Ostitch: MIR applied to musical instruments

Abram Hindle
Software Engineering Group
University of Victoria
abez@uvic.ca

ABSTRACT

The paper discusses the use of MIR in computer music instruments. This paper proposes and implements a performance time MIR based instrument (Ostitch) that produces "audio mosaics" or "audio collages". Buffering, overlapping and stitching (audio concatenation) algorithms are discussed – problems around these issues are evaluated in detail. Overlapping and mixing algorithms are proposed and implemented.

Keywords: Collage, Overlapping, Mosaicing

1 INTRODUCTION

Music Information Retrieval (MIR) has been commonly used for querying repositories of sound and music. One aspect of MIR that has been somewhat explored has been applying MIR techniques to live music performances. Often MIR is applied to conventional instruments (Kapur et al., 2004b) or voice samples (Kapur et al., 2004a).

What I propose in this paper is an instrument which uses MIR technology to produce and replace sound from a corpus other sounds. This instrument is controlled by sound much like a filter but unlike a classical filter it can learn about the sound coming in and use that sound to produce new sounds. The main instrument demonstrated in this paper is a performance-time (real-time for music performance purposes) audio collager. Using either a corpus of sound or a growing corpus of incoming sound, similar sounds are queried and played instead of the original signal. An instrument like the collager (Ostitch) attempts to imitate or use the control signal as "inspiration" for its decisions.

MIR can be used to improve human computer interaction with a computer , especially those interactions of skilled musicians. A musician could play a tune to query for a song or use their instrument to control a computer instrument. Interfaces could be based on simple pitch trackers to more complex feature trackers which use domain specific features to describe the instrument input (Kapur et al., 2004b).

MIR can also be used to provide cost effective input to a computer music instrument. A user can simply use a microphone and their own voices or sound making tools to provide an input signal (Kapur et al., 2004a). The voice is an excellent source because it can produce a very wide range of sounds and noises, from vowels and consonants to throat and mouth sounds. The voice is also a very natural instrument for most people.

The inputs commonly associated with a computer, the keyboard and mouse, often are not expressive enough to replicate the input a user might provide with a real instrument. Thus why not let the user use the real instrument with all of its expressiveness to control another instrument?

1.1 Related Work

Schwarez has produced a PhD thesis about audio concatenation (Schwarz, 2004), thus many of these related works are extracted from his thesis:

- PlunderPhonics , by John Oswald is a thesis and a project based around the idea that music is just as usable as sound for producing music, unfortunately music is heavily bound by copyright. Oswald argues that instruments used to produce sound but now if one uses a music player device as an instrument (like a turntable) it confuses the distinction between sound and music. Oswald argues for the "plundering" of other sound to compose and produce new sound. Regardless, DJing and music aggregation are very similar to sound concatenation (Oswald, 1999).
- *Manipulation and resynthesis with natural grain* by Hoskinson , discusses granular synthesis in respect to music concatenation. Samples of 1024 bytes were used with an overlap of 256 samples (in section 3.2 I discuss my overlapping algorithm, this algorithm here would be considered to be an instance of my algorithm where $n = 8$) (Hoskinson., 2002).
- *SoundMosaicing* by Steven Hazel , is an implementation of a sound mosiacer. Hazel focused on trying to achieve near perfect reproduction of sound

through variable size samples. One example provided was the replacement of screaming chimp samples with George W. Bush samples. Hazel does not focus much on the stitching which makes many of the pops noticeable (Hazel, 2001).

- *Musical Mosaicing* by Zils and Pachet , deals with concatetive sound synthesis and constraints used in matching and choosing samples. The focus in this paper was to make a more musical collage. In that respect Zils and Pachet tried to better match the pitch of the music they were imitating through pitch choosing constraints (Zils and Pachet, 2001).

- George Tzanetakis mentioned audio mosaicing but did not provide much detail about it (Tzanetakis, 2002).

- La Legende des siecles 2002 (Schwarz, 2004), was described by Schwarz as a performance art piece in 2002 where the performers used a corpus of audio samples which replaced their audio output; the samples were also pitch shifted and the volumes changed.

- *Mosievius* by Lazier and Cook is the first paper to actually discuss in any sort of detail interactive "mosaicing". Mosievius had the neat idea of mosaicing based upon features rather than input sounds. With the "sound sieve" one could traverse a feature space by playing arbitrary samples whose features were similar to the those along the path (Lazier and Cook, 2003).

- Catepillar is an offline music concatenation system which focused on finding the best path in a corpus to reconstruct the music. *Data-Driven Concatetive Sound Synthesis* was a PhD thesis written by Diemo Schwarz about his engine. Schwarz went so far as to employ hidden Markov models and post processing steps like pitch shifting (Schwarz, 2004).

- *Audio Textures: Theory and Applications* , provided a more artistic use of MIR and concatetive sound synthesis. In the paper the authors describe using a sound similarity matrix on a small sample of music. Then by using the small sample and its self similarities they extract an audio texture. An audio texture is a mini model of the sound which could be played that represents the smaller sound but stretched over a longer time (while still sounding similar). One use of audio textures was to replace lost frames of audio (Lu et al., 2004).

- *Query-by-beatboxing*, was about a technique to query music by beatboxing (simulate beats with the mouth). It was a good example of using other input interfaces to provide query data (Kapur et al., 2004a).

- *MATConcat*, is a non-performance time concatetive sound synthesis engine implemented in Mat-lab. Sturm uses the Hazel "Bush Chimp" example as a benchmark. MATConcat is more of a graphical GUI enabling the user to explore the mosaic more (Sturm, 2004) .

1.2 Contribution

My contribution is the ad-hoc testing and implementation of certain kinds of stitching and overlapping techniques used to produce mosaics for interactive use. As well as providing information about the performance time use of audio collage.

2 SYSTEM DESCRIPTION

Ostitch is an audio collager. It makes audio collages or audio mosaics out of input signals based upon a corpus of audio or the input signal itself.

There are 2 main sources of signal inputs to Ostitch, one is through the sound-card whether it be CD, Line In, or a microphone, the other is through files. Output can be either or both file output and sound output.

User input can be directed into the system via the command-line or remotely via the UDP port where "setter" commands can be sent (it is one way communication). UDP setter commands include commands that turn classifiers on and off, start or stop recording commands. UDP is used for an interface as GUIs can be created which is decoupled from the running instrument process. UDP is explicitly used for its connectionless and latency related attributes.

Command-line options include sampling rate (`-sr`), number of overlap chunks (`-o`), number of samples per block (`-ol`), FFT frame size (`-fft`), which window to use as an envelope of the samples (`-han, -ham, -black, -bart`), and feature disabling options (`-nozc, -norms, -noflux, -noroll, -nocent`).

File IO switches include input corpus (`-i file`), save rendered output (`-o outfile`) , no recording to the database (`-norec`), no sound output to the sound card (`-nosound`).

FFT Mixers can be chosen using the following switches: `-lowpass`,`-hipass`, `-dither` , `-pass` , `-dither` .

2.1 Usage Scenarios

Ostitch can be used for various purposes with various results, some of these scenarios include:

- Voice Replacement - Using a corpus of someone else's voice, the user speaks into a microphone. The samples produced by the user are replaced by the other person's voice samples. This is very entertaining if the other user is not of the same gender as the user.

- Word Replacement - using a corpus of speech, the user speaks into the microphone to have their speech sounds replaced by speech sounds in the corpus. One could produce poetry this way.

- Song Imitation - Uses the corpus of one song to imitate the other. This could be done with a musical instrument – you could play music of another type with the instrument you had. Other interesting uses

include taking a few songs by one artist and then providing a song for it to imitate (by the same artist or not).

- Singing Music - With song imitation you can sing into the microphone and hear music outputted, not just vocals but actual music. For instance, blowing into the microphone with a heavy metal corpus could result in heavy guitars being played because the energy of blowing into a microphone and heavy metal are pretty similar. Classical music is quite dynamic and affords easy interaction with the human voice.

- Sequencing Synthesizer - If one was using a synthesizer as input you could use various notes to sequences samples from the corpus (assuming the notes were consistent). Playing multiple notes at the same time would access different samples rather than the samples accessed by each note individually.

- Radio Exploration - using a radio as input, one can change the output simply by changing radio stations. Radio is a great source signal because it can provide a changing and complex input signal ranging from music to static to talk radio.

3 PROBLEMS

In this section I will discuss various problems that make this research interesting and slightly different from other sound mosaicers. Concatenation of sounds fragments and the mixing of the sounds during concatenation will be referred to as stitching. Sample size refers the size of the chunks of audio being mixed.

3.1 Performance Issues

With any kind of stitching, each kind of stitching sounds different when different sized samples are used. If samples are small then the grains of sound are small, thus the lower frequencies will be interrupted and often lost. If the grains are large the source of the grain becomes more and more obvious. The more obvious the source of a grain, often the less enjoyable it is to hear. A medium size grain seems more perceptually pleasing (at least to the author).

Overlapping is useful as it joins samples tighter together and allows for time domain or frequency domain mixing. There are some caveats with overlapping. Overlapping can result in a pitch shifting of the sound, the pitch usually increases. The less the overlap the less noticeable the pitch shift. Although the less overlap the higher chance of a popping sound. Overlapping samples of different size can produce much more musical results.

3.2 Simplified Overlapping Algorithm

A performance time overlapping algorithm with minimal buffering was implemented (see figure 1). This algorithm would have mix blocks (triangles) and blit blocks (rectangles). Mix blocks would be mixed together using a mixing algorithm while blit blocks would be copied verbatim. Figure 1 demonstrates the overlapping algorithm based on n number of blocks (number of blocks being the number of equally sized segments we cut our samples into) . Each outlining rectangle indicates a block of time being for playback. Samples starting on the left side are buffered from the last call to the overlapper.

Mix indexes are important as the iterate the job of the overlapper for the current time unit. Mix indexes are decremented from $n-1$ to 1 for cases where $n >= 2$ (where n is the number of blocks).

Note how the time units are numbered with mix indexes from 0 to $n-1$ where n is the number of blocks. The first case is a special case so normally mix indexes start at 1. Mix indexes are significant because a time unit of mix index 1 implies that 2 samples plus the last sample are needed to complete that frame (notice how for $n = 2$ all the frames have a mix index of 1). Also when the mix index is 1 that implies there are 2 pairs mix blocks. When a mix index is greater than 1 there is only 1 pair of mix blocks. When a mix index is greater than 1, let m be the mix index, the $n-m$ to $m-1$ blocks (zero indexed) of the last buffer will be blitted into the output buffer, then the m indexed block of the buffer will be mixed with with the first block of the first supplied buffer. Then the $n-m$ blocks after the first block of the first new buffer will be copied to the output buffer. If the mix index was 1, the last block of the last buffer will be mixed with the first block of the first new buffer, the middle $n-2$ blocks of the new buffer will be blitted to the output buffer and then the last block of the new buffer will be mixed with the first block of the second buffer. In this case the second buffer will be copied as the lastbuffer for the next iteration.

The algorithm is described in point form below (assume copying to the outbuffer is done in order as to avoid indexing the outbuffer):

- let m be the current mix number
- let sample1, sample2 be the two chosen samples.
- let lastsample be the buffered samples from last time
- let outbuffer be the output buffer, the size of a sample
- if ($n < 2$) then
  - copy sample1 to outbuffer
- else
  - if (m = 1) then
    - mix the last block of lastsample with the first block of sample1 to outbuffer
    - copy blocks 1 to $n-2$ of sample1 to the outbuffer if $n > 2$
    - mix the last block of sample1 with the first block of sample2 ot outbuffer
    - copy sample2 to lastsample
  - else
    - copy blocks $n-m$ to $m-1$ of lastsample to outbuffer
    - mix blocks m of lastsample and 0 of sample1
    - copy blocks 1 to $n-m+1$ of sample1 to outbuffer

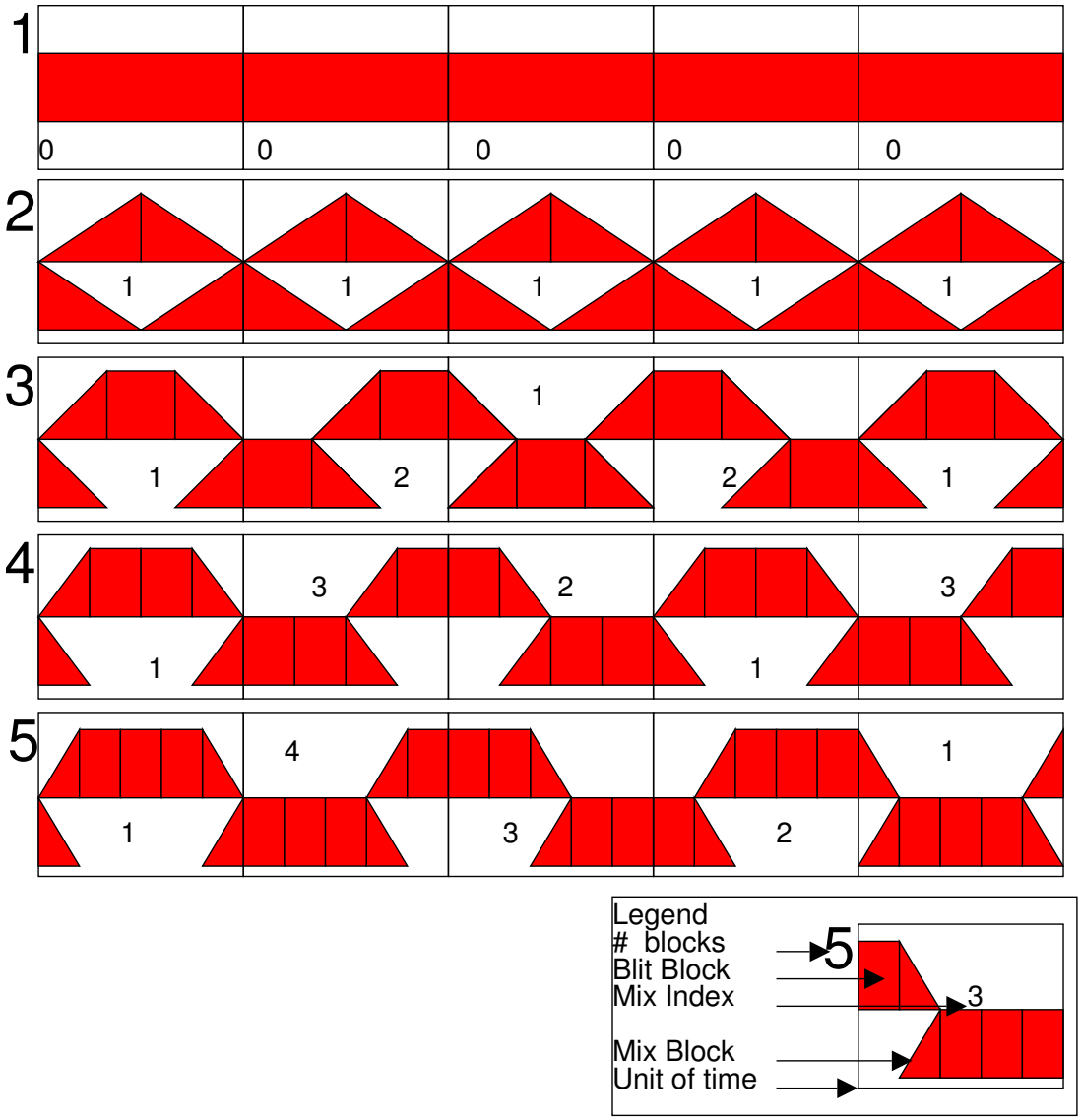

Figure 1: Overlapper Algorithm

* copy sample1 to lastsample

m := (m - 1) modulus (n - 1) + 1

3.3 Time Domain Stitching

Time Domain stitching is the simplest form of stitching. One can simple concatenate the time domain data in order to stitch the data. Simple concatenation is problematic because it can introduce pops or other extraneous noise.

Solutions for dealing with pops and noise are to window or envelope the samples using windows such as the Hann window or a ADSR envelope. One problem with enveloping is that it can add other sounds, such as the sound of the envelope. For examples if we have a block size of 1024 samples and a sampling rate of 44100 and we apply a Hann window to each block, we will have produced another mixed wave at around 43 hz. Of course that wave could be canceled out but if we had a quiet signal holding above 0 we would produce a time domain wave form that looked like many concatenated Hann windows – a far cry from silence.

We found that time domain wise overlapping worked pretty well although one would have to mix the edges of the sample to avoid popping. If there was too much of an overlap ($n = 2$ or multiple samples played at the same time which were similar) the pitch seems be shifted and it sound like we were playing samples at double rate.

If random samples are chosen the pitch can be perceptually increased because there are not a lot of continuous lower frequencies. This happens more if the edges of the samples are mixed out.

There are multiple ways to mix time domain sound. High pass and low pass filters, weighted averaging, dithering are an example of a few. Figure 2 refers to FFT based mixing but many of the examples are attributable to time domain mixing as well.

3.4 FFT Based Overlapping and Stitching

FFT Based overlapping allows "filter" based overlapping. Frames refer to blocks of data of the FFT size which are subsections of samples. One difficultly with FFT overlapping is if the FFT is too large and there aren't enough overlapping frames the mixing algorithms won't work well and the mixing could be rather abrupt and harsh. Thus one problem with FFT Based overlapping is that it can easily bring in extra noise.

There are many kinds of ways to mix FFT frames; one can simply sum FFT frames for more linear mixing (it probably would have been computationally cheaper to use linear mixing in the time domain). With access to the FFT of frames it is very easy to apply a steep high and low-pass filter to each frame and the summate the frames.

Another mixing type is dithering. Dithering is where samples from each FFT are interlocked in a dithered pattern. As the one FFT frame mixes more into the other, it has a higher proportion of the FFT samples.

See figure 2 for a diagram of Fourier Transform Based Mixers.

3.5 Collage

What collage consisted of was:

- Read in an input sample
- Normalize the samples
- Extract features from the sample:
  - Zero Crossing Rate - The number of times the time domain waveform crosses the $y = 0$ line.
  - Flux - The amount of change from the last sample.
  - Roll-off - A ratio of the low energy versus the high energy.
  - Centroid - The center of the distribution of energy.
  - RMS - A measurement of average energy.
- Then find the nearest neighbors to that sample and use those as input to the overlapper.
- Inserting the sample into the database (if recording was on)
- Playing output

The parameters of the collager were the sampling rate, the overlap chunks, the number of samples per "sample" (sample refers to a block of audio read in), the FFT size, and the envelope used for time domain stitching.

Similar chunks were found by using a nearest neighbor algorithm with Euclidean distance (Mahalanobis distance was not used as it would require continuous recalculation of the covariance matrix – Mahalanobis distance makes more sense to use on static corpuses).

The collager can be furthure parameterized by modifications to the chunk similarity algorithm, as it was found during performance that sometimes inaccuracy in the piece selection was musically more interesting than high accuracy of selection.

4 IMPLEMENTATION

The tools used by Ostitch include (**bold** items indicate that it is needed to run Ostitch):

- **OCaml 3.0.8**
- **extLib** - an OCaml library full of useful modules like IO modules for dealing with other programs.
- swig - Swig was initially used to interface with SNDLib but proved to be much too complex to be useful.
- **SNDLib** - SNDLib is a creation of mine (currently updated for this project) that plays audio out to ALSA and allows reading audio from ALSA.
- **OCaml C Bindings** - are heavily used to bind SNDLib to OCaml, this is probably the best way to integrate C code with OCaml.

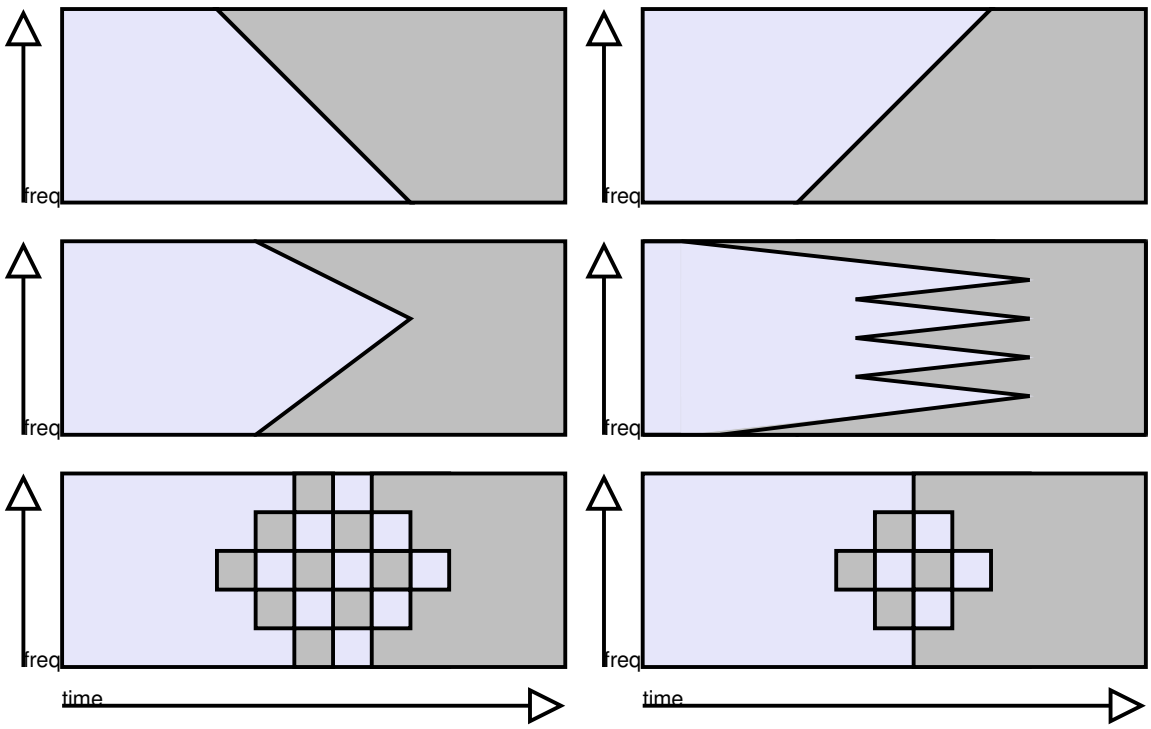

Figure 2: From Left To Right, Top to Bottom: Low-pass mixer, Hi Pass Mixer, Bandpass Mixer, Smear Mixer, Big Dither Mixer, Small Dither

- **ALSA** - Used to play audio. ALSA caused many problems. Buffer overruns while writing out to the sound card occurred frequently even this is a blocking interface! Overruns really shouldn't occur if blocking is provided.
- **GNU Compiler Collection** - GCC was used to compile SNDLib and the UDP Client.
- **GNU Make** - GNU Make tracks dependencies and compiles Ostitch.
- **Perl and Tk** - Perl and Tk were used to provide a detachable GUI to Ostitch.
- Marsyas - Provided initial FFT code and some classifier code to Ostitch. Mainly acted as a reference implementation.
- CSound - provided a reference for implementing some feature

Most of the important functionality was written in OCaml. C was used primarily to talk to ALSA and provide a STDIN to UDP client program. OCaml was chosen because it is a functional, type-safe language who's speed rivals that of C . OCaml is an elegant and clear language which made writing much of the code very easy and intuitive.

OCaml suffers from some debugging issues (such as backtraces and clarity of compiler error messages). OCaml also suffers from painfully poor syntax in regards to floating point numbers (+ and – are for integers , +. and *. are for floating point numbers). Other weird OCaml related issues are the fact that OCaml does not use native Ints or floats, extLib was required to allow OCaml to read and write native types.

OCaml's C integration is very nice and really consists of includes full of type munging macros.

OCaml was appropriate for the project as the program deals with audio which is stream based – audio filters are also usually recursively defined (OCaml supports tail recursion). Unfortunately when I tried to use OCaml with linked lists representing streams of sound I ran into grave garbage collection problems, the program was eating memory very quickly. Due to this I had to switch to an array/block based architecture which complicated things a little bit. For instance it meant I had to use more statements (rather than expressions) than I'd like to. I had to be explicit about what was mutable and what wasn't. Also I couldn't use functions like map because that would require making a new array. The program mostly relies on static buffers, some functions will make their own buffers (hidden by closures).

One challenge was importing an FFT into OCaml. I did not want to use FFTW because I didn't feel like munging around with FFTW types, I wanted to use types that I could manipulate. So I ported the FFT from Marsyas. Unfortunately when I attempted a reverse FFT the algorithm would take an inconsiderate amount of CPU time. A reverse FFT shouldn't be that complex. So I ported over a FFT I had written in Java (which was already array bounds checked and somewhat statically typed). The new FFT worked well, the most useful aspect of the new FFT was that I had to explicitly provide output arrays for real and imaginary values. Even more helpful was that reals and imaginary values were separated into their own arrays

(a `FFTData` type was created to handle these 2 arrays). In summary the FFT and porting the FFT to OCaml caused no end of trouble. I was able to re-implement the FFT without using references or mutables (except for the arrays which have mutable values).

SNDLib was a C library I had built to gain ALSA sound output. I modified it a bit to allow for OCaml integration and to enable it to read sound as well. SNDLib was wrapped in the Sndcaml module to provide an interface to SNDLib within OCaml.

Commlib is a module I wrote which enables remote control of a program via commands sent to a UDP port that Commlib listens on. This style of interaction is very nice as you can run one continuous instance of a GUI and Ostitch can be stopped and restarted. Unfortunately this does not lend to very good feedback inside of the GUI. Perl and TK were used to provide a simple GUI to Ostitch.

Audio is a module which contains miscellaneous audio functions. These functions include windowing algorithms, feature extractors like zero crossings, RMS etc, euclidean distance, flux, overlappers, and numerous array utility functions like `arrayapplyi` which is like a in place map, `array2fold` which folds 2 arrays into one scalar value, `arrayfoldi` which folds 1 array into one scalar value but provides the indices of the values being folded.

Findarg is a simple module much like getopt but much much simpler, it supports flags and string based input from the command-line.

Ostitch calls upon all these and the overlapper to produce a usable system. One of the difficulties of Ostitch was to provide near real-time performance. Issues arise in the size of the buffers, not overflowing the output buffer and scheduling the reading in of samples while outputting samples at the same time. Some buffering inside the overlapper were needed to deal with the fact that samples were consumed at a faster rate than they were read or played at. Of course buffering is always an issue when doing any sort of audio programming.

5 CONCLUSIONS

In summary it is hard to tell how well certain aspects of the project work because most of the project is so perceptual. Artistically Ostitch seems to have merit, it is quite fun to scream into a microphone as heavy metal samples replace your screaming.

For live performance I think this instrument is a success. I plan to use it to perform music at the next Victoria Noise Festival, and I gave a small demo in front of George Tzanitakis's MIR class. With corpus pre-loading,one can have predetermined music sets without worrying about training.

Lessons learned from the instrument are to be weary of overlapping and sample size. Bother of these parameters can increase the perceived pitch of audio. Small sample sizes sound noisy and poppy, they don't provide any hooks into real audio. Medium sized samples provide some coherency while still being separate from the audio they were extracted from.

5.1 Future Work

There are many future directions one can take.

One direction would be useful would be to implement some perceptual metrics based on MOS (Mean Opinion Score) or at least attempt to model MOS. I would like to test the quality of the various parameters against users or at least models of users.

5.1.1 Stitching

There are various kinds of Stitching that could be explored.

For sample selection, edge metrics might be useful where you match the start and end of the sample to infer a more smooth transition.

Non-linear mixing would also be appropriate such as logarithmic scaling of the audio in the time domain.

A good question to evaluate is "Do we need samples to be played at a consistent time, can we play samples at any time?". It might be the case that the proposed algorithm was not as good as say randomly layering samples at various start times.

Different mixing techniques should be evaluated as well. Maybe convolving the audio might produce a nicer transition between samples.

5.1.2 Collage

Other directions for audio collage that should be evaluated include:

- Modify the samples - use time warping and pitch shifting to modify and layer samples for overlapping. Time stretching and folding would allow for the lower frequencies to be better represented. This has been done to a certain extent already.
- Multi-length chunks - chunks of audio should not be as discrete as they currently are, one should be able to grab an arbitrary chunk out of the database and start layering on top of the current sounds.
- More Features - more features should be extracted to allow for more accurate sample retrieval (this has been done already elsewhere). It would be good to evaluate the expense of certain features and whether or not they contribute to the accuracy of the chooser.
- Feature Selection Parameters - there should be parameters such as jitter and "nth best choice" which allow for and emphasize error in sample choice.
- Sample fairness - weight samples so that the least frequently used samples get choosen as well.

5.1.3 Applications of MIR to Performance

In the more general case of using MIR to performance there are some areas that others or I should evaluate:

- Features as instrument parameters - would the output of a zero crossing, flux or centroid be appropriate input for an FM synthesizer? How quickly do these parameters change? What the properties of change and thus how appropriate are some of these features for controlling the parameters of other instruments. Can

we have parameters which map well to the source? Pitch tracker has been done quite a lot in the computer music arean.

- Natural Mappings of voice features to music - The voice is very versatile and very controllable it would be helpful to effectively exploit the voice further, especially for using a microphone as interface to an instrument.

- Audience Based Measures - There should be some exploration of audience measurement or audience interaction with MIR. Audience interaction seems to mostly consist of aggregates and direction (e.g. which group in the audience is the loudest etc).

References

Steven Hazel. Soundmosaic. http://thalassocracy.org/soundmosaic/, 2001.

Reynald Hoskinson. Manipulation and resynthesis with natural grains. Master's thesis, University of British Columbia, 2002.

A. Kapur, M.Benning, and G. Tzanetakis. Query-by-beatboxing: music retrieval for the dj. In *Proc. Int. Conf. on Music Information Retrieval (ISMIR)*, 2004a.

A. Kapur, G. Tzanetakis, and P.F. Driessen. Audio-based gesture extraction on the esitar controller. In *Proceedings of the International Conference on Digital Audio Effects*, pages 17–21, October 2004b.

Ari Lazier and Perry Cook. Mosievius: Feature driven interactive audio mosaicing. In *DAFX 2003*, 2003.

Lie Lu, Liu WenYin, and Hong-Jiang Zhang. Audio textures: Theory and applications. In *IEEE Trans. on Speech and Audio Processing*, volume 12, pages 156–167. Institute of Electrical and Electronics Engineers, Inc., March 2004.

John Oswald. Plunderphonics. http://www.plunderphonics.com, 1999. URL `http://www.plunderphonics.com/`. Last accessed April 2005.

Diemo Schwarz. *Data-Driven Concatenative Sound Synthesis*. PhD thesis, Universit Pari, January 2004.

Bob L. Sturm. Matconcat: An application for exploring: Concatenative sound synthesis using matlab. In *ICMC 2004*, 2004.

George Tzanetakis. *Manipulation, Analysis and Retrieval Systems for Audio Signal*. PhD thesis, Princeton, 2002.

A. Zils and F Pachet. Musical mosaicing. In *Proceedings of DAFX 01*. University of Limerick, December 2001.