SMS synthesizer, phases are usually ignored for both sinusoidal and residual data in favor of generating what ever is necessary to produce a contiguous sound at the moment of synthesis (see 2.2.3). Once again, frame resampling completely ignores sound events that should not be stretched in time, well-known as 'transient smearing'.

Amplitude modification is also trivial and can be applied to both sinusoidal and noise components separately. Otherwise, amplification is very similar in the time-domain.

Filtering can be accomplished by increasing or increasing the amplitude of a range of the spectral data. Frequency modifications to individual sinusoidal components or a harmonic sound distinguish that frequency range from the rest of the unitary sound. Doing this on one or just a few partials results is an electronic effect, in the form of a pure sine tone, yet transposing a larger set creates an interesting juxtaposition between the original data and the altered set, each containing a separate harmonic structure.

3.1.2 Mid-Level

With further parametrization of the sinusoidal and residual data, it is possible to change qualities of the sound commonly described as *timbre*. Interesting manipulations in this category are discovered through research in acoustics, instrumentation, physical models, and many times through plane experimentation.

If a *fundamental frequency (f0)* is assumed as the first sinusoidal track, some basic manipulations can be performed by deviating the harmonics according to this estimation. Perfect harmonics of a fundamental frequency would be at exact integer multiples of this frequency, but this is really only possible within synthetic sounds. In many cases, the exact cause of inharmonicity is not well-known as it is hard to analyze due to compacting causes. One understood cause of inharmonicity is with plucked or struck strings, such as those of a guitar or piano. These instruments are known to exhibit a harmonic 'stretch' factor, especially in extremely high and low frequency ranges where there is an increase in the string tension [Fletcher and Rossing, 1991]. This can be replicated by first calculating the harmonic deviance from a perfect harmonic and then adding this value back to the 'stretched' harmonic, based on a stretch factor. Another related possibility is to simply scale the value of inharmonicity for each partial, making it vary from perfectly harmonic, either more or less in the same direction. These two manipulations are displayed in figure 3.1. Many other possibilities can be discovered through experimentation with manipulating the spectral data in attempts to recreate similar effects that may be suggested by the patterns found in extracted harmonic sinusoidal tracks. Precise manipulation of inharmonicity, possibly yielding more interesting results, would require in-depth research using precise samples (see [Dubnov and Rodet, 2003] for a proposed model of the harmonic deviance in a bowed cello).

An effective way to manipulate the sound without changing its harmonicity is to only modify the magnitudes of the harmonics. It is most helpful to first compute the spectral envelope of the harmonics, as explained in section 2.1.3. One then has a generalized curve that can be shifted, expanded, contracted, or tilted, in order to give emphasis to certain spectral regions in a sound (see figure 3.2). Using the spectral envelope in this way can be seen as frequency domain filtering, where the frequency response of the filter is derived from a natural sound. As the residual component is also defined by a spectral envelope, the same effects can be performed to manipulate the frequency distribution of noise within the signal.

Other modifications are suggested through research in acoustics or other scientific fields. For example, it is well-known that reducing the magnitude of all even harmonics will create a spectral quality characteristic of clarinet and other 'closed

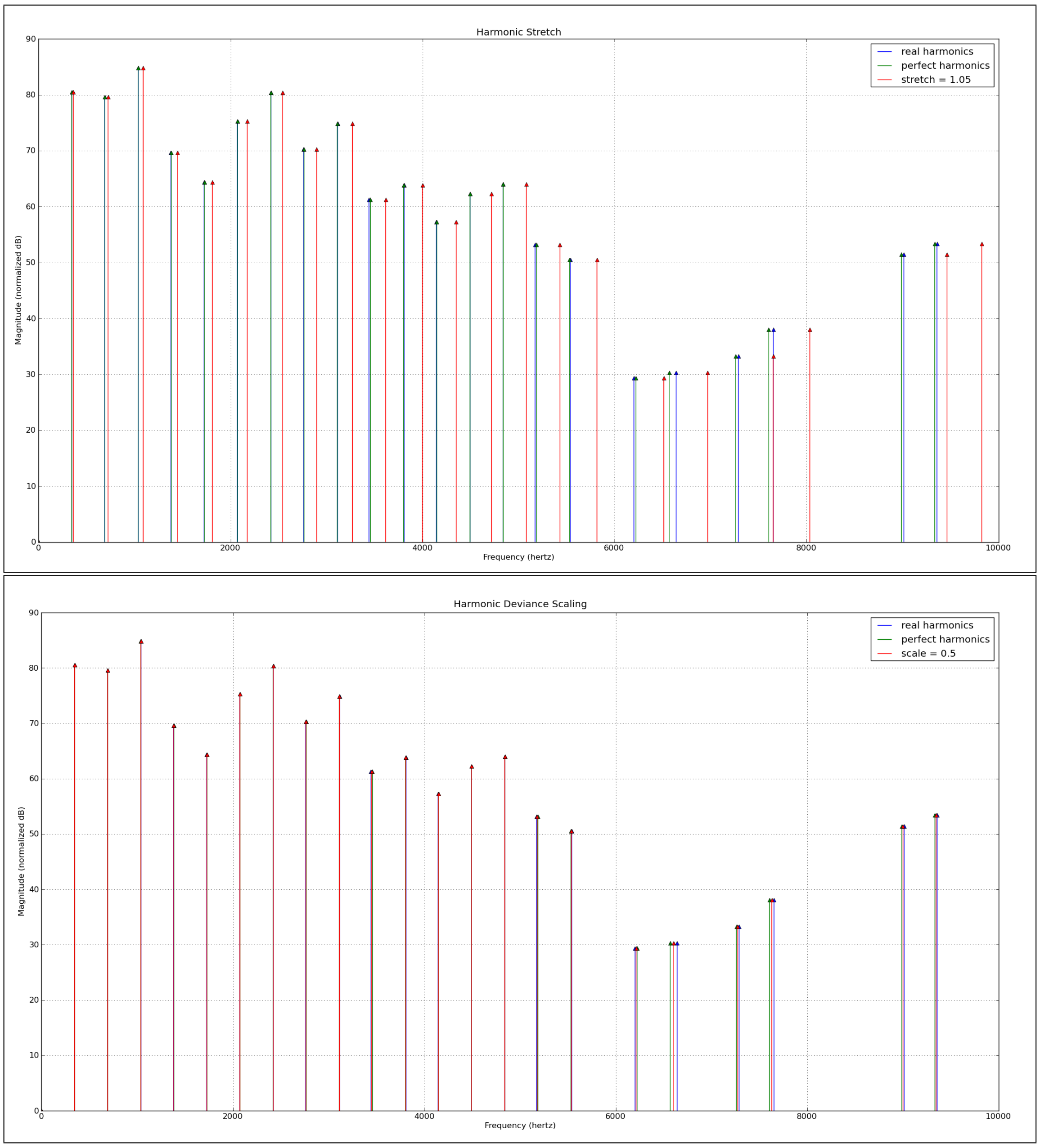


Figure 3.1: Simple examples of modifying harmonic locations. The top example 'stretches' the harmonic ratio by adding a factor of .05 of the fundamental frequency to each harmonic location. The bottom example scales the harmonic deviance by a factor of .5, making the harmonics closer to the ideal locations.

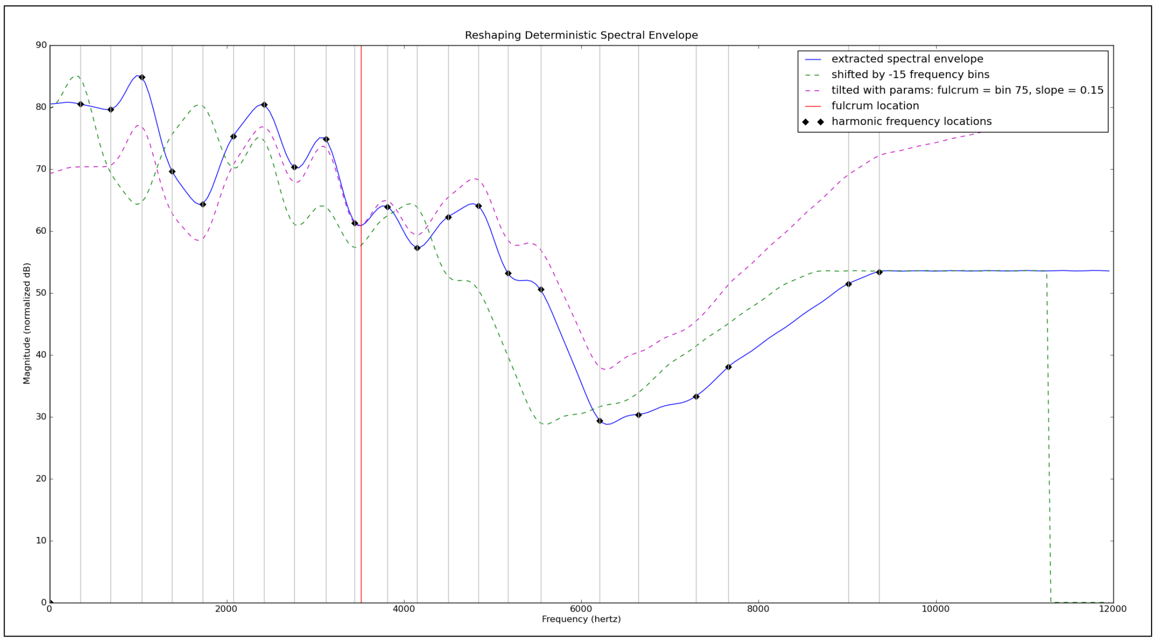


Figure 3.2: Possible methods for reshaping a spectral envelope in order to assign new magnitudes to harmonic partials. The diamonds show the original harmonic frequencies and the solid blue line represents the harmonic spectral envelope. By shifting or tilting the envelope, one can then use the original frequency locations (grey vertical lines) to find new magnitudes.

pipe' instruments [Fletcher and Rossing, 1991].

3.1.3 High-Level

While the effects mentioned in the previous section give good control over timbrel characteristics, they are not completely uncoupled. Either manipulating partial locations or magnitudes will change the spectral envelope. Moreover, increasing magnitudes in spectral regions more sensitive to the ear will increase the overall loudness of the sound. Spectral characteristics are coupled in acoustic instruments as well; in fact this coupling is what gives each instrument its unique strengths in creating some sounds, yet drawbacks in others. The key to successful music instruments is not their uncoupled control of sound quality, but the exact opposite; how to couple the controls in ways that allow for a unique and interesting sound from the instrument. One can view the high-level of sound manipulation in electronic instruments as a strive for an intuitive and expressive control of *sound features*.

In recent works, there have been many attempts to control spectral data based on its 'musical content' [Amatriain et al., 2003]. Some of these types of effects may include converting from male to female voice, dark to bright intensity, or smooth to rough texture. Most types of content-based manipulations tend to assume metaphors stemming from a traditional understanding of musical instrument qualities. Moreover, most methods for transitioning from one of these qualities to another is by annotating sounds and running batch descriptor processing to find other sounds that exhibit similar spectral characteristics. Then, one assigns a method to 'morph' [Serra, 1994, Fitz et al., 2002] between multiple analysis files in order to match the characteristics of the target timbre (recent examples using real-time SMS synthesis include [Verfaille et al., 2006, Grill, 2008].

The limitation within the content-based arena of manipulations is that one must define the *content*. If the target timbre is that of an acoustic instrument, the maximal result will always similar to what you could create without a computer. While a high fidelity acoustic instrument synthesizer has definite uses (i.e. removing the need to own the instruments and/or orchestra), it remains to be seen whether such techniques have the ability to expand in the direction of sound expressitivity. In other words, it is difficult to find new sounds if one is targeting the timbre of existing sounds.

In specific cases, there are other alternatives to creating high-fidelity synthesis by using existing knowledge of natural sounds. One case is to manipulate the spectral envelope with formants. Schwarz [1998] proposes to represent the spectral envelope as *fuzzy formants*, or a formant region within the magnitude spectrum defined by a center, lower, and upper point. He then suggests that one can shift the formant up or down in frequency with reasonably good results.

3.2 A Method for Real-Time Manipulations

In this section I will present an approach to gaining high-level control of spectral data through parametrization and real-time manipulations of many low to mid level controls. Similar concepts parallel some acoustic instruments that possibly take years to achieve control over what can be considered 'basic' manipulations in many electronic music instruments. For example, learning to play major scales in key on a violin remains the focus of any beginner violinist, yet after mastering this ability they have commanding control over possibly the most melodically expressive instrument ever created. It is clear that expressive parameters of the violin, such as bow position and speed, are closed tied to both melodic and rhythmic control as well, even more so because the violinist is forced to control all parameters simultaneously to play anything.

High-level manipulations, like those mentioned in section 3.1.3, seem to necessitate an intermediate mapping layer, usually drawing directly from the control mechanisms of acoustic instruments. *SSynth* [Verfaille et al., 2006] is a real-time

SMS instrument aimed at emulating the control found in acoustic instruments by mapping synthesis parameters, such as time index and file (including morphs between multiple files, consisting of wood, wind, and brass instruments) according to a database of time-changing spectral features such as pitch, intensity and spectral centroid. A physical controller is then mapped to a manipulation *result*, after handing the task of finding the necessary modifications in order to achieve this up to machine learning and musical feature annotations. This technique of creating expressive control is promising in acoustic instrument emulations, yet does not offer advice for controlling the sound analysis data 'outside' the acoustic instrument.

On the other hand, low- to mid-level manipulations can be manipulated as coupled or independent of each other in real-time, requiring minimal parameters of control within reasonably well known ranges. The resulting sounds from real-time 'fiddling' with spectral parameters varies between interesting, not so interesting, and irritating. While the best mappings of parameters for controlling spectral data is not yet know and likewise hard to concretely define, a logical method for discovering workable combinations is to expose many parameters at once and experiment in real-time. Through usage within a greater musical context, one can find innovative ways to combine basic manipulations into instruments with complex control of melody, loudness, and timbre, yet different from familiar acoustic instruments.

3.2.1 Continuity in Time

It is well known that the manner in which natural sounds vary in time contributes largely to their unique timbre. There are always subtle variances when playing acoustic instruments that will cause a flux in the spectral features we extract and use to recreate the original sound. The bow and finger pressure, resonances in the body, or even spatial variances are all complications within a violin recording that lend to the intricacies of harmonics that are now used within a system where the violin is abstracted.

There is still much to learn about the source of these minute details and how to recreate similar effects, which is a large focus of *physical modeling* techniques. Yet, there is little that can be discovered from looking at the low level spectral data one acquires through sinusoidal decomposition. An exact reconstruction of spectral data will maintain these intricacies by traversing the frames in the same order they were created. Furthermore, in many cases it is unnecessary to start at the beginning of a sound and stop at the end, maintain a constant speed, or even direction when indexing the frame-based data, in order reproduce sounds resembling the original acoustic instrument.

Any smooth curve, provided the region indexed has stable features, will maintain the original timbre to a high degree of fidelity. Figure 3.3 shows two examples of smooth time indexing and the resulting progression of sinusoidal tracks. The first

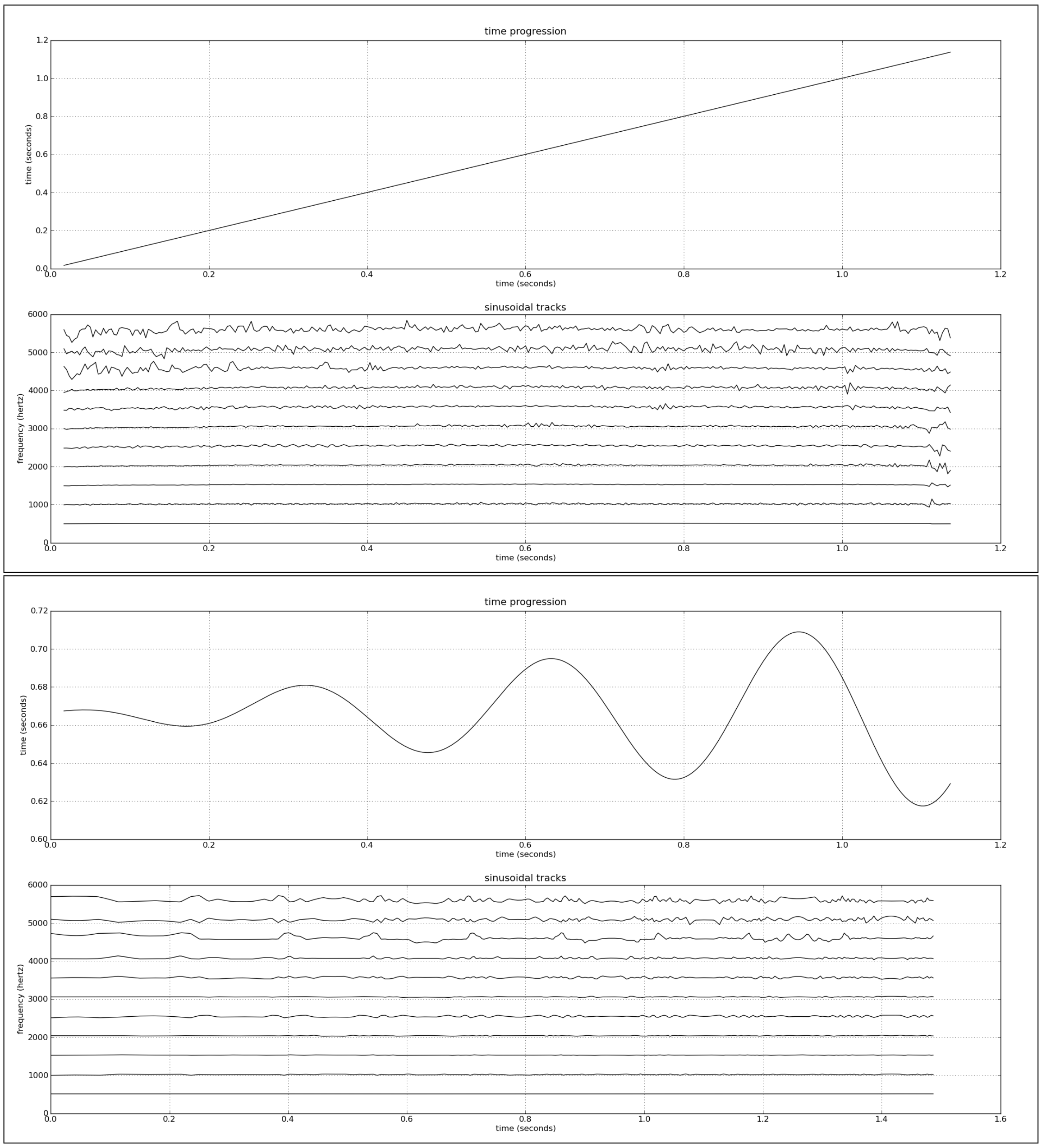


Figure 3.3: A crude example of re-ordering the time progression of sinusoidal tracks. On the top is the normal, beginning to end, time indexing. The bottom shows the same file traversed using a growing cosine shape, still maintaining frame-to-frame smoothness (although becoming more erratic as the oscillation magnitude increases).

is a normal beginning to end traversal, while the second traversal oscillates with increasing speed in either direction over time. Although it is difficult to see the minute details of the evolution in these sinusoidal tracks, those extracted from a wind instrument sample, the synthesis of both traversals sounds almost identical.

The sinusoidal tracks in the bottom of figure 3.3 were created from only the data within eighty milliseconds of the original file, about twenty-four spectral frames at three hundred frames per second. The traversal begins in the middle of the analysis data, which will create an abrupt sound unless a smooth amplitude fade is imposed.

Lastly, flexibility of traversing the sound data with arbitrary time indexing is greatly increases by removing the limitation of integer frames through interpolation. Any index curve with only integer values will not provide a traversal smooth enough to create the illusion of the original timbre if the index is slowly advancing. Research has shown that in all but extremely fast changing sounds, linear interpolation of spectral data results in a synthesis almost identical in perception to a synthesis with a more complicated interpolation scheme [Wright and III, 2005][1] On the other hand, using interpolation between frames and moving back and forth within only a short segment reproduces the time-varying qualities of the original sound quite well.

3.2.2 Continuity in Modifications

For the same reasons that time traversal needs to be smoothly changing, modifications should also ensure a gradual change if aspects of the original timbre should remain. Although low to mid level manipulations, like those mentioned in section 3.1, are changing the data, the hidden intricacies are for the most part preserved as a whole provided one does not change anything *too much, too fast.*

Park et al. [2007] proposes to perform spectral modifications that evolve smoothly by first extracting features from the sound and then defining methods for altering the feature. Modifications are performed by gradually interpolating in time from the feature-extracted parameter to the target parameter. The system is implemented in Matlab with a toolbox that allows for visualizing the modifications. The so called *Feature Based Synthesis* thus requires a target value before any modification can occur, resulting in a modification of many data frames. When manipulating the spectra in real-time, it may be more difficult to define a 'target' feature parameter, as every feature is also varying in time. Furthermore, not all features are directly manipulable (ex. spectral centroid 4.3.2) but would necessitate one of many possible schemes for altering the value.

The approach taken in this paper is much simpler, as it does not require feature analysis or parametrization. Instead of starting from an extracted feature value,

[1]Extremely fast changing sounds are not very well modeled as sinusoids or residuum any way, but should probably be modeled as transients.

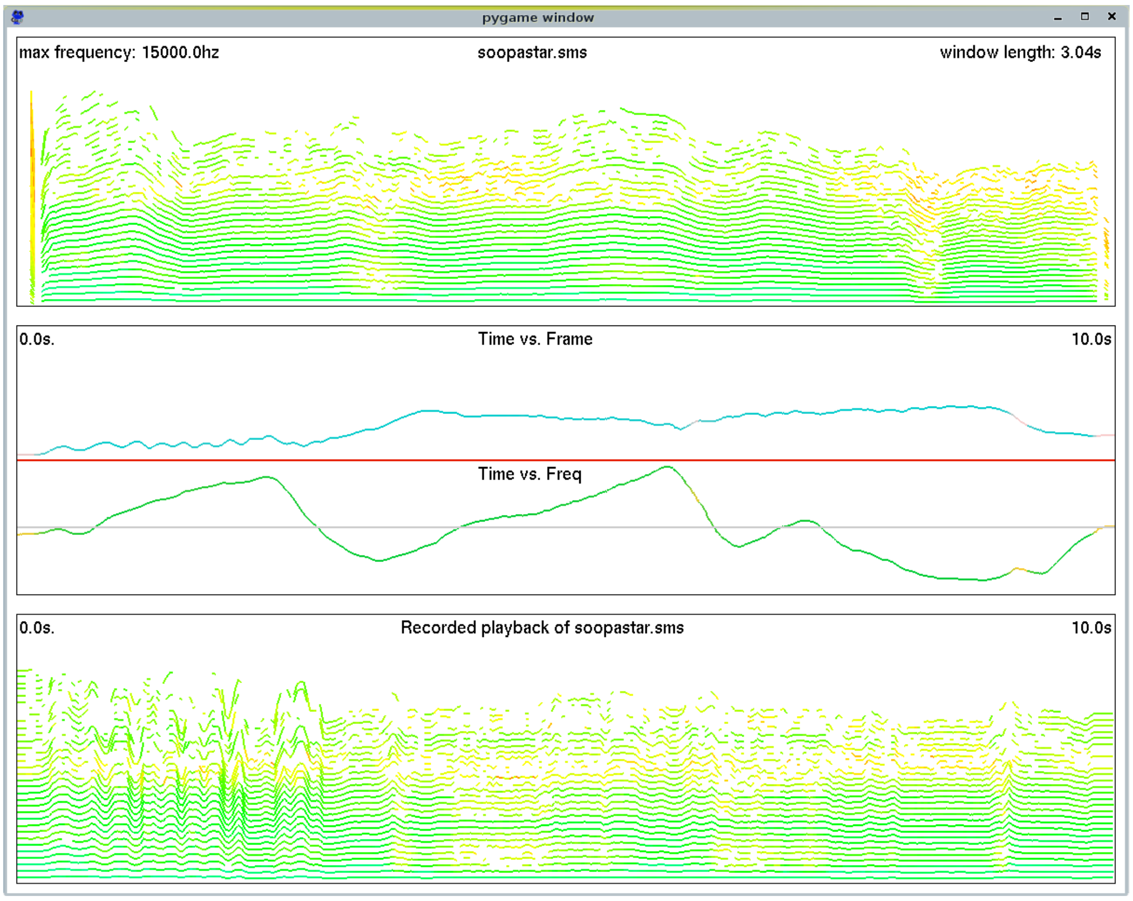


Figure 3.4: An example showing time / frequency flexibility. On the top is a representation of singing voice about 3 seconds long. The middle two x/y plots are 10 seconds recorded from the stylus pen, controlling time (index into the model) and frequency as described. The bottom plot shows the resulting tracks that were synthesized according to these manipulations, containing intricacies similar to the original.

one can introduce an addition, subtraction, or other simple operation gradually in time to the data frame. This will allow the spectral data to begin synthesis with its original values and become increasingly modified as parameters are increased.

Figure 3.4 is a example application for displaying the resulting sinusoidal tracks that are manipulated in real-time with continuous time, frequency and amplitude control[2]. The sinusoidal tracks extracted from a singing voice sample are displayed in the top section, which was traversed for ten seconds according to the recorded curves in the middle. The output data values were recorded and displayed in the bottom section. In the beginning of the recorded traversal, it is apparent that the region synthesized was varying rapidly (the first formant of the sample), but in the later section the track evolution is quite smooth. The frequency transpose has much less effect on the resulting tracks than the time index, as the pitch of this sample is varying rapidly as well. Yet, by reducing the speed of the traversal (in about the middle of the ten second recording, the traversal was near stationary), one can smoothly control the pitch transpose.

The other types of manipulations mentioned in section 3.1 also produce smoother results of the region of data traversed is stable. Manipulations such as spectral envelope shifting, harmonic stretching, or even envelope morphing can be combined to create complicated manipulations, but one must experiment with parameter values in order to find pleasing results.

3.3 Mapping

In order to make any digital sound representation interactive, it is necessary to abstract the underlining computational processes by mapping manipulation parameters to gestural input. Gestures can be captured by any type of digital signal, nowadays only restricted to the imagination and budget of an instrument designer.

Jordà [2005] provides an in depth theoretical overview of mapping concepts and ergonomics, as well as surveying the history of devices that have been used in controlling digital music instruments. I will discuss here those instruments and fundamental mapping concepts involved in controlling arbitrary spectral data as instrument. One important concept throughout Jordá's work is that digital instrument design is better considered a craft rather than science, as many of the decisions involved depend on the experience and motivation of the designer.

[2]The graphical interface and synthesis engine for creating this plot will be discussed in chapter 5, however it is useful for displaying what happens to sinusoidal tracks when they are manipulated in the manner discussed within this section

3.3.1 Uncoupled Parameters

The obvious and most common mapping in computer systems is simply *one-to-one*, where one controller is mapped to one parameter. For reasons of experimentation, it is valuable to be able to manipulate one parameter without effecting others, providing a feeling of the perceptional results of a given manipulation. Many of the possible methods of altering spectral data are not common to acoustic instruments, nor are their results. Moreover, some manipulations, such as harmonic deviation schemes, may produce very subtle results that will be indistinguishable if manipulated in conjunction with more manipulations that tend to have, perceptually, more extreme effects.

Within the context of this work, there is one parameter which seems to be so perceptually important that it needs an isolated method of control. Because of the manner in which time indexing is given arbitrary control, moving in any direction within a sound that may contain various types of spectral content can cause drastic changes in the resulting synthesis. Thus, the data indexing parameter needs precise control because it has the possibility to alter every perceptually relevant feature of the sound, regardless of whether or not the feature is well-defined.

Another parameter, pitch, might seems so perceptually potent that it needs to be have a solitary control as well. It is indeed common in electronic music to hear melody controlled independent of any other musical dimension, yet this is probably one of the least favorable characteristics of electronic music from the opinion of acoustic musicians. It is well known that melodic expression is created by more than pitch content, but with concepts of rhythm and phrasing, largely depending on dynamics. Timbrel control in piano melodies is expressed through chord voicing and in violin with the position and speed of the bow.

It is clear that electronic music instruments, including the one focused within this thesis, contain too many parameters of control for interactive manipulation via one-to-one mapping. While it is useful for attaining familiarity of the certain manipulations and their effects, one must move to more complex mapping schemes in order to increase control of musical expression.

3.3.2 Coupled Parameters

While one-to-one mapping this is a good choice for many scientific applications that require a precise control environment, the complex timbres of music instruments are usually due to their coupled nature of controls:

> Blowing harder on many wind instruments not only affects dynamics but also influences pitch, in such control difficulties is where, in fact, the expressiveness may lie for many acoustic instruments [Jordà, 2005].

Moreover, most acoustic instruments rarely contain more than two or three independent control mechanisms that require simultaneous manipulation. As already mentioned, something even as basic as pitch manipulation will usually effect other parameters of the timbre, such as brightness or dynamics. In acoustic instruments, these coupled controls usually evolve through centuries of design experimenting.

Yet, this aspect of mapping is something that an electronic music designer has much more control over. Intuitive coupling, such as pitch and brightness or dynamics and harmonic distortion, undergo experimentation and fine-tuning at a much faster rate, especially within a real-time programming environment (see 2.2.2). Likewise, absurd or unnatural coupling is feasible, which may lead to unexpected results, good or bad.

Throughout this work the main goal has been to create an environment for experimenting with new timbres through spectral manipulations. As such, the details of coupled mappings can only be discovered through practising with the instrument and using a variety of mapping schemes.

3.3.3 Wacom Tablet as Controller

While the choice for controller is largely up to the user and concepts which they are familiar with, the Wacom Tablet[3] has been used as the main musical controller throughout this work. The primary reason for this choice has always been due to the precision of the stylus; a pen-like object with absolute, continuous control of six couple controls (*absolute x*, *absolute y*, *x-tilt*, *y-tilt*, *pressure*, and *distance* from the surface) and one trigger (touch)[4]. Some of these controls naturally seem fit for our available parameters; pressure, or force, is a common controller of dynamics, and is thus mapped to the overall amplitude of the synthesis. Pitch is commonly portrayed on a vertical axis, such as in various music notation schemes, and is likewise intuitively mapped to the vertical location of the pen[5]. Most importantly to the data we are manipulating, frame indexes, representing original time of the analysis sound, are mapped from left to right along the absolute x axis, a tradition most likely leading back to the organization of text. This standard has been used almost exclusively in organizing musical material; trying to find new ways of representing time flow using other dimensions usually only clutters the coherence of the display.

These days, as anyone who searches the internet for videos related to music and a tablet, it seems the usefulness of this device meant for graphics applications has

[3]Wacom Intuos website: `http://www.wacom.com/intuos/`

[4]There are other also buttons on the stylus and tablet, along with sliders of various types depending on the model, but these rarely seem as fit for musical control as the main controls mentioned above.

[5]Section 5.3 will discuss both the drawbacks and gains for mapping a pitch transpose to a continuous controller such as absolute y position of the stylus.

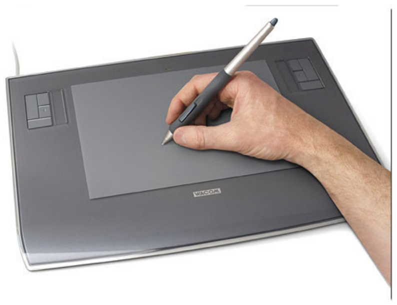


Figure 3.5: The Wacom Intuos 3 Graphics Tablet and Stylus (pen).

proved its worthiness as a sound controller. Even using a Wacom Tablet to control additive synthesis is now a relatively old concept Zbyszynski et al. [2007]: Matthew Wright has been performing with sinusoidal models in Max/MSP for almost 10 years now. While he uses a tablet for similar reasons - it has precise, absolute, and coupled parameters - the mapping scheme presented here is much different. Wright decides to map many sounds to the tablet surface, hand-annotated in a grid-like fashion. This gives him access to many sounds at any moment, yet with far less concentration on time or frequency accuracy. In the implementation presented in this paper, one complete analysis sound is mapped across the length of the tablet, usually kept short enough so that distinct sections in the data can be consistently located by the stylus tip without much effort. In this manner, the tablet is a very appropriate controller for the sound data dealt with in this thesis.

Other authors and researchers who have used the tablet as controller have done so because it is seen as a 'low-cost' or 'off-the-shelf' controller. Both of these terms are true to the device, yet it remains most valuable to interactive spectral manipulations because of the precise sampling in the absolute x and y coordinates of the flat surface. These effectively give control of precise locations within the data and, simultaneously, the possibility for micro-tonal pitch control. Yet, the fact that the device uses the usb protocol warrants some precaution to ensure that control data is consistent, as usb guarantees no real-time consistency. To this aim, the device is sampled at an audio block-rate to ensure that short or long bursts of usb data do not degrade the granularity of controlling the instrument.

Chapter 4

Interactive Visualization

Unlike acoustic instruments, a digital computer processes information in a manner incomprehensible to a musician, whether it be binary numbers or text in a programming language. At the same time that a computer processes sound data in a completely non-artistic medium, it can produce vibrant images and animation through precise geometric shapes, textures, and color scales, with the same data. So then, a viable method for revealing what is 'under the hood' is to think of creative methods for data visualization, and as of late this has become a crucial aspect in computer music systems.

This chapter focuses on visualizing techniques of spectral data in order to solve two problems within the realm of real-time manipulations. The ultimate goal is to reveal the secrets hidden within a complicated set of sound data to a human performing with a sound. Both operate by receiving the same control parameters handled by the synthesizer in real-time

First, a visual interface depicting 'what the synthesizer is doing' is introduced. This provides an informative view of the synthesis process, aids in experimenting with new types of data modifications, and facilitates brainstorming new possible data modifications.

Second, methods are proposed for representing an entire sound sample in a visualization that informs a performer what musical content is within the sound; a *feature representation.* It seems that as more computation is necessary to convert sound parameters to sound, the system tends to mask possible ways to manipulate the sound within its automated tasks. A computer is, by design, efficient and consistent with data reproduction. Yet, a computer can hardly determine what is artistically meaningful or useful, as this subjectivity is unique to the artist. This is the case in most common displays of spectral data, in which remain at a level easy for a computer to interpret yet far from how a human actually hears the sound. The sonogram, although it is clearly more informative than a waveform, seems to contain too much information to use as an aid in controlling a musical instrument.

If one can visualize the content of the sound before deciding how to manipulate the sound as an instrument, in a manner that reveals the musical potential of the sound, they gain intuition and control in their performances.

While visualization options may appear limitless, it is difficult to create sound to visual mappings that make sense from a musician's, or listener's, viewpoint. Spectral Data visualization has a rich history for as long as there has been digital spectral analysis (the first sonograms were created even before digital computation [Potter, 1945]), probably due to its dense nature. For this reason, my main goal here to use represent the data in a way that gives insight into the 'content' of the sound by further abstracting spectral data into spectral features. The visuals are kept simple and few numerical cues are utilized, as exact numbers do not help musical manipulations in most cases.

My overall approach to the visual aspects of this thesis is towards using the interface interactively and intuitively while manipulating sound. Chapter 3 discussees the type of manipulations possible, which suggests using additive or multiplicative modifications to the spectral data, controlled with continuously sampled parameters. It is clear that not all the manipulations will create good sound results all the time. Some modifications will work better depending on the sound content, which is, in this system, constantly changing. These uncertainties are lessened within the visual interface presented here.

4.1 Re-coupling the Sound and Representation

The high level sound representations used within the scope of this thesis (specifically the techniques reviewed in section 2.1.2), are in themselves disconnected from music. In the process of developing the representations through analysis software, the concern normally focuses on creating data perceptually identical to the analysis sound. At the point of entrance of this work, which concerns how to manipulate this data as a music instrument, the data requires a more interactive interfacing approach in order to be playable. A fundamental requirement is to control the synthesis in real-time. Then, an actions are immediately tied to sound production as in the natural world once again; the user feels like he/she is creating the sound. When trying to discover new sounds with spectral data by simply turning knobs and faders, it is difficult to understand what effects are useful because the data content is hidden from the user.

Acoustic instruments tend to depend on a functional control layout, while feedback is provided in various *haptic* forms. This would be hard to replicate in an instrument based on arbitrary spectral data. How could you physically distinguish the differences between bright and dark sounds, harmonic and inharmonic? A more practical way to re-couple this type of synthesizer is to use real-time visual feedback.

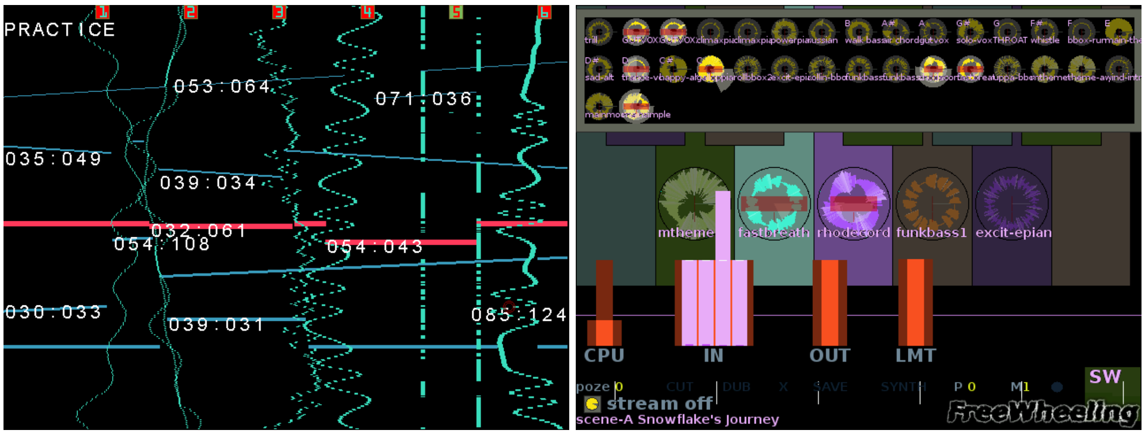


Figure 4.1: (Left) *FMOL*'s first visual interface, called *Bamboo*. This visual interface attempts to reveal just about everything going on in the sound engine [Jordà, 2005]. (Right) *Freewheeling*, an interactive application for loop-based music improvisation. The spinning wheels represent time-domain samples (one revolution makes one loop).

> The 'audio' feedback presented to the performer in an interactive visual form, intuitively helps the understanding and the mastery of the interface, enabling the simultaneous control of a high number of parameters that could not be possible without this visual feedback Jordà [2005].

Providing a functional visual layout is important when learning any new instrument; it helps a beginner make a connection between gestures and sound and an experienced user think of new possibilities. In Jordä's own instrument, *FMOL* (see left figure in 4.1) he aims to reveal all the controls of a sophisticated 8-track sound engine, with each track having a large set of audio plugins and parameters. The interface contains vertical lines representing the tracks and horizontal lines representing the effects that manipulate the tracks. When a track is causing sound, its corresponding vertical line oscillates as a visual cue that helps to connect the sounds a performer hears and the manipulations performed with input controllers.

In instruments that are based on sound samples, the same gesture can result in a multitude of sounds, so it is therefore most helpful to display some information about the sound before it is synthesized. *Freewheeling*[1] is an innovative approach to displaying samples in a manner conducive to improvisation (see right figure in 4.1). The following quote from the application's website describes the design philosophy:

> It works because, down to the core, it's built around improv. We leave mice and menus, and dive into our own process of making sound.

[1] *Freewheeling*: http://freewheeling.sourceforge.net/

While the display is quite simple, the features of the sample are portrayed well in the revolving 'wheels'. This waveform display accompanied by a descriptive name are enough to allow a performer to interactively start and stop a set of samples via simple trigger gestures. As the waveforms revolve, it is easy to localize each component of a performance that may contain many simultaneous samples looping at the same time. Very little text is necessary in this setting, as one quickly adapts to the interface with a few performances.

Freewheeling, along with most other sound applications, are designed to give a user simultaneous access to a set of samples in order to create music (in this case improvised, although most others are based more on composition).. Applications based on spectral analysis/synthesis techniques do not need many sounds. While some have decided to go this route, using a large data base of pre-analyzed sounds as in [Verfaille et al., 2006] and[Grill, 2008], I find that there is plenty of creative and exploratory potential within one spectral representation, provided the sound source is interesting to start. Still, real-time spectral synthesizers have not seen the such informative interfaces as waveform-based systems; most designers tend to either disregard the interface and hide the underlining computations or display everything in a scientifically oriented interface.

4.2 Towards a more Interactive SMS Interface

The SMS-based instrument presented within this thesis further necessitates the need for visual feedback, as the resulting sound not only depends on which sample is used, but also *where* in the sample one is currently playing and *what features* this section contains. Manipulating with one section of an analysis can result in a completely different sound than another section, along with any modifications such as those proposed in chapter 3. Yet, gestural control remains identical. This system benefits from revealing how the data varies in time as one performs manipulations in order to distinguish how different manipulations work in different sounds..

The first step in creating an interface more conducive to interaction is to reveal what the synthesizer is synthesizing. Within the SMS data structure, as computed in section 2.1.2, there are three distinct components in every frame; sinusoidal tracks, a deterministic spectral envelope (an extra computation step to facilitate manipulations), and a stochastic spectral envelope. Figure 4.2 shows a minimalist display of this data within a single window. There is no grid because exact frequency or amplitude values are not necessary in this view; the ear does not process sound based on exact values (unless this ear belongs to a classically trained pianist). Instead, a value for the maximum frequency is displayed, while the maximum magnitude is realized to be ranging from 0 to 100 decibels, (100 corresponding to a linear amplitude of one)[2].

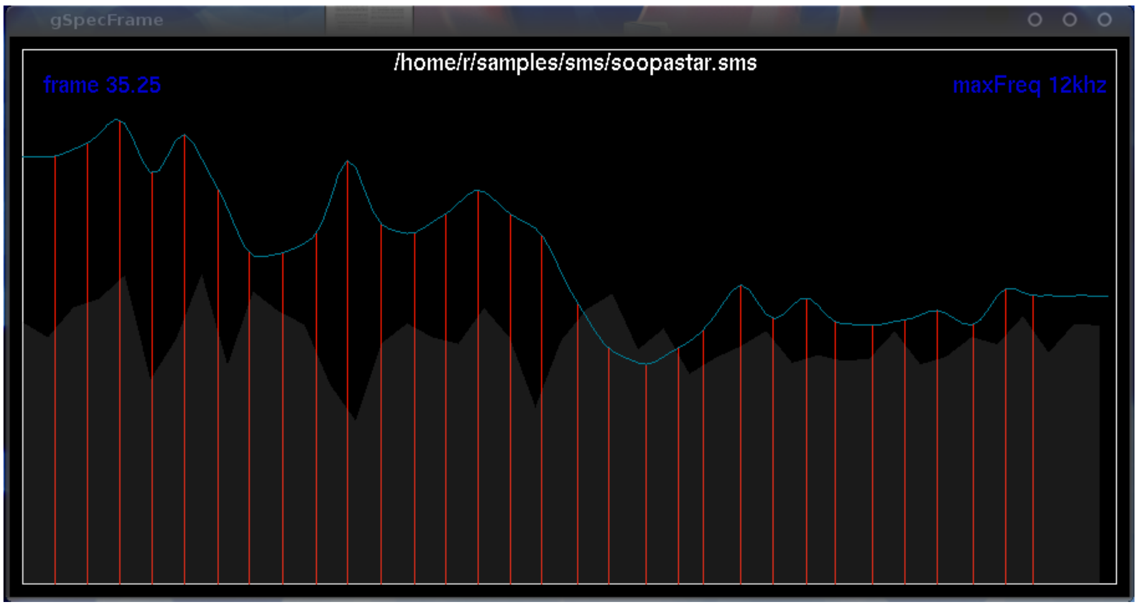


Figure 4.2: Spectral data of one SMS frame: sinusoidal tracks (red vertical lines), deterministic spectral envelope (blue curve), and stochastic spectral envelope (gray shape in back).

This frame display is meant as an interface to reveal, in real-time, a visual depiction of the data within the current synthesis buffer. This includes modifications; if harmonics are transposed, they should be shifted in the display interface. If either component's gain is adjusted, the color should change to reveal this. Modifying the spectral envelope, harmonic deviance, or other features should be revealed within this plot. In this way, the *reason* why a certain manipulation is pleasing or abrupt may become apparent via properties of the visual representation. Furthermore, in such a display it is easy to see when a sound's data components are changing erratically from frame to frame, which would usually cause an odd sounding synthesis.

4.3 Displaying Time-Varying Spectral Data

Common methods for displaying time-varying spectral data are quite dense. *SMSTools* (see figure 2.1) and popular sonogram tools display the content of each

[2]This decibel normalization is one of many that happens to simplify many computations. It seems most scientific applications normalize a linear amplitude of one to zero dB and everything else below, but this is not the case in the software I primarily deal with (libsms and pd) so it is convenient to push everything above zero.

frame vertically, using colors to represent magnitudes, so that left to right depicts how data changes in time. The data resolution is usually limited by the available pixel resolution and as such many times there is not enough resolution to show all the data. Some choose to display the third dimension as another axis instead of color [Bécares, 1998, Bresson and Agon, 2004], commonly known as a *waterfall* plot. Yet, although it is aesthetically pleasing to see complex three-dimensional shapes, they are both slow to render and complicated to navigate in real-time.

In the system developed within this thesis, there has always been a need for displaying how the data changes over time in order to better understand how to intuitively navigate the sound in real-time. A sonogram view was the first to be used, for practical and traditional reasons, yet it always seemed to display unnecessary or redundant information as an instrument. For example, harmonic sounds will have up to forty or so partials to display, yet the pitch is determined according to only the fundamental frequency. Intricacies in spectral envelope, either deterministic or stochastic, take on the form of a color variation within a region, but the exact region boundaries are hard to determine or understand when manipulating the data in real-time. Harmonic deviance are completely unexplained in the sonogram view; all that can be deduced is that there *is* deviance.

The rest of this section will discuss some experiments of abstracting the time-varying spectral data into a visual representation more related to how the 'sound data is heard'.

4.3.1 A Higher-Level Spectral Data Representation

As a sound representation increases in its level of abstraction, its data becomes closer to how we naturally perceive the sound. Serra writes:

> The auditory system functions as a spectrum analyzer, detecting the frequencies present in the incoming sound at every moment in time Serra [1989].

To take this analogy further, the human auditory system also abstracts features of the detected frequencies into high-level information used to recognize the sound. It is clear that the brain condenses this information into features useful in identifying various sounds in musical and everyday situations. Thus, It seems to be useful to present a visual representation that abstracts from the low-level, time-changing partial frequencies to higher-level, time-changing spectral features. Visually representing the data as time-varying features can possibly facilitate an understanding of manipulations, how they are audibly perceived.

For facilitating navigation of samples within a large online database, *The FreeSound Project*[3] displays sounds as a combination of time and frequency content. Figure 4.3

[3]The FreeSound Project: `http://www.freesound.org`

shows a sample of melody played on saxophone, thus having time-varying spectra. With this visualization, one can see the overall amplitude shape of the sound and, at the same time, get a good idea of whether the frequency spectrum is relatively high, low, or fluctuating. Waveforms are normally varying too rapidly to display a time-varying line and are instead generalized as a 'blob' created by the low and high amplitudes, sampled at a pixel resolution. As this blob is normally just filled with a constant color, this approach is much more informative in revealing what content is in the sample without hearing it.

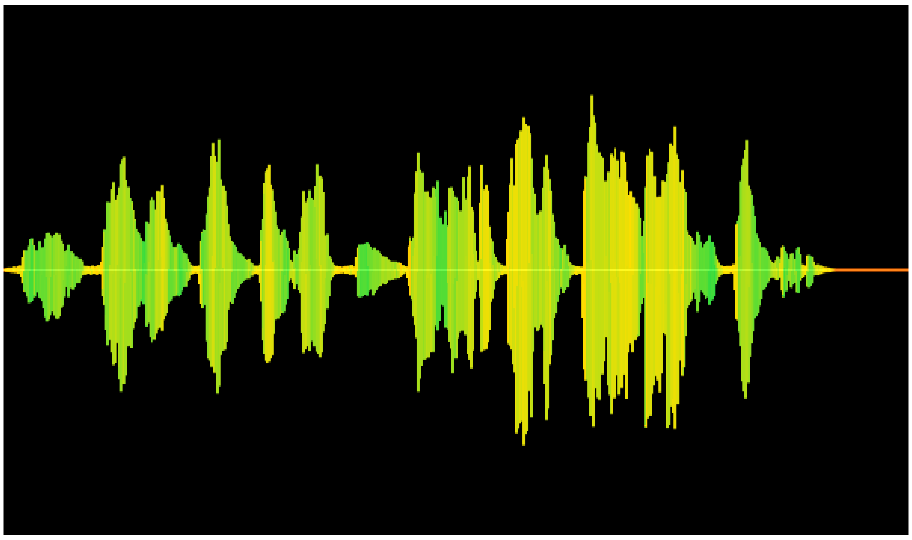


Figure 4.3: Waveform view of a saxophone sound sample, colorized according to the spectral centroid, used to create the waveform images for *The FreeSound Project*.

When dealing with spectral data, one can go much further in *revealing* the content of the sound through data visualization. Figure 4.4 is a diagram representing the process of abstracting the data and its common visualization at each step, until we reach spectral features. One starts with a waveform that reveals how amplitude fluctuates in time, but otherwise masks perceptual information. After a transformation via the STFT, it is clear the sample is harmonic with a time-varying pitch. Next, SMS analysis extracts the harmonics and reveals that there is high pitched noise information (the gray background). Finally, the time-varying spectra is broken down into features with a single value each frame, each contributing to a unique characteristic of the image.[4]

While the computation time increases and becomes more problematic high up on the 'abstraction ladder', it also significantly decreases the amount of data. In

[4]The resulting display is provided here to suggest a possible visualization of the features, among many. An evaluation of its effectiveness and possible alternatives are discussed in section 5.3.

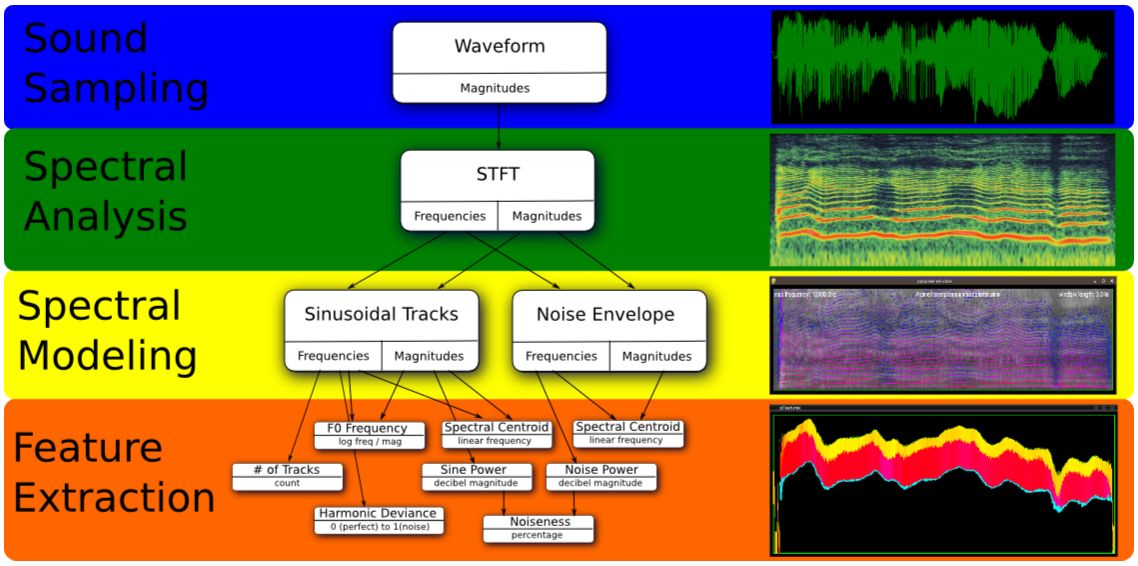


Figure 4.4: *Diagram of extracted features detectable in one frame of sinusoidal and noise data.*

contrast to plotting every sinusoidal track in a frame, mapping extracted features to an alternative visual representation can immensely reduce visual clutter and computation time in order to display the sound representation. Each extracted feature in figure 4.4 is represented by one value per frame, whereas there are somewhere around 40 sinusoidal tracks and 256 residual coefficients in each frame of the visualization one step below. Still, some features inherit straightforward visual representations (such as pitch) while others remain largely experimental (how does one visually conceive harmonic deviance?). In the end, it will take interacting with many sounds and discovering how each feature should be visually represented according to the ease of connection between the audio and visual feedback.

4.3.2 Computation and Visualization of Spectral Features

The following is a non-extensive list of frame-based spectral features that I have experimented with for visualizing the time-varying perceptual qualities contained in a sinusoids plus residual sound representation. The computation of the features is explained along with perceptual significance and how this can be linked to a visualization, an easy task in some cases while problematic in others. The features presented here can accurately be computed from a ‘successful’ sinusoids plus noise decomposition has already been computed, yet may create a bogus visual otherwise.

The time axis is not discussed here because it is created by displaying the frames consecutively, thereby revealing time-varying features from left to right. Spectral envelope is also left out, as it is difficult to describe with few parameters. Further abstracting the spectral envelope into a more condense visualization is beyond the constraints of this thesis, although it would surely be most useful.

Amplitude

Amplitude is arguably the most fundamental feature of a sound in any domain. Within a sinusoids plus noise decomposition, it can be computed for each individual components. Serra and Bonada [1998] calculate the amplitude of the sinusoidal component, A_{sin}, by summing the magnitudes and expressing as dB:

$$A_{sin} = 20 \log_{10} \left(\sum_{i=1}^{I} a_i \right) \tag{4.1}$$

where a_i is the linear amplitude of the ith partial track and I is the total number of tracks. The amplitude of the residual is summed in the same manner by summing the linear magnitude of I bins and representing as dB.

In order to calculate the total amplitude, one sums the amplitudes of the individual components before converting to dB.

Amatriain et al. [2003] suggest to represent the amplitude on a power scale in dB:

$$P_{sin} = 10 \log_{10} \left(\sum_{i=1}^{I} a_i^2 \right) \tag{4.2}$$

Loudness has been considered as one of three 'main axes in a sound transformation', although it is clear that amplitude does not completely describe loudness. Still, representing amplitude in a logarithmic scale largely describes this most basic feature of a sound.

Visualizing the amplitude as a feature is probably best in the time-domain representation; that is, the louder the sound, the larger the image. This can be replicated with spectral data as well, although one can portray the ratio of the two component amplitudes by dividing the overall shape according to the respective calculated amplitudes. One other remark is that a time-domain sound representation is usually near symmetric around the horizontal axis. This information is redundant and unnecessary in the frequency-domain portrayal of a waveform, although some might find it familiar. Still, that the information is different (the representation is not a waveform), it seems that breaking from the tradition of symmetrical amplitude seems appropriate.

Pitch

Pitch is the perceived fundamental frequency of a harmonic sound. The fundamental frequency is the Within SMS data, it is the first sinusoidal track. The algorithms for obtaining this data vary (see Section 2.1.2) between implementations and an in-depth explanation of the scheme used within this work is described in [Cano, 1998].

In musical applications, it is useful to represent the pitch frequency based on the Western musical scale, which divides octave into twelve equal *semitones* [Puckette, 2007]. It is furthermore useful to express pitch logarithmically, as in the MIDI specification:

$$m = 69 + 12log_2(f/440) \tag{4.3}$$

where f is the input frequency of the pitch and m is the output MIDI value. 69 represents the 'A-440', the concert pitch of an A note above middle C. This can then be mapped to a line that vertically varies in a linear fashion.

Without a well-defined layout scheme of the pitch, especially since the pitch is arbitrarily changing within the data we are dealing with, it becomes extremely difficult to created meaningful melodies. Jordà [2005] takes note of Jeff Pressing's opinion on the varying ability for different instruments to improvise melodically:

> The design of some instruments allows more precise visual feedback and more categorical kinesthesic feedback than others. This is almost certainly why sophisticated improvisation using advanced pitch material is more difficult on the violin than the piano, and extremely challenging for the vocalist [Pressing, 1988].

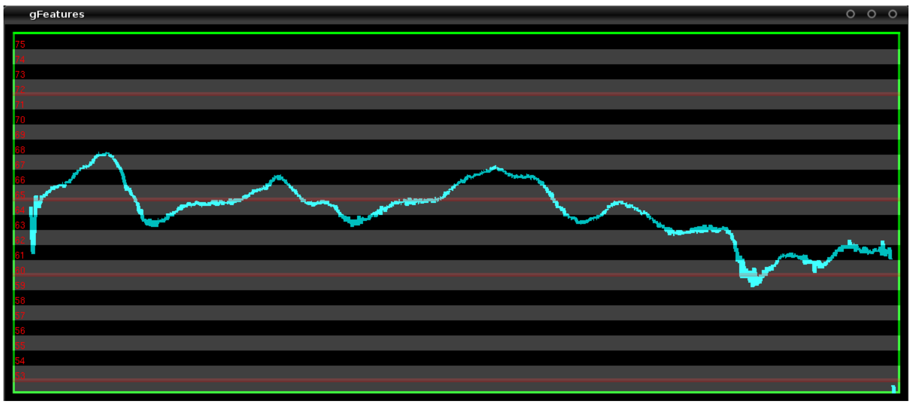


Figure 4.5: The time-varying fundamental frequency extracted from an SMS analysis, then mapped onto a MIDI grid in order to reveal precise pitch content.

The piano contains possibly the most intuitive interface for melodic creation because of its precise, repeating style and clean layout in space. In order to draw on these strengths, a powerful method of revealing exactly what the pitch is at any frame in our spectral data is to map the fundamental frequency to *pianoroll*, symbolic of the piano keyboard and what is commonly used to display MIDI information. Figure 4.5 shows this mapping, along with the MIDI note numbers of each key on the far left. One significant difference is that our spectral data does not contain *discrete* pitch values, nor should it. So, the integer fundamental value is correlated with the lower edge of each 'key', or horizontal bar. The keyboard is positioned by calculating the mean fundamental frequency, then centering the keys and pitch around this.

Spectral Centroid

The *spectral centroid* defines the 'balance point' of the magnitude spectrum, defined in Serra and Bonada [1998] as:

$$Centroid = \sum_{k=0}^{N-1} \frac{k}{N} f_s \times \frac{|X(k)|}{\sum_{k=0}^{N-1} |X(k)|} \tag{4.4}$$

where $X(k)$ is the magnitude of spectral bin k, N is the number of spectral bins, and f_s is the sampling frequency.

For the residual component, the spectral centroid is immediately computed from the above formula. For sinusoidal data, it is easier to replace $\frac{k}{N} f_s$ with discrete sinusoidal frequencies and X_k with discrete sinusoidal amplitudes.

Spectral centroid is known to be closely correlated the musical perception of *brightness* [McAdams et al., 1995] and is invaluable for distinguishing between different timbres. As such, it makes sense to represent spectral centroid with color, brighter colors signifying higher frequency content within the frame (as in Freesound's display: 4.3. When visualizing the amplitudes of sinusoidal and residual components separately, it is logical to use their respective spectral centroids as a color fill. One difficulty arises in deciding what scale of frequency to use, as the sinusoidal and residual components tend to have very different ranges of spectral centroids. Furthermore, if a linear frequency scale is used, high frequency content may show a drastic variance in color that is largely unperceptive.

Harmonicity

A *harmonicity* measurement describes how far a the data is from purely harmonic. That is, how much the harmonics stray from the perfect, ratio locations. Peeters [2004] defines the harmonicity factor as:

$$inharmo = \frac{2}{F_0} \frac{\sum_{i=1}^{I} |f_i - (F_0 \times i)| \times a_i^2}{\sum_{i=1}^{I} a_i^2} \tag{4.5}$$

where F_0 is the estimated fundamental frequency of the f, a partials. The second term weights the partial distortion by the energy of the frame. A resulting value of zero would indicate a perfectly harmonic sound, while a value of one should indicate that the average deviance is directly in-between the ideal harmonic locations.

Representing harmonicity as a visual is a difficult task. One would ideally want a generalized and reduced display, yet a line or color hardly seems fit. A texture visual seems appropriate (ex., crosshatch), where the density or regularity of the texture is mapped to the calculated harmonicity. For simplicity in computation and as a first step, I have mapped harmonicity to either the color or alpha (transparency) component of the fundamental frequency, although neither fit well.

4.4 Future Prospects

This section has introduced a concept of visualizing sound data based on its features with a single graphical object, which seems to be largely unexplored within the field of research. It is a promising method for displaying the sound within an application that is geared towards real-time usage, where in a sonogram data representation may contain too many graphical objects to quickly comprehend. This most certainly seems like a creative way to visualize the sound and may lend some help within the artistic processes of improvisation and composition. Furthermore, it helps one to hear components of the sound that are otherwise congested into a whole; to aid in bringing the subliminal to the conscious.

The features described here are far from complete. Serra and Bonada [1998], Amatriain et al. [2003], and Peeters [2004] define and discuss many more features that could be correlated to a descriptive image of the sound. Yet, before these can be incorporated, more experimentation needs to be conducted with the few but powerful features described in this chapter. Mapping boundaries, scales, with which sounds each feature is more likely to be descriptive, and making sure bogus data is not misinterpreted are all issues that need further investigation before one can move forward with a more sophisticated visual.

It is tempting to begin annotating some of the sound features based on qualities that are easy to recognize by a human yet still hard to describe computationally. For example, describing all the harmonics of a saxophone by an analyzed fundamental frequency and some image resembling the sound of a saxophone (say, shining brass). Yet while this may provide at first a more useful way to distinguish harmonics of one sound from another, it seems that further scientific research will lead to better results in the long run. By comparing statistics of the spectral features in different sounds and how they vary, one can learn how to bring out the differences through visualizations and incorporate these findings into the visual feature representation that is begun here. With time, I believe such a representation will evolve into a powerful way to visualize sound, while at the same time verifying the integrity of the data which produced the features.

Chapter 5

Implementation and Evaluation

The approaches to real-time spectral manipulations presented here have been carried out over the past two years, consistently building on the notion that arbitrary frame indexing will facilitate the experimental exploration of various types of sounds. In order for this to sound good, it is first of all necessary that a low-latency computer is used to operate the software. In my experiences, buffer sizes of 256 samples or less, which corresponds to 5.33 milliseconds at a 48k samplerate, is fast enough to make the instrument *feel* like you are controlling it in real-time. This has been achievable in the Mac OS X operating system, although the instruments presented here are commonly used from a PC with Debian-based Linux distribution.

The implementation here does not attempt to encapsulate all possible methods of control, but instead aims to expose as many as possible for experimentation within a real-time programming environment. This approach is especially fun because while encountering unforeseen problems or effects along the way of implementing various manipulations, one stumbles upon new ones that would have gone unheard if the initial program was written in a more confined environment. If nothing else, the effect of constantly listening to a program as it is being built keeps the engineering portion of a project lively.

5.1 *Trax*: A Pure-Pd Approach

The first, naive attempt I made at designing a real-time instrument used Pd both for synthesis and data visualization, and included a small step sequencer that could be used to automate time indexing (pictured in figure 5.1). While it had its flaws, many ideas and features were incorporated from *Trax* into my next, more sophisticated instrument, presented in the next section. As *Trax* has been used both in performances and compositions for a quite some time now (although now retired), here is a review and reflection to provide insight into design directions taken thereafter.

5.1.1 Overview

Trax [1] is a real-time synthesizer based on sinusoidal modeling an additive synthesis (see 2.1.2), disregarding any residual representation for the sake of simplicity. The implementation is in the form of two externals programmed in C (see 2.2.2) and many ‘patches’ in Pure Data - a graphical system for programming real-time audio software. It basically works as follows:

1. A sound file is analyzed in an external program, such as *CLAM SMSTools* [2], or *Spear* [3], specifically designed for sinusoidal analysis.

2. the resulting analysis data is saved to file in *SDIF* file format and loaded in *Trax* to a memory buffers.

3. data is then visualized using Pure Data’s graphical data structure mechanism (see bottom figure in 5.1).

4. data index and some basic modifications are controlled via the *Trax* main GUI patch (see top figures in 5.1) or using a Wacom Tablet.

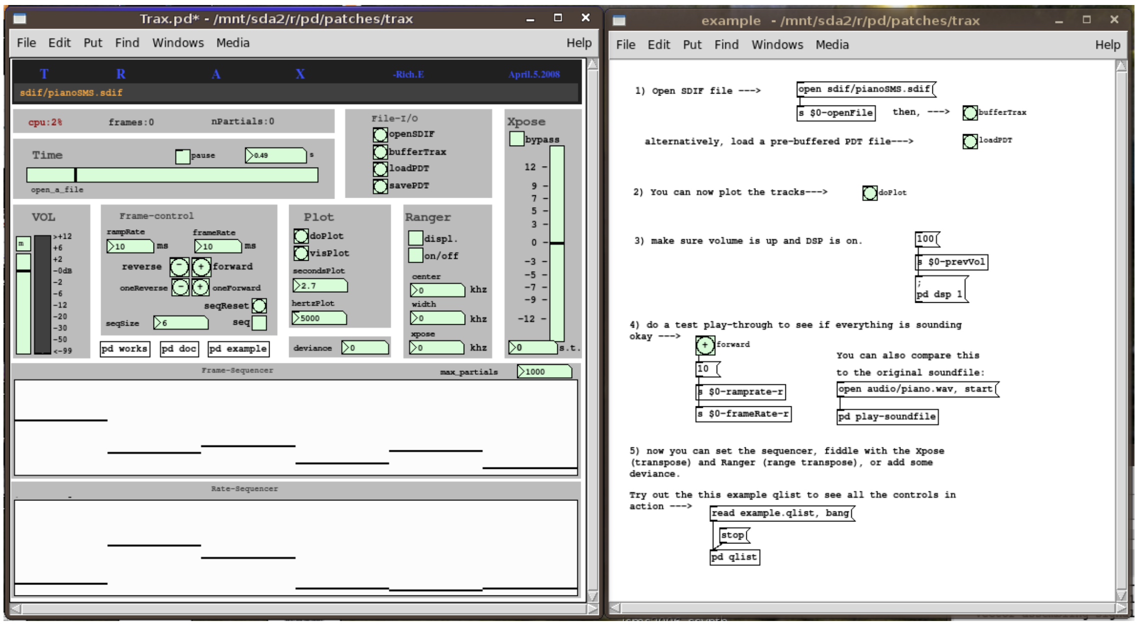
5. resulting partial frequencies and amplitudes are sent to an oscillator bank external that creates a sound waveform, which is then given to the software audio output to be delivered to the speakers.

The data visualization originally grew out of the necessity of knowing whether the analysis was correctly imported and, furthermore, I could correctly access frames and tracks stored within the data structures. Programming with Pd’s visual data structures was new to me at the time of this implementation (see [Barknecht and Wall, 2006] for a better explanation of this unique feature). It immediately seemed like a powerful way to visualize the success or failure of the organization of data held within the structures, which became dense and complicated quite fast in my case. Another enticing feature was the possibility to edit the data in real-time with the mouse or more complicated macros, thereby seeming to immediately offer avenues for manipulation.

5.1.2 Evaluation of *Trax*

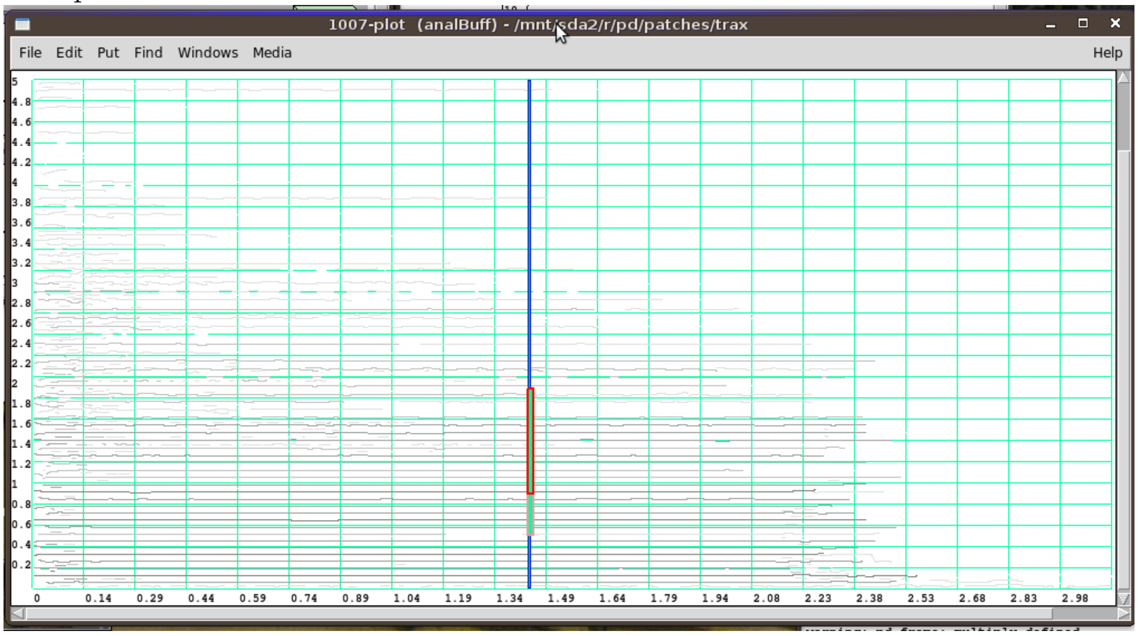
The evaluation here is based on using the instrument within musical scenarios; performing within an experimental improvisation group and personal compositions.

[1] *Trax*: http://mtg.upf.edu/files/personal/trax.tar.gz
[2] *CLAM SMSTools*: http://clam-project.org/
[3] *Spear*: http://www.klingbeil.com/spear/


Trax.pd* - /mnt/sda2/r/pd/patches/trax
File Edit Put Find Windows Media Help
T R A X -Rich.E April.5.2008
sdif/pianoSMS.sdif
cpu:2% frames:0 nPartials:0
Time pause 0.49 s
open_a_file
File-I/O openSDIF bufferTrax loadPDT savePDT
Xpose bypass
VOL
Frame-control rampRate ms frameRate ms reverse forward oneReverse oneForward seqReset seqSize seq
Plot doPlot visPlot secondsPlot hertzPlot
Ranger displ. on/off center khz width khz xpose khz
deviance
pd works pd doc pd example
Frame-Sequencer max_partials
Rate-Sequencer
example - /mnt/sda2/r/pd/patches/trax
File Edit Put Find Windows Media Help

```
1) Open SDIF file --->        open sdif/pianoSMS.sdif
                              s $0-openFile   then, --->   bufferTrax

 alternatively, load a pre-buffered PDT file--->           loadPDT


2) You can now plot the tracks--->       doPlot


3) make sure volume is up and DSP is on.          100
                                                  s $0-prevVol
                                                  ;
                                                  pd dsp 1

4) do a test play-through to see if everything is sounding
okay --->
          forward
                              You can also compare this
          10                  to the original soundfile:
          s $0-ramprate-r     open audio/piano.wav, start
          s $0-frameRate-r    pd play-soundfile

5) now you can set the sequencer, fiddle with the Xpose
(transpose) and Ranger (range transpose), or add some
deviance.

Try out the this example qlist to see all the controls in
action --->
           read example.qlist, bang
             stop
           pd qlist
```

Figure 5.1: (top left) The main graphical interface for *Trax*, showing all the available controls. The bottom two arrays are time-sequencers, which will manipulate the horizontal bar on the top left, representing time index into the analysis file. Other controls are for visualizing the data, loading files, overall frequency transpose, and a range-based frequency transpose. (top right) And example patch for loading a file, plotting it, and playing the instrument via data stored in a text file. (bottom) The data visualization patch in *Trax*. The horizontal axis represents time and the vertical axis represents frequency. The gray lines are time-varying sinusoidal tracks and their color represents the mean magnitude of each track. The line in the middle represents the current frame location, the two boxes on top of it represent a ranged frequency transpose.



There exists a few performances containing my name on Youtube and two compositions that I have placed on the internet[4]. Below are three fundamental criterion that I found valuable for evaluation.

Quality of Sound Creation Using the Interface

First and foremost, the ability to create high quality sounds is the most important aspect of any music instrument. Many things play into the sound quality of a spectra-based synthesizer, such as the method of digital signal processing or analysis parameters. Within this design, the important factors were the method of additive synthesis and how the data was delivered to the synthesizer.

The synthesis implementation was quite slow; it could manage to synthesize a few hundred partials in one frame but only if slowly traversing the frames. Sections with dense partial tracks conflict with the audio loop and occasionally cause the GUI to freeze. This is partly because the implementation of $\boxed{\text{oscbank}\sim}$ used some unoptimized loops based on criterion for how to determine whether a partial is born, evolving, or dead. Yet, the main reason was that control and audio were processed in the same buffer, meaning all control data had to be finished within the time of 64 samples. This system works based on a system of track indexes so as to save data space, yet in retrospect such data conservation seem unnecessary in modern times.

In sound representations with few partials, such as single-note acoustic instrument samples, a good synthesis can be created with flexible time control. These sounds work well in Trax, while other more complicated sounds may distort more often due to audio dropouts and visual clutter. I actually had quite some fun playing with an analyzed section of a Webern chorale; although the resulting sounds were rich and exciting, it sounded nothing like humans singing.

Ability to Control Parameters in Real-Time

The mapping strategy was very similar to what was explained in section 3.3.3; a Wacom Tablet is used to control frame index and frequency transpose via the absolute X and Y axes, respectively. There is also a *ranger*, a transpose for only adjusting the frequencies within a certain range, specified via X/Y tilt and a button on the stylus.

There is excellent ability to manipulate the frame index in real-time with the Wacom Tablet, as the sound data is spread out so that one can traverse 10 or 1,000 frames within the same time. To make sure the synthesizer was not overloaded by such fast gestures, the tablet's parameters were sampled at Pd's block rate.

Changing individual sinusoidal parameters is almost pointless from a musical point of view, as it only creates an 'electronically' sine tone amidst the original

[4]online recordings: http://soundcloud.com/rich-e last visited: August 24, 2009

sound. It did, however, give insight into what we are actually hearing in an additive synthesizer, by all of the sudden making an overtone at 1,000 hertz or higher quite audible. The frame based method for transposing sections of sinusoidal tracks works much better and producing a unique sound effect. Still, its visualization is poor, as there is little correlation between the transpose (moving based on a linear scale) and the visualization (logarithmic scale). It is also quite easy to move the ranger controls to where no significant effect can be heard, or a very significant modulation is created.

Graphical Interface Effectiveness and Efficiency

As emphasized in chapter 4, in order to make any arbitrary sound playable, it is important to be able to visually recognize qualities of a sound based on its representation. For instance, a transient will create a very abrupt sound when it is played back, so it is important to know this ahead of time in order to avoid abrupt or out-of-place musical performance.

The ease of recognizing sound features is poor compared to even a common sonogram. Within the Tk drawing tools, the API that Pure Data uses to display the data, the color of each line cannot change. Consequently, tracks that become suddenly loud do not show these time-varying features.

Another drawback of this system is that it is nearly impossible to implement zoom/pan functionality in real-time. The entire plot has to be redrawn in order to do this, which takes up to a few seconds for some analysis files with dense sinusoidal data. Still, the vertical bar is a good indicator of what sounds are going to be heard, as it nears a section of dense tracks (e.g., many lines).

The display of frequencies is quite cumbersome. It was necessary to incorporate some way of visualizing frequency transpose, which is best performed via the exponential scale explained in 4.3.2. Yet, displaying sinusoidal tracks based on the same scale looks horrible, as the harmonic structure is condensed in higher frequencies. So, I decided to display the tracks based on linear frequencies while creating a MIDI-like grid in the background, thereby completely disconnecting the visualization of transpose and pitch content. There is virtually no way to visualize exactly what pitch you can create without analyzing the output signal with a pitch detector after the sound is already created.

As already mentioned as part of a problem concerning audio quality, graphical drawing computation largely interfered with Pd's audio deliverance. Loading a file into a data struct was quite fast, yet plotting the data caused the audio to halt for up to a few seconds. This makes the system hard to be used in dynamic situations where one wants to load new data on the fly.

5.1.3 Foreseen Possible Improvements

Most importantly, it was clear that the graphical interface needed to be implemented in a different thread, working at its own speed. This allows Pure Data ample computation time in order to avoid audio dropouts. A separate Pd thread could be created in a number of ways; it is commonly achieved by starting two instances of Pd and connecting them over a network, although in newer versions of Pd you can do this with one main instance that spawns another Pd and can pass both audio and control signals. Yet, a more flexible graphical toolkit would allow the data to be drawn more expressively, as well as providing for panning and zooming, so using two Pd threads did not seem like the appropriate way to go. Something based in OpenGL seemed most appropriate.

The synthesis method could use improvements so that it could accurately handle the fluctuations in partial tracks that are born and die. Furthermore, a pure-Pd residual synthesis (using no externals, just Pd's built-in FFT objects) was planned, yet this system seemed to be problematic from the start. It would be difficult to make sure the resulting components could be 'melded' back together (see 2.1.2) and the residual data created with CLAM was difficult to interpret.

In general, both visualizing the data manipulation with a sonogram and time bar and viewing manipulations with simple rectangles was not descriptive of what was being heard. Pitch transposition was very powerful in this system, although impossible to create any sort of sophisticated melodic phrasing. Controlling more complex manipulations seemed outside the capabilities of this instrument.

5.2 *SmsPlayer* Instrument Implementation

SmsPlayer, so-called because of the current lack of a more creative name, is the combined set of Pd patches for sound control and synthesis and python scripts for interactive visualization (see 2.2.3). The main graphical interface is shown in 5.4, showing a feature-based display of the entire sound file on the top and the data currently within the synthesizer on the bottom. This is one of three views, depending on the prominent features within the data, that one would use while trying to create musical manipulations, while there are two more for viewing the data in more detail (further explained in section 5.2.4)

At the point of manipulating the data while watching the OpenGL user interface, all parameters are controlled by either the Wacom Tablet or a MIDI keyboard with rotary knobs and sliders. Before this, at the long and experimental phase of deciding how to map the parameters to these controllers, one is using the Pd interface (the top layer is shown in 5.3). It is possible here to independently control parameters, fine tune HID mapping, and save/load state files of the current mapping scheme and synthesis parameters..

The following sections provide more detail into the different components and decisions taken throughout the implementation. Appendix A provides more sound examples, screenshots, and links to online videos demonstrations.

5.2.1 Creating the SMS Data

SMS data is normally created either with the terminal application `smsAnal` (see 2.2.3) or with a python equivalent script. As of this writing, creating a good spectral representation of more complicated sound with libsms is still a process of guessing the correct parameters. If a sound is harmonic, in a medium range of frequencies, and stable (which does cover many sounds ranging from voice to orchestral instruments), a good result can be obtained with default parameters to the analyzer. For many interesting sounds, one must inspect the sound prior to analysis (ex. a spectrogram is useful) and adjust parameters to such an extent that it is not yet beneficial to analyze the sound in a real-time environment (such as with the analysis external $\boxed{\text{smsanal}}$). As of this writing, it is best to create a personalized script for each sound I wish to use, honing in on the parameters that produce the most accurate results, then load the data into Pd with $\boxed{\text{smsbuf}}$.

5.2.2 The Real-Time Synthesizer

The SMS synthesizer presented here assures that any data frame can be synthesized immediately with just a few parameters. Sophisticated data modifications, such as those mentioned in section 3.1.2, are processed in real-time if they are turned on, although off by default. All parameters are sent with Pd's message system and stored as internal variables, then sampled every time another SMS frame is going to be reconstructed to waveform. Regardless of exactly what value is given to any of the parameters, synthesis will continue while audio computation is running.

In $\boxed{\text{smssynth}\sim}$'s audio loop, there are three main library functions that the necessary work. `sms_interpolateFrames()` first samples the current floating point index within the range of the frames stored in an $\boxed{\text{smsbuf}}$ and creates a new frame with interpolated values. `sms_modify()` then modifies the resulting frame based on a set of flags within a parameter structure, and finally `sms_synthesize()` blindly synthesizes the modified frame into audio samples that will be delivered at Pd's blockrate.

Separate from Pd, the blockrate specified within the synthesis parameters decides the time for linear interpolation from one frame to the next, occurring between any two arbitrarily sampled frames. By default, the window size in $\boxed{\text{smssynth}\sim}$ is 128 samples, set by the limit on FFT sizes within libsms. This can be increased to any power of two by providing an additional argument when creating $\boxed{\text{smssynth}\sim}$, although larger blockrates have yet to show drastic improvements in computation time (which is normally too low to be a concern, at least for monophonic purposes) or synthesis fidelity.

In some cases, it may be desirable to turn off either sinusoidal or residual synthesis. Both can either be completely turned off in real-time or their amplitudes can be adjusted separately. This aids in understanding the quality of each component's representation and localizing features of the sound. For instance, the residual synthesis occasionally sounds like wind, thereby inadvertently making the result sound more synthetic. Or, the deterministic synthesis may be distorting because of bogus modification parameters, which may be slightly masked if the residual synthesis is loud.

The synthesis parameters are programmed directly into libsms so that other applications can take advantage of identical algorithms at a C level. Then, each programming API calls the function to perform the manipulation independent of its own programming paradigm. In Pd, this equates to adding a C method with a symbol attached that can be recognized in the real-time environment, then sending a message to $\boxed{\text{smssynth}\sim}$ containing the symbol and parameter values. Within the C method, checks are conducted to make sure the values are sane, but otherwise everything is handled within libsms.

Although modifications can be seen as part of the process of synthesizing the data, much effort has been made to separate all modifications from the synthesis routines. This aids in refactoring the original code, continuously checking if analysis and re-synthesis produces the same results for test files. With this separation, others may also avoid the libsms modification routines and use their own, while still benefiting from the synthesis engine. Almost all the low to mid level modifications described in chapter 3 have been implemented within the `modify.c` file of libsms.

5.2.3 Control Mapping

Since the conception of *Trax* (see 5.1), a Wacom Tablet has proved an appropriate controller for the constraints of the SMS instrument in this thesis. The model used within this system is the Intous II, 6 by 11 inches (15.2 by 27.9 centimeters). This USB-powered tablet provides all the features that Wacom provides while remaining quite portable; small enough to fit in a backpack while large enough to provide ample space for gestures.

The Wacom data is received in Pd using Hans Steiner's `linuxevent` or `hid` objects, available in both Pd's svn[5] and Pd-extended[6]. As the data regularity of USB can vary depending on other processes in a computer, the control values are

[5]Pd's svn: `http://sourceforge.net/projects/pure-data/`

[6]Pd-extended: `http://puredata.info/downloads`

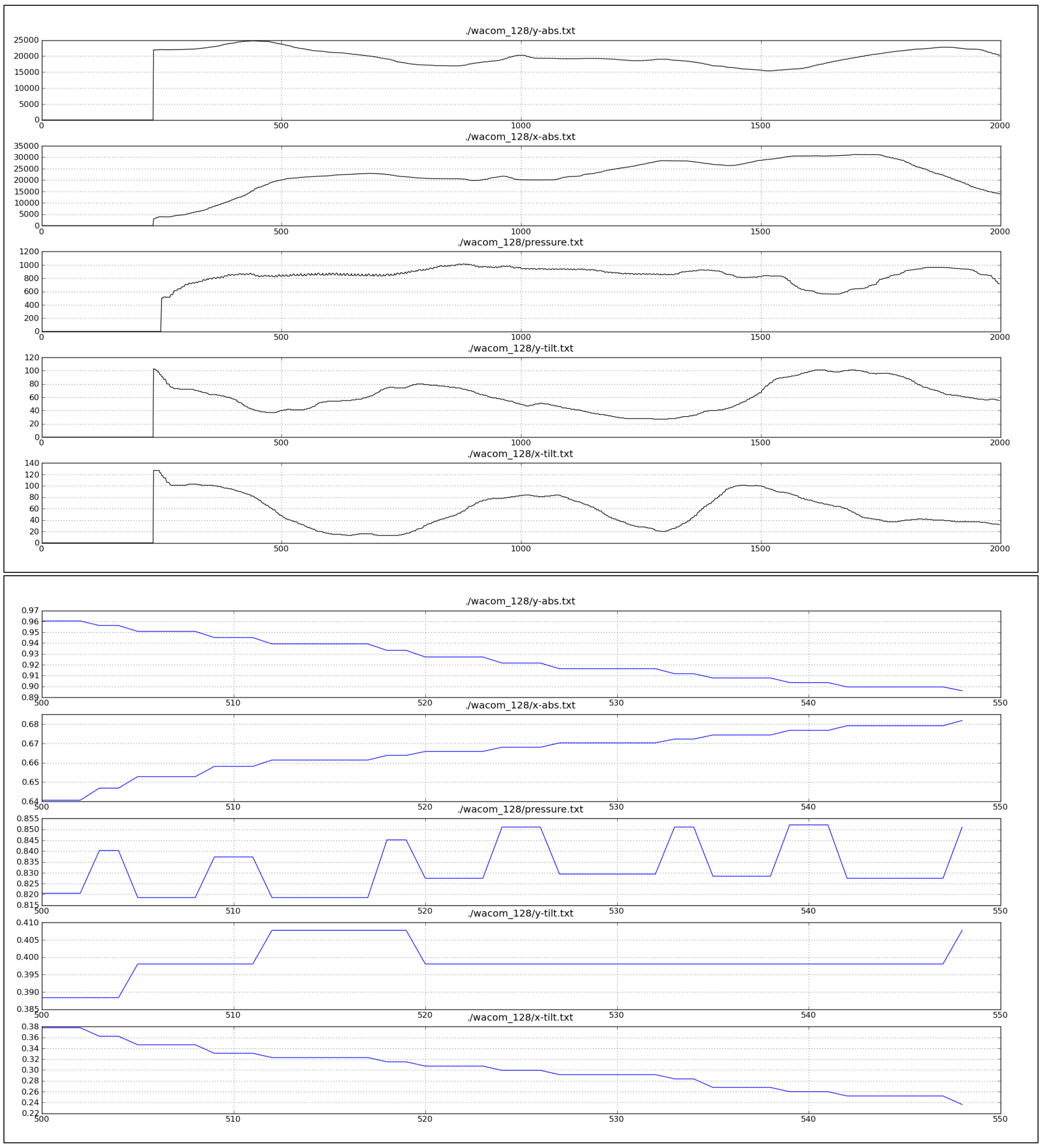


Figure 5.2: The wacom data, sampled in Pd with the $\boxed{\text{bang}\sim}$ object for 2,000 frames. The top figure shows the overall sampling of each of the five coupled controls, while the bottom displays frames 500-549, normalized from zero to one.

used to set synthesis parameters that are only evaluated at the beginning of every audio block. Figure 5.2 displays the data of the five continous and coupled controls offered by this tablet version, sampled every synthesis buffer (128 samples). The top figure shows the data values sampled for 2,000 frames (5.33 seconds). The bottom figure shows 50 normalized frames (about 14 milliseconds) of the same data, revealing some interesting information about the capacity of this device as an instrument. Th pressure shows some small oscillations about .02 in magnitude. As the tilt axes, only ranging from 0-128, seem to be stable despite their low fidelity. In this sampling, the Y-tilt is near stationary while the X-tilt is constantly descending It is difficult to deduce any information about the accuracy of the absolute axes from their display here, yet they have always had a natural mapping (explained below) and controlling these parameters has been responsive and accurate.

The X absolute position naturally maps to a frame index into the currently loaded data file, spanning the entire length of the file. For files up to about ten seconds in length, this gives good control over time-changing features.

The Y absolute position is mapped to frequency transpose, allowing an octave in each direction and centered around zero transpose. On the current tablet in use, this means moving the stylus tip a quarter inch, or 0.6 centimeters, will transpose the sound by one semitone. Although this mapping is adjustable on the fly, keeping a consistent mapping allows one to improve their skill in melodic improvisation. Furthermore, transposing many sounds more than an octave does not normally sound good, while an two octaves provides ample room for melodic creation.

The stylus pressure is most naturally mapped to an overall amplitude gain in decibels. One problem in early usage of the tablet was that when ending a musical phrase, the last recoded gesture would occasionally be a positive and audible value, while no more parameters are changing. In order to make sure the instrument only creates sound when the stylus is touch the tablet surface, the tip is also mapped to amplitude gain as a multiplier of one if it is touching and zero if not touching. The oscillation of pressure values seen when sampling the data probably will not create a large difference when mapped to gain, but instead may help in providing constant variability.

The above mappings are fixed, as they feel natural while controlling this instrument. The other parameters that need controlling, such as stochastic gain, envelope modifications, and harmonic frequency / amplitude modifications, remain experimental in controlling. It is desirable to find settings that distribute some or many of these parameters among the X and Y tilt control, searching for a medium between ease of expression and variability. In order to streamline the experimentation of these mappings, a system of 'automapping' has been created in the form of Pd abstractions to permit all parameters the possibility of receiving control data from the wacom parameters, a variable step sequencer, and a midi keyboard with sliders. These are kept on the main Pd patch (see figure 5.3, ultimately out of site during performance. Experimentation with mapping these parameters is explained in section 5.3.3.

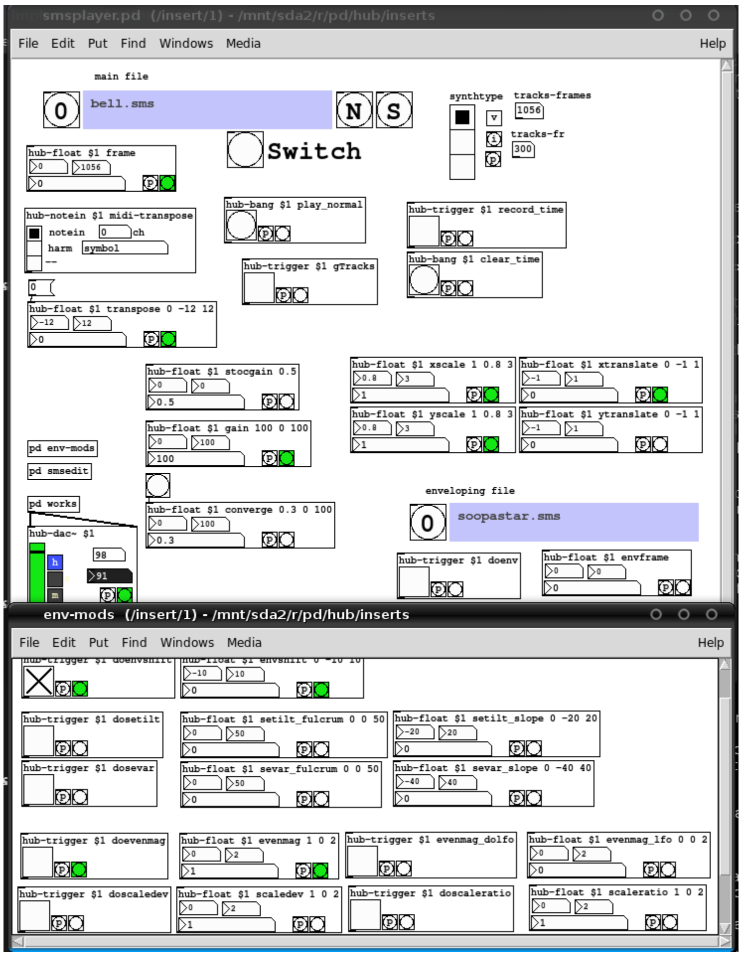

smsplayer.pd (/insert/1) - /mnt/sda2/r/pd/hub/inserts
File Edit Put Find Windows Media Help
main file
bell.sms
N
S
Switch
synthtype
tracks-frames
tracks-fr
hub-float $1 frame
hub-notein $1 midi-transpose
notein
ch
harm
symbol
hub-bang $1 play_normal
hub-trigger $1 record_time
hub-trigger $1 gTracks
hub-bang $1 clear_time
hub-float $1 transpose 0 -12 12
hub-float $1 stocgain 0.5
hub-float $1 xscale 1 0.8 3
hub-float $1 xtranslate 0 -1 1
hub-float $1 yscale 1 0.8 3
hub-float $1 ytranslate 0 -1 1
pd env-mods
pd smsedit
hub-float $1 gain 100 0 100
pd works
hub-float $1 converge 0.3 0 100
enveloping file
soopastar.sms
hub-dac~ $1
hub-trigger $1 doenv
hub-float $1 envframe
env-mods (/insert/1) - /mnt/sda2/r/pd/hub/inserts
File Edit Put Find Windows Media Help
hub-trigger $1 doenvshift
hub-float $1 envshift 0 -10 10
hub-trigger $1 dosetilt
hub-float $1 setilt_fulcrum 0 0 50
hub-float $1 setilt_slope 0 -20 20
hub-trigger $1 dosevar
hub-float $1 sevar_fulcrum 0 0 50
hub-float $1 sevar_slope 0 -40 40
hub-trigger $1 doevenmag
hub-float $1 evenmag 1 0 2
hub-trigger $1 evenmag_dolfo
hub-float $1 evenmag_lfo 0 0 2
hub-trigger $1 doscaledev
hub-float $1 scaledev 1 0 2
hub-trigger $1 doscaleratio
hub-float $1 scaleratio 1 0 2

Figure 5.3: Main Pd patch and subpatch for assigning and manipulating all parameters. The `hub-trigger` and `hub-float` abstractions have automap functionality for the wacom tablet, midi keyboard, and a variable step-sequencer. The ranges of values, linear or logarithmic, are set with initial arguments or with the top two number boxes in each abstraction. The big number box reveals each parameter's current value. If the rightmost 'bang' (the circles) is green, the parameter has been mapped to something, which can be reported with the leftmost bang.

5.2.4 The Graphical Interface: *gSmsPlayer*

The following section describes the graphical interface that is used as an aid in understanding the characteristics of the currently loaded sound and visual feedback of the real-time manipulations. The implementation, called *gSmsPlayer*, is programmed using the *pysms* (see 2.2.3), *Pygame*[7], *PyOpenGL*[8], and *NumPy* python modules. One large benifit of programming the interface in python is, other having extensive libraries for any type of graphics manipulations desired within a easy-to-read scripting language, is that video is processed in a seperate thread from Pd, thereby further guaranteeing real-time audio synthesis.

Figure 5.4 shows the main window, including the five optional displays. The bottommost display is the frame-based display of the spectral data that corresponds to what is currently processed by the synthesizer. It reflects all synthesis parameters; switching off the synthesis of either component will make its visual representation disappear, or changing their respective gains will make them darker or brighter.

The top portion of the larger figure shows the entire sound file that is loaded. The current synthesis frame is visualized by a vertical line that is gray when gain is zero (ex., the stylus pen is not touching the tablet) and increasingly bright as gain increases. This is also reflected in the frame-based display by a frame number in the top right. The display here is the current visualization used based on features. The original version is shown in the small, top-left figure, which is different only in one aspect: the first attempt displayed fundamental frequency on top of the pianoroll (see 4.5), then stacked overall amplitude of the sinusoidal on top of this and overall amplitude of residual data on top of that. While the display seemed descriptive of the components relative to each other, the display is lop-sided, especially when transposing the sound (which shifts the overall image so that the fundamental frequency still corresponds to the correct MIDI value). To help fix this, the second attempt uses fundamental frequency as a divider between sinusoidal and residual amplitudes, as displayed in the largest section of 5.4. In both versions, the spectral centroids of the respective components are mapped to color using a logarithmic scale. The harmonic deviation is mapped to the color of the pitch, although it fails at revealing any information because erroneous data seems to offset any subtle inharmonic changes.

The top-right display is used when the sound is inharmonic, in which the first sinusoidal track would give no correct information for pitch content of the sound. Instead, the sinusoidal amplitude begins in the middle of the display and extends upwards, while the residual extends downwards. The result is very similar to a time-domain waveform view in logarithm display, except non-symmetrical. The pianoroll is removed because the it does not help in determining the pitch of a

[7]PyGame: http://www.pygame.org

[8]PyOpenGL: http://pyopengl.sourceforge.net/

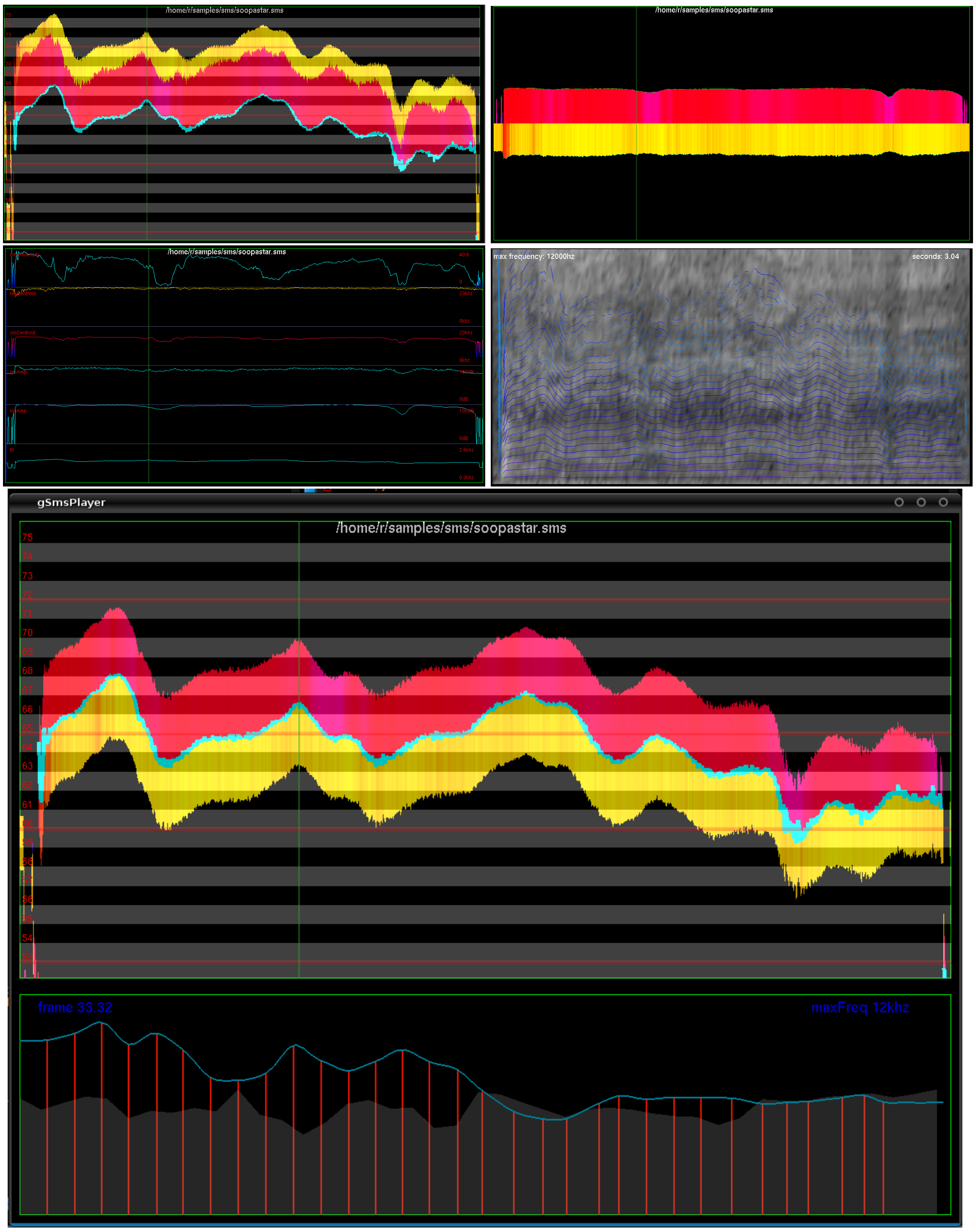


Figure 5.4: Screenshots of *gSmsPlayer* loaded with the same singing voice sample used throughout this thesis. The top four images are optional views of the overall data, displayed in the larger section. The bottom display always reflects the current, real-time synthesis data.

sound, whether transposed or not. The sound can still be manipulated melodically, although judgment is left up to the ear in this case[9]

The next display in 5.4, the lower-left of the small figures, shows the extracted features separated with there own independent axis. This suggests where information is erroneous (such as the beginning of this sound, or all of the harmonic deviance analysis) and helps in creating knew data visualizations. The last small display, bottom-right, shows the original display that resembles a sonogram. Here, residual magnitude bins are represented with a gray scale while sinusoidal tracks are represented on top with magnitude-varying color.

At the time of this writing, the overall display is unchanging accept when transposed, where-in the feature-based image is shifted to match the corresponding pianoroll. As such, all five displays are loaded as OpenGL display lists, which are then processed in real-time to create the visuals with minimal CPU usage.

5.2.5 Linking Pd and Python

Pd and Python communicate over the UDP protocol using high-level OSC Wright and Freed [1997] modules: In Pd, messages are sent using the $\boxed{\text{sendOSC}}$ object that is available in both Pd's svn and Pd-extended. In python, the messages are received using a pure-python OSC implementation[10]. As of this writing, both applications are run best in their own terminal, although it is certainly possible to run Pd from within the python script or vice-versa (calling python from Pd). Either step would decrease start-up time and make the instrument feel like a whole.

As they run in separate threads, both Pd and Python need their own copy of the sound data in memory, although this amount of memory consumption is not an issue in today's computer systems. A large benefit of placing all modification code within libsms is that both ends, the audio engine and graphical interface, will compute the same modifications to the data by using the same functions and modification structure. In this manner, whenever a message is sent to $\boxed{\text{smssynth}\sim}$ it is also sent to $\boxed{\text{sendOSC}}$ with the message "/smsplayer/" prefixed. The message is received in python by binding a function to each message.

Network bandwidth has not yet been an issue in this implementation. Still, as the graphics frame rate is much slower that the rate of Pd, one can optimize the network bandwidth by down-sampling the OSC messages sent by a factor of eight.

[9]Following any sort of traditional theories when manipulating an inharmonic sound as harmonic will not help anyway. One might as well make up their own in this case.

[10]Python SimpleOSC module: `www.ixi-software.net/content/body_backyard_python.html`

5.3 Evaluation

In order to correctly evaluate the instrument presented in this thesis, experimentation will need to be conducted within musical applications for quite some time. This type of evaluation is quite subjective, wherein one performs with the instrument for an extended time and imagines what type of new effects can be introduced to create interesting sounds. In the evaluation presented here, I will list the strengths and weaknesses of the system of the instrument, and also evaluate the preliminary set of implemented effects.

5.3.1 Interactive Control of SMS Data

The instrument presented here provides excellent control of pitch, loudness, and timbre through the use of arbitrary frame indexing and frame-based modifications. The possible sounds are flexible and rich, creating a powerful interactive instrument from minimal sound data. Since partial tracks are interpolated on the fly from one frame to the next (with appropriate phases), one can easily improvise for an arbitrary length of time with only a model that contains a few interesting features (see figure 3.4).

When controlled with the Wacom Tablet, data is continuously sent the synthesizer that varies enough to consistently sound 'alive'. Articulations are straightforward and natural; a very realistic vibrato is produced by moving the stylus in a circular motion around a desired pitch. Sophisticated melodies can be created with more ease due to the pianoroll visual aid.

Without using display lists, *PyOpenGL* is too slow to interactively modify partial tracks, and as such it is difficult to improve interactivity of the sound representations presented in this thesis. If the visualization is computed in real-time, it is possible to visualize permanent data modifications, a feature desirable of an interface for music composition. Yet, in this implementation, computing the sonogram visualization requires about 50 percent CPU usage while computing the feature-based representation requires about 50-100 percent, depending on the frame resolution. Both methods could most surely be optimized, yet the largest optimization would be a port to C++. As the PyGame code is a direct wrapper around SDL[11], the structure of the C++ code would be nearly identical as the prototype in python.

5.3.2 Synthesis Improvements

The quality of synthesis is in general quite good if the data it is synthesizing is accurate. However, many problems are introduced by sounds that are harder to analyze (and would probably create good music material).

[11]SDL: `http://www.libsdl.org/`

One of the largest problems discovered during this thesis is that sporadic partials, ones that are born and die repeatedly, cause problems when using a spectral envelope for modifications. The problem arises mainly when trying to interpolate between frames, which allows for one to manipulate a much smaller section of the sound with more expression. However, the spectral envelope interpolation (pre-computed during the analysis phase) does not take into account partial track interpolation, causing partials that would otherwise ramp into an audible amplitude to start with the amplitude of the spectral envelope. An obvious solution, though undesirable, is to compute the envelope in real-time, before any modifications are performed to the data. Another solution is to improve the data through better post-processing techniques, like the methods mentioned in section 2.1.2.

5.3.3 Manipulation Experiments

The goal of this thesis was to define and create a real-time synthesizer that streamlined the experimentation of spectral modifications, allowing the user to test the effectiveness of effects interactively with both their ears and eyes. In the end, the evaluation most effective will be undergone throughout the process of implementing and testing new modifications, and in attempting combine them to create higher level manipulations. Within the time span of this thesis, there was not enough time to warrant this full evaluation, although preliminary experimentation is described here.

Bar far the largest gain from this system is an improved ability to transpose any harmonic data with pitch accuracy. This is due to the feature-based representation, in which the fundamental frequency is mapped to a visual pianoroll in the MIDI scale. Pitch control is a fundamental manipulation that showed promise when controlled with the continuous Wacom Stylus, yet difficult to master without this visual aid. Early on, pitch was sometimes delegated to a MIDI keyboard in order to stay in tune, although it was clear that much of the expression that the stylus contained when simultaneously controlling pitch and dynamics was lost. With the higher-level display of pitch and only a short time practicing melodies within a given scale were achievable with decent accuracy and rhythmic control. After more practice, it seems highly likely that melodies will be controllable with more expression by modifying the data and still maintaining precise control of the pitch.

The modification changing the amplitude of all the even harmonics is most effective in real-time. It easily allows the sound to become more or less 'hallow', while otherwise maintaining all timbrel characteristics. One experiment has been in controlling this parameter with a *Low Frequency Oscillator (LFO)*, which modulates the amplitude of even harmonics with variable speed and depth. This and similar mechanisms (such as a [psuedo]randomizer) are quick and easy to build in Pd with pre-designed abstracts. The first attempt turned out less effective than desired;

if the modulation is two fast or strong, it sounds like one copy of the original is synthesized an octave higher in frequency.

Concerning the method of exposing some envelope modifications, such as tilt or variation, the current implementation is quite tedious. For every type of contortion, such tilting or shifting, the following is necessary:

- a function in libsms is added to libsms
- additional members are added to the modify structure handed to the main modify function, for turning on/off the modification and controlling it
- functions are added to $\boxed{\text{smssynth}\sim}$ source code to receive the structure's parameters
- message commands are added in a Pd patch to send the values in real-time to both $\boxed{\text{smssynth}\sim}$ and over the network to python
- functions are added in python for dealing with the parameters and a binding is made for this function

As the number of possible ways to modify an envelope are quite limitless, this begins to be a repetitive and inefficient method of adding new modifications when the idea arises. An alternative could be to use a Pd array as a filtering function and then interpolate the original with this[12]. Then, one could easily implement various types of 'filtering envelopes' directly in Pd without the need for compilation. Yet this method is more complicated in making sure that modifications are introduced smoothly in time, an important requirement for the real-time manipulations suggested here.

On the other hand, the ability to shift the envelope while retaining its shape is a powerful effect for both sinusoidal and residual components. The sound is noticeably disconnected from the original when doing this, yet not in an abrupt or unpleasant manner. By linearly interpolating frequency bins of the spectral envelope, it is possible to introduce the effect in a more subtle manner.

Sadly, reducing the harmonicity of a sound with the rudimentary methods shown in 3.1 normally result in something less pleasant than the original, explaining why acoustic instrument designers strive to create their instruments as harmonic as possible. Manipulations involving a modification to harmonic locations will probably need to rely more heavily on acoustic instrument analysis and physical modeling techniques in order to reposition harmonic locations without reducing the quality

[12] $\boxed{\text{smssynth}\sim}$ can already use spectral envelopes from a secondary file in a simple, real-time method of 'morphing' (see 3.1.3), although this concept of manipulation is outside the scope of this thesis

of the data. It did seem, however, that quickly adjusting the inharmonic parameters, either the deviance or ratio, from very extreme back to the original at a fast rate, produced the pleasant effect of an electronic sound converging into something meaningful. Although this was not the original manipulation intended, it may be an viable tool for expressing some musical ideas, especially in combination with other effects.

Chapter 6

Conclusions and Future Prospects

Designing experimental manipulations of spectral data is both rewarding and frustrating. Sometimes, one can create sounds that are new yet familiar at the same time, one can apply new ideas to existing sounds and thereby explore music creation with a new avenue. Other times, the resulting sound is hideous and abrupt and one wishes only to resort back to the timbres of acoustic instruments that we have come to love.

The interface presented in this thesis shows great potential in the ability to search and explore new sounds in the form of spectral data. The next step is to use the instrument for an extended time, to become familiar with the available manipulations and learn how to combine their effect in order to improve the dynamics of the instrument.

Many improvements can be made to the feature-based display of SMS data in order for the image to make a closer correspondence to what one hears when traversing that section of the sound. These improvements will also require experimentation in both visual programming and using the instrument.

Many other directions can be taken from here as well, starting with the tools created in order to efficiently synthesize spectral data in real-time. An interface for real-time analysis of sounds, using visual aids and $\boxed{\text{smsanal}}$ could lead to new possibilities in live performance with SMS techniques.

A Wacom Tablet provided for natural control of the frame-based SMS data, as it allows one to easily navigate the time axis in an intuitive way. Yet, as the system still suggests the use of a visual aid in order to understand what sounds will be created beforehand, a graphics tablet could further couple the data, sound, and visualization. As of this writing, graphics tables do not offer control mechanisms comparable to the tablet currently used, such as precise X/Y location, pressure, and tilt. Until then, separating the controller and visual is necessary.

The sound representations introduced here for interactive navigation of spectral data could also make great use in music composition, improving the visualization of computer-aided composition. The visuals are both informative to the viewer and computer, creating a link between the two paradigms that has become separated in electronic music composition. Compositional prospects of the techniques introduced in this thesis would also improve if methods were created for visualizing permanent data editing. As of this writing, doing so forces a complete redraw of the OpenGL display lists.

The ability to edit the SMS data in real-time would be a valuable tool for both creating new sounds and improving the analysis data in general. Real-Time editing would further make the system presented here well-fit for composition, as modifications can be organized in time to create a larger work. In retrospect, it is necessary to design an interface for efficient exploration in order to know what is possible. Thereafter, interesting avenues are easier to recognize and pursue.

Appendix A

Sound Examples

This section displays some example analysis data and visual representations used by the instrument presented within this thesis. Information about the sound, its analysis, and screenshots of the graphical interface are provided here, while the original/synthesis sound files and resulting SMS files are online at:
http://mtg.upf.edu/people/eakin?p=Examples

This web address also contains links to video examples of instrument manipulations, along with information regarding choices and retrospect. While the demonstrations are not very musical in themselves, they present the instrument at its earliest stage of usage, when it is still the most experimental.

In all but the first example in this appendix (the *soopastar* sample used throughout this work), there is a main screenshot that shows the currently synthesized frame data on bottom and the primary view of the entire sound (based on spectral features). The other images below are alternative views of the entire sound as explained in section 5.2.4.

A.1 Singing Voice

Throughout this thesis I have used the same singing voice sample almost exclusively, which is also the sample file used within the example scripts included in libsms. It originates from the *Freesound Project*, with the following address:
http://www.freesound.org/download/4228/4228_NoiseCollector_soopastar11.wav

I have renamed it to 'soopastar' for the use within this context, but it is otherwise unaltered. Images of the visual interface when using this file are in 5.4. I particularly like this sample because it has many fluctuations in most of the spectral features that I use within this thesis, while at the same time it is a very ordinary sample. There is a video example online in which I experiment with shifting, tilting, and varying the sinusoidal spectral envelope while performing with the sample.

SMS header information:

```
smsHeader:
    nFrames         : 912
    iFrameRate       : 300
    nTracks    : 60
    nStochasticCoeff : 128
    iFormat          : harmonic
    iStochasticType  : approx
    nEnvCoeff : 128
    iMaxFreq : 12000
    iEnvType  : fbins
    iSamplingRate   : 44100

text_string: >
    created by smsAnal with parameters: format 0, soundType 0,
    analysisDirection 0, windowSize 4.00, windowType 2,
    frameRate 300, highestFreq 12000.00, minPeakMag 0.00, refHarmonic 1,
    minRefHarmMag 30.00, refHarmMagDiffFromMax 30.00, defaultFund 100.00,
    lowestFund 50.00, highestFund 1000.00, nGuides 100, nTracks 60,
    freqDeviation 0.45, peakContToGuide 0.40, fundContToGuide 0.50,
    cleanTracks 1, iMinTrackLength 40,iMaxSleepingTime 40, stochasticType 1
```

A.2 Ocarina Note

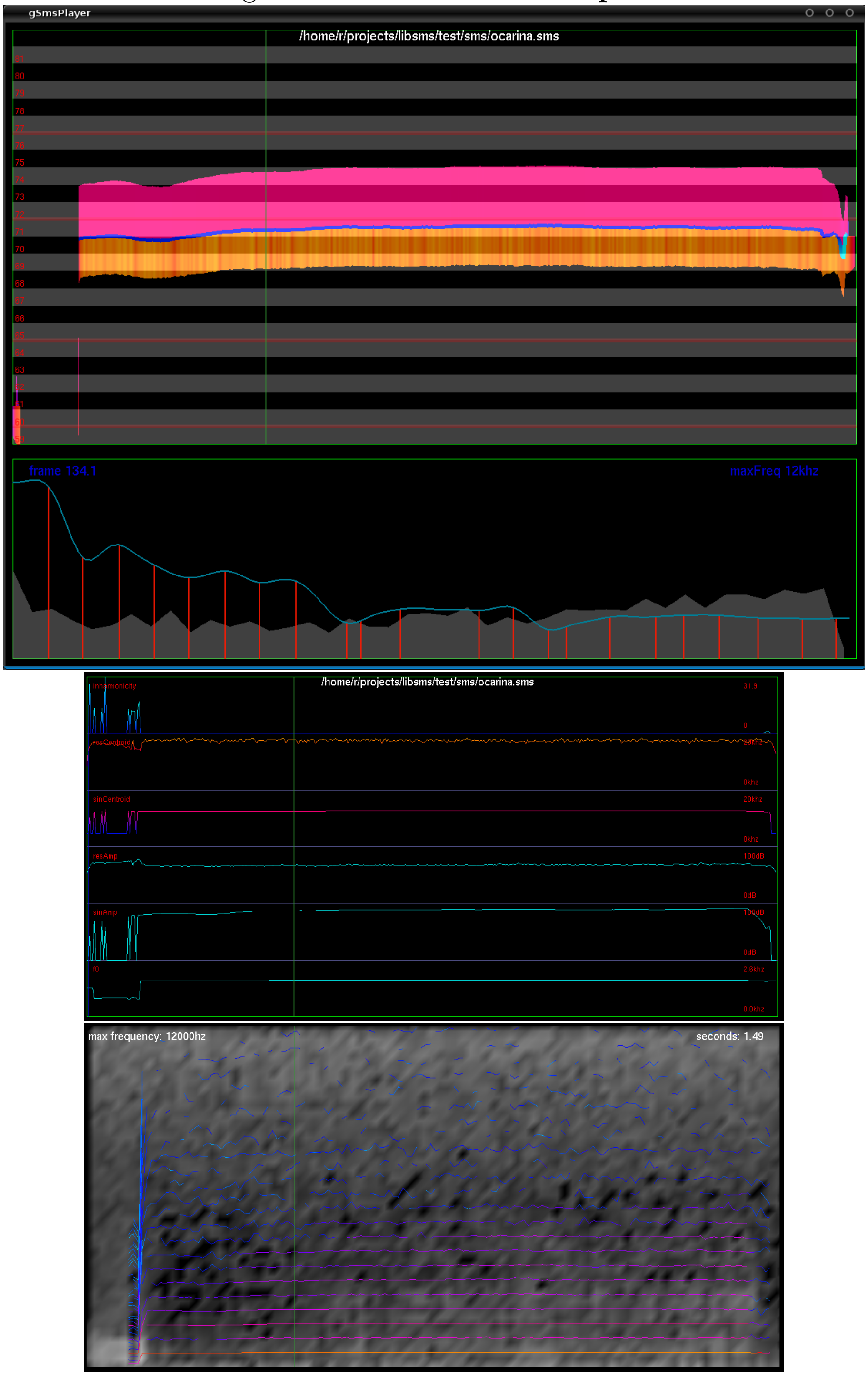
An ocarina note with slight fluctuation in amplitude and pitch, lasting 1.5 seconds. There was no fundamental frequency located in the beginning of the file, an error during analysis. This section is only softer than the data following immediately afterward and is unnecessary to perform with the data as an instrument.

Notice the residual spectral centroid (bottom of the main figure) is varying from orange to red consistently while the residual amplitude stays the same. This may be an important characteristic of this type of sound.

SMS header information:

```
smsHeader:
    nFrames         : 447
    iFrameRate       : 300
    nTracks    : 60
    nStochasticCoeff : 128
    iFormat          : harmonic
    iStochasticType  : approx
    nEnvCoeff : 128
    iMaxFreq : 12000
    iEnvType  : fbins
    iSamplingRate   : 44100

text_string: >
    created by smsAnal with parameters: format 0, soundType 0,
    windowSize 3.50, windowType 2, frameRate 300, highestFreq 12000.00,
    minPeakMag 0.00, refHarmonic 1, minRefHarmMag 30.00,
    refHarmMagDiffFromMax 30., defaultFund 100.00, lowestFund 50.00,
    highestFund 1000.00, nGuides 100, nTracks 60, freqDeviation 0.45,
    peakContToGuide 0.40, fundContToGuide 0.5, cleanTracks 0,
    iMinTrackLength 40,iMaxSleepingTime 40, stochasticType 1
```

Figure A.1: **Ocarina Example**



A.3 Saxophone Melody

This saxophone sample originates from the *Freesound Project* with the following web address:
http://www.freesound.org/download/46798/46798_uauaua_mec7.wav

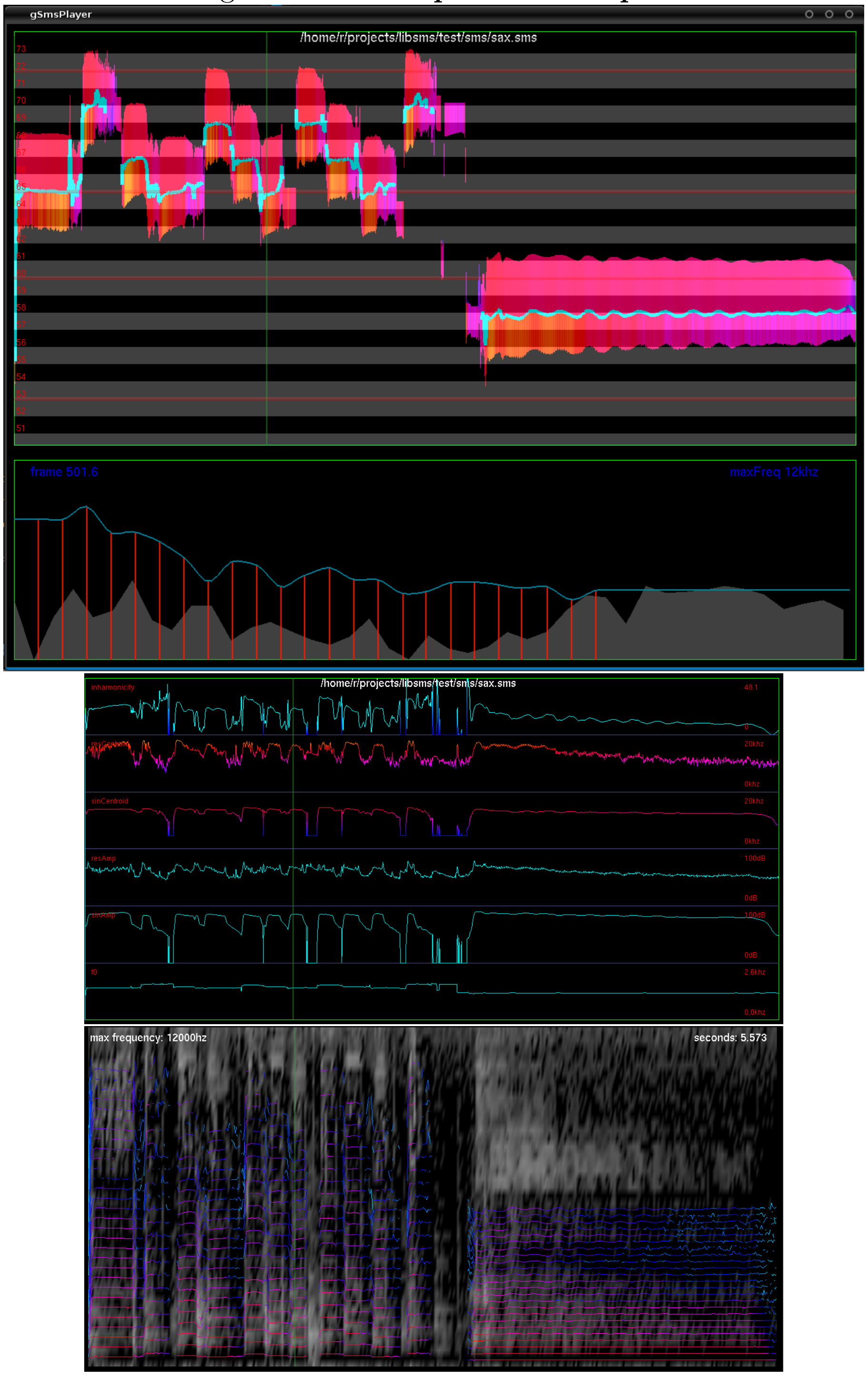
It consists of stoccato notes within a major scale followed by a long vibrato note, as depicted in the main view that contains a pitch mapping to MIDI pianoroll. The spectral centroid of the saxophone is quite high, close to that of the residual.

There is a video online that shows an attempt at modifying harmonic locations by adjusting the ratio to the fundamental while also modifying the even harmonic amplitudes to make the sound more ‘hollow’.

SMS header information:

```
smsHeader:
    nFrames         : 1672
    iFrameRate       : 300
    nTracks    : 60
    nStochasticCoeff : 128
    iFormat          : harmonic
    iStochasticType  : approx
    nEnvCoeff : 128
    iMaxFreq : 12000
    iEnvType  : fbins
    iSamplingRate   : 22050

text_string: >
    created by smsAnal with parameters: format 0, soundType 0,
    analysisDirection 0, windowSize 3.50, windowType 2,
    frameRate 300, highestFreq 12000.00, minPeakMag 0.00,
    refHarmonic 1, minRefHarmMag 30.00, refHarmMagDiffFromMax 30.00,
    defaultFund 100.00, lowestFund 50.00, highestFund 1000.00,
    nGuides 100, nTracks 60, freqDeviation 0.45, peakContToGuide 0.40,
    fundContToGuide 0.50, cleanTracks 0, iMinTrackLength 40,
    iMaxSleepingTime 40, stochasticType 1
```

Figure A.2: **Saxophone Example**



A.4 Shakuhachi

The shakuhachi sound example is a section of the file used in the ICMC 2000 Analysis/Synthesis Comparison section [Wright et al., 2001]. I received the sample from the following website:
http://archive.cnmat.berkeley.edu/SDIF/ICMC2000/sounds.html

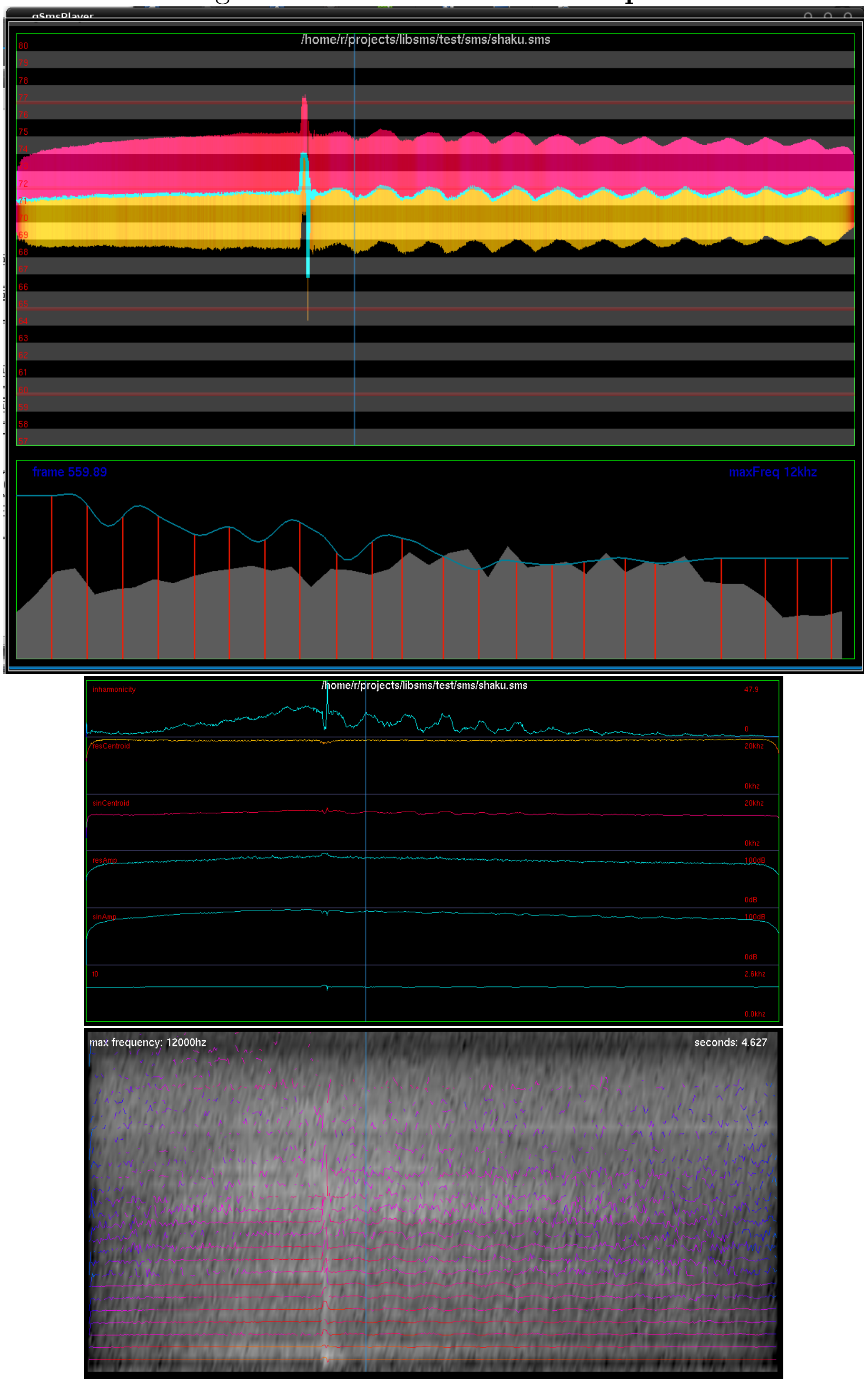
The section I chose to use contains first about two seconds of a subtle glissando from B-flat to C, then short fluctuation characteristic to the shakuhachi, followed by a vibrato on C for two and a half more seconds. Note that these sections are much easier to distinguish in the feature-based representation rather than the sonogram plot.e

The online example video using this sound is an attempt at playing a folk song while using the fluctuation in the middle of the file for accentuation.

SMS header information:

```
smsHeader:
    nFrames         : 1388
    iFrameRate       : 300
    nTracks     : 60
    nStochasticCoeff : 128
    iFormat          : harmonic
    iStochasticType  : approx
    nEnvCoeff : 128
    iMaxFreq : 12000
    iEnvType  : fbins
    iSamplingRate   : 44100

text_string: >
    created by smsAnal with parameters: format 0, soundType 0,
    analysisDirection 0, windowSize 3.50, windowType 2,
    frameRate 300, highestFreq 12000.00, minPeakMag 0.00,
    refHarmonic 1, minRefHarmMag 30.00, refHarmMagDiffFromMax 30.0,
    defaultFund 100.00, lowestFund 50.00, highestFund 1000.00,
    nGuides 100, nTracks 60, freqDeviation 0.45, peakContToGuide 0.40,
    fundContToGuide 0.50, cleanTracks 0, iMinTrackLength 40,
    iMaxSleepingTime 40, stochasticType 1
```

Figure A.3: **Shakuhachi Example**



A.5 Bell

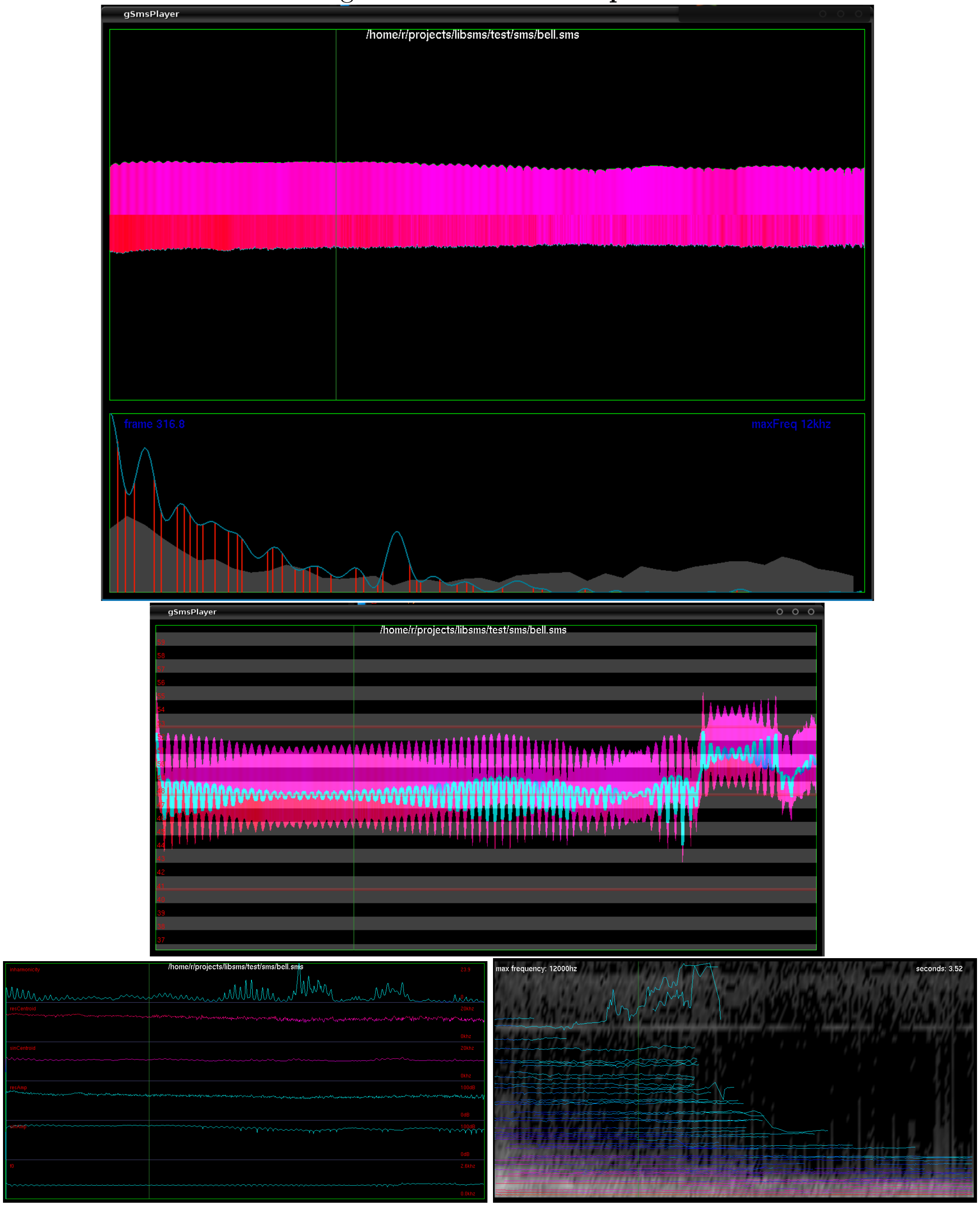
This bell sample is included with Pd as an example for additive analysis and synthesis within the patch "doc/4.data.structures/14.partialtracer.pd". Here it is an example of using an inharmonic sound within this system for interactive navigation and manipulation. Note that the feature representation which makes use of the first harmonic as fundamental frequency is not help for describing the sound. As such, either the inharmonic display (in the main window here) or the sonogram can be used.

There are video examples of using the bell sound as a melodic instrument online; one using a MIDI keyboard for discrete pitch control, and the other using the tablet stylus as in other examples.

SMS header information:

```
smsHeader:
    nFrames         : 1056
    iFrameRate       : 300
    nTracks    : 60
    nStochasticCoeff : 128
    iFormat          : inharmonic
    iStochasticType  : approx
    nEnvCoeff : 256
    iMaxFreq : 12000
    iEnvType  : fbins
    iSamplingRate   : 44100

text_string: >
    created by smsAnal with parameters: format 1, soundType 0,
    analysisDirection 0, windowSize 3.50, windowType 2, frameRate 300,
    highestFreq 12000.00, minPeakMag 0.00, refHarmonic 1,
    minRefHarmMag 30.00, refHarmMagDiffFromMax 30.00, defaultFund 100.00,
    lowestFund 50.00, highestFund 1000.00, nGuides 100, nTracks 60,
    freqDeviation 0.45, peakContToGuide 0.40, fundContToGuide 0.50,
    cleanTracks 1, iMinTrackLength 40,iMaxSleepingTime 40,
    stochasticType 1
```

Figure A.4: **Bell Example**


gSmsPlayer
/home/r/projects/libsms/test/sms/bell.sms
frame 316.8
maxFreq 12khz
gSmsPlayer
/home/r/projects/libsms/test/sms/bell.sms
/home/r/projects/libsms/test/sms/bell.sms
max frequency: 12000hz
seconds: 3.52

List of Figures

Bibliography

J. Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 25(3): 235–238, 1977.

M. Alonso and K Hansen. More DJ techniques on the reactable. 2008.

X. Amatriain and P. Arumi. Developing cross-platform audio and music applications with the clam framework. In *Proceedings of International Computer Music Conference*, pages 403–410, Barcelona, Spain, 2005.

X. Amatriain, J. Bonada, and X. Serra. Spectral processing. In *DAFX: Digital Audio Effects*, pages 373–438. John Wiley & Sons Publishers, 2002.

X. Amatriain, J. Bonada, À Loscos, J. L. Arcos, and V. Verfaille. Content-based transformations. *Journal of New Music Research*, 32(1):95–114, 2003.

T. H. Andersen. In the mixxx: novel digital DJ interfaces. In *Conference on Human Factors in Computing Systems*, pages 1136–1137. ACM New York, NY, USA, 2005.

F. Barknecht and N. Wall. 128 is not Enough-Data structures in pure data. In *LAC2006 Proceedings*, page 61, 2006.

J. M.J Beauchamp. Fundamental frequency estimation of musical signals using a two-way mismatch procedure. *Journal of the Acoustical Society of America*, 1993.

David M. Beazley. SWIG: an easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, pages 15–15, Monterey, California, 1996. USENIX Association. URL http://www.swig.org/.

B. M Bécares. SMS3d: an application for the visualization of SMS data. In *Proceedings of COST*, 1998.

J. Bresson and C. Agon. SDIF sound description data representation and manipulation in computer assisted composition. In *Proceedings ICMC*, page 520–527, 2004.

P. Cano. Fundamental frequency estimation in the SMS analysis. *Proceedings of COST G*, 6, 1998.

O. Cappe and E. Moulines. Regularization techniques for discrete cepstrum estimation. *Signal Processing Letters, IEEE*, 3(4):100–102, 1996. ISSN 1070-9908. doi: 10.1109/97.489060.

A. de Cheveigne and H. Kawahara. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111:1917–1930, 2002. URL http://www.ircam.fr/pcm/cheveign/ps/yin.pdf.

Shlomo Dubnov and Xavier Rodet. Investigation of phase coupling phenomena in sustained portion of musical instruments sound. *The Journal of the Acoustical Society of America*, 113(1):348–359, 2003. doi: 10.1121/1.1518981. URL http://link.aip.org/link/?JAS/113/348/1.

K. Fitz, L. Haken, and P. Christensen. Transient preservation under transformation in an additive sound model. In *Proc. International Computer Music Conference*, Berlin, Germany, 2000.

Kelly Fitz, Lippold Haken, Susanne Lefvert, and Mike O'Donnell. Sound morphing using loris and the reassigned Bandwdith-Enhanced additive sound model: Practice and applications. 2002. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.8144.

Neville H. Fletcher and Thomas D. Rossing. *The Physics of Musical Instruments*. Springer, 1991. ISBN 0387941517.

Adrian Freed, Xavier Rodet, and Philippe Depalle. Synthesis and control of hundreds of sinusoidal partials on a desktop computer without custom hardware. volume 2, pages 1024–30, Santa Clara, CA, 1993. DSP Associates.

T. Galas and X. Rodet. Generalized discrete cepstral analysis for deconvolution of Source-Filter system with discrete spectra. In *Applications of Signal Processing to Audio and Acoustics, 1991. Final Program and Paper Summaries., 1991 IEEE ASSP Workshop on*, pages 0_71–0_72, 1991.

J. Gordon and J. Strawn. An introduction to the phase vocoder. *Proceedings, CCRMA*, 1987.

Thomas Grill. *Two-dimensional gesture mapping for interactive real-time electronic music instruments*. Master thesis, 2008. URL http://grrrr.org/pub/grill-2008-master-2d_gesture_mapping.pdf.

K. F. Hansen. The basics of scratching. *Journal of New Music Research*, 31(4): 357–365, 2002.

David Jaffe and Lee Boynton. An overview of the sound and music kits for the NeXT computer. *Computer Music Journal*, 13(2):48–55, 1989. ISSN 01489267. doi: 10.2307/3680040. URL http://www.jstor.org/stable/3680040. ArticleType: primary_article / Full publication date: Summer, 1989 / Copyright © 1989 The MIT Press.

Sergi Jordà. *Digital Lutherie*. PhD thesis, Universitat Pompeu Fabra, 2005.

A. Loscos, J. Bonada, M. Boer, X. Serra, and P Cano. Voice morphing system for impersonating in karaoke applications. Berlin, Germany, 2000.

Stephen McAdams, Suzanne Winsberg, Sophie Donnadieu, Geert Soete, and Jochen Krimphoff. Perceptual scaling of synthesized musical timbres: Common dimensions, specificities, and latent subject classes. *Psychological Research*, 58(3):177–192, December 1995. doi: {10.1007/BF00419633}. URL http://dx.doi.org/10.1007/BF00419633.

R. McAulay and T. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34 (4):744–754, 1986.

J. A. Moorer. Signal processing aspects of computer music: A survey. *Computer Music Journal*, 1(1):4–37, 1977.

C. Palombini. Pierre schaeffer, 1953: towards an experimental music. *Music and Letters*, 74(4):542–557, 1993.

T. H Park, J. Biguenet, Z. Li, C. Richardson, and T. Scharr. Feature modulation synthesis (FMS). In *Proc. ICMC, 2007*, 2007.

G. Peeters. A large set of audio features for sound description (similarity and classification) in the CUIDADO project. *CUIDADO Project Report*, 2004.

Larry Polansky and Tom Erbe. Spectral mutation in soundhack. *Computer Music Journal*, 20(1):92–101, 1996. ISSN 01489267. doi: 10.2307/3681275. URL http://www.jstor.org/stable/3681275. ArticleType: primary_article / Full publication date: Spring, 1996 / Copyright © 1996 The MIT Press.

R. K. Potter. VISIBLE PATTERNS OF SOUND. *Science*, 102(2654):463–470, 1945. ISSN 0036-8075. doi: 10.1126/science.102.2654.463. URL http://www.sciencemag.org/cgi/content/citation/102/2654/463.

J. Pressing. Improvisation: methods and models. *John A. Sloboda (Hg.): Generative processes in music, Oxford*, page 129–178, 1988.

M. Puckette. Combining event and signal processing in the MAX graphical programming environment. *Computer Music Journal*, pages 68–77, 1991.

M. Puckette. The patcher. In *Proceedings of the 1988 International Computer Music Conference*, page 420–429, 1988.

M. Puckette. *The theory and technique of electronic music*. World Scientific Pub Co Inc, 2007. URL http://crca.ucsd.edu/~msp/techniques.htm.

Miller S. Puckette. Pure data. In *Proceedings of the 1996 International Computer Music Conference*, pages 269–272, San Francisco, CA, 1996. URL http://crca.ucsd.edu/~msp/software.

Edward Resina. SMS composer and SMS conductor: Applications for spectral modeling synthesis composition and performance. In *Proceedings of 1998 Digital Audio Effects Workshop*, 1998.

A. Robel and X. Rodet. Efficient spectral envelope estimation and its application to pitch shifting and envelope preservation. page 30–35, 2005.

X. Rodet and D. Ph. Use of LPC spectral estimation for analysis, processing and synthesis. In *1986 Workshop on Appl. of Digital Sig. Process. to Audio and Acoust., New-Paltz, New York*, 1986.

N. Schnell and D. Schwarz. Gabor, multi-representation real-time analysis/synthesis. In *Proc. of the 8th Int. Conference on Digital Audio Effects (DAFx'05)*, Madrid, Spain, 2005.

Diemo Schwarz. *Spectral Envelopes in Sound Analysis and Synthesis*. PhD thesis, IRCAM, 1998. URL http://recherche.ircam.fr/equipes/analyse-synthese/schwarz/da/.

Diemo Schwarz and Xavier Rodet. Spectral envelope estimation and representation for sound Analysis-Synthesis. *PROCEEDINGS OF THE ICMC*, 351:351—354, 1999. doi: 10.1.1.30.630. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.630.

X Serra. Musical sound modeling with sinusoids plus noise. In *Musical Signal Processing*, Studies on New Music Research, pages 91–122. Swets & Zeitlinger, 1997. ISBN 90 265 1482 4.

Xavier Serra. Sound hybridization techniques based on a deterministic plus stochastic decomposition model. In *Proceedings of the 1994 International Computer Music Conference*, Denmark, 1994.

Xavier Serra. *A System for Sound Analysis-Transformation-Synthesis based on a Deterministic plus Stochastic Decomposition.* PhD thesis, CCRMA, Dept. of Music, Stanford University, 1989.

Xavier Serra and Jordi Bonada. Sound transformations based on the SMS high level attributes. In *Proceedings of the Digital Audio Effects Workshop*, Barcelona, Spain, 1998.

J. O. Smith, X. Serra, Center for Computer Research in Music, Acoustics, and Stanford University. *PARSHL: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation.* CCRMA, Dept. of Music, Stanford University, 1987. URL `http://ccrma.stanford.edu/~jos/parshl/`.

Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications.* W3K Publishing, second edition edition, 2007. ISBN ISBN 978-0-9745607-4-8. URL `http://ccrma.stanford.edu/~jos/mdft/`.

Julius O. Smith. *Spectral Audio Signal Processing, October 2008 Draft.* 2008. URL `http://ccrma.stanford.edu/~jos/sasp/`.

E. Tellman, L. Haken, and B. Holloway. Timbre morphing of sounds with unequal numbers of features. *Journal of the Audio Engineering Society*, 43(9):678–689, 1995.

V. Verfaille, J. Boissinot, P. Depalle, and M. M. Wanderley. SSynth: a real time additive synthesizer with flexible control. In *Proceedings of the International Computer Music Conference (ICMC'06)*, 2006.

T. S. Verma and T. H. Y. Meng. Extending spectral modeling synthesis with transient modeling synthesis. *Computer Music Journal*, 24(2):47–59, 2000.

M. Wright, A. Chaudhary, A. Freed, S. Khoury, and D. Wessel. Audio applications of the sound description interchange format standard. In *proceedings of the Audio Engineering Society 107th Convention*, 1999a.

M. Wright, R. Dudas, S. Khoury, R. Wang, and D. Zicarelli. Supporting the sound description interchange format in the Max/MSP environment. In *Proc. ICMC*, 1999b.

M. Wright, J. Beauchamp, K. Fitz, X. Rodet, A. R\öbel, X. Serra, and G. Wakefield. Analysis/synthesis comparison. *Organised Sound*, 5(03):173–189, 2001.

Matthew Wright and Adrian Freed. Open sound control: A new protocol for communicating with sound synthesizers. page 101–104, 1997.

Matthew Wright and Julius O. Smith III. Open-Source matlab tools for interpolation of SDIF sinusoidal synthesis parameters. In *Proc. Int. Computer Music Conference (ICMC)*, 2005. URL http://cnmat.berkeley.edu/publications/open_source_matlab_tools_interpolation_sdif_sinusoidal_synthesis_parameters.

Matthew Wright, Edmund Campion, and Michael Zbyszynski. Design and implementation of CNMAT's pedagogical software. 2007. URL http://cnmat.berkeley.edu/publication/design_and_implementation_cnmats_pedagogical_software.

Michael Zbyszynski, Matthew Wright, Ali Momeni, and Daniel Cullen. Ten years of tablet musical interfaces at CNMAT. In *Proceedings of the 7th international conference on New interfaces for Musical Expression*, pages 100–105, New York, 2007. ACM. doi: 10.1145/1279740.1279758. URL http://portal.acm.org/citation.cfm?id=1279758.