

TimeStretch Prototype



Audio Timestretching

The objective of the project is to lengthen a selection of digital audio, so that it is slower, but so that it also maintains the subjective properties of the un-processed audio and contains a minimum of defects.

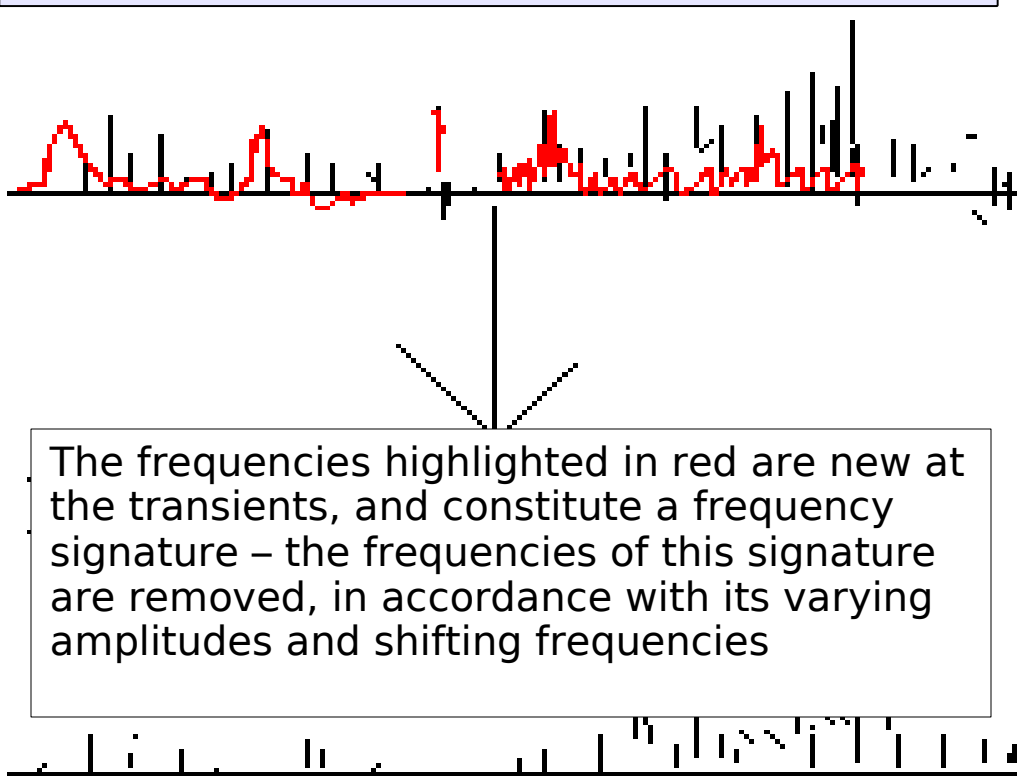
The Algorithm

"Frequency Signature Identification and Progressive Disintegration"

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



The Progress

I decided to use an audio/graphical interface library called JUCE, because it's cross-platform, but is a library for C++, so can perform faster real-time calculation than Java. The reader probably hasn't heard of it – that's because it's pretty underground and rather new (in fact, extremely new to the Linux). The going was tough, initially. The debugging process was difficult – there were perplexing compiler errors in the actual library code, which I found very surprising.

Coding proceeded at a rather smooth pace after I got the library compiled and the demo project linked and compiled correctly (this took, I believe, just the 1st quarter and a little bit of the 2nd). I created several demo projects and prototypes (see screenshot in upper left), each time exploring a different segment of the JUCE API and protocol. There were the phantom segmentation faults and the confusing hours of reading the documentation, but not anything I couldn't handle. A big snag-up came when I tried to implement audio, involving a confusing mix-up with documentation and library versioning and multiple attempts to create a simple audio player.

Then time was up for the project, at least until late in 4th quarter – what you're reading right now is part of a long process of content creation and paperwork that currently precludes the necessity to get farther with the project. I hope this information will give you a good sense of the techlab timeframe!

Comp-sys Techlab Tips!

Many of you reading this poster will be endeavoring to complete a comp-sys techlab project of your own some time in the future. Here are some tips that some of you may find useful (and some may not):

- Use Java

Why use Java? Why, for a number of good reasons! Putting aside all actual language preferences or capabilities, Java may make life easier for you in a number of ways:

1. It's largely self-contained. That means that, for most purposes, you won't have to scout out, research, download, compile, install, learn to use, debug or beta-test any external libraries or API's – and those activities can bog up roughly 100% of your time.
2. You're working in the system provided by the computer systems lab, not in the comfort of your home system – you don't have much control over anything, and you don't have much processor power to work with. Java and all its fixings are already set up for you here, and you won't have to worry about finding an IDE or messing with the settings – and if something in Java breaks, dozens of people will notice and complain. If you use C++ and want a good IDE, or don't want to have to wait for the sys-admins to come and fix things that you don't have access to, you're going to have to have a rather good grasp of Linux to be able to install and maintain everything you need in your home directory.
3. Chances are, there will be people in your class who are much, much, **much** better and more experienced coders than you are. Chances are, they know Java, and are willing to help you – and you don't want to be stuck trying to explain the intricacies of this or that obscure library to someone before you can get the help you need.

- Keep Track

The most important thing is that you have enough words written down to document whatever's happening – even if you feel you're not doing anything particularly important or reaching interesting conclusions, you still need to document as much as you possibly can so that your supervisor can assess your performance. You have to have written, concrete material – otherwise, your grade could be anything, and it's probably not something you want to take a chance with.