Audio Timestretching

Adam R. Lederer
Thomas Jefferson High School for Science and Technology
Alexandria, Virginia

June 12, 2006

Abstract

Audio Timestretching is the process of modifying a selection of digital audio so that it plays back at a different tempo (i.e. is longer or shorter). The trivial method is to simply extend the length of each sample (or an interpolated equivalent for non-integers), but this changes the frequency of the audio, which is an undesired side-effect. The goal of this project was to change the tempo of the selection of digital audio while maintaining the greatest possible subjective similarity to the original selection, avoiding the modification of pitches and the addition of artifacting.

1 Introduction

Most current timestretching algorithms work best on monophonic material as such, I decided that the main focus of my research would be to separate individual frequency elements from each other. To do this, I decided I would build frequency signatures by going to transients (i.e. hits, notes) and identifying the frequencies which appear or change at these locations.

The algorithm consists of four parts. After an initial pass-through to build initial signatures, the second pass transfers all of the appropriate signals to the signatures of best fit. In the third pass, what's left (which should be any signals that enter gradually and without transients) is divided similarly into frequency signatures. The fourth pass separates these.

After all (or most) of the amplitude of the signal is transferred into individual frequency signature audio samples, a comparatively simple interpolation in the frequency domain can be applied to each harmonic of each frequency signature, and the audio can then be summed back together

2 Background

Timestretching is currently quite possible in certain situations. A program entitled "Melodyne" by Celemony seems to be at the forefront of timestretching technology, as well as pitch-shfiting - it works with very little artifacting, but is only for use on monophonic material, and its calculation isn't performed in realtime. There are many algorithms, including granulization and its somewhat equivalent analog of frequency-domain analaysis and subsequent shifting, and there are many applications and plugins that aim to implement it. However, I endeavored to implement my unique approach, and to code the application in a manner that could be cross-platform and conceivably real-time ready.

3 Technology Used

3.1 JUCE

JUCE is a cross-platform audio/video/GUI library that is currently being developed by Jules, of Tracktion fame. Tracktion itself was implemented on a JUCE framework. I chose to use JUCE because it's a C++ library, and therefore should be suitable for real-time use, as well as because it's cross-platform, so that I could minimize the porting work. JUCE also has the capability to extend its framework for use in VST plugins, which would be useful for the real-time version.

3.2 Linux

Linux is an open-source operating system on which I decided to do my main development. I chose to develop on Linux mainly because the Computer Systems Lab here at TJHSST runs Linux - it was the only real option.

4 Progress

The development progress didn't go smoothly - not smoothly at all. From the beginning, I was facing incompatibilities and source code errors beyond my ability to easily deal with - I did surmount them, though, surprisingly enough, enough to get JUCE compiled and installed in my home directory (which helped me produce some source code error reports to Jules, the JUCE developer). After that, I spent the rest of my time (which was rather short, discounting the time necessary to develop research materials and documentation, as well as the time I spent snagged by incidental bugs and administration issues) developing a number of prototypes.

In each prototype I endeavored to explore a different aspect of JUCE, in order that I be able to code the final product. First, I developed a prototype to explore the basic application framework of JUCE - then I researched and implemented some of the component/GUI/visual functions. In a third prototype I focused on action callbacks and user response. Most importantly, in my fourth prototype, I looked into JUCE's audio playback functions, but found them extremely confusing. Later I would discover that I had been experiencing the effects of a mismatch between documentation version and library version, but at that point I knew nothing of that, and proceeded to be frustrated continuously in attempting to implement audio playback and audio device queueing until the end of my research experience.

5 Results and Discussion

The TimeStretching concept I had in mind never panned out, but I will readily describe my research experience as having been very educational. The code itself wasn't entirely above my level, so that I could handle it, but the administration and incidental challenges with which I was faced were trials I was totally unprepared for, and as such I learned a great deal in attempting to sort them out. I do, however, have some interest in implementing the TimeStretching algorithm I've developed at some point in the future, as well as in doing some more intensive (and real) research. This is work that a future party could aim to take on, although if that party is not myself, it's unclear as to whether my experience in working on this project would be of any use to them.

References

[1] Celemony, "Melodyne", *http://www.celemony.com*

[2] Wikipedia, "TimeStretching", *http://en.wikipedia.org/wiki/Timestretching*

[3] John Arroyo, "Phase Vocoder", *http://dsp.mixin.com/index.html*