Zeeshan Lakhani

December 13, 2007

DST: Final Project...

"With the term "time stretching" we mean the contraction or expansion of the duration of an audio signal (time compression, time expansion, time scaling—signal processing term)" (Zolzer 205). My final project, completed in Matlab, is a function/program that allows a user to process a signal with two different time-scaling algorithms (at the user's discretion): one that sits firmly in the time-domain and another that deals in the time-frequency domain. The latter, a little something called a *phase vocoder*, dates back to 1966 and has been used and modified may times over. The former, *SOLA (synchronous overlap and add*), was introduced into 1986 by Roucos and Wilgus (Loscos 58); it is simple computationally and does have real-time ability (though not in this project's scenario). The goal of such techniques is to avoid changes of shifts in frequency/pitch content. Obviously, other errors and artifacts do creep up, but I will refer to those later this paper. The main point however—the one with the most practical panache—is that these implementations make good, interesting, and usable (from a compositional standpoint) sounds.

Let's begin with the first case, which is ***SOLA*** (in regards to the order of my code). The basic procedure is organized like this:

a) The input waveform is segmented into overlapping frames of a set window length. These frames are spaced apart by user-chosen hop size. The first frame is directly output to the output, a.k.a. synthesized signal, which—in the loop—has a shifted step size, based on alpha: a user-set scaling factor. Alpha factors below 1 speed up and compress the signal; factors above 1 stretch and slow down the celerity of the signal; a factor of 1 keeps the signal intact.

b) Cross-correlation is used in this algorithm to find the discrete-time lag of maximum similarity between each analysis frame and the synthesized signal within a user-designated overlapping region. This is the formula:

$$R_m(k) = \frac{\sum_{j=0}^{L_0-1} y(mS_s + k + j)x(mS_a + j)}{\sqrt{\sum_{j=0}^{L_0-1} x^2(mS_a + j)\sum_{j=0}^{L_0-1} y^2(mS_s + k + j)}}$$

The correlation (equation) function determines the best 'point' to overlap frames. These optimal, “best-position” (Dorran 12) overlapping blocks are weighted by a cross-fade (fade-in and fade-out) and the rest of the frame is added/copied to the output (the output vector contains the overlap, the cross-fade, and the leftover samples). This process occurs for every iteration of the loop—creating an overlap and add situation. Look:

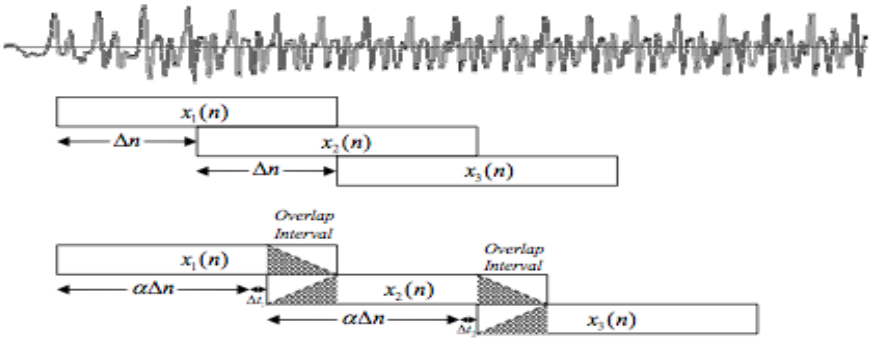

Figure 2.2 SOLA time-scale

Again, the overlap interval length (in samples) and the time-scaling alpha factor is part of the function's input parameters (decided by the user). This correlation-based method preserves the pitch, magnitude, and phase of the input signal, which I can attest to, even though I don't trust my ears all too often. “The *SOLA* algorithm is robust in the presence of correlated or uncorrelated noise, and can improve the signal to noise ratio of noisy speech” (Sanjaume 15).

This technique runs well for shorter sample inputs; however, it bogs down for full-length songs, especially in regards to time expansion. The use of the correlation function (*'xcorr'* in Matlab) is considered computationally expensive (not the entire algorithm per se). Work has been done to further this method through the practice of autocorrelation, adding an FFT into the mix.

In processing various signals, I have found that polyphonic examples struggle with time compression, sounding annoyingly chaotic. With expansion, percussion and polyphonic pieces sound elucidated and deliberate, which is positive for any computer music tasks. Monophonic signals seem to

work well in both cases.

The second approach is known by many as the ***phase vocoder***. As stated earlier, this method involves taking the STFT (Short-Time-Fourier-Transfom) of a signal and separating the real (magnitude) and complex (phase) elements of a spectrum. There are a myriad of avenues in regards to the implementation of this technique, with the main models being *Filter Bank Summation* and the *Block-by-Block Approach* (Zolzer 272). For this project, I decided to work with a variation of the latter, approach (quicker), one using integer time-stretching ratios—considered to be a more direct and elegant implementation (without unwrapping the phase). I took these logical steps:

a) This process begins by segmenting the signal into predetermined (N-length) windows, moving in a loop by the user-input hop size. Then, a window type is applied (i.e. Blackman, Hanning). Then, the infamous Fast Fourier Transform is taken.

b) Moving into the spectral domain, the phase and magnitude values are separated. This allows for a direct multiplication step, involving the magnitude, the phase, and the stretch ratio. The stretch ratio is calculated by dividing the output's (time-scaled) hop size by the original signal's hop size. Again, both of these numerical evaluations are user-picked. Once the mathematical calculation is completed, an IFFT of the spectrum is implemented. This leads to a new, scaled set of numbers—back in the necessary time-domain. Each iteration of the loop overlaps and adds this newly generated vector to the output, and voila: a time-scaled signal is ready to be heard!

Compared to the *SOLA* algorithm, the *phase vocoder* follows a more linear, to-the-point system of orders. For me, in regards to listening, I was more attracted to the vocoder's sound, rich in texture, especially in dealing with time expansion. However, this richness is probably due an actual 'flaw' of the algorithm. *Phase vocoders* are known to suffer from audible drawbacks such as 'smearing' and 'reverberation.' These 'issues' definitely do affect texture, which could be negative or positive, depending on the musician's/scientist's need. The frequency content remains intact from the original input signal, which is the streamlined goal of the time-scaling process. However, one might argue otherwise. The new, output signals (of both algorithms) sound different from their predecessors

(obviously); so, I had and have a difficult time distinguishing frequency changes, if such changes actually do arise. More testing is listed on the agenda.

In any case, both algorithms (*SOLA* and *phase vocoder*) allow for signal processing, new music, and a foundation for more research. The *phase vocoder*, a historical approach, is still being mended and redressed—a plethora of implementations exist; it's almost ridiculous. Also, I added a third case to this time-scaling project/function. It is comprised of the same phase vocoder algorithm that I used in case two, but with a catch: it decimates the input signal before algorithmic processing. Decimation low-pass filters the initial waveform and then downsamples it. This procedure allows for swifter computation. The output waveform is played at a lower rate, but it's just another option, permitting the user to go further and farther.

So, this was my project. The function is indubitably usable: I used it to redefine samples from Ennio Morricone's *Ecstasy of Gold*, which is awesome!

Works Cited

Dorran, David. Audio Time-Scale Modification: PhD Thesis. Diss. Dublin Institute of Technology, 2005.

Loscos, Alex. Spectral Processing of the Singing Voice. Diss. Pompeu Fabra University, 2007.

Sanjaume, Jordi Bonada. Audio Time-Scale Modification in the Context of the Professional Audio Post-Production. Diss. Pompeu Fabra University, 2002.

Zolzer, Udo. DAFX: Digital Audio Effects. England: John Wiley & Sons, LTD, 2002.