AUDIO ANALYSIS IN PWGLSYNTH

Vesa Norilo
CMT,
Sibelius Academy
Helsinki, Finland
vnorilo@siba.fi

Mikael Laurson
CMT,
Sibelius Academy
Helsinki, Finland
laurson@siba.fi

ABSTRACT

In this paper, we present an incremental improvement of a known fundamental frequency estimation algorithm for monophonic signals. This is viewed as a case study of using our signal graph based synthesis language, PWGLSynth, for audio analysis. The roles of audio and control signals are discussed in both analysis and synthesis contexts. The suitability of the PWGLSynth system for this field of applications is examined and some problems and future work is identified.

1. INTRODUCTION

PWGLSynth is an audio synthesis subset of the visual programming language PWGL[1]. Most of our prior work deals with generating control information from the integrated notation package ENP [2] and applying it to, for example, physical models of acoustic instruments.

Recently, our system's input capabilities have been added in the form of realtime audio and MIDI inputs as well as a forthcoming streaming sound file player. These extensions provide motivation to examine the viability of PWGLSynth in the case of audio analysis.

The rest of this paper is organized as follows. In the first section, 'Analysis vs. Synthesis', we discuss the kinds of abstractions and signal models commonly used in programming for these two scenarios. The second section, 'A Case study: f0 estimation in PWGL', presents the details of our pitch estimation algorithm. The implementation details are discussed in the section 'Design of analysis boxes'. Finally, the viability of PWGLSynth for audio analysis and future improvements are outlined in the section 'Further development'.

2. ANALYSIS VS. SYNTHESIS

In some ways, audio analysis and sound synthesis can be viewed as opposites. What is reversed is the flow of signal, from high level to low level or the opposite. In this case, signals related to human perception such as 'pitch' or 'amplitude' are considered high level signals while the signal containing the largest amount of raw information, the audio signal, is considered a low level signal.

In sound synthesis, we devise algorithms to create low level audio signal from a set of high level signals that have musical meaning. A lot of seminal work has been done in the field, and the challenges in sound synthesis often deal with very fine musical nuance or radically novel timbres. In audio analysis, we try to retrieve high level meaning from a raw waveform, which is a task less ideally suited to a computer. The situation is quite different from sound synthesis, and work is still being done to achieve robust detection of some fairly basic musical parameters such as pitch or tone color.

Traditionally in the field of sound synthesis, high level signals are considered control signals. The separation of signals to control rate signals or control events and audio rate signals is well established, and serves to optimize the computation of a synthesis patch. While the control signal paths are equally if not more important than audio signal paths in determining the musical timbre and personality of a synthesizer, they require a slower update rate than the actual audio computation.

A typical implementation of a synthesizer would run the control section separately from the inner DSP loop, conceptually 'before' the synthesis computation is performed. The control section would then update some signal slots available to the synthesizer part, which is run as a separate subroutine. The dependency of the synthesis computation on the control computation is in this case implicitly recognized by always evaluating the control section before synthesis section. This general scheme is also the one for which PWGLSynth was originally designed [3].

For audio analysis, the situation is reversed. The patch is driven by audio input, either from a sound file or a real time audio input, generating high level events or signals from the low level data stream. Now the signal rate tends to decrease instead of increasing as we traverse the patch downstream. Analysis and synthesis processes are not formally reciprocal by any means, but the processing stages of each kind of patch resemble a mirror image of the other. One major focus of this paper is the difficulities this presents to the current PWGLSynth signal model and potential avenues of improvement.

3. A CASE STUDY: F0 ESTIMATION IN PWGL

As a case study, we examine a variant of the harmonic product spectrum[4] estimation method for fundamental frequency of a monophonic signal. This algorithm was developed by the first author for a violin intonation trainer software Sonic Finetune, a Sibelius Academy project [5]. This section will briefly outline the algorithm.

A harmonic product spectrum based method was chosen over the currently popular autocorrelation methods after initial testing. In this method, several compressed copies of the spectrum are created. In each copy, the frequencies of the components are divided by an integer number. The product of all these copies has peaks at the frequencies that exhibit strong overtones, as overtones are spaced at integer multiplies of the fundamental frequency. The frequency resolution limitations of HPS were overcome with phase

unwrapping, a technique adapted from phase vocoders[6]. In many ways, this is similiar to the Max MSP fiddle-object,[7] which improves HPS frequency resolution by phase unwrapping the dominant frequency bin.

For the current algorithm, a partial analysis is carried out, contrasting the phase behavior of each FFT bin to that of a steady state sinusoid over a number of subsequent FFT frames. This technique will favour steady, sustained tones and achieves quite a high frequency resolution with relatively small frame size.

A product spectrum is then computed over the frequency as a continuous domain with the detected partials represented as delta functions. Frequency uncertainty is allowed between a set of delta functions near each other since the partial analysis is a heuristic method. The final frequency value for the product spectrum delta function is computed as an amplitude weighted average of the set of delta functions that contributed to the product. This technique allows for a relatively small FFT frame with large overlap to be used for a relatively high analysis time resolution without compromising the frequency resolution for monophonic signals, as the frequency resolution for this algorithm is more dependent on the window step size than the FFT size.

A high data rate in turn allows for some data redundancy which can be leveraged during post processing. This makes the algorithm well suited for examining some fine details of vibrato, for instance. In informal testing with emphasis on passages requiring high time resolution, the algorithm performed at least equally well or better on most sound clips than an excellent implementation of the autocorrelation pitch detection by McLeod and Wyvill in the Tartini software package[8]. Full scientific evaluation of the performance of this algorithm is dependent on continued funding for the Sonic Finetune project.

4. DESIGN OF ANALYSIS BOXES

The most straightforward way to implement the pitch detection algorithm for PWGL would be to create a single monolithic box that converts an audio signal into f0. However, this is not preferable for a number of reasons. Since PWGL is a programming environment, generality and modularity are prioritized over extreme simplicity. The most beneficial approach would be to create a set of analysis boxes that can be combined in various ways to realize different analysis algorithms, much as synthesis algorithms have traditionally been accomplished in PWGLSynth.

However, this brings up a design problem: we have previously spent much effort on avoiding multiple protocols within the patch[9]. The traditional example of this division is the presence of separate control rate and audio rate paths within a system. This division is not present in PWGL, greatly simplifying patching and reducing the number of box types and permutations needed by the user.

A set of analysis boxes will have yet another class of signal needs, such as transforming the signal between time domain and frequency domain. Integration of these kinds of signals into the PWGLSynth framework while achieving as much reusability as possible from the previous set of synthesis boxes is the goal of the analysis box design.

4.1. FFT transform box

When transforming an audio signal into the frequency domain, the waveform signal model in PWGLSynth no longer directly applies. The most practical implementation is to provide a short time Fourier transform based box that buffers the audio signal and transforms each buffer into the frequency domain. The applications and benefits of this scheme are common to much of fundamental signal processing. Our implementation offers the standard set of choices in window size, step size, zero padding and windowing. Together, these parameters allow for applications ranging from analysis to efficient convolution.

The extended multichannel capacity of PWGLSynth[10] is leveraged in formatting the output of the FFT box. For a N-point transform, the output signal is a N-channel vectored signal containing the real and imaginary parts of the positive frequency side of the transformed signal. The components are mapped as follows:

Table 1: *FFT transform data format*

0	...	$n/2$	$n/2+1$	...	$n-1$
bin 0	...	bin $n/2$	bin 1	...	bin $n-1$
re	re	re	img	img	img

This non-interleaved format will allow the user to split the output into a real part and imaginary part vectors with minimal overhead. This scheme in turn allows the user to employ the existing rich set of vector operations with FFT transform results.

The FFT box will, in addition, provide refresh events to notify the downstream patch whenever a new transform is computed.

4.2. Partial analysis

The partial analysis computes the phase delta for each frequency bin between subsequent FFT frames. Phase delta history is stored for a user defined number of adjacent FFT frames. While the FFT analysis result is a series of phase and amplitude vectors for a discrete set of frequencies determined by the window size, the phase vocoder algorithm is able to exchange phase information for increased frequency resolution.

The next step is to compute a more accurate frequency value for each FFT bin, examining the behavior of the phase of the bin over a a number of FFT frames. In the traditional phase vocoder algorithm, the phase change between two frames is examined as an indicator of what is the precise frequency of a sinusoid the FFT bin is picking up. In a number of cases the phase behavior can be caused by transients or changing signals, and this can result in a phase vocoder picking up erroneous partials.

A steady state sinusoid will produce a constant phase delta between frames. In our algorithm, the average phase delta of the history set is examined along with the deviation of phase delta. The phase unwrapped partials are inversely scaled by the phase delta deviation, thus decreasing the weight of detected partials less closely resembling steady state sinusoids.

Finally, partials that are close to each other will be merged, the resulting partial having the total amplitude of the two partials and a frequency that is an amplitude weighted average of the partials. This step is carried out to compensate for the tendency of the phase vocoder analysis to produce a cluster of pseudo-partials near the actual components of the input signal. With suitable parameters, the merged average partial of such a cluster very closely matched the source signal. Merging the partials also reduces the combinatory load of the eventual f0 analysis.

In a synthetic test, the algorithm was able to follow a the fundamental frequency of a rich geometrical waveform within an error margin of $0.2Hz$ with a frame size of 1024 samples with 8 overlapping frames, with the sampling rate of $44.1kHz$. This will in turn yield a data rate of of $344.5Hz$ with a time uncertainty in the range of $20ms$ due to window size.

The partial analysis box output will consist of a user-defined number of partials sorted by their relative amplitude. The output data format is as follows, the data represented by a vector with length $2n$ for n partials.

Table 2: *Partial analysis data format*

0	1	2	3	...	...	$2n-2$	$2n-1$
freq	amp	freq	amp	...	...	freq	amp
partial 1		partial 2		...		partial n	

4.3. F0 estimation

Like in the traditional HPS method, our estimator creates spectral copies compressed in frequency and multiplies them for a product spectrum. In this case, unlike the usual FFT spectrum, the set of partials from the partial analysis module will be used. Due to the heuristic nature of the algorithm and also the fact that exact equality for floating point math is undefined in computer hardware, exact rigor in multiplying the compressed spectra together is not followed.

Mathematically, the set of partials is a group of delta functions over a continuous frequency axis, and only partials with exactly the same frequency will have a non-zero product. That is why fuzzy matches are accepted when multiplying partial amplitudes over the frequency axis. For example, multiplying two partials whose frequencies differ by less than a user defined threshold will produce a partial with a frequency between the two multiplicands and a product amplitude. Also, a floor amplitude value will be added to each multiplicand amplitude to prevent a missing harmonic from canceling out the whole product. The most prominent frequency component of the product spectrum is treated as the f0 candidate.

The tweaking of the partial merge threshold, the number of partials contributing to the f0 estimation and the floor value control the response quality and robustness of the algorithm.

4.4. Pre- and post-processing

Before analysis, the signal is bandpass filtered with a corner frequency of the lowest fundamental frequency expected. This reduces the amount of less significant low and high frequency content in the signal and emphasizes low frequency partials relative to higher partials. This in turn increases the robustness of the analysis and reduces the rate of octave errors in the detected pitch.

The frequency biasing is performed with standard audio filters. In addition, averaging and median filters can be used on the f0 output to smooth out noise and blips in the analysis result.

4.5. The algorithm

A PWGLSynth patch displaying the full implementation of the algorithm is show in Figure 1. The sound source for the patch is a realtime audio input and the resulting f0 analysis is used to drive a simple pitch following sinusoid oscillator.

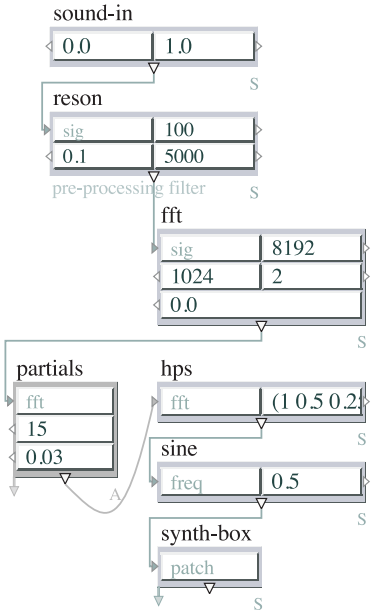

Figure 1: *A patch featuring the f0 estimation algorithm*

5. FURTHER DEVELOPMENT

Implementing audio analysis on PWGLSynth reveals some disadvantages in the current signal model, resulting from the fact that PWGLSynth was originally strictly designed for the synthesis scenario. In this context, signals flow from sparse control signals to dense audio signals. For audio analysis, the case is the opposite. For instance, a method for computing the power spectrum of an audio signal is presented in the patch shown in Figure 2.

However, the way this patch is evaluated in the current implementation is suboptimal, as the mul-vector, add-vector, accum and sqrt boxes run at the audio rate, operating on very large vectors on every single sample frame. There is currently no way to communicate the FFT update rate to these audio rate boxes unless they specifically implement a control rate interface. This in turn would require the box developer to implement processing routines twice, once for audio rate and once for control events. More generally, this problem persists for any patch that generates lower rate signal from a higher rate signal.

In contrast, a similar algorithm implemented in Pure Data - like environment would have the benefit of packaging each transformed buffer as a single message, and downstream nodes can simply respond to receiving messages. The downside is the problem of combining event streams that is a requisite in the ubiquitous class of nodes that combine a set of parameters to produce a single result. This scheme requires the user to explicitly control the scheduling of the patch by using bang events and observing somewhat arbitrary scheduling rules.

The need for a signal model that retains the generality of our current model while providing extended support for a diverse range of signal rate changes within a patch independent of individual box

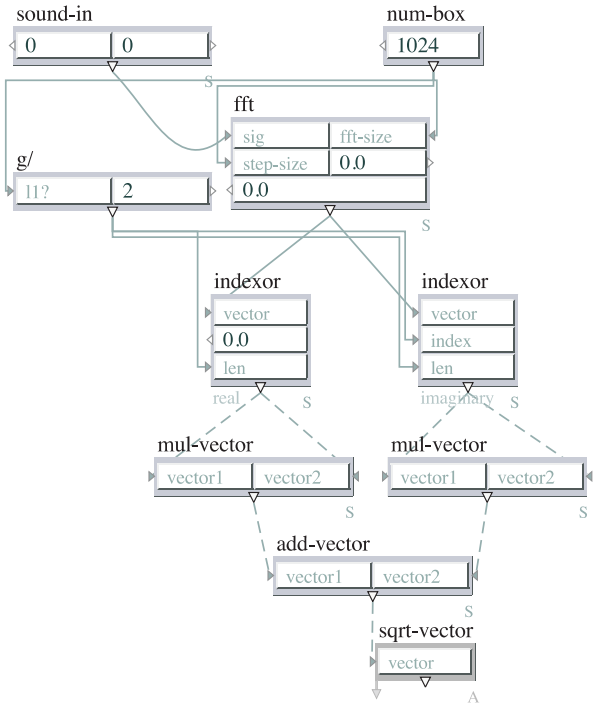

Figure 2: *A patch featuring a power spectrum computation.*

implementation becomes apparent. The ideal model would combine a computationally optimal DSP scheduling with freely mixing signal rates, yet without a need for specialized control and audio rate boxes. A natural application of such a system is a patch containing both analysis and synthesis elements. A simple example of a combination of these two paradigms is a pitch following oscillator, but the implications are much broader, especially in the context of live performance.

The authors have proposed a new signal model combining data driven and request driven processing. The model involves analyzing the state space of a patch and generating dependency based procedures for each state that contain the minimal set of operations required to synchronize the rest of the states in the patch whenever a state update is triggered. This model was designed to address the needs of audio analysis and other cases, where the signal flow tends to be more complicated than in the common control and synthesis case.

6. CONCLUSIONS

In this paper, we examined the general suitability of PWGLSynth for audio analysis and an implementation of a fundamental frequency estimator specific to PWGLSynth. The presence of analysis suggests the intriguing possibility of combined analysis/synthesis patches. The topology and the conceptual modularization and mapping of the algorithm into general purpose PWGLSynth boxes was presented. Some features of the current signal model were found suboptimal for the audioanalysis case, and future work addressing the improvement of the model was laid out.

7. ACKNOWLEDGMENTS

The work of has been supported by the Academy of Finland (SA 105557 and SA 114116).

8. REFERENCES

[1] Mikael Laurson and Mika Kuuskankare, "PWGL: A Visual Programming Language for Computer Aided Composition, Music Analysis and Sound Synthesis," 2004.

[2] Mika Kuuskankare and Mikael Laurson, "Expressive Notation Package," *Computer Music Journal*, vol. 30, no. 4, pp. 67–79, 2006.

[3] Mikael Laurson, Vesa Norilo, and Mika Kuuskankare, "PWGLSynth: A Visual Synthesis Language for Virtual Instrument Design and Control," *Computer Music Journal*, vol. 29, no. 3, pp. 29–41, Fall 2005.

[4] M. R. Schroeder, "Period histogram and product spectrum: New methds for fundamental frequency measurement," *Journal of the Acoustical Society of America*, vol. 43, no. 4, pp. 829–834, 1968.

[5] Noa Nakai, "Viulunsoiton opetus ja teknologia," M.S. thesis, Sibelius Academy, Helsinki, Finland, 2008.

[6] James Moorer, "The use of the phase vocoder in computer music applications," *Journal of the Audio Engineering Society*, vol. 26, no. 1/2, pp. 42–45, January 1978.

[7] M. Puckette, T. Apel, and D. Zicarelli, "Real-time audio analysis tools for pd and msp," 1998.

[8] Philip McLeod and Geoff Wyvill, "A smarter way to find pitch," in *Proceedings of International Computer Music Conference*, Barcelona, Spain, September 2005, pp. 138–141.

[9] Mikael Laurson and Vesa Norilo, "Recent Developments in PWSynth," in *Proceedings of DAFx 2003*, London, England, 2003, pp. 69–72.

[10] Mikael Laurson and Vesa Norilo, "Multichannel Signal Representation in PWGLSynth," in *Conference on Digital Audio Effects*, 2006.