SMSPD, LIBSMS AND A REAL-TIME SMS INSTRUMENT

Richard Thomas Eakin
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
rich.eakin@gmail.com

Xavier Serra
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
xavier.serra@upf.edu

ABSTRACT

We present a real-time implementation of SMS synthesis in Pure Data. This instrument focuses on interaction with the ability to continuously synthesize any frame position within an SMS sound representation, in any order, thereby freeing time from other parameters such as frequency or spectral shape. The instrument can be controlled expressively with a Wacom Tablet that offers both coupled and absolute controls with good precision. A prototype graphical interface in python is presented that helps to interact with the SMS data through visualization. In this system, any sound sample with interesting spectral features turns into a playable instrument. The processing functionality originates in the SMS C code written almost 20 years ago, now re-factored into the open source library, *libsms*, also wrapped into a python module. A set of externals for Pure Data, called *smspd*, was made using this library to facilitate on-the-fly analysis, flexible modifications, and interactive synthesis. We discuss new transformations are introduced based on the possibilities of this system and ideas for higher-level, feature based transformations that benefit from the interactivity of this system.

1. INTRODUCTION

A musical instrument needs to be capable of expression. It should not only have an interesting and dynamic sound, but it also needs to be playable. The well-known techniques of Spectral Modeling Synthesis (SMS) can easily create rich and interesting sounds, but playability can only be achieved through compromises in order to intuitively control this dense parametric model in real-time. Many approaches choose to re-synthesize the data similar to its acoustic source; for example, a saxophone melody analyzed, segmented and reconstructed with a new order of pitches. On the other hand, focusing on one note in this melody, if parametrized accordingly, presents an abundance of compositional material throughout its many evolving harmonics and energy bands. By limiting the length of the sound source to a few well defined events, one can explore the spectral richness of an analyzed acoustic event through a system based on real-time SMS synthesis, thereby turning these events into playable instruments. Our instrument not only lends itself to musical interest, but also gives the performer an improved understanding of the underlying structure in sounds we may sometimes take for granted.

This paper discusses the ongoing development of a spectral modeling synthesizer in the Pure Data [13] real-time audio environment. It stems from the original SMS system developed in [15], now in the form of the open source C library *libsms*[1]. The library is undergoing constant re-factoring and additional tools from new research in spectral modeling techniques, but at its core contains very powerful synthesis routines that provide for a powerful real-time instrument within a lightweight code base.

Using this library, externals for analysis, synthesis, and editing were created in Pure Data (Pd) called *smspd*. Once this system was established, we were presented with many new and innovative ways to use these tools within such a powerful and flexible programming environment. To begin however, it seemed most appropriate to design a playable instrument, hoping to be as expressive as the acoustic instruments used to create our SMS data. To this aim, the following design choices were established along the way:

- analysis is treated as a separate non-real-time procedure.
- the overall sound length is kept short enough to allow one to focus on distinct regions of data with interesting features.
- the choice of synthesis frame (time index) is unrestricted: any order of frames produces a smooth synthesis, chosen in real-time.

While these decisions may seem quite limiting in the face of many advancements to SMS techniques, this paper aims to show how they lead to an instrument that allows for an interactive exploration of the acoustic phenomena organized within the spectral data in a very musical way.

The rest of this paper is organized as follows. Section 2 provides an overview of the analysis / synthesis methods used and discusses related software. Section 3 introduces *libsms* and tools for real-time synthesis and visualization that use the library. Section 4 describes the implementation of an interactive SMS-based instrument in Pd. Finally, Section 5 discusses the benefits of our approach and provides room for future work in both interface design and modifications.

2. BACKGROUND AND RELATED WORK

2.1. Sinusoids Plus Noise Model

There are many approaches to modeling spectral data that all try to describe a signal in the frequency domain with more useful parameters than what is first obtained with short-time Fourier analysis [1]. Within the scope of this paper, the Sinusoids (or Deterministic) Plus Noise (or Stochastic) [17] model is used for analysis, transformation and synthesis. This model is useful because it effectively parametrizes a large variety of sounds in a manner that offers a number of meaningful manipulations [18] and is well fit for real-time synthesis. The detailed processing techniques are described in [4], while here is only a brief introduction.

[1] libsms homepage: http://mtg.upf.edu/static/libsms

In sinusoidal analysis, a sequence of FFT frames are computed using the STFT. Peaks in the resulting magnitude spectrum are then detected and tracked from frame to frame. The result is a set of sinusoidal 'tracks' with time varying frequencies, magnitudes and phases [12, 20]. With only sinusoidal tracks, it is possible to generate a reconstruction that captures the acoustic qualities of many musical instruments.

If the sound is known to be harmonic, a property common to many musical instruments, the first track can be seen as the fundamental frequency of the sound source. This information can then be used to improve the peak-picking process by looking for partials that are multiples of the fundamental.

There are two basic methods for reconstructing a time-domain signal from sinusoidal tracks. The traditional method is with an oscillator bank, where-in an array of frequency/magnitude/phase are made into time-varying sinusoids by a table-lookup. The resulting waveforms are successively summed into a complex waveform. A more efficient method is to sum the partials while still in the frequency domain and use the inverse-FFT algorithm to convert all partials to a complex waveform at once [8]. Both techniques interpolate the frequencies and amplitudes between frames to create smoothly evolving sinusoids.

A stochastic residual helps to 'fill in' the representation of many sounds that contain noisy characteristics, such as the breath in voice or the string slipping in bowed instruments [15]. The residual is obtained by re-synthesizing the sinusoidal tracks and subtracting the resulting waveform from the original. If the residual is considered as purely stochastic, which is the case in a perfect deterministic analysis, it is only necessary to store the magnitude spectrum of the residual.

The inverse-FFT algorithm is used to reconstruct a time-domain waveform of the stochastic residual from the analyzed magnitude spectrum and randomly generated phases. To reconstruct the original signal, the two time-domain waveforms are summed together.

In both sinusoidal and noise components, the data is computed and stored at a frame-rate fast enough so that linear interpolation from frame to frame creates the illusion of a smooth acoustic evolution in the synthesis. Transients, such as onsets, are not accurately represented in a sinusoids plus noise decomposition[2] because they evolve too fast to be accurately reproduced at a frame-rate low enough to be useful when modeled as sinusoids, yet their phase values are not stochastic. Some systems extract the transients before sinusoidal and stochastic residual analysis and treat them independently [21], while others re-sample the frame-rate in sections where transients are detected [7].

2.2. Real-Time Flexibility with Magnitude Only Analysis / Synthesis

Some types of sounds, especially those with smooth evolution, do not benefit from using original phase values in re-synthesis [15]. If original phases are disregarded in the extracted sinusoidal tracks, new phases can be calculated from any 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 that the next frame's phases can smoothly continue. It is a similar case with the residual data; phases are disregarded and regenerated randomly when needed.

[2] While the result of modeling transients as sinusoids might go strangely haywire, there have been many instances when using a malformed model produced very interesting musical sounds

Modifications in time are trivial if phases are generated on the fly at the time of synthesis. A constant interpolation time from frame to frame is first provided (in the IFFT re-synthesis 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 discontinuities.

2.3. Modifications of the Analysis Data

The Sinusoids Plus Noise model is chosen for our work because it permits many types of modifications, making it possible to create new sounds that contain a rich timbre derived from an acoustic source. The most common manipulations are frequency and time scaling the entire data set by a constant [12, 17]. These modifications make the model popular because they do can be performed independent of one another, without distorting the timbrel characteristics of the sound.

The magnitudes of individual or groups of sinusoidal tracks can also be modified, which produces similar effects to time-domain filtering. Modifying individual frequencies creates a distortion to the overall acoustic structure of the sound, an interesting effect unattainable within the original sound's capabilities.

The noise component can be modified independently of the sinusoidal component, yielding a number of effects. The easiest modification is to emphasis or deemphasis the noisiness by modifying its overall gain. Also, the brightness of the sound can be modified by increasing the gain of the residual in the higher frequency range.

Higher level modifications can be accomplished by extracting features from the SMS data [18]. The spectral shape, or envelope, of both sinusoidal and noise components can be calculated and modified with very interesting results. In sounds that contain a distinct spectral shape independent of pitch (fundamental frequency), re-applying the spectral shape of the original data after transposition can help to preserve timbre characteristics [15]. Two spectral envelopes, from different sources, can be used to create the effect of a time-varying 'hybrid' from one sound to another [16].

Some sound features can be described with one value per analysis. For example, a harmonic distortion can be applied by multiplying the frequency of every partial by a non-integer. Another possibility is to approximate the spectral tilt with a line and use the slope coefficient as a controller, thereby changing frequency magnitudes depending on their location in the spectrum.

Feature-based modifications are useful for manipulating SMS data in real-time without necessarily needing a 'perfect' analysis. They produce interesting sound effects, but the results may be unpredictable. There are other systems that use feature extraction and classification techniques to preform more musically meaningful transformations based on machine learning [3], but usage of these techniques is beyond the scope of this paper.

2.4. Software Approaches to Synthesizing SMS Data

There are many different approaches for processing SMS data, which vary depending on the application at hand. Here is a short review of various SMS software systems whose histories contribute to the system presented in this work.

SMS was originally used in terminal-based applications that were controlled and tested with bash scripts. In the original SMS C package, MusicKit [9] was used to create modifications over time

with envelopes.[3] A score file was read that directed functions for time stretching, amplitude, frequency and hybrid modifications.

Eduard Resina later created the Windows application *SMS-Composer* [14], 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 [2], containing *SMSTools*, has been developed to allow for a collection of effects combined with SMS synthesis. Analysis and Transformations of the SMS model are organized in XML scripts.

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 [11], 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.

Micheal Klingbeil's Spear is a convenient standalone application for different types of analysis, re-synthesis, and editing of sinusoidal tracks [10]. The model he works from is not based on SMS, but is mentioned here anyway because this application has a very nice interface for interacting with extracted sinusoidal tracks. It is a straightforward and lightweight tool that displays sinusoids on a time/frequency axis, resembling a sonogram. Various editing tools are available such as frequency and time region selectors, the most interesting of which is the 'lasso' tool for hand picking a region of sinusoidal tracks. *Spear* is a powerful application for sound analysis, although difficult to use for musical purposes - all modifications must be exported to file. However, *Spear* can export sinusoidal tracks to *SDIF* files [22], thereafter usable in a variety of other applications. For example, the first author wrote externals in *Pd* for importing *SDIF* 1TRC frames created in *Spear*, and then synthesizing with an oscillator bank[4]. There are also objects within the Max/MSP environment for additive synthesis based on analysis data stored in *SDIF* files [22, 23].

3. THE (RE)BIRTH OF LIBSMS AND [SMSPD]

This section provides a brief introduction to *libsms*, an open-source C library for spectral analysis, synthesis and modifications.

Many years ago, a lightweight, C-based library was developed for spectral modeling analysis and synthesis based on the sinusoids plus stochastic model [15]. Last touched around 1990, the code contained many functions that did not work on modern day computers. For example audio input/output depended on the NeXT sound file format and the script-based sequencer was based on the NeXT MusicKit [9]. The audio routines themselves were clearly kept separate from these design decisions and worked quite well from a debian-based linux OS with only a minimal amount of work to make the included terminal programs analyze and synthesize a variety of sounds. Some of the original changes include incorporating Erik de Castro Lopo's *libsndfile*[5] for flexible sound file input/output, changing to floating-point computation, and revamping memory allocation, while there is also a growing list of other changes packaged with the library. *Libsms* aims to be a cross-platform, GNU licensed C library, capable of handling many types of spectral-based algorithms, usable within a variety of applications.

While there have been many advancements to SMS analysis since 1990, the synthesis algorithms are virtually the same. Not only do the synthesis routines in libsms produce high fidelity results, they use very little CPU power on modern day computers.

For the sinusoidal component, both table-lookup oscillator bank and inverse-FFT methods can be used and are interchangeable. For residual re-synthesis, the magnitude spectrum is approximated with stochastic phases [19].

The analysis routines in libsms require many frames of data due of a circular buffer[6]. On the other hand, the synthesis routines are completely frame based, only needing one frame in history for interpolation. In this sense, the synthesizer produces one frame of latency before constructing an audio waveform.

Currently, there are 3 types of programs that use libsms: command-line tools, a SWIG-wrapped python module (with examples), and Pd externals. A brief description of the Pd externals, the **smspd** library, is given here, while information about to the command line and python tools can be found in the library package or homepage.

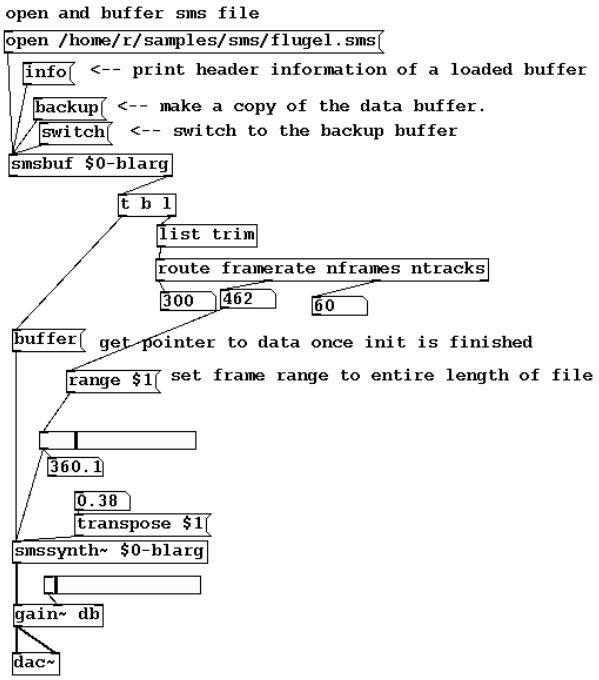

Figure 1: *Example Pd patch for loading a pre-existing file into* ***smsbuf****, then synthesizing it with* ***smssynth~***

There are 4 externals that all operate on a common **smsbuf**

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

[4] Pd externals **sdiflists** and **oscbank** can be found in the Pd svn repository or: `http://mtg.upf.edu/people/eakin?p=Pd%20Stuff`

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

[6] While this circular buffer significantly improves results, current work in modularizing the analysis processes in order to make this circular buffer optional will make real-time analysis in Pd possible.

buffer, similar to how Pd's **delread**$\sim$ and **dealwrite**$\sim$: 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 1. 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.

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

smssynth$\sim$ 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.

The choice to make **smsbuf** an object within Pd was made hoping that it would simplify further advancements in data modifications. While **smsedit** is currently a fairly basic external for low level modifications, it requires very little code to access SMS data. Furthermore, there are certain types of manipulations where multiple data files are necessary. For example, **smssynth**$\sim$ can create 'hybrids' of two data sets by imposing a spectral envelope of one sound onto the partials of another [16], requiring access to two **smsbuf** objects. On the other hand, some effects, such as polyphony and harmonization, are easily created with multiple **smssynth**$\sim$ objects that all have access to a single **smsbuf**.

4. SMS INSTRUMENT IMPLEMENTATION

This section describes an instrument created for performing with SMS data in real-time with *smspd*. The Graphical interface in python for interactive visualization is introduced. Video examples can be watched online[7].

4.1. Preparing the Analysis

Creating a good spectral representation of a sound with libsms is not automatic. 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. Once a favorable analysis is stored to file, it can be recalled into an **smsbuf** in an instant. Nonetheless, the ability to analyze sound from a Pd array is implemented and deserves more experimentation.

4.2. Synthesis Parameters

The parameters for controlling the synthesis are few. They are specified in real-time and are meant to interactively control the synthesis process. Regardless of exactly what value is given to any of the parameters, synthesis will continue while audio computation is turned on.

Time Frame

Time is specified by a frame number relating to the analysis window, usually between 2 and 20 milliseconds. The synthesis window hop size (in samples) effects the interpolation time between frames. By default, the window size in **smssynth**$\sim$ is 512 samples or 10.6 milliseconds at a sampling rate of 48k/sec.

The next frame to synthesize is specified as a floating point number; a new data frame is created by interpolating between two integer frames. Because of this, one can either specify frames with arbitrary distance inbetween (ex. frame 10.2 followed by 1010.5) or by only a slight change (ex. frame 10.2 followed by frame 10.05 or 10.7). Any frame succession will produce a smooth additive synthesis of naturally evolving partial frequencies (although large jumps will obviously create artificial interpolations). Thus, long files may contain *too much* material for performance and shorter sounds with distinct timbres become favorable. Even a half second sound file can be turned into an instrument that is interesting to play for an entire musical work.

Frequency

Another parameter in the synthesizer is a frequency control, which transposes every partial according to a well-tempered scale. As discussed in Section 2.3, it may be desireable to maintain the original sound's spectral envelope in order to preserve timbre characteristics. Spectral enveloping techniques have been added to libsms, based on the discrete cepstrum spectral envelope described best in [6]. It has been very useful for maintaining formant characteristics in voice sounds, while it tends to distort other sound types such as brass instruments. The current implementation allows for both disregarding and maintaining the original spectral envelope. seen

Gain

Gain is controlled by either global magnitude (multiplying the output signal by a constant directly in Pd) or adding/reducing the residual component's magnitude. The gain of individual partials can also be modified with **smsedit** (see sec. 4.6).

Synthesis Type

In some cases, it may be desirable to turn off either sinusoidal or residual synthesis. Both component can be turned on or off in real-time. This helps immensely to understand the quality of each component's representation. For instance, the residual synthesis occasionally sounds like wind, thereby inadvertently making the result sound more synthetic.

4.3. Wacom Tablet as Controller

While the choice for controller is largely up to the user when it comes to Pd, ranging from store-bought electronics to table-top LCD's, this section discusses mapping choices for using a Wacom Tablet as a high-precision controller for **smssynth**$\sim$. Using a Wacom Tablet to control additive synthesis is not new [25]: Matthew Wright has been performing with sinusoidal models in Max/MSP

[7] Video examples of real-time SMS instrument: `http://mtg.upf.edu/people/eakin?p=Examples`

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 uses a set of pre-analyzed sounds in his performances, hand-drawn onto the tablet surface in a grid-like fashion that makes a wide variety of sounds available at any time. In the implementation presented in this paper, one set of analysis data is mapped across the length of the tablet, usually kept short enough that distinct sections in the data can be consistently located by the stylus tip without much effort.

The tablet currently used is a Wacom Intous, 6x11 inches[8]. This model provides a pressure control, which is naturally mapped to the output signal's gain. The synthesis frame is mapped to the absolute X-axis and frequency transpose is mapped to the Y-axis, centered around the middle of the tablet. The actual amount of transposition is variable, depending on the effect desired. For instance, an expressive vibrato is easier to control if the maximum transposition is only about one whole-tone, but melodies are easier to create with up to an octave transposition in either direction. A midi keyboard has also been used for more discrete pitch transpositions according to the fundamental frequency of the harmonic.

The stylus tilt is also used, but as it is difficult to control both absolute position and tilt simultaneously, it is better to map tilt to parameters that do not dramatically effect the produced sound. For example, X-tilt is optionally mapped to reverberation and Y-tilt to residual magnitude.

The Wacom data is received in Pd using Hans Steiner's *linux-event* or *hid* objects, available in both Pd's svn and Pd-extended.

The above mapping works well in most cases and is used by default. Considering the flexibility of Pd, however, it is sometimes preferable to map parameters to other controllers, a sequencer, or even a signal analysis patch. In performances and recordings, the first author changes the mapping scheme on the fly to incorporate other processing techniques available in Pd.

4.4. Interactive Visual Representation

Controlling **smssynth**$\sim$ is difficult without first becoming familiar with the sonic features of an analysis. If one practices using the above described system with a given sound, with time they will grow familiar with the characteristic sections and develop an ability to intelligently perform with the material. However, this time spent could be unnecessary. By watching a sonogram-like plot of the sinusoidal tracks and a time-bar that indicates the variable frame position, one only needs to perform with the instrument presented here for a short time before understanding the sonic features at hand. In this section, we discuss an implementation of a coupled visual aid to the SMS synthesizer presented above that makes use of *Pygame*[9], *PyOpenGL*[10], and *pysms*, the SWIG [5] wrapped *libsms* module.

According to the sonogram analogy, partial tracks are displayed from frame zero to the end of the file across the x-axis, while frequency increases to a fixed point (usually around ten-thousand hertz) along the y-axis. Color is used to represent the decibel magnitude of each partial track. This creates easily recognizable formant structures and onsets that are not cluttered by the noise of a signal.

Originally, the display of every partial breakpoint was visually movable, which was also coupled to the analysis data. This proved burdensome because moving just one track did not create an interesting sound effect, but only made one partial component a distinguishable sinewave in the midst of an acoustically rich sound. Instead, it is more useful to modify groups of partials at a time, making sure that changes occur gradually. The modified sections are visualized by adding color to one of the RGB components, making it easy to know when a deviation from the original sound representation is about to be heard (see figure 2).

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. After moving the time bar over a focused section of a sound a few times, one can make a connection between the graphical representation and sonic characteristics, thereby enhancing the ability to control the sound synthesis.

The graphical interface needs its own copy of the analysis data in memory, although this amount of memory consumption is not an issue in today's computer systems. While this creates maintenance work for making sure modifications are properly represented, the data needs to be accessed differently by synthesis routines (frame to frame) than in the GUI (partial to partial). Therefore, the structure of code in both components is more efficient and easier to read.

4.5. OSC: Linking Pd and Python

Pd and Python communicate using high-level OSC [24] modules: In Pd, messages are sent using the **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[11]. This supports the decision to use multiple programming environments that complement each other: Python provides a large selection of graphical tools and is good for database management, while Pd is designed for complex audio processes in real-time.

4.6. Modifications Separated from Synthesis

Pd is a good choice for high-level audio processing because it is flexible and contains a strong time scheduler for both audio and control data. It seems necessary, however, that all SMS-related manipulations need to act on the low-level data buffer in C. Editing large data sets, like SMS data for example, in patches is cumbersome and bug prone. For this reason, the SMS synthesizer presented here assures that any SMS data frame can be synthesized immediately with just a few parameters, while complicated modifications are developed in other externals or *libsms*.

In **smssynth**$\sim$'s audio loop, there are three main library functions that do the necessary work (see the doxygen documentation for details):

sms_interpolateFrames() takes a floating point index into a file and produces an interpolated frame of data.

sms_modify() modifies the resulting frame based on a set of flags within a parameter structure.

sms_synthesize() blindly synthesizes the interpolated and modified frame

Currently, modifications in sms_modify() are: pitch transposition, envelope maintenance, and superposition of an additional

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

[9] PyGame: `http://www.pygame.org`

[10] PyOpenGL: `http://pyopengl.sourceforge.net/`

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

envelope from a separate analysis file. These modifications create almost no CPU overhead. The system is meant to support a growth of modifications within the library, without effecting the functionality of existing applications.

The modifications available in **smsedit** are low level, generalized methods that serve as an example/proof-of-concept of how to access the data within an **smsbuf**. The edits are performed one frame at a time, either on a range, every other, or a single partial track. Modifications are currently limited to deterministic data, including frequency and amplitude modifications by addition, multiplication, or introducing a random deviance. It is quite easy to add new modifications at the C level when they come to mind, or build more sophisticated, time-varying modifications within pd patches (see figure 3).

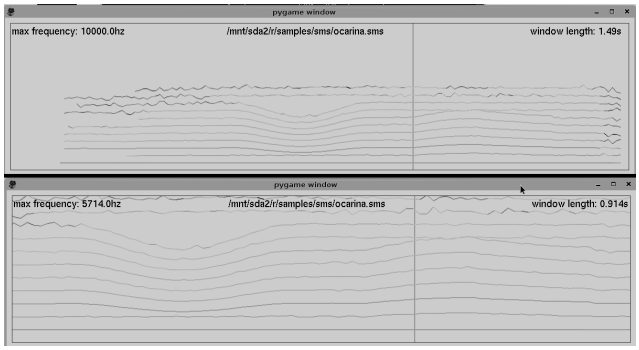

Figure 2: *The top image shows the interface for visualizing sinusoidal tracks extracted with libsms. The vertical bar represents the current synthesis frame in Pd's **smssynth**~. Edits were made, using **smsedit** by multiplying consecutive frame's partial frequencies with values stored in a Pd table. The bottom image is the same analysis data, zoomed in near the frame of synthesis.*

Editing the data with **smsedit** creates irreversible distortions to the unity of the original sound representation. For this reason, **smsbuf** has the ability to keep a 'backup' of the original analysis that can be recalled. With a 'converge' message to **smsedit**, the data can gradually shift towards the original values. With this in combination with the low-level editing functionality in **smsedit**, one can write Pd abstracts to perform interesting modifications that distort sections of the spectral model while leaving others, and then return the model to its unity. Such distortions sound very electronic while maintaining a sense of acoustic structure from the section of the model unmodified.

5. REFLECTIONS AND FUTURE WORK

5.1. New Achievements in Interactive Control of SMS Data

As mentioned in section 4.2, this real-time implementation of SMS synthesis provides for great flexibility in time and frequency, creating a powerful interaction with the sound. 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 4).

It is best to constantly provide new control data to the synthesizer. If time and frequency are both kept stationary, the resulting synthesis immediately sounds stale, as the synthesizer is

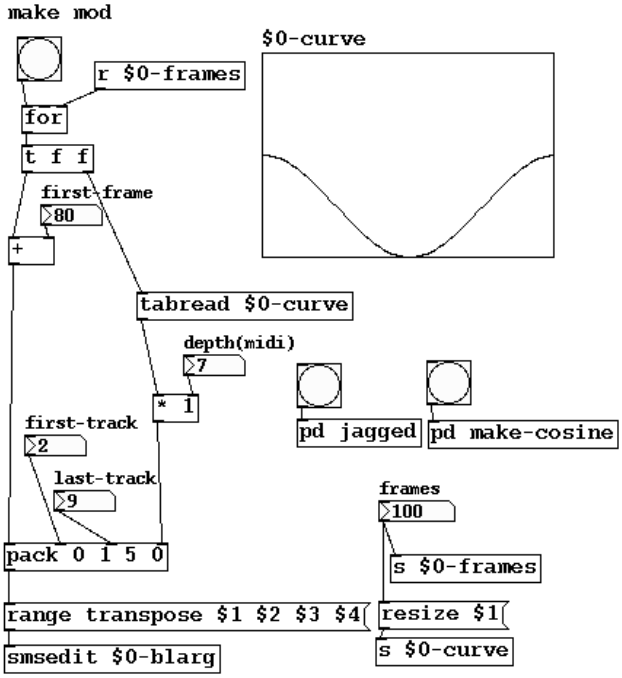

Figure 3: *Example of **smsedit**, using a Pd array as a template for transposing consecutive frame data. The modification is visualized in figure 2.*

re-iterating over the same set of data and there is no partial movement. This is not a hard task with the Wacom Tablet, since time, frequency and magnitude are all coupled with any small gesture from the stylus. Articulations are straightforward and natural; a very realistic vibrato is produced by moving the stylus in a circular motion around a desired pitch.

5.2. Interface Improvements

PyOpenGL is too slow to interactively modify partial tracks. There are still many desireable manipulations that are not yet feasible to visualize, such as bending or warping groups of sinusoidal tracks in real-time. If one takes advantage of OpenGL display lists to pre-compile the plotting commands, CPU usage is reduced from around 50% to around 2%. Yet, one can not change the visual aspects of the independent partials in a display list, thereby loosing many interesting benefits of the graphical representation. A port of the python code to C/C++ is soon to be undergone, where drawing without display lists will be more efficient. As the PyGame code is a direct wrapper around libSDL, the code will be nearly identical as the prototype in python.

5.3. More Modifications

Manually editing the SMS data has usually resulted in turning an acoustic sound into an electric one. This may be desirable in some instances, but if an acoustic coherence is sought, higher level manipulations that take place over time need to be implemented within **libsms**. One research direction is towards extracting 'feature templates' from analysis data that can then be generically used

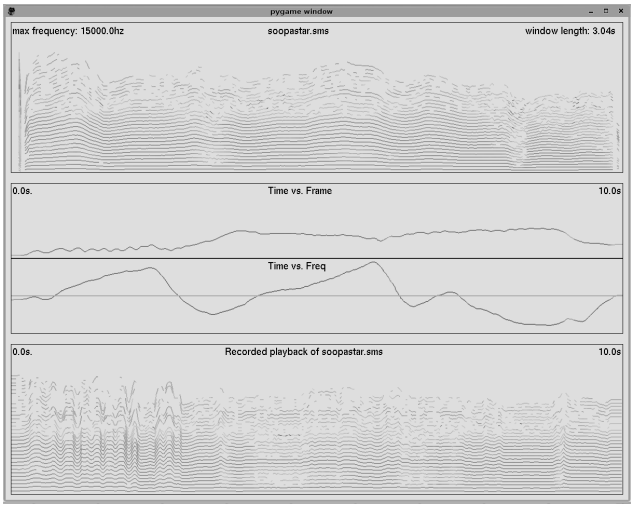

Figure 4: *An example showing time / frequency flexibility.: On the top is a representation of singing voice about 3 seconds long. The middle two x/y plots are 10 seconds recorded from the stylus pen, controlling time (index into the model) and frequency as described. The bottom plot shows the resulting tracks that were synthesized according to these manipulations, containing intricacies similar to the original.*

to modify a sound representation interactively. Then, we hope to extend the interactive control of our system to high level modifications that will yield to greater possibilities in expression.

6. CONCLUSIONS

In this paper we introduced a method for controlling SMS synthesis in a flexible way that allows for real-time interaction, based on a well-tested and now re-factored SMS code base. The flexibility given to the order of synthesized frames provides a framework for exploring spectral synthesis in new and innovative ways. With the re-birth of libsms, its wrapper in python, and real-time externals for Pd, there is new room for exploring state-of-the-art spectral processing techniques. We have discussed methods for adding more interaction and expressive capabilities to the instrument presented here. There is much work to undergo before the true benefits of this system can be fully realized.

7. ACKNOWLEDGMENTS

Many thanks to Miller Puckette, Tom Erbe, and Shlomo Dubnov. Thanks to everyone at MTG, especially Günter Geiger whose help in design ideas is indispensable. Thanks to John Glover for his contributions in libsms.

8. REFERENCES

[1] 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.

[2] 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.

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

[4] X. Amatriain, J. Bonada, and X. Serra. *Spectral Processing*, pages 373–438. John Wiley & Sons Publishers, 2002.

[5] D. 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.

[6] O. Cappe and E. Moulines. Regularization techniques for discrete cepstrum estimation. *Signal Processing Letters, IEEE*, 3(4):100–102, 1996.

[7] 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.

[8] A. Freed, X. Rodet, and P. 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.

[9] D. Jaffe and L. Boynton. An overview of the sound and music kits for the NeXT computer. *Computer Music Journal*, 13(2):48–55, 1989. ArticleType: primary_article / Full publication date: Summer, 1989 / Copyright © 1989 The MIT Press.

[10] M. Klingbeil. Software for spectral analysis, editing, and synthesis. Barcelona, Spain, 2005.

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

[12] 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.

[13] M. S. Puckette. Pure data: Another integrated computer music environment. In *Proceedings of the 1997 International Computer Music Conference*, pages 224–227, San Francisco, CA, 1996.

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

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

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

[17] X. Serra. *Musical Sound Modeling with Sinusoids plus Noise*, pages 91–122. Studies on New Music Research. Swets & Zeitlinger, 1997.

[18] X. Serra and J. Bonada. Sound transformations based on the SMS high level attributes. In *Proceedings of the Digital Audio Effects Workshop*, Barcelona, Spain, 1998.

[19] X. Serra and J. S. III. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, pages 12–24, 1990.

[20] J. O. Smith, X. Serra, C. for Computer Research in Music, Acoustics, and S. University. *PARSHL: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation*. CCRMA, Dept. of Music, Stanford University, 1987.

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

[22] 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*, 1999.

[23] 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*, 1999.

[24] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. page 101–104, 1997.

[25] M. Zbyszynski, M. Wright, A. Momeni, and D. 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.