Techniques for Virtual Instrument Development

Darius Alexander Johnston Chatfield

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Masters in Music Research

September 2017

Contents

Included Assets

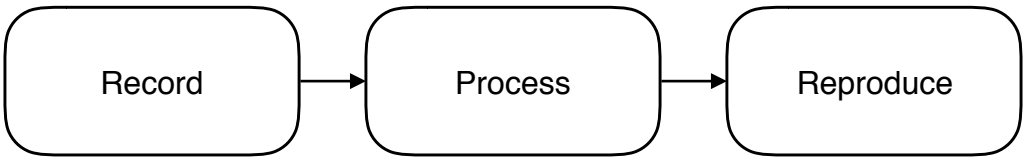
- PDF - Techniques for Virtual Instrument Development
- java-classes
  - README
  - classes
- Software
  - Dynamic Time Warping
    - Application
    - Source code
    - User manual
    - Video tutorial
    - Video demonstration
    - Interactive demonstration
    - Walkthrough
  - IR Timbral Modification
    - Application
    - Source code
    - User manual
    - Video demonstration
    - Walkthrough
  - Sample Rate Retuning
    - Application
    - Source code
    - User Manual
    - Walkthrough
  - Metadata Mining
    - Application
    - Source code
    - User manual
    - Interactive demonstration
    - Walkthrough
- Samples
  - Unprocessed Samples
  - Processed Samples

Word count: 13,803

1 Introduction

The project seeks to address the question of how to produce sampled orchestral instruments with high timbral flexibility whilst attaining high quality and realism with a relatively small data storage demand than current commercial solutions. This problem arises as modern orchestral music production demands an ever wider variety of high quality sounds, pushing up the demands on computer resources and the time cost of producing virtual instruments. This project addresses how a small number of samples can be used to meet the demands of virtual instrumentation.

Production of orchestral virtual instruments can be broadly split into three stages:



1. Recording:

   For example, this may be carried out in a dry studio or concert hall with a wide variety of microphone selection and position, or may involve capturing non-standard musical sounds such as instrumentalist breath intake or valve sounds for brass instruments.

2. Processing:

   This may include noise reduction, normalisation, cutting, and naming at a basic level, but could involve tuning, EQ, or other processing.

3. Reproduction:

   A sample engine follows DAW commands to manage playback of samples such as commands on sample selection, volume, pitch bend, effects, etc…

This project focuses its attention on developing techniques for stage 2, processing audio in non-traditional ways to deliver a commercial solution within the practical limitations of stage 1 and 3. Stage 2 was identified as an under utilised stage, acting as a conduit for transferring the characteristics of performed sound in stage 1 to reproduction in stage 3. Traditionally, orchestral sample instruments have been produced with a focus on stage 1 and 3, often with only minimal processing usually confined to editing cuts and clean-up (eg. replacing unsuitable samples, de-noising, etc).

The techniques developed for stage 2 in this project are:

- Dynamic Time Warping – resampling audio to maximise phase correlation
- Timbral Control – use of convolution to create different spectral characteristics
- Sample Rate Retuning – a method of pitch correction via a variable resampling rate
- Metadata Mining – using DFTs to extract and process pitch envelope information

The project started out with the intention of recreating characteristics of human performance by developing techniques to be carried out at stage 3. However, to achieve the project goal of high flexibility and realism without the use of extensive computer storage resources, the samples delivered by stage 2 (and implicitly recording in stage 1) were required to follow specific rules. For example, much of this project addresses the problems with providing flexible variation in replication (stage 3), being that crossfading of any two samples (such as dynamic layers) without precisely matching phase would produce phasing issues; a chorus or doubling effect, being particularly problematic for the realism of single voiced instrumentation.

To solve this doubling problem required two techniques: Dynamic Time Warping (DTW) and Sample Rate Retuning (SRR) which deal with phase and pitch conformity, while Metadata Mining (MDM) deals with the reintroduction of realistic human vibrato characteristics to sample instrument performance in stage 3. The final technique, Impulse Response Timbral Modification (IRTM), is similar to DTW in that it produces timbral variance with high phase cohesion, yet is capable of providing alternative benefits to the capabilities of DTW.

2 Dynamic Time Warping

2.0 Introduction

Music creators in DAWs may often require the ability to create realistic, expressive imitations of real instruments. Burton (2015) describes sonic perception being split into six parts; pitch, duration, loudness, timbre, sonic texture, and spatial location. Many of these perceptual characteristics are handled reasonably well by sample engines and DAWs, such as pitch, duration, and spatial location. One of the most important factors in this is the ability to control musical dynamics (often referred to as 'expression' in a MIDI context). Although traditional sampling methods have been capable of reproducing a note as a single static dynamic level since the early samplers, problems arise when dynamics are required to modulate throughout the course of a note for the sake of musical expressivity.

2.1 Exposition

The first of these perceptual problems is the relationship timbre has to loudness. It is entirely possible to play back a french horn sample at a brassy *fortissisimo* and change the loudness throughout the note to the loudness of a *pianissisimo.* But without the mellow timbre of the *pianissimo*, the relationship between timbre and loudness has in the physical world is not digitally replicated.

Two ways sample engines have traditionally dealt with controlling dynamics over user-defined parameters is to use a MIDI continuous controller (CC) to fade between two or more samples of different varying dynamics. The problem with this method is that when fading between two dynamic levels, slight variations of pitch between the two can create audible phasing/doubling artefacts where the separate voices can be distinguishable or filtering can be heard.

Another solution that sample library developers have used is the mass-content creation of many dynamic shapes, such as crescendos and diminuendos of varying length. This technique of sampling benefits from a natural sound, as the expression comes directly from the musician. However, this expression is fixed, which results in either libraries with limited expressive value or the production of libraries with a large storage and RAM overhead to provide convenience and flexibility to the end user (EastWest (2015) boasts 680GB of samples for their Hollywood Orchestra package). Large libraries are becoming less of a problem for users to accommodate as storage becomes increasingly plentiful but may provide problems for virtual instrument producers in the increasing size and cost of production.

One of the solutions proposed to solve the phasing problem, and the topic of this section of this paper, is to restructure the waveform of the sample to minimise phase differences between the crossfaded samples (Samplemodeling, 2007). When cross-fading with any mix of two or more processed samples, phase differences should be small enough to render any individual sample indistinguishable from the overall mix, having largely reduced comb filtering by aligning the phase. In order to produce samples, this DSP must be carried out before introduction to the sample engine as it is relatively demanding on computer resources.

This technique is expected to have several limitations to its use. As described in the introduction to this project, many of the limitations are derived from the interaction between the three stages. This processing tool (part of stage two) allows stage three (sample engine replication of sound) more flexibility of expressive material while keeping use of computer resources to a minimum. The caveat is that the nature of this real time resampling of expressive parameters (eg; dynamics, pitch bend, …) means any time sensitive audio information in the sample, such as reverb, must be added after the reproduction of sound by the sample engine. For example, if reverberation was recorded as part of a set of bassoon samples; G#3, at dynamics, *p*, *mf*, *f*, processed using DTW, and reproduced in a sample engine - the resulting dynamic crossfades would also crossfade the reverb for impulses that happened before the crossfade. For this reason, any change in dynamics would mean the reverb would fade in and out simultaneously with the dynamic layers instead of diminishing over time from an impulse, as is the expected behaviour of reverberations. Perhaps a more invasive problem would be a change of pitch over time to imitate vibrato; the same behaviour would be seen as with the dynamic layer, with reverberations being unnaturally retuned along with the direct waveform from the instrument. For this reason, the samples must be dry with reverb added after expressive parameters.

2.2 Process

The process is split broadly into three stages; onset alignment, phase analysis, and reconstruction. Each stage has several parts to complete which are mostly automated, requiring only minimal user input to guide or correct the process.

In onset alignment, the first stage, a master sample is chosen to be the template that all the rest of the samples (slaves) follow in phase and timing. In onset alignment, transients of notes can be defined. More precise phase analysis happens in the next stage to refine the position and pitch of the waveforms into phase. The final stage involves the creation of warped samples and finalising to complete for review and output.

2.2.1 Onset Alignment

The first major stage in this process is to identify the onsets of notes, defining a starting frame to analyse the rest of the sample.

The way the program deals with this is by setting up “sync points”, where the user can define where in a pair of samples the start of the note is. If sync points were not defined, the phase analysis would have no concept of where slave waveforms should relate to the master, meaning note onsets between the master and slave would likely start at different points when played back simultaneously, which would be detrimental to achieving a realistic sound.

An automated method for detecting onsets would be appropriate than manually defining onsets by hand, as it is likely a single advanced instrument could require many hundreds of samples. Finding a solution to detecting onsets included the consideration of a few different methods; envelope analysis and spectral energy analysis.

The Precedence Effect in psychoacoustics states that any two complex sounds heard less than 40ms apart can be perceived as a single sound, although simple sounds may require as little as 5ms difference (Wallach, Newman, & Rosenzwig, 1949. pp. 335-336). Although this psychoacoustic effect was tested for perceptual localisation, it divulges highly relevant information for the boundaries of sonic ‘fusion’ - perceiving two sounds as one. The method employed will have to be precise enough to detect the perceived onset within this timeframe. The onset detection system should aim to detect onset reliably within 40ms, if not substantially less than 40ms in the pursuit of accuracy.

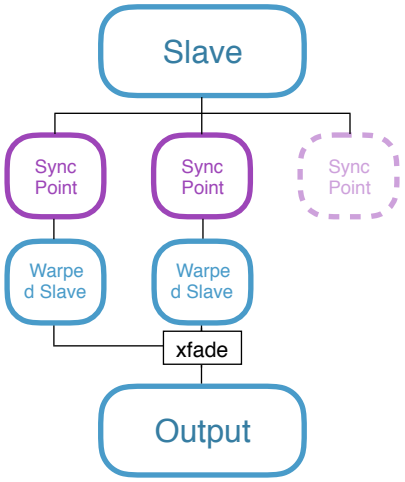
A range of methods for onset detection was explored and trialled, everything from a simple RMS threshold detection to more complex FFT window analysis and interpretation such as using adaptive whitening (Stowell & Plumbley, 2007) and negative log likelihood (Bello et al., 2005) to improve spectral analysis results. After exploring these options, a custom process was designed which avoided an intrinsic problem with ‘blurring’ caused by FFT windowing, yet produced more reliable results than simple amplitude threshold detection.

The developed process involves taking amplitude averages (ie; RMS) of windows across the whole sample, separated by a consistent ‘hop’ period. Change of amplitude is then calculated from each window to the next. The process then groups adjacent windows of the same direction of change (+/-) into groups, and identifies the group with the largest positive change as the onset. An additional advantage over using a set amplitude threshold is that the process works equally well for both quiet and loud samples, as it accounts only for changes in amplitude.

A shortcoming of this system producing unreliable results, is with samples of a complex amplitude envelope. For example, double tongued samples with two distinct notes or accented *fortepiano* notes. This method will not distinguish between the amplitude change of the first note and the amplitude change of the second note, and may therefore choose the amplitude change of the second note as the largest positive change group.

A second problem arises in identifying the *perceptual* start of a note. For instance, it is common in brass instruments to apply air pressure before the lips vibrate, sometimes causing a noisy ‘pop’ before a tone is created. Similarly, in string instruments, it can be likely to hear the bow hit the string before bow movement produces a note. These noises and in instances where the transient may be unclear or weak (eg; very soft attacks) not only causes problems for the algorithm, but are issues in subjective perceptual alignment.

The process was later adapted, starting a fundamental change in the way DTW handles samples. The aim was to detect releases of notes by searching for the largest negative cumulative sum group, as was done with onset detection but for the largest positive cumulative sum group. Providing this feature meant that a slave sample could be warped to match up with both the attack and release of a sustained note, preventing notes from finishing early. Making multiple sync points between a master and save meant that the sync points were required to diverge for their own phase analysis in the next stage, as they would follow an entirely different phase path. Not only would they require their own analysis, but would also require their own warping, thus creating one warped sample for each sync point between the master and slave. This would create a subset of slave samples for each slave (one warped sample for each warp point of the slave), which would require recombining to produce an output sample.



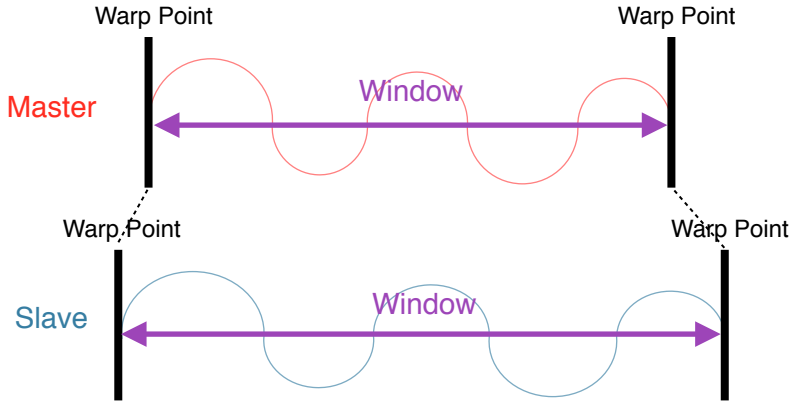
Also included at this stage was the functionality to add user specified arbitrary sync points for many-to-one relationships between the slaves and master sync points. For instance, the onset of a slave could be synchronised to the release of a master, which could be beneficial for creating bow direction changes for string instruments. It also lays the foundation for creating loop points, allowing samples to seamlessly crossfade on itself to an earlier point in the sample within the sample engine.

In conclusion, this model for controlling sync points is highly flexible and the method of onset/release detection is fairly reliable, but further development of a more advanced statistical analysis would likely yield much more robust results - especially for samples of a complex envelope.

2.2.2 Phase Analysis

The phase analysis stage is split into two steps; the “initial analysis” and “continued analysis”. Although onset alignment broadly aligns the amplitude envelope of the samples, the initial analysis is concerned with refining that to a phase accurate starting point, after which the continued analysis warps the rest of the sample to match phase against the master.

The analysis stores its results in a separate database (or “warp file” when saved to disk) for each sync point relationship. The entries in the database describe warp points - a sample position of the master and a sample position of the slave sample it relates to. A pair of “warp points” define a window for the master and slave, with the aim being that the waveform in the window is in phase. The idea of the phase analysis is that these warp points are repositioned in time to resample the slave to match the phase of the master sample.



2.2.2.1 Initial analysis

The initial analysis refines a sync point to a phase accurate position. This happens by analysing the correlation of windows of the waveforms against each other, offsetting the slave in time by different amounts to see where waveform correlation is at its most. Previous versions of DTW

analysis suffered from only taking into account offset which, while its effects were minimal on waveforms of very similar frequency content, any deviation to this rule proved difficult to identify phase coherent peaks using the single dimension (offset), and entirely failed when trying to align two samples of different pitch. Therefore a second variable was added to the analysis; stretch, which compresses and lengthens the slave waveform to analyse how well it matches the master. Adding a second dimension of stretch multiplies the processing time as, for every value of stretch, the program must reprocess all the offset values with the new stretch value. Using brute force calculations, the program compares the correlation (calculated by the product sum) of the two waveform windows, storing the result in the relevant cell of the matrix for the offset and stretch values. Improvements to this model may involve using interpolation on fewer analyses to cut down processing requirements.

Once the initial analysis has been completed, the best phase match (or “phase peak”) is automatically selected for user review. The program has been made to pause at this point for the user to review whether the automatically selected phase peak is correct. Onscreen feedback displaying the two dimensional matrix and the resulting windows of the master and slave permits the user to assess whether the desired correlation point has been identified, and allows a graphical interface to select a custom phase match if required. Much later, feedback of the preceding and following “continued analysis” was added (described in section 2.2.2.2) as it was found that this was useful information to inform the offset of the phase peak - as the harmonic content can differ at the onset compared to the stabilised tone, locking the initial analysis into an offset to the fundamental later on in the sample. An improvement to the process could automatically include the resulting product sum of the audio around the initial analysis to resolve this issue.

The initial analysis creates two entries into a database (aka “warp file”) with the initial analysis. These two “warp point” entries each describe a sample position that the slave relates to on the master sample, and are the positions that define the start and end of the initial analysis window.

Pitch identification was used with an attempt to provide automatic values for the initial analysis parameters. However, inaccuracies in pitch detection, especially when analysing transients, meant this created more problems than it solved. A more advanced model with statistical analysis could provide a stable and useful feature in future.

2.2.2.2 Continued Analysis

The continued analysis picks up where the initial analysis ended, by using one of the warp points from the initial analysis and creating a new warp point further on in the sample to define a new window. The warp point set out in the initial analysis is considered to be in phase, being one of two

warp points of a window that has been determined to be in phase. Therefore the continued analysis only requires the new warp point under scrutiny to be adjusted in time, lengthening and compressing the waveform and storing the product sum in the same way as the initial analysis. This analysis is one dimensional, as the new warp point position is the only variable. Once the new warp point position has been determined by calculating the peak phase, the new warp point is added to the warp file and the analysis repeats, using the newly created warp point as a constant and creating another warp point for analysis.

The parameters of the continued analyses can be defined separately from the initial analysis and generally may have window sizes much larger and of relatively small stretch range. The stretch range can be significantly decreased as the continued analysis attempts to guess the next warp point via linear extrapolation, using the estimation as the centre of the stretch values in the matrix. The benefits of this feature become *much* more pronounced as the requirements of stretch modification increase, such as warping two notes of different pitches together or unstable tuning. This also helps the analysis stay locked to the fundamental defined in the initial analysis, preventing the slave attempting to switch to locking to a harmonic that may have a better phase match.

This model was subsequently adapted to create warp points flexibly, going both forwards and backwards in time from the last known warp points. This was essential for the adaptation earlier described for allowing sync points to be set at the end of samples, and also allows sync points to be set in the middle of samples. The program automatically identifies when to switch analysis direction and when it has completed the analysis. On completion of the analysis, the program automatically linearly extrapolates an extra start and end point that covers the full range of the slave sample, and saves the warp file to disk with a unique name describing the master sample, the slave sample and the synchronisation point it refers to.

This method does have several flaws. Most notably is that the analysis was only built to work in integer sample positions, which means samples with higher pitch and smaller window size can be adversely affected by inaccuracies. This could be observed in the data, by viewing how the slave sample had been warped over time, often seeing the number of extra samples it had to stretch alternating between two adjacent integers as it over-compensated then under-compensated, causing fluctuations in pitch in consecutive windows.

This could be resolved in several ways: Firstly, by using a floating point sample position, which could be interpolated to find the peak of maximum phase by finding the zero crossing of the first derivative of a quadratic function. Secondly, by reverse engineering the interpolation method used

in reconstruction. The interpolation method, Catmull-Rom spline (explained in section 2.2.3), would require heavy CPU use due to the number of coefficients required for each sample calculation, but would analyse exactly for the results being produced by the warping algorithm making the algorithm extremely accurate. A compensation on the latter solution could be made; firstly by carrying out linear interpolation to find warp points, the set of warp points could be improved by using the Catmull-Rom spline to refine the positions of each warp point, perhaps over several passes of the dataset.

2.2.3 Warping

Once warp files containing warp points have been collected, they must be interpreted by appropriate means for audio output. Significant time was spent on one major aspect - research into interpolation methods: During the development of earlier versions of DTW it was found in order to interpolate between the samples between two points, one must make two interpolations. Firstly interpolate the sample position (floating point value), and secondly interpolate the resulting sample amplitude using adjacent samples to accurately represent the warped waveform.

When considering sample position interpolation, one of the most basic forms, linear interpolation, was explored and dismissed quickly as the acute changes of gradient (*f'(x)*) created an audible sudden modulation in pitch as the windows adapt their rate to reduce phase difference.

The next method of interpolation studied was polynomial interpolation (Bourke, 1999). Limitations in the flexibility of the programming language, Max MSP, was identified which resulted in learning and utilising Javascript, creating code that could build polynomials to a variable *n* exponents, defined by the number of coordinates fed to the method. It was found while polynomials to the third, fourth or fifth power resulted in significantly smoother curves, this still resulted in a too sharp change of *f'(x)* when transitioning between windows, although significantly less pronounced than linear interpolation. As more coordinates were added to the polynomial, it required additional exponent powers to be added to the polynomial. While this increased its smoothness, it decreased the function's stability due to Runge's phenomenon, occasionally causing *wildly* fluctuating pitch.

Bezier curves were explored as a method of creating yet smoother curves, but the shortcoming with Bezier curves were that they did not pass through all the data points, and became ever-more divergent from the set points as more ordinates were added to the method. By not passing directly through the warp points, the effectiveness of the initial analysis and continued analysis was being undermined.

The next iteration of interpolation explored was of a subset of hermite spline called a Catmull-Rom spline (Bourke, 1999) (Van Winkle, 2005). This method improved on the Bezier curves by passing through each data point, while maintaining an absolutely smooth gradient transition from one window to the next.

Once a sample position is interpolated in time, a sample amplitude must be calculated to write to the new phase-aligned sound file. This process of interpolation followed very much the same process as interpolating the sample position resulting in using the use of the Catmull-Rom spline. This could potentially be simplified to a cubic interpolation, as Arnost (n.d.) suggests there is little difference in quality between cubic and hermite interpolation for sample amplitude calculation.

However, the Catmull-Rom spline could be deemed a highly appropriate interpolation method for use in audio (Niemitalo, 2001. pp 12). It is a CPU intensive calculation therefore should only be carried out in offline processing. The Javascript code, for a number of the interpolation methods above, were later reprogrammed in Java, significantly improving calculation speeds and reducing processing time by around 10-15 times. This is not only an economic but also an ergonomic improvement to processing the hundreds of samples likely needed to make an advanced virtual instrument.

2.2.4 Crossfades

Each sync point between the master and a slave creates a unique sample based on the waveform of the slave (discussed in section 2.2.1), but matching the waveform phase and sync point timing of the master. As the slave-to-sync-point relationship is a one-to-many relationship, multiple samples can be created from the slave sample sync points, giving each slave a subset of warped samples that each have a unique sync point timed to the master. To create an output sample, the subset must be combined in a way that ensures the correct sample from the subset is heard at the correct sync point - in time with the related master sync point.

This process is largely automated, as a few predictions have been made. Firstly, the timings of the sync points are known, allowing the program to determine the order of the subset, and points to start/end crossfades between the subset samples. Furthermore, the phase relationship of the subset is known, as all are based on the waveform structure of the master and the harmonic content is expected to be largely the same over the sample, meaning the crossfade type should be linear to avoid creating unequal amplitudes over the range of the fade. Another assumption that can be made of crossfades is that they should avoid being too close to any sync point. Testing and experimentation showed that the crossfade length often worked well between 100ms and 500ms.

The program uses these assumptions to calculate position and length of the crossfade, providing on-screen feedback to allow adjustments.

2.3 Limitations

This technique was expected to have limitations, which were confirmed in testing. Most notably, the instrument samples must be reverb/reflection-free, monophonic, and solo. Because the nature of time-domain warping, any resonating or noisy sounds such as bow noise, reverberation, or sympathetic resonances will be warped and re-pitched along with the prominent waveform.

This technique also has shortcomings at warping samples with reverb for another reason; the harmonics of the captured audio do not remain in phase with each other as complex reflections and reverberations from the room construct and deconstruct, affecting phase in an unpredictable way. This effect was seen by filtering single partials and examining their relative phase movement next to the filtered fundamental. Dry samples did not exhibit this problem.

2.4 Conclusion

The program can successfully convert a set of samples into a phase-matched and time-matched set. While this technique has benefits, the caveats are that no recorded ‘real’ reverb can be processed, and only some performance characteristics can be captured. For example, a set of static-pitched dynamic sustains could be processed, but human vibrato could not be included because of the pitch unifying nature of DTW.

Demonstrations of warped samples can be found in the Dynamic Time Warping folder inside the submitted materials.

3 Impulse Response Timbral Modification

3.0 Introduction

In creating music, it can be advantageous to crossfade between two samples, such as dynamic layers or articulation technique. This use of musical modulation is a widespread feature in modern virtual instruments, but is not always carried out in a way that provides optimal results for sonic flexibility or quality. For example, it is a well known issue that crossfading can often create phasing or doubling artefacts. Below is just one example of many forum posts describing problems with even the most popular and expensive commercial virtual instruments (Bachrules, 2014) (SyQuEsT, 2006). This timbral modification tool aims to resolve inherent phase issues with crossfading samples of similar pitch.

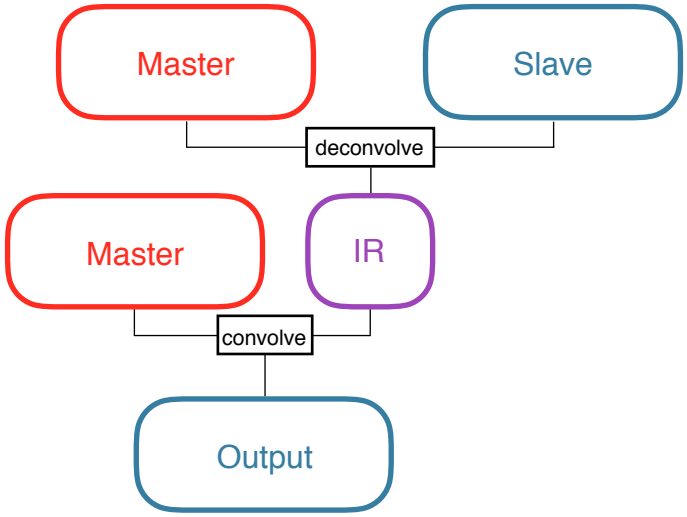
Impulse Response Timbral Modification (IRTM) is intended for making one sample sound more like another sample while maintaining a linear phase relationship, so that on simultaneous playback, the samples may be cross-faded to any degree with no phasing. While this technique is similar to DTW in terms of outcome, it is intended to be significantly faster, but with some reduction in reliability and flexibility of outcome. These attributes make this tool appropriate for processing material of secondary importance, such as a ‘vibrato layer’ (modulating between a slightly different tonal quality to the sustain sample), articulation technique (eg; sul-ponticello, mute, or harmonics), or perhaps even to replicate a ‘bad’ sample (eg; using a *forte* sample to modify into a *mezzoforte* sample that was badly performed).

Essentially, this is an improvement on using a manual EQ curve to modify a sample sound - for example increasing a high shelf band to imitate harsher harmonics of louder instrument dynamic. To accurately change the sound of a sample, IRTM uses computer analysis to create a high definition EQ automatically through the use of impulse responses.

These samples will have to go through preprocessing before being introduced to the sample engine, creating separate samples that can be crossfaded for timbral control. It is important that any modifications to frequency does not offset the phase of the waveform, avoiding phase issues and comb filtering on simultaneous playback. To maintain phase relationship between the samples, the filter type should be linear.

3.1 Exposition

The concept IRTM uses is a comparative analysis of frequency responses by use of deconvolution. Take an example of two samples, a master and slave; by deconvolving sample the slave over the master, an impulse response (IR) can be calculated that gives the master a similar frequency response as the slave when convolved with the IR. The resulting waveform is that of a sample that approximates the same “energy-time” shape of the master, but with the “frequency-energy” of the slave - In more simple terms, the master sample is filtered to sound like the slave. Phase relationships between the newly convolved sample and sample A are dependant on the phase attributes of the deconvolved IR, which in this case can be calculated as a linear phase filter. This linear phase filter is key to this technique, as any adjustments to the phase of the filter may cause undesirable phase patterns between crossfaded samples in the sample engine.



As with DTW, this tool aims to have the capability to process several dry mono channels of coincident-mic’d samples to allow a user-defined mix of channels to be played back in the sample engine. Due to the comparative nature of deconvolution, this, in theory, is entirely possible even with non-coincident microphones and with reverberations.

3.2 Process

As a starting point, the HISSTools Impulse Response Toolkit (2012) provides a means of creating and editing impulse responses in a flexible and modular way. IRs are more often used for capturing the reverberation and frequency profile of a space but may also be used to measure the frequency response of an acoustic object such as a speaker.

The ITRM software features controls for a number of parameters. Master sample selection, phase, crop, and smoothness are accessed in the interface to affect the IR of the slave sample deconvolved over the master sample. Each control has benefits and caveats, and the user must decide which trade-offs another parameter may resolve.

The phase control is perhaps the most important attribute to the impulse for the intended use, as it determines the phase relationship between the master and output slave. It also affects where the pre-ringing and/or post-ringing occurs. For instance, a percussive sound should generally be convolved with a minimum phase impulse to avoid pre-ringing and protect the transient. Maximum phase is generally not used. As a default, the program is set to linear phase as this is the required setting for the intended use of creating linear phase relationships between samples.

Cropping the length of an IR is to balance how audible the ringing and pre-ringing is against the ‘resolution’ (quality) of the impulse. Over-cropping may result in an inaccurate IR as information is lost in the crop, while under-cropping may result in pre/post ringing.

IR smoothing can be used to remove inconsistencies and sharp, sudden changes in the frequency response which may result in fluctuating frequencies or harmonics (Wilkinson, 1998). The use of more smoothing can make pre/post ringing more subtle, as the energy of the IR becomes more clustered towards a point.

Setting up the correct process order was essential to avoid losing data, creating the highest quality outcome. For instance, cropping the length of the IR removes data, (losing ‘resolution’ in the low frequencies at first, with higher frequencies affected as more is cropped) - one might expect that retaining the most frequency domain information yields the best result; but not for IRTMs intended use. The nature of filters causes ringing; the more extreme the gain of a frequency, the more it rings, with the phase of the IR determining the balance of post-ringing to pre-ringing.

There are two different types of smoothing the HISSTools provides: high and low quality. High quality smoothing takes much longer to process. It may be worth adding the option for a first pass of low quality smoothing before cropping, although this may not be conducive to the goal of attaining a high quality outcome in an efficient way.

While IR smoothing can reduce the ringing of the filter, getting the best aspects of both resolution from a long IR and reduced ringing from the filter is not entirely viable due to the processing time involved for smoothing an uncropped IR, especially with multiple samples queued. In testing, only

a small amount of data was lost in cropping the IR length pre-smoothing, therefore it made sense that smoothing was to be carried out after cropping.

A further cropping is required after convolution of the master sample and the impulse response to ensure the centre of energy of the IR does not delay the phase of the newly created convolved output sample. If this was not addressed, the newly created sample may not (in instances where phase is not minimum) be delayed by some percentage of the impulse response, dependent on the phase setting and IR length. As a result, when the newly produced samples are played back simultaneously they would not be remotely in phase, thus defeating the purpose of this tool. The mathematics behind this is simple, needing only to cut the beginning of the newly convolved sample by the length of the IR, multiplied by the phase of the IR (expressed between 0. & 1.). This cropping throws up new problems such as the possibility of the sample failing to start at a zero crossing, resulting in an audible click when playing back the sample. This can be solved with a simple fade, which is solved automatically by the softwares feature with a fade-in and fade-out. The impulse response can also extend the sample. Although this does not undermine the purpose of this tool, it may be worth providing options to crop and fade the end of the sample to maintain equal input and output sample lengths.

After convolution, it is likely that some of the produced samples will peak and distort in clipping file-types such as 16-bit & 24-bit audio. As the samples are calculated in float-32, they are a non-clipping format which can be normalised to fit within range of a standard file type without error. A few solutions to this may be normalisation, group normalisation, or limiting. Consider an example of three crumhorn sustains on A3, *piano*, *mezzoforte*, and *forte*. Using the *mezzoforte* as a master, impulse responses are deconvolved, processed and convolved with the *mezzoforte* sample to create three dynamic samples with linear phase. Assuming the input samples are at recorded gain (not normalised), the impulse responses created would affect the *mezzoforte* sample to create louder and quieter convolved output samples in imitation of the *piano* and *forte* samples. In this case, it would be appropriate to provide a group normalisation within a context of that particular note, A3. However, group normalisation per note in this way may provide inconsistencies with a separately processed note, such as the three sustains on A#3. To further complicate this, if multiple microphone channels were to be used, normalising without considering the peak amplitude across all microphones and samples could mean different channels become unpredictably louder relative to the other channels. The only two seemingly rational ways of maintaining a normalised and consistent sample set (without manual post-production adjustments) is to normalise from the maximum peak of all the samples across microphone channels, notes, and articulations, or to normalise each sample to its own maxima, then compare original and convolved samples to produce a table of metadata that the sample engine can use to adjust the gain of each sample to

an A-weighted level. Considering the wider projects planned heavy use of metadata in areas such as pitch and vibrato envelopes (covered in section 5), the latter may be preferable. The impact these different situations will have on the program will require a flexible process whereby a user can essentially choose which elements to include in a normalisation sub-group from a choice of articulation, channel, or note. However, such a complex feature is not necessary for the current stage of the project, but would add functionality vital to a sampled instrument project being produced to different specifications.

3.3 Conclusion

The software can successfully alter the frequency response of a sample to match the frequency of another with a linear phase relationship. However, the effect of filter ringing can be a substantial problem for percussive notes unless sacrificing the IR length (therefore resolution) of the impulse response, or moving the phase relationship towards minimum (losing benefits of linear phase).
At the current stage of development, the software is more ideally suited to sustained, static pitched sounds, but further developments could improve the overall performance of the program. This might include the ability to divide a sample into multiple parts, applying the IRTM process to each one separately. This may provide benefits such as providing more focused frequency modifications, and the ability to change the settings for each part of the sample. For example, the transient of a sustained note can be set to have low pre-ringing, the body of the sustain could have a longer (therefore accurate) filter, and the tail and reverberation of the sustain can be compared specifically to that of the other sample.

4 Sample Rate RetunIng

4.0 Introduction

The sample rate retuning (SRR) program was designed to retune multiple coincident microphone channels to steady pitch, while maintaining their phase relationship. This method of tuning was explored as available tuning methods from commercial software does not provide the guarantee phase will be kept between samples. Like DTW, sample rate retuning can be used to reduce the tuning difference across crossfaded velocity layers but without a phase-locked relationship to a master, which helps reduce audible phasing artefacts.

Although this method may appear to be inferior to DTW, this method of tuning becomes vital when there is no master waveform to guide phase difference reduction. For instance, sample rate retuning is an optional stage to be performed on master sample(s) before DTW, reducing tuning inaccuracies that would be mirrored in the slaves to the master if they were not corrected beforehand.

Other instances in which this situation arises is in legato samples consisting of two notes; a launch note and a land note most often of differing pitch. Often, a launch sustain sample is being played when the legato sample is crossfaded in on the launch note, then plays through to the landing note, and crossfaded to the landing sustain note. While it may be possible in future versions of DTW to match phase of the landing note and landing sustain to make the crossfade seamless, the launch note must be ignored by the DTW process as not to retune it. This results in the launch note having the potential to be out of tune, clashing with the launch sustain and causing comb filtering for the duration of the crossfade. It would not be possible to replicate DTW for a launch note and legato sample, as the time in which the crossfade must happen is undetermined until the sample engine calls the order. In which case, the best that can be done is to match the perceived frequency of the launch sustain and the launch note of the legato sample, reducing comb filtering and only causing an unknown amount of phase change of the waveform during crossfade, potentially causing constructing or deconstructing frequencies. It would be interesting to see SRR make its way into the integrated workflow of DTW for processing legato samples.

Another use for SRR could be to unify differently pitched notes to one pitch. For example, by tuning with SRR and the addition of DTW processing, a set of trombone sustains could be cross-faded in a sample engine as the playback rate is modified to match their original rate. How this can be achieved would be to tune with SRR the seven positions (each sounding a semitone apart)

belonging to a lipped harmonic on a trombone. Each of these would use preliminary SRR processing to resolve them to a single pitch before alignment in DTW. As these samples would all be able to crossfade with each other with minimal phase issues, a glissando could be controlled in the sample engine by varying the playback rate of all samples in unison and fading in the sample that most closely represents the original pitch. The intention of this process is to maintain the nuance and characteristic of each note while allowing the user and engine maximum plasticity with minimal detriment to the quality of sound.

The benefits of using a time domain retuning method such as this over a frequency domain method, such as using an FFT (fast Fourier transform) pitch transformation, include the guarantee of maintained phase relationship and benefits of retaining the original sonic characteristics of a sample. For instance using FFT based processing with a long window is suitable for sustained sounds, however, long windows are known to be destructive to transients such as note onsets and releases. Designing a more sophisticated FFT method was explored, intending to use two or more different analysis windows simultaneously designed for a transient and a sustained sound, analysing the spectral continuity to generate a cross-fade envelope between the simultaneous iFFT'd waveforms. However, designing such a complex system currently has no benefit over a more simple solution such as the time domain method proposed.

4.1 Process

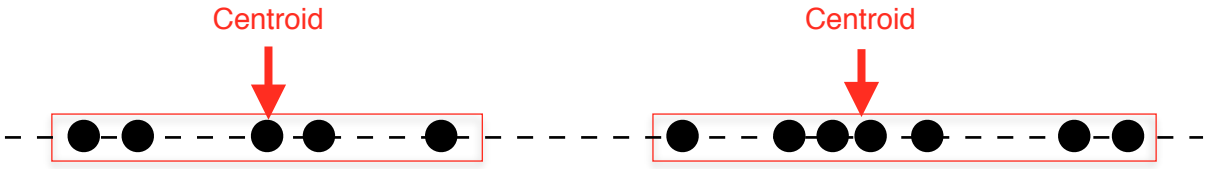
To modify the sample to the correct rate (and thus pitch), an envelope describing the perceived pitch of the sample must first be identified. For this, IRCAM's ircamsdescriptor~ object is used to obtain frequency information, along with other useful information such as waveform energy. Designing a frequency analysis tool was deemed an unnecessary step as existing technologies could be quickly implemented. Although fairly reliable, pitch detection using the ircamsdescriptor~ object can result in inaccurate or sporadic pitch detection especially at low amplitudes. Several offline mathematical processes have been implemented to reduce or negate the effects of these inaccuracies including K-means clustering, a variation of the Ramer-Douglas-Peucker (RDP) algorithm, and energy-based rate scaling.

4.1.1 K-Means Clustering

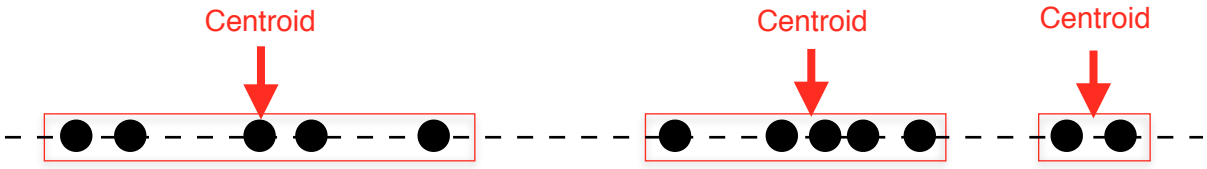
K-means clustering is the first step towards a more sophisticated and capable tuning method, allowing for detection of multiple groups of data (Likas, 2003), such as multiple notes as seen in legato samples. The automatic pitch analysis executed on the sample would benefit from an analysis technique that acknowledges that there maybe be more than one note, and the possibility of erroneous data. For example, the previous method tested, filtering pitches that fall outside a standard deviation of the mean of detected pitches, works well for filtering the range of pitches

detected (ergo potential errors) in a sample of a single note. However, this often falls short of the requirements when there is more than one perceived note, such is the case in legato samples. K-means clustering is an unassisted machine learning technique used to split similar data into a specified or calculated number of data clusters (Jain, 2010), which is a more appropriate technique for dealing with pitch data extracted from a sample.

The initial step of analysis involves determining how many clusters are to be analysed for. This could be a user defined number based on knowledge and feedback about the sample, or could be based on the outcome of analysing for various numbers of clusters and determining how many number of clusters best define the dataset (as discussed later in this section).



12 datapoints (black) split into 2 clusters (red)



The same 12 datapoints split into 3 clusters.

Having decided on a number of clusters, an initial centroid is defined for each cluster. The centroid defines the centre of a cluster of which all the datapoints closest to that centroid belong to, calculated by a means of distance to each one. If a datapoint moves too close to another centroid, the datapoint can become part of the other cluster. There are several methods of defining initial centroids that affect the resulting clusters. Two methods were considered here; the Forgey method (Forgey, 1965) and random partition (Hand, 2005). The Forgey method selects a random value for the centroid within the range of the dataset. Random partition randomly selects a datapoint in the dataset, setting the centroid to that value. Random partition has the benefit of every cluster being defined by at least one data point as a starting point, while the Forgey method risks clusters having no datapoints in them if the centroid is randomly placed somewhere with few datapoints. The preference in this instance is for the Forgey method, as data is not expected to be noisy but can be incredibly wrong often due to errors in misidentifying note harmonics as the fundamental within the ircamdescriptor~ object's internal calculation. The Forgey method ensures a more evenly weighted distribution throughout the frequency range, avoiding the catastrophic possibility of selecting a misidentified harmonic as a centroid. In using the Forgey method to randomly select a value in the

data range, it was noted that higher pitches had undue prominence to being set as a centroid due to the linear calculation of random frequency points used. To resolve this, a harmonic random function was developed to evenly distribute the initial centroids across the pitch range:

$$r = \min + \min\left(\log_2\left|\frac{\max}{\min}\right| r_0^1\right)$$

Harmonic random (r) uses the minimum (min), maximum (max) from the dataset, and a random variable between 0 & 1

Once centroids have been defined, each component of the dataset is assigned to the closest centroid, creating clusters of datapoints around centroids. In linear scales the minimum absolute difference between the centroid and the data point defines which centroid a data point belongs to. However, the difference calculation implemented here takes account that pitch is perceptually nonlinear, and instead calculates the difference in octaves given by this function:

$$d_{ij} = \left|\log_2(p_i) - \log_2(c_j)\right|$$

The difference (d) is calculated using the i'th datapoint (p) to the j'th centroid (c)

Once these clusters have been defined, the centroid of these clusters must be redefined as the average of the data points. Again, in Euclidean mathematics, this average would be a mean giving k-means clustering its name, but may also be a median average; k-median clustering. However, both of these are not entirely appropriate a the nonlinear centroid calculation. Hamerly and Elkan (2002) determine a harmonic mean calculation is not only more appropriate in many instances, but mathematically correct for the nonlinear data here. The clustering method implemented uses a harmonic difference to calculate a new value for the centroid of the cluster using the datapoints belonging to the cluster, taking into account any datapoints that have joined or left the cluster. Once the centroids have been recalculated, the calculation of new clusters and centroids is repeated with until the centroids do not change with each repetition.

This analysis can be done multiple times to obtain more reliable results, as a random element is involved in this process. In this instance, the analysis is repeated a number of times, taking the median centroid as the definitive centroid for that cluster. The analysis can repeat not only several times for a set number of clusters, but can repeat for different numbers of clusters. Having obtained reliable centroids for various numbers of centroids analysed for, an analysis of these results can determine the minimum number of clusters that reduce error in the clustering. Usually rate distortion theory or v-fold cross validation is used to determine the minimum number of

clusters required, however, as the dataset is expected to be highly correlated clusters except for a tiny proportion of misidentified data points, a more basic calculation may be implemented. For any cluster that contains a relatively small percentage of the data points it can be determined that the data points (if any) in that cluster are highly likely to be misidentified harmonics, thus determining the number of clusters analysed for is at least one too many. The number of clusters analysed for can be decremented until all clusters hold a sizeable portion of the dataset.

Currently, the analysis of clusters involves only one dimension, frequency. Future versions of this clustering may also take into account the time where pitches were extracted from a sample, giving raise to the conceptualisation of notes existing rather than just frequencies in extraneous time. Further to this, additional dimensions could be added to time and pitch such as waveform energy or other attributes that could be supplied by the ircamdescriptor~ object. The benefit of including a larger dataset with more varied, and potentially more reliable, data includes the likelihood of a more reliable outcome but may result in an increased computation cost.

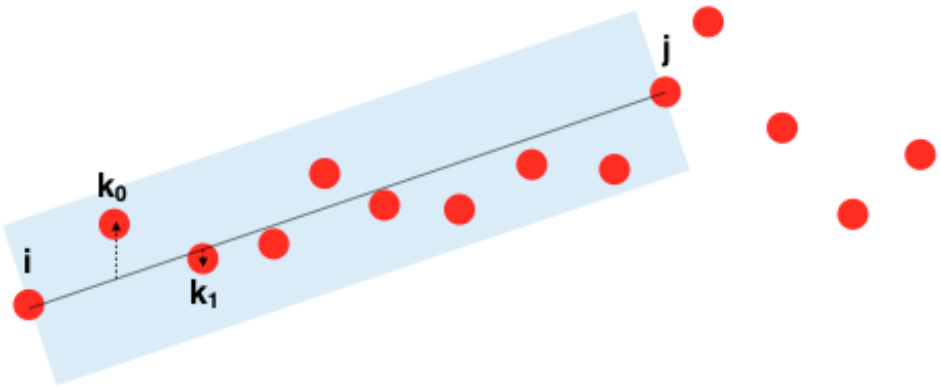
However, this model did not make its way into the final SRR software, as the methods described above suffered from inaccurate pitch detection entries in the dataset. While samples that returned few pitch detection errors showed fairly reliable detection of numbers of notes, single-note samples observed the misinterpretation of two notes with more than just a few errors in pitch detection. This implies that this method is somewhat reliable, but an error reduction stage prior to this could be of great benefit.

4.1.2 Ramer-Douglas-Peucker Algorithm

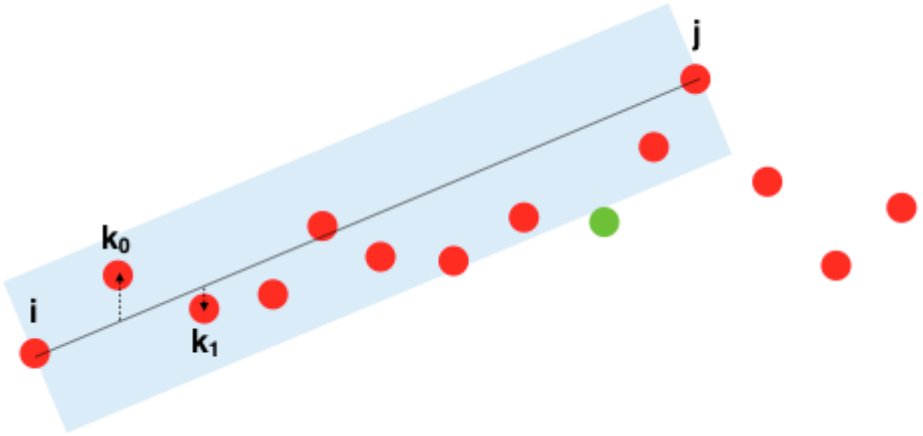
Having obtained pitch information from a sample, it was found that very frequent sample analysis often resulted in many small fluctuations of pitch. While the hop size of data returned into the database could be increased by the analysis attributes of ircamdescriptor~, this reduced the accuracy of pitch detected in areas of more rapid change of pitch. Slowing the rate at which the frequency is modified allows the sample to be tuned less aggressively, giving the opportunity for small amounts of pitch variation to remain. This may provide some ‘life’ in the sample, rather than a robotic resulting pitch. The amount of data was also detrimental to the performance of the program, often taking an unergonomic time to carry out calculations for display and function. The solution to this is to collect frequent data entries then reduce the dataset to the data entries that best describe the envelope, removing data points that contribute little to the detected pitch curve. Having researched methods to do this such as methods for low-pass filtering, Ramer-Douglas-Peucker algorithm (RDP) was selected as an appropriate method of reducing data entries that did not affect the overall shape of the envelope (Heckbert & Garland, 1997).

The basis of RDP is to linearly interpolate between the first and last points on the dataset, *i* and *j*, with an array of *k* points between them. The process then identifies the *k* point furthest from the interpolated *i*→*j* line. This furthest *k* point is then used to define an *i*→*k* line, where any other *k* point between them are discarded if within a defined range (as opposed to domain).

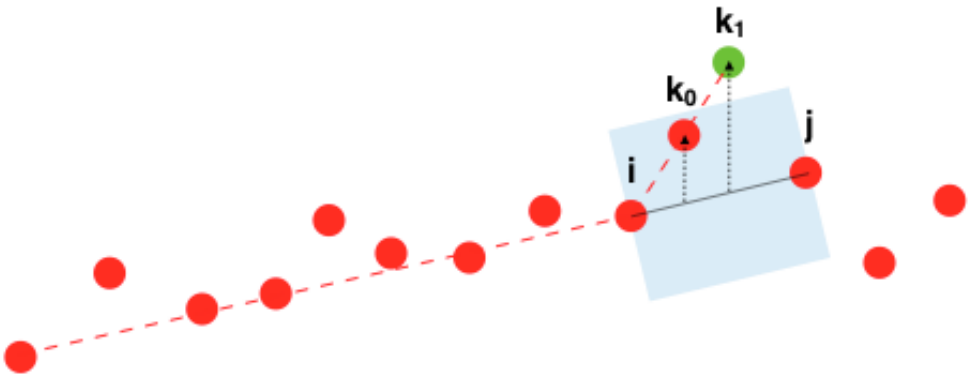
In testing with this data reduction method, some modifications were made to the RDP algorithm to suit this function's requirements. The modifications went through several stages of testing, instead focusing on a variable end-point, *j,* that iterates away from the start-point, *i*, and analysing all points in between array [*k0, k1, …*] to determine whether they are within a defined range (area shown below in blue) of the interpolated line between the start point and end point. Any point found to be outside the range is not rejected, but becomes the new start point. This is repeated until the end of the dataset.



Here all points fall within the range of the interpolated line between i and j. The method continues with the next iteration of j.

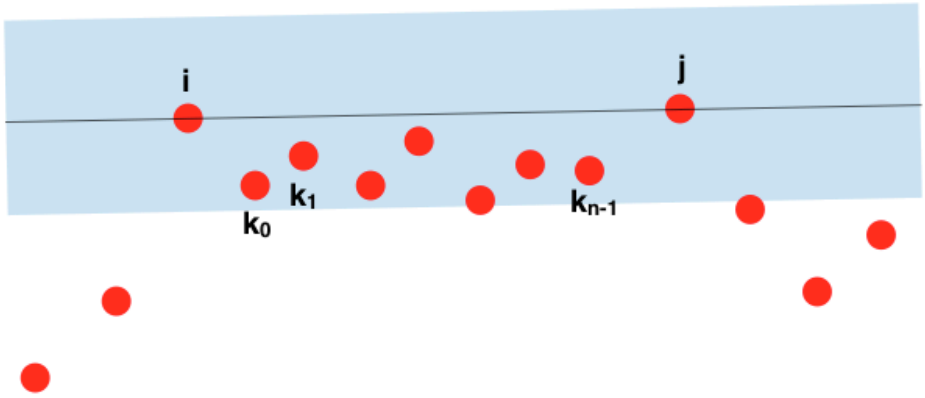


The next iteration of j returns one value that is out of range (green).



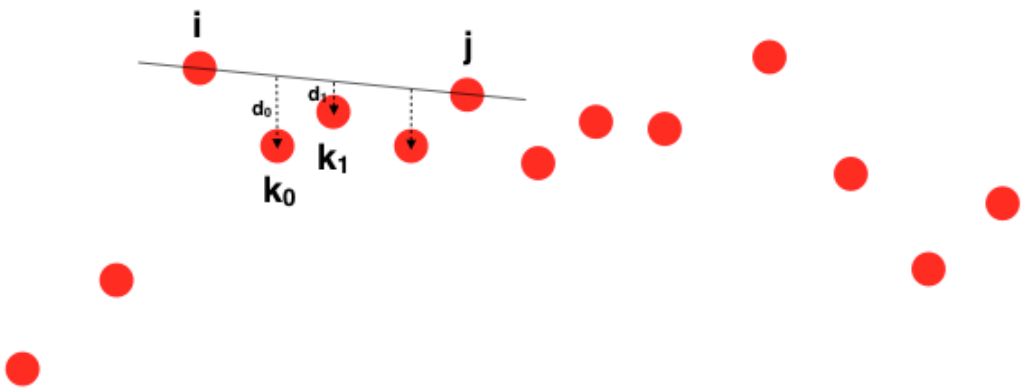
i is assigned as the out of range value of the previous iteration, and the analysis continues until a new value of k is found to be out of range. The red dashed line indicates the line calculated in the previous and current iteration.

This method proves to be faster in calculating and rejecting small fluctuations in pitch than the RDP algorithm, obtaining very similar results. However, there were circumstances where the reduced data did not represent the original curve accurately in neither RDP or my altered method. This occurs when points *k*, between *i and j*, are skewed above or below the interpolated line. This problem could be exacerbated by the method of smoothing used, Catmull-Rom (covered in section 4.1.4), which could arc smoothly away from the shape of the data.



Points k are shown skewed below the i to j interpolated line, and the range of rejected datapoints in blue. In this instance Catmull-Rom smoothing would arc above i and j, away from the original data.

This representation was improved upon with another minor change to the algorithm, calculating the absolute cumulative distance of *k* points from the interpolated line. Any array of *k* that skewed too far negatively or positively are represented in the new reduced data, acting somewhat similar to a localised linear regression. However, if the array *k* is made of equally positive and negative changes, the cumulative difference tends to zero, allowing for the analysis to continue with a new iteration of *j*.

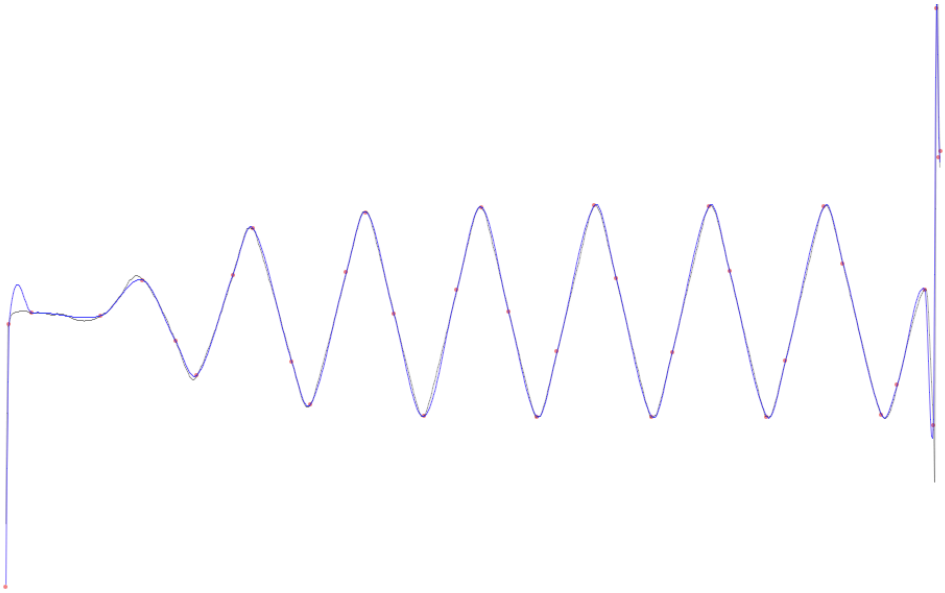


The points, k, skew negatively, which accumulate and trigger the method to set a new value of i.

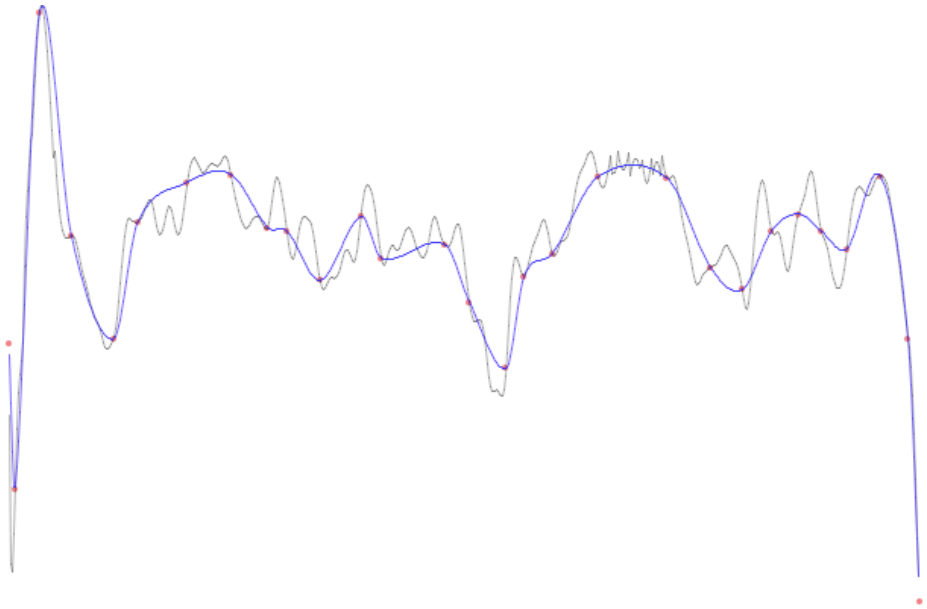
$$f > \left| \sum_{i=0}^{k} d_i \right|$$

The sum of the difference of all points k to the interpolated i - j line is always less than the threshold, f, defined by the user.

In real-world cases, this method reduces data suitably, maintaining the shape of the curve with negligible deviation from the original even under conditions of high data reduction settings. However, there are cases when extreme (often erroneous) fluctuations in detected pitch causes the curve smoothing algorithm to return an inaccurate representation. Solutions to this have yet to be discovered, but speculatively may involve a further refinement to the position of the remaining *k* points to more accurately represent the original dataset.



Data reduced pitch curve (blue) and its relative points (red) showing a 94% reduction of data of the original curve (black). Note the inaccuracies caused by erroneous data points especially at the beginning of the curve.



Another data reduced pitch curve (blue) and its relative points (red) showing a 96% reduction of data of the original curve (black).

Further refinement to this method included the change of the measure of distance of $i \rightarrow j$ to k, was to change Pythagorean distance to harmonic distance. This resolved an issue with data reduction in higher frequencies being over exaggerated as pitch is perceptually exponential in scale compared to linear measurement in difference of frequency-time.

$$d_{ji} = \left| \log_2(p_i) - \log_2(c_j) \right|$$

Difference (d) is calculated with each datapoint (p) and each centroid (c)

The final refinement to this method involved redefining the harmonic distance to a *harmonic area*, calculating the cumulative area between the linearly interpolated *i to j* line and for all k_{n-1} to k_n lines. This proved to be more effective at reducing a positive or negative skew than harmonic difference.

4.1.3 Energy-Based Rate Scaling

Another cause of pitch analysis errors was when the waveform was low in volume. This was likely due to noisy or unbalanced harmonics against the fundamental at the beginning or end of notes. Energy-based rate scaling reduces the effect of retuning when the waveform energy drops below a defined threshold, and linearly reduces the retuning effect until there is no energy. This is achieved by scaling the target frequency (descried in section 4.1.4) towards the detected frequency, reducing the modifying effect of resampling and retuning.

This method, while useful, is a temporary substitute for a more complex analytical method to reduce erroneous measurements, investigated in section 4.1.1 K-Means Clustering.

4.1.4 Resampling

Having collected data on detected fundamental frequency, a target fundamental frequency can be set for tuning, after which the process of retuning to the target can commence. In this method, the target frequency and detected frequency are used to create a variable sampling rate curve, similar to the way DTW resamples a waveform. Each target frequency and detected frequency curve undergoes its own interpolation with a Catmull-Rom spline (discussed in section 2.2.3), then the interpolated values of target frequency are then divided by the detected frequency. The mathematical basis for this is elegant and simple, as this division provides a rate of a rate (the first derivative of sample rate) which defines how many samples (floating point) to jump forward to either compress and raise, or stretch and lower, the length and frequency of the sample. This method is further simplified as the rate in which this is carried out is always exactly once for every sample of the input buffer.

For example, a section of audio with the detected fundamental frequency of 200Hz and target fundamental frequency of 400Hz means:

target/detected = 400/200 = 2

Jumping the sample index of the input audio forward by two samples, thus compressing the waveform by a factor of two and raising the fundamental frequency by an octave reaching its target fundamental. Similarly, with a target fundamental frequency of 100Hz:

target/detected = 100/200 = 0.5

The source of the output audio takes the index 0.5 samples ahead of the last sample index, slowing the output to achieve the target fundamental frequency. This floating point index is cumulative, always being added to the last calculated floating index. The floating point index is then passed to the method to calculate sample amplitude, read from the original sample using Catmull-Rom interpolation. This method had to be developed this way, as an implementation of the DTW method (section 2.2.3) resulted in a phase offset of the retuning curve by half the period between data points. Any significant offset of this nature would lead to ineffective tuning results, as the resampling rate would not align with the pitch data extracted from the sample.

Another problem caused by changing resampling rate is that any silence before or after the sample transients start can be lengthened or shortened, especially in cases of extreme retuning. To maintain the approximate timings of the input sample, the output sample is written to a temporary buffer that is large enough to accommodate any likely result, as calculating the size of the buffer would require a pre-calculation, essentially running the method twice. This would be an inefficient use of computer resources. Once the tuning process has been completed to the temporary buffer, it is trimmed into an output buffer in a more reasonable size for delivery to the user. This prevents samples with an overall slower rate (or lower frequency) from potentially being clipped if the output buffer was set to the same length as the input buffer, and also prevents an unreasonably long buffer from being delivered to the user. This memory behaviour is a caveat of using Java to code this part of the program, but is far outweighed by the speed at which it can perform this operation compared to Javascript, which provides a native function for arrays of variable length.

4.2 Conclusion

In conclusion, this process is a basic implementation which offers a starting point for developing capabilities for multiple notes in a single sample to be tuned as a preprocessing stage to DTW. This process also has the potential to be applied to multiple coincident microphone channels, if the same pitch correction curve were applied to a set of coincident recordings.

5 Metadata Mining

5.0 Introduction

The current approaches to virtual instrument building involve simulation of various characteristics of performance such as controls for simulating vibrato, dynamics, pitch variation, etc. Metadata Mining (MDM) investigates whether it is possible to improve the realism by assisting the MIDI performance with data that can describe measured characteristics in a real performance. For example, often vibrato of a sampled instrument is simulated with an LFO applied to pitch, usually consisting of a sine or triangle wave, and sometimes with an element of randomness to imitate human performance variations. This part of the project seeks to investigate whether this method can be improved upon by providing the sample engine with data on the shape and depth of a performed vibrato, transforming real human attributes into a format which can be replicated in the sample engine.

5.1 Exposition

Extracting pitch information on characteristics such as pitch and vibrato envelopes helps to solve a problem caused by dynamic time warping (DTW) processing, or indeed other forms of tuning/pitch-correction. For a group of samples, for instance four notes on french horn, *pp*, *p*, *mf*, and *f*, on F#4, any type of retuning (eg; pitch correction) or inability to use vibrato samples (in the context of DTW) means ‘real’ pitch envelope information can be lost. In the context of using DTW, static notes are recorded to be warped together, leaving no ‘real’ vibrato information available to the user in the sample engine, where previously, discrete vibrato samples supplied this functionality. To enable the sample engine to play back vibrato, two options were considered: The first would involve supplying the sample engine multiple distinct groups of samples with varying characteristics of vibrato, such as non-vibrato, fast vibrato, or slow vibrato samples for the user to select. The other option would be to introduce a definable pitch variation to a group of non-vibrato samples.

The former would provide a whole range of logistical problems such as the recorded samples needing their vibrato to be perceptually in synchronisation with each other so the tone and pitch match as the DTW does its processing, as well as extra requirements for computer resources in storage and memory. This lack of flexibility is also not conducive for the project goal of high sonic flexibility.

The latter benefits from its flexibility to the user, allowing the custom control of both vibrato speed and depth over time, also requiring less computer memory and storage to hold different groups of

vibrato samples. However, one of the caveats of this approach is that it may make the note sound artificial due to unrealistic pitch or tone variation expected from a physical instrument. The proposed approach to this problem is to use pitch envelopes taken from a vibrato performance to imitate as closely as possible the pitch variation of the performance in the sample engine. Tonal qualities of vibrato may possibly be dealt with by the timbral modification tool (outlined in section 3 - IR Timbral Modification).

In studying pitch variation from instrument performances, a further expansion of the description of pitch variation as vibrato was needed to describe the behaviour observed. Four types of pitch variation were identified:

- Vibrato - a deliberate, wide, repeating oscillation
- Envelope - often a reactive adjustment to pitch, or technique such as portamento/fall-offs
- Detune - a consistent offset of the resting pitch
- Fluctuation - small, fast, unpredictable fluctuations

To follow one of the project aims, allowing a high degree of flexibility to the user, these variation types could be split up into discrete interface controls, allowing for vastly more ways to control sound. To give a perspective on scale, a single standard MIDI CC prescribed to *pitch variation* could provide 128 degrees of control, while four sliders for each type of variation would give just shy of 270 million combinations.

To separate these variation types into distinct curves, the pitch data from a real instrumentalist performance can be analysed for the properties of each type, and split into different curves. One of the most distinctive characteristics of the pitch variation types is their frequency of oscillation, with detune being the slowest, going through envelope, vibrato, and fluctuation to the fastest. This characteristic can be used to split the different types of variation into separate variation types through the analysis of data with discrete Fourier transforms (DFT - section 5.2.1).

5.2 Process

The collection of data starts out with a similar process to SRR (Sample Rate Retuning, section 4), collecting only pitch in a determined range of frequencies using the ircamdescrptor~ object in Max MSP. However, the method of defining the frequency range differs from SSR as the method of analysis (DFT - section 5.2.1) required a centre point of oscillation, a ‘0’ point - in this case, defined by the fundamental of the note. A note range value can then be used to limit detected frequencies to within set boundaries. This model would see improvement with the use of statistical analysis to reduce errors - especially at low amplitudes.

Processing of this data continues with the use of a discrete Fourier transform (DFT) for analysis, then filtered and reconstructed with the inverse DFT function.

5.2.1 Discrete Fourier Transform

Having obtained metadata from a sample using the ircamdecriptor~ object, the next step is to analyse the frequencies of this pitch information so that it may be utilised for quantitative analysis and processing. The Discrete Fourier Transform can be used to quantify the amplitude and phase of a given frequency over a defined time window. For the purpose of this project, very low frequencies will be analysed for an expected vibrato range between 5-7Hz (Fletcher, 2001. pp100. Sundberg, 1994. pp49), widening this range from 0Hz to around 20Hz to fully capture the shape of the pitch data captured from performance.

One important aspect of the DFT is the characteristics of the window, most notably the length of the window; affecting how many samples are included in the analysis, and the window function; affecting the main lobe and side lobe characteristics. A Hann window is a good general purpose window but has a wider main lobe than others such as Hamming and Blackman-Harris (Zölzer, 2011).

In this project, DFT was approached in an experimental way. Usually, to perform a Fourier transform on a sequence of samples, the samples are equally spaced period at a sample rate. In planning the MDM software, the nature of the data to be used in the DFT was considered. From techniques explored in SRR (section 4), there could be data refinement techniques implemented that rejects some data as errors. This would mean the sample rate would not be consistent (unless the rejected data was resynthesised). To make the algorithm more forgiving to the lack of reliability of the incoming data, sample position used time (in seconds) rather than a sample index for the Fourier transform. As the nyquist frequency is usually defined as half the sample rate, it becomes impossible to define as a constant if the rate is changing. Setting a nyquist frequency to an average rate yielded some unexpected results, as the Fourier transform tries to suppress any ‘missing’ or ‘misplaced’ samples to zero amplitude by affecting the high frequency bins in extreme ways, leading to erroneous high frequency oscillations in the resynthesised waveform. The solution was simply to cap the frequency analysis at a lower limit, not allowing the analysis of high frequency oscillations.

5.2.2 Filtering

Frequency filtering is used to split the pitch data from a real performance into separate components. In this version of the MDM software, the data is split into two of the four types of pitch variation identified; vibrato and envelope.

The DFT analysis results of phase and amplitude becomes easy to modify for each frequency bin, needing only to multiply the amplitude of a given bin to change its prominence. It would be possible, for instance, to multiply all frequencies above 10Hz by zero and the rest by 1, creating a brick-wall low-pass filter at 10Hz. This is not an ideal solution as brick-wall filters often create undesired artefacts such as ringing (Wilkinson, 1998).

A preferred method of filtering frequencies is to imitate a Butterworth filter which can be modelled with the following equation:

$$a_f = \frac{1}{\sqrt{1 + \left(\frac{f}{c}\right)^n}}$$

The amplitude of a given frequency bin (a), uses the frequency of the bin (f), a crossover frequency (c), and the filter order (n)

During testing, it was found that the order could be changed to a negative number to create a high-pass filter:

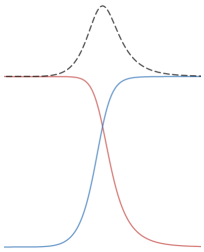
$$a_f = \frac{1}{\sqrt{1 + \left(\frac{f}{c}\right)^{-n}}}$$

In testing these functions, an accentuation of the crossover frequency was observed which was creating an inaccurate representation of the summed frequencies. This outcome would not be suitable for the proposed system of summing these filtered envelopes in the sample engine as the crossover frequency would be more prominent in any oscillations. Therefore, a filter that would crossover at -6dB (a doubling of amplitude rolloff) would be required to produce a flat response with no undesirable boosting of any frequency.

By squaring these functions a Linkwitz-Riley filter can be made, which importantly creates a crossover at -6dB when summed (Rane, 2005). These two functions were further adapted by substituting filter order with a more precise dB/octave adjustable slope which also gives the user a more readily interpretable measure than an abstract filter order value.

$$a_f = \frac{1}{1 + \left(\frac{f}{c}\right)^{\left(\frac{\pm n}{3}\right)}}$$

An interactive Butterworth filter can be seen in the link below (Chatfield, 2017), where the summation of frequencies at the crossover can be observed, and how squaring the function resolves this.



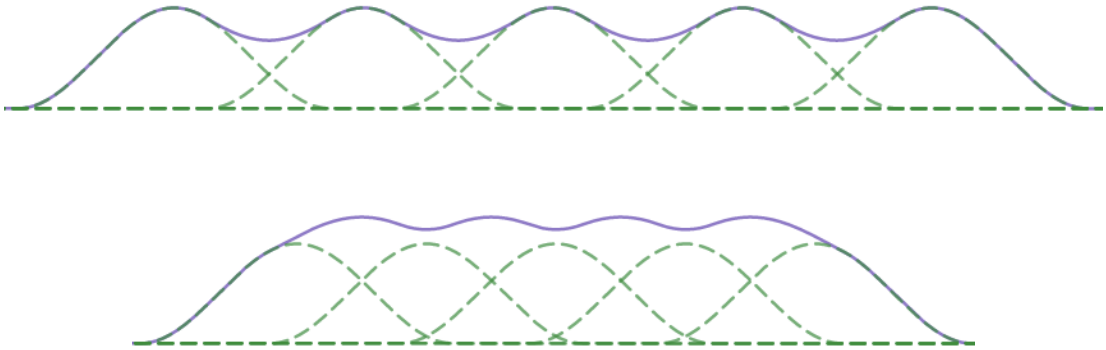
Butterworth filter - https://www.desmos.com/calculator/nknvpcqfcm

Once the amplitudes of the frequency bins have been modified to reflect the filter settings, they are transformed back to cartesian coordinates in preparation for the inverse discrete Fourier transform to resynthesise them into time domain of each pitch variation type, *vibrato* and *envelope*.

5.2.3 Inverse DFT

Much like the DFT, the iDFT (inverse DFT) determines the real and imaginary amplitude of a given sample and sums these values of each window and frequency, multiplied via the associated window function, turning a frequency domain representation into the time domain.

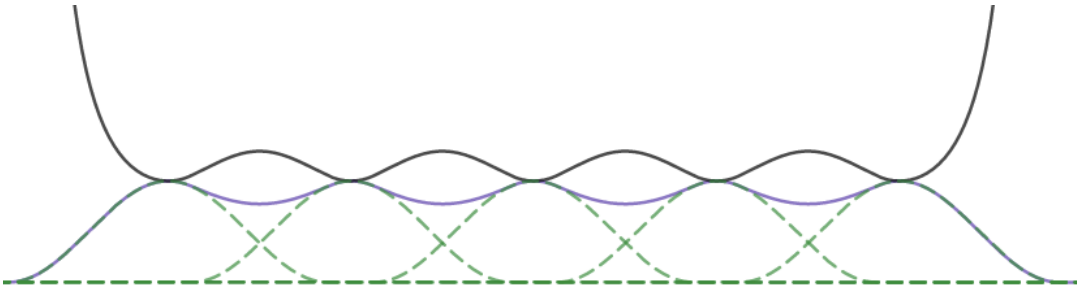
Each frequency of each window is accumulated by multiplying the real component by a cosine wave and imaginary by a sine wave of the given frequency. This accumulation of overlapping windows creates a problem with the reconstructed waveform - affecting the amplitude and creating a wavy summation of the windows.



Multiple $\sin(x)^2$ windows (green) summed (purple), with a big hop above, and a small hop below.

This process of summation describes the overlap-add method (Zölzer, 2011. pp. 191), however, this is not suitable as it causes the recreated waveforms to reflect the inconsistent envelope of the summed curve.

To resolve this, a type of averaging was considered. The *weighted average* sums the amplitude of a sample in a frequency bin, multiplying it by the amplitude of the window function. This means sample amplitudes that are to the extremities of the window are reduced significantly in weight compared to the amplitudes in the middle of the window. To avoid the summation problem discussed above, the sum of weights of each sample from each window is tracked, then used to rationalise each sample so that the amplitude is normalised - no matter how many windows or the weight of those windows contribute towards the amplitude of a single sample. Mathematically, this can be described as multiplying the multiplied and summed frequency bin amplitudes by *1/ summed window function.*



The new line (black) defines the multiplication function of the sample to normalise the summed amplitudes (purple).

In viewing results in testing, the recreated filtered waveforms suffered from some unforeseen characteristics. If the sample started or ended not equal to the fundamental (such as in analysing recordings of glissandi or fall-offs), the DFT interprets the extremities of the dataset to mean anything else is ‘equal to zero’ (ie; the fundamental). The DFT suppresses the lack of data to zero, causing a high frequency ‘click’ at the beginning and end of the dataset, which is reproduced in the high-pass filtered waveform. For this reason, the use of DFTs do not appear to be ideal for all types of separation of *all* the pitch variation types. It may be more suitable to explore different kinds of analysis for the variation types that are not based on oscillations; envelope and detune. A first-pass with a different analysis of the data (eg; moving average) to log and remove the low-frequency envelope might work better, allowing the DFT to interpret the vibrato and fluctuations more accurately.

5.3 Development

To finalise the separate variation types for use in a sample engine, they must first be multiplied to the correct scale and format for the chosen sample engine, whether that be a scale (eg; between -1 & 1) or a MIDI MSP & LSB of pitch bend messages. The sample engine would require the ability to read this data at a determined rate, applying the read values to the resampling rate of the played audio samples, perhaps even applying to other parameter modulation such as dynamic layers or

vibrato layers to affect the tone, as Sundberg (1994) shows several parameters are affected in unison with vibrato.

To fit with the project's goal of flexibility, further processing to the variation types can be carried out. For example, if the sample engine was to play back a static sustain, attaching a vibrato curve straight from a real performance, the vibrato data and the stained note would certainly not align for notes of different lengths, meaning the vibrate oscillations could simply run out and stop. For this reason, the proposed method of processing vibrato (and indeed other variation types) is to separate each type into component parts. Vibrato could be cut into *vibrato attacks*, *vibrato sustains*, and *vibrato releases*, allowing for *vibrato sustains* to be looped at the zero-crossings and even for different types of components to be mixed and matched. A more simple system could automatically generate envelopes for how much pitch variation happens throughout the note.

For this feature in flexibility to work, the suggested method would be to separate the captured vibrato data into separate oscillations of one period, splitting them at the zero-crossing. This would allow a number of different things to happen; vibrato oscillations could be repeated indefinitely for an unknown note length, oscillations could be randomly selected from a database of single oscillations to recreate a varied performance, and vibrato of different characteristics (eg; speed, depth, style) could be chopped in on-the-fly on command from the DAW.

5.4 Conclusion

This proof of concept shows that data potentially useful for virtual instrument production can be extracted from real performances, as shown in the interactive demonstration Max MSP patch found in the Metadata Mining folder. Although this program deals only with a few of the identified pitch variation types, it could also be expanded to include other potentially useful perceptual parameters such as loudness, S/N ratios, harmonic/fundamental balance, or other characteristics.

The use of metadata can be extended beyond the scope of analysing one sample; by analysing a group of samples from a live performance, the way parameters behave in context with one another could be observed. For example, observing vibrato characteristics over the pitch range of an instrument could yield results such as seeing vibrato rate increase with increasing pitch, which could be replicated in the sample engine.

6. Conclusion

The research focus of this paper was to increase the realism of virtual instruments while keeping within realistic boundaries of commercial technology, with limitations like processing power and digital storage space. The processes investigated in this research are not expected to directly reduce the data needed to make cumbersome virtual instruments that focus on comprehensive recording, but is expected to drastically improve the flexibility of virtual instruments using smaller data footprints.

Although all of the software/processes are in their infancy, they show the concepts have promise. The small-scale tests implemented in this research will have to be expended further to determine the effect they will have on a comprehensively sampled instrument.

References

Arnost, V. (n.d.). *Comparison of Interpolation Algorithms in Real-Time Sound Processing.* Retrieved from https://www.maximalsound.com/mastering/interpolation%20methods.pdf

Bachrules. (2014) *Once again - velocity X-Fade and phasing*. Retrieved from https://www.vsl.co.at/community/posts/t37875-Once-again---velocity-X-Fade-and-phasing#post230506

Bello J.P., Daudet L., Abdallah S., Duxbury C., Davies M., Sandler M.B. (2005) *A Tutorial on Onset Detection in Music Signals*
Retrieved from https://pdfs.semanticscholar.org/4afa/5e20cbbc5300c51dd9e16e20674d257a3f39.pdf

Bourke, P. (1999). *Interpolation methods*.
Retrieved from http://paulbourke.net/miscellaneous/interpolation/

Burton, R. (2015). *The Elements of Music: What Are They, and Who Cares?* Victoria: The Australian Society for Music Education. Retrieved from http://asme2015.com.au/the-elements-of-music-what-are-the-and-who-cares/

Chatfield, D. (2017). *Butterworth Filter.* Retrieved from https://www.desmos.com/calculator/eqepskabiq

Fletcher, N.H. (2001). *Vibrato in Music*. Acoustics Australia. Retrieved from https://newt.phys.unsw.edu.au/music/people/publications/Fletcher2001a.pdf

Forgey, E.W. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classification. *Biometrics*, *21,* 768–769.

Hamerly, G. Elkan, C. (2002). *Alternatives to the k-means algorithm that find better clusterings.* Retrieved from http://people.csail.mit.edu/tieu/notebook/kmeans/15_p600-hamerly.pdf

Hand, D.J. (2005). Optimising k-means clustering results with standard software packages. *Computational Statistics & Data Analysis*, *49*(4), 969-973.

Harker, A., Tremblay, P. A. (2012). *The HISSTools Impulse Response Toolbox.* Retrieved from http://eprints.hud.ac.uk/14897/

Heckbert, P., Garland M. (1997). *Survey of Polygonal Surface Simplification Algorithms.* Retrieved from http://www.cs.cmu.edu/afs/cs/user/garland/www/Papers/simp.pdf

Hien, C. (n.d.). *Chis Hien Ultra Realistic Virtual Instruments*. Retrieved from http://www.chrishein.net/web/Willkommen.html

Jain, A. (2010). Data Clustering: 50 Years Beyond K-means. *Pattern Recognition Letters, 31*(8), 651-666.

Likas, A. (2003). The global k-means clustering algorithm. *Pattern Recognition*, *36*(2), 451-461.

Rane, B. (2005). *Linkwitz-Riley Crossovers: A Primer.* Retrieved from http://www.rane.com/note160.html

Niemitalo, O. (2001). *Polynomial Interpolators for High-Quality Resampling of Oversampled Audio.* Retrieved from http://yehar.com/blog/wp-content/uploads/2009/08/deip.pdf

Stowell, D., Plumbley M. (2007). *Adaptive Whitening to Improve Real-Time Audio Onset Detection*. Retrieved from http://c4dm.eecs.qmul.ac.uk/papers/2007/StowellPlumbley07-icmc.pdf

Sundberg, J. (1994). *Acoustic and Psychoacoustic Aspects of Vocal Vibrato*. Retrieved from http://www.speech.kth.se/prod/publications/files/qpsr/1994/1994_35_2-3_045-068.pdf

Syquest. (2006). *Instrument solo + Velocity X-Fade = doubling ?.* Retrieved from http://www.vsl.co.at/community/posts/t8769-Instrument-solo--Velocity-X-Fade--doubling#post65299

Tommasini, T., Seidleczek, P. (2004). Sample Modeling. Retrieved from http://www.samplemodeling.com/en/technology.php

Van Winkle, L. (2009). *Introduction to Splines*. Retrieved from http://codeplea.com/introduction-to-splines

Wallach, H., Newman E. Rosenzwig M. (1949). The Precedence Effect in Sound Localization. *The American Journal of Psychology, 62,* 315-336.

Wilkinson, S. (1998). I Want to Take You Higher. *Electronic Musician*, *14*(10), page 34.

Zölzer, U. (2011). *DAFX: Digital Audio Effects* (2nd ed.). John Wiley & Sons

Copyright Statement