Implementation of a Playback Controllable Morphing Algorithm in Max

Víctor Llinares Vacas
722525

Bachelor thesis submitted as a part of requirements for obtaining the Bachelor's Degree in Telecomunication Systems, Sound and Image Engineering at the Universitat Politècnica de València, Escola Superior de Gandia

Faculty of Media
Institute of Sound And Vibration Engineering (ISAVE)
Hochschule Düsseldorf, University of Applied Sciences

Supervisors:
Prof. Dr. Michael Oehler
Dipl.-Ing. Frank Meuter

Düsseldorf, December 2016

To my family, for their support, love and encouragement.

STATEMENT OF ORIGINALITY

I declare that I am the sole author of this bachelor thesis titled Implementation of a Playback Controllable Morphing Algorithm in Max.

The work contained in this thesis has not been previously submitted for a degree or diploma at any other higher education institution.

The data, concepts, softwares and tools that have been taken directly or indirectly from other sources have been acknowledged and referenced.

DATE:

SIGNED:

Abstract

Sound Morphing, it's a transformation in wich two sounds from different sources are gradually hybridized. In this Thesis, the morphing concept will be defined, trying to avoid the ambiguity surrounding it in the Sound field. Moreover, a brief study of several Morphing algorithms will be realized, and finally, a morphing playback controllable algorithm will be developed in Max.

Keywords: Sound Morphing, Algorithm, Playback controllable, Max

El Morphing de sonido, es una transformación en la que se hibridizan gradualmente dos sonidos de fuentes distintas hasta llegar a un punto intermedio. En esta tesis, se define el concepto de morphing, tratando de evitar la ambigüedad que lo rodea en el campo del sonido. También se llevará a cabo una breve investigación sobre diversos algoritmos de morphing , así como el desarrollo en el software Max de un algoritmo de morphing de sonido con control sobre la reproducción.

Palabras clave: Morphing de sonido, Algoritmo, Control, Max.

Contents

1. Introduction

Sound Morphing is one of the most discussed and interesting sound transformation techniques, due to its creative potential and flexibility. It's been used in fields like computer music and synthesizers, speech research or even in psychoacoustic experiments.

The concept of creating hybrid elements is always interesting, and in spite of its greatest popularity in the image field, the abstract factor surrounding an hybrid between two sounds makes it, in my opinion, even more fascinating if possible. The creation of an intermediate timbre between two musical instrument, for instance, can lead us to imagine a new timbre realm representing a third unexisting instrument.

Despite its popularity, it seems to be a lot of controversy regarding the true nature of the process, evidencing the necessity of setting up boundaries between what should and what shouldn't be called Sound Morphing. In this thesis, an attempt in clarifying the blurry barrier that separates Sound Morphing from other audio techniques will be carried out. Also several algorithms for both analysis and Sound morphing will be described and finally an algorithm that will allow the user to have a playback control over the morphing process will be developed in Max.

Before the final algorithm programming section I will explain the choices I made along the way to implement it and will introduce some theory about sound analysis and resynthesis to accompany the whole project with a theoretical complement about the subject.
During the theoretical part of this thesis, I'll try to illustrate the content with Max examples, as well as explain how Max deals with sound processing regarding to what's relevant for the topic.

1.1 Thesis Structure

- Chapter 2: Sound morphing concept is defined. Also, an attempt to clarify the blur surrounding this technique is performed by making a distinction with different sound processing approaches that are conceptually close to sound morphing.

- Chapter 3: Reviews some of the already developed, and still in development, sound morphing algorithms.

- Chapter 4: Deals with the theoretical background of sound processing and Max software, introducing the unavoidable Fourier analysis, for performing alterations in sound at a frequency domain level.

- Chapter 5: playback controllable sound morphing algorithm

- Chapter 6: Conclusions and future work.

- Chapter 7: Aknowledgments

- Chapter 8: References

2. Background

2.1 Sound Morphing Definition

Sound morphing, is a sound transformation technique that bridges the gap between two (or more) different sounds, gradually transforming a sound object into another.

Some may take this definition to the letter, calling morphing what's just a simple mix or a completely different technique. The essential difference between Sound Morphing and the rest of the miscalled morphing techniques is that in a sound morph, only one identifiable sound stream should be perceived by the listener throughout the transformation, for what we need to interpolate values at a frequency domain level.

To better understand the procedure, it seems somehow easier to think about morphing in the Image field (Image or video morphing), being a technique which the average reader will be much more familiar with, having seen it in innumerable movies.

Figure 2.1 Example for image morphing. (Caetano, Rodet, 2011)

For instance, having two different human faces as an input and a target, to gradually transform the Source into the target, interpolating values from both objects is needed in order to achieve a perceptually satisfying intermediate representation. If instead of interpolation from both pictures, we just exchanged parts of each pictures, even if it was under certain parameters, one could easily identify those parts from each of the inputs, because those parts

will never change to an intermediate point, and that's the most relevant differentiating aspect of a morph.

Unlike video morphing, in which most of the time reaching the target image in a gradual way is the main purpose, in audio morphing we find the most important information in the results that are perceptually intermediate to the objects treated, i.e. the transformation itself.

2.2 Fundamentals of Sound Morphing

From a perceptual point of view, we could easily differentiate morphing from other audio techniques inasmuch as the result is a unique identifiable output stream, meaning that we are not perceptually able to detect either of the inputs by listening to the morphed outcome. This is already enough to rule out most of the commonly misnamed morphing techniques such as simple mixing, convolution or cross-synthesis among others. Here, we're understanding outcome as the sound generated between the two sources. It's obvious that during a morphing process, where the sound “travels” from the source sound to the destination sound, the listener will be able to identify both the source and destination as different sound entities, but if we took just the part of the morphing process were we find the actual morphed sound, it wont appear to the listener that one of the sounds or it's characteristics is affecting the other in any way, but a third entitie is created in between both the source and the destination.

Following, I'll explain several audio processing techniques that are commonly misunderstood as being analogous to sound morphing.

The first obstacle that I faced when I started this thesis, was to start reading and learning about morphing from articles that were calling morphing processes that were not. It's curious how this concept is misunderstood, and due to it's mixing or blending nature, it's used to describe other techniques that consist of similar conceptual process. This misunderstanding,

is given and aggravated due to the lack of a strict mathematical approach to sound morphing. For example, no one would call convolution a simple mixing process, since the word convolution is directly linked to a mathematical approach that restricts it's use to a single defined process. I find then, that a more useful way to restrict the use of Sound Morphing to a single process is to specify, define and differentiate what the other conceptually similar processes are.

2.3 Differentiating Other Sound Processing Techniques

2.3.1 Cross-synthesis

Cross-synthesis is a technique that maps characteristics of one sound to another using spectral or amplitude envelopes from one sound to affect a second one. One example could be impressing the spectral envelope of a voice speech into an instrument sound, or simply use the amplitude spectrum of one sound with the phase spectrum of another. The result are well-known effects like “talking-guitar” or any kind of musical vocoder in which it can be easily distinguish elements of the two sound sources.

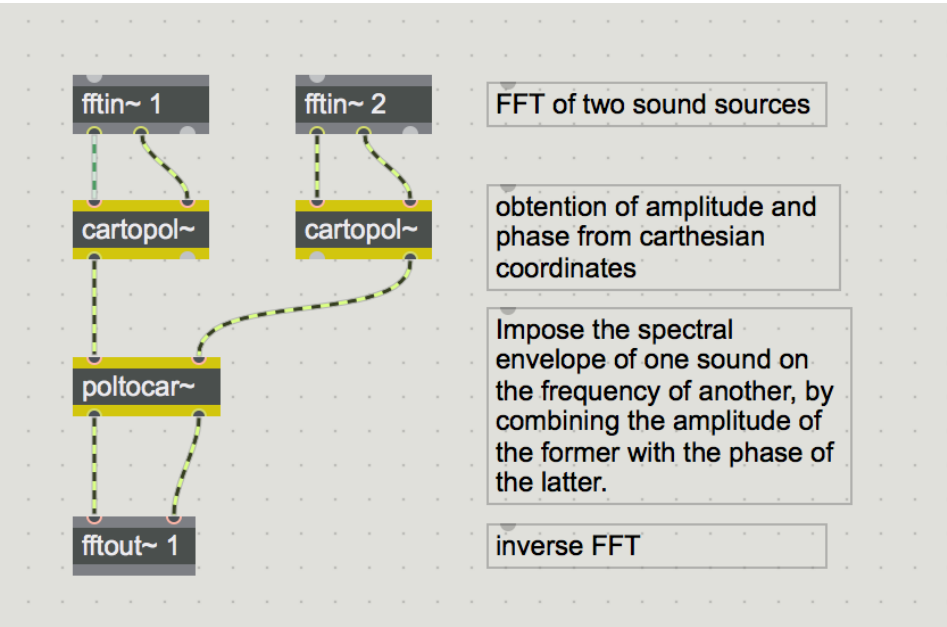

Figure 2.2 Simple corss-synthesis in Max (Dobrian, 2001)

2.3.2 Convolution.

Convolution is a mathematical way of combining two signals to form a third signal. Using the strategy of impulse decomposition, systems are described by a signal called the impulse response. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response (Smith, 1997, pg. 107).

Convolution is used in Digital Signal Processing in order to characterise a system. To summarize, if we know the Impulse response of a system, i.e. the response of a system when the input gets excited by an Impulse, we know the behaviour of the system hence we are able to calculate what the output will be for any possible signal.

From a more perceptual or musical point of view, convolution is mostly used for emulating rooms and enviroments responses for reverbs. Nonetheless, a convolution in time-domain is the same as a multiplication in frequencie-domain, so as showed in the next example, one could use the the fft (Fast Fourier Transform) of two different sounds and multiply it in the frequencie domain to obtain a third one.

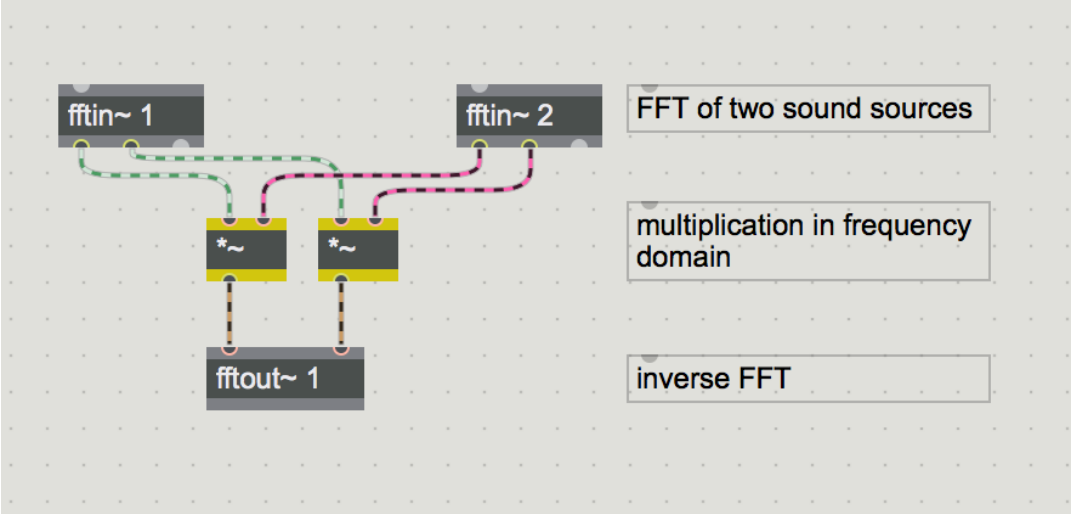

Figure 2.3 A simple type of spectral convolution (Dobrian, 2001)

It's true that, if we understand morphing as a hybridisation of sounds, convolution as showed before is creating a third sound from two different sources in a very straight-forward approach. Some authors will include this in the morphing family while others, will discard it claiming that in audio morphing, similar features are matched and interpolated, and for doing so, the system must analyse the signals and determine how to interpolate them. In this paper, we consider convolution a valid hybridisation process but different in concept and definition to a sound morphing procedure.

As we'll see in the next section, there are algorithms that use features from each of the inputs of a morph, matching similar qualities, instead of simply interpolating, to achieve a more perceptually meaningful result. Caetano and Rodet (Caetano, Rodet, 2011) make a distinction between Interpolating and Morphing, in which interpolating acts on the parameters of a model while Morphing includes by definition hybridisation of perceptual qualities. This implies that a previous analysis of the signals is required in order to obtain features of each input that help to reach (and control) a satisfying outcome. In this work, the Morphing described by Caetano and Rodet is just considered a more perception accurate Morhping algorithm without ruling out interpolating techniches wich will be also considered morphing.

3. Morphing Aproaches

3.1 Sound Morphing using Loris and the Reassigned Bandwidth-Enhanced Additive Sound Model

This section is mainly based and extracted from References [19] and [20].

Additive synthesis, represents each sound as a collection of sine wave components, or partials. This allows independent fine control over the amplitude and frequency characteristic of each partial in a sound. As a result, a wide variety of modifications are possible with additive synthesis, including frequency shifting, time dilation, cross synthesis, and sound morphing.

The reasigned bandwidth-enhanced additive sound model is kind of an improved sinusoidal model, developed by Kelly Fitz and Leopold Haken, in which sound waves are modeled as a collection of sine waves called partials. This partials are not strictly sinusoidal, a bandwidth-enhancment technique is employed in order to combine sinusoidal and noise energy creating *bandwidth-enhanced partials* (sine waves with noise) with time-varying parameters.

In order to implement efficient real-time timbre manipulations, they developed a stream-based representation of partial envelopes. *Envelope parameter streams* provide amplitude, frequency, phase, and noiseness envelopes (or Bandwidht) for each partial. Bandwidth envelopes represent noise energy associated with each partial and constitute an important extension to additive sine wave synthesis. Noise envelopes were developed in order to have a homogenous representation of both sinusoidal energy and noise energy of a sound.

The method of reassignment (Auger and Flandrin, 1995) improves the time and frequency estimates used to define partial parameter envelopes. This model yields greater resolution in time and frequency than is possible using conventional additive techniques, and preserves the temporal envelope of transient signals, even in modified reconstruction (Fitz, Haken, and Christensen, 2000).

Envelope Parameter Streams

Data streams encode envelope parameters for each partial (Haken 1995). The envelope parameters for all the partials in a sound are encoded sequentially. Typically, the stream has a "block size" of 128 samples, which means the parameters for each partial are updated every 128 samples, or 2.9 ms at a 44.1 kHz sampling rate.

Envelope parameter streams are usually created by traversing a file. The file contains data from a non-real-time(Loris) analysis of a source recording. A parameter stream typically passes through several processing elements. These processing elements can combine multiple streams in a variety of ways, and can modify values within a stream. Finally, a synthesis element computes an audio sample stream from the envelope parameter stream.

The synthesis element implements *bandwidth-enhanced oscillators* (Fitz and Haken 1995) with this sum:

$$y(t) = \sum_{k=0}^{K-1} \left(A_k(t) + N_k(t)\, b(t) \right) sin\left(\theta_k(t) \right)$$

$$\theta_k(t) = \theta_k(t-1) + 2^{F_k(t)}$$

where
y is the time domain waveform for the synthesized sound,
t is the sample number,
k is the partial number in the sound,
K is the total number of partials in the sound (usually between 20 and 160),
A_k is partial k's amplitude envelope,
N_k is partial k's noise envelope,
b is a zero-mean noise modulator with bell-shaped spectrum,
F_k is partial k's log frequency envelope,
$theta_k$ is the running phase for the kth partial.

Figure 3.1 Bandwidth-enhanced oscillators sum. (Fitz and Haken, 2003)

The noise envelope Nk is their extension to the additive sine wave model. Rather than use a separate model to represent noise in the sounds, Fitz and Haken define this third envelope (in addition to the traditional Ak and Fk envelopes) and retain a homogenous data stream.

Analysis Parameters

The analysis and representation of transients is a well-known problem for additive synthesis. The onset of a sound, in particular, is psychoacoustically important (Berger 1964, Saldanha and Corso 1964) and is difficult to analyze with sufficient time accuracy. The time-domain shape of an attack is distorted because the window used in the analysis of the sound cannot be perfectly time-localized.

Conventional additive analysis performs a sequence of short-time Fourier transforms. The time domain signal is windowed, with overlapping windows used for successive transforms. The result of each transform is mapped to the time at the center of each window. If a window of data is centered just before the onset of a sound, the left (early) samples in the window

precede the attack and the right (later) samples in the window include the attack. If this situation is not explicitly detected, the attack is blurred. The time-domain shape of transients is distorted even if, at each window, the analysis guarantees phase-correctness of each partial. With the reassigned bandwidth-enhanced greater resolution in time and frequency can be achieved and temporal envelope of transient signals is better preserved. It can be configured according to two parameters: the instantaneous frequency resolution, or minimum instantaneous frequency separation between partials, and the shape of the short-time analysis window, specified by the symmetrical main lobe width in Hz.

The frequency resolution parameter is the minimum distance in frequency between partials meaning, it will control the density of partials.

The shape of the short-time analysis window governs the time-frequency resolution of the reassigned spectral surface, from which bandwidth-enhanced partials are derived

An analysis window that is short in time, and therefore wide in frequency, yields improved temporal resolution at the expense of frequency resolution. Spectral components that are near in frequency are difficult to resolve, and low-frequency components are poorly represented. A longer analysis window compromises temporal resolution, but yields greater frequency resolution. Spectral components that are near in frequency are more easily resolved, and low-frequency components are more accurately represented, but short-duration events may suffer temporal smearing, and short-duration events that are near in time may not be resolved.

The use of time-frequency reassignment allows us to use long (narrow in frequency) analysis windows to obtain good frequency resolution, without smearing shortduration events. However, multiple short-duration events occuring within a single analysis window still cannot be resolved. Fortunately, the improved frequency resolution due to time-

frequency reassignment also allows us to use short-duration analysis windows to analyze sounds having a high density of transient events, without greatly sacrificing frequency resolution.

Sound Morphing using loris

Sound Morphing using traditional additive sound models for quasi-harmonic sounds basically consist in a weighted interpolation of the time-varying frequencies and amplitudes of the corresponding partials in the source sounds.

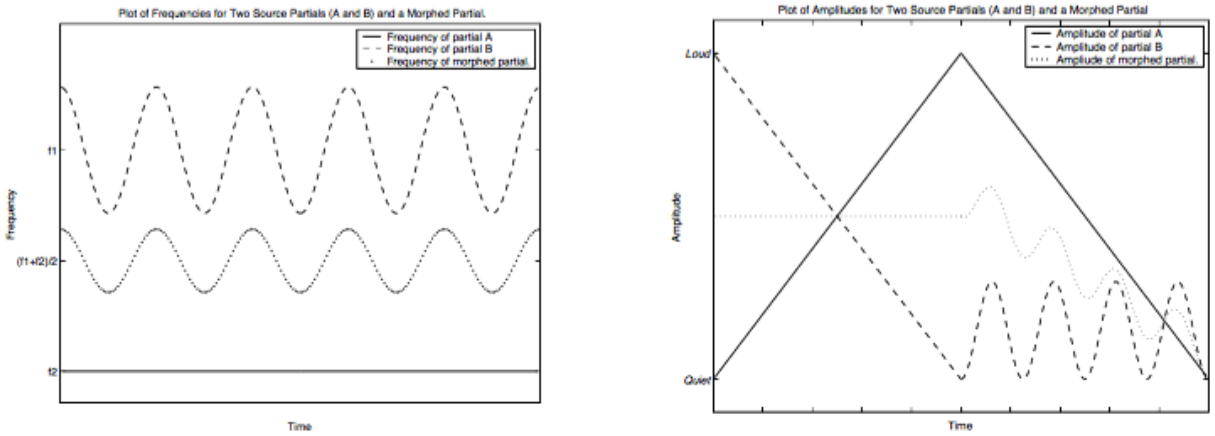

Figure 3.2 Equal-weight morphing of a hypothetical pair of partials by interpolation of their frequency (left plot) and amplitude (right plot) envelopes. The source partial envelopes are plotted with solid and dashed lines, and the envelopes corresponding to 50% (equal-weight) morph are plotted with dotted lines. Note that these envelopes are artificially generated to illustrate the morphing operation, and do not correspond to any real sounds. (Fitz, Hakken, Lefvert and O'Donnell, 2002)

In Loris (the software that implements the reassigned bandwidth-enhanced additive sound model) sound morphing is achieved fundamentally as in traditional additive sound models, so, interpolating the time-varying frequencies, amplitudes, and bandwidths of corresponding partials obtained from reassigned bandwidth-enhanced analysis of the source sounds. The difference though remains in the process of partial construction.

Three independent morphing envelopes control the evolution of the frequency, amplitude, and bandwidth, or noisiness of the morph.

Using the reassigned bandwidth-enhanced additive model (Haken, Fitz, and Christensen 2002) even very noisy quasi-harmonic sounds can be represented by a single partial for each harmonic, simplifying and improving the morphing process.

For non-harmonic or polyphonic sounds though, explicitly establishing correspondences is necessary due to the lack of obvious correspondence between partials or the opposite, having many possible correspondences between partials. In this case, correspondence between partials in the source sounds are established by channelizing and distilling.

Partials in each source sound are assigned unique identifiers, or labels, and partials having the same label are morphed by interpolating their frequency, amplitude, and band-width envelopes according to the corresponding morphing function. The product of a morph is a new set of partials, consisting of a single partial for each label represented in any of the source sounds.

In Loris, channelization is an automated process of labeling the partials in an analyzed sound. Partials can be labeled one by one, but analysis data for a single sound may consist of hundreds or thousands of partials. If the sound has a known, simple frequency structure, an automated process is much more efficient.

Channelized partials are labeled according to their adherence to a harmonic frequency structure with a time-varying fundamental frequency. The frequency spectrum is partitioned into non-overlapping channels having time-varying center frequencies that are harmonic (integer) multiples of a specified reference frequency envelope, and each channel is identified by a unique label equal to its harmonic number. The reference (fundamental) frequency

envelope for channelization can be constructed explcitly, point by point, or constructed automatically by tracking a long, high-energy partial in the analysis data. Each partial is assigned the label corresponding to the channel containing the greatest portion of its (the partial's) energy.

The sound morphing algorithm described above requires that partials in a given source be labeled *uniquely*, that is, no two partials can have the same label. In Loris, distillation is the process for enforcing this condition. All partials identified with a particular channel, and therefore having a common label, are distilled into a single partial, leaving at most a single partial per frequency channel and label. Channels that contain no partials are not represented in the distilled partial data.

Labeled and distilled sets of partials are morphed by interpolating the envelopes of corresponding partials according to specified morphing functions. Partials in one distilled source that have no corresponding partial in the other source(s) are crossfaded according to the morphing function. Source partials may also be unlabeled, or assigned the label 0, to indicate that they have no correspondence with other sources in the morph. All unlabeled partials in a morph are crossfaded according to the morphing function.

The various morph sources need not be distilled using identical sets of frequency channels. However, dramatic partial frequency sweeps will dominate other audible effects of the morph, so care must be taken to coordinate the frequency channels used in the distillation process. Though the harmonic frequency structure described by the channelization process may not be a good representation of the frequency structure of a particular sound (as in the case of a non-harmonic bell sound for example), it may still yield good morphing results by labeling partials in such a way as to prevent dramatic frequency sweeps.

3.2 Sound Morphing by Feature Interpolation

This section is mainly based and extracted from References [7] and [8].

One of the most spread uses of Sound Morphing has been music compositions or musical purposes (sound design...). When we imagine a morph between two different musical instruments, one could think that the result should be a third hybrid instrument placed in between the source and target instruments, therefore sharing features with both, or more accurate, being perceptually intermediate to those source and target instruments. Unlike in the algorithm described before, which basically consist in interpolating the parameters of the model of a sound regardless of perceptual features, in this algorithm, obtaining sounds whose values of features are intermediate to those of the source and target sounds is the main goal.

Using this strategy, the authors aim to be able to measure the perceptual impact of the morphed sounds using feature values as an objective measure. Also, linear changes in the sound features may lead to linear perceptual changes.

In this work, parameter refers to coefficients from which sounds can be resynthesized, while feature refers to coefficients used to describe or identify a particular aspect of a sound.

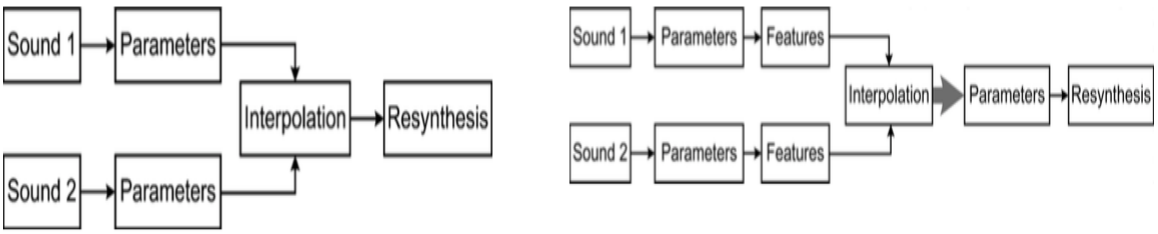

Figure 3.3 Morphing scheme using interpolation principle (left) compared to the morphing by feature interpolation principle (right). (Caetano and Rodet, 2013)

In figure 3.3, we can see the main difference between morphing using the interpolation principle and morphing by feature interpolation. It's true that, if we are looking for linear varying perceptually relevant features in a morph, representing sounds by adjusting parameters of a model and linearly interpolating those parameters can't assure that changes during morph will be perceptually linear. If we want the result to sound perceptually intermediate, we need to develop techniques to interpolate perceptually motivated features. On the other hand, resynthesize sounds directly from feature values (particularly when the features are correlated to perceptual characteristics of sounds) is not a straight-forward process, or it's not possible in theory.

Features used in this process are acoustic correlates of timbre dimensions obtained by perceptual studies, such that sounds whose feature values are intermediate between two would be placed between them in the underlying timbre space used as guide.

Acoustic correlates of timbre spaces

This section has been extraced from [] pp.

Multi Dimensional Scaling (MDS) techniques figure among the most prominent when trying to quantitatively describe timbre. Grey (Grey and Morrer, 1977) investigated the multidimensional nature of the perception of musical instrument timbre, constructed a three-dimensional timbre space, and proposed acoustic correlates for each dimension. He concluded that the first dimension corresponded to spectral energy distribution (spectral centroid), the second and third dimensions were related to the temporal variation of the notes (onset synchronicity). Krumhansl (Krumhansl, 1989) conducted a similar study using synthesized sounds and also found three dimensions related to attack, synchronicity and brightness. More recently, Caclin (Caclin, McAdams, Smith, and Winsberg, 2005)studied the perceptual relevance of a number of acoustic correlates of timbre-space dimensions with

MDS techniques and concluded that listeners use attack time, spectral centroid and spectrum fine structure in dissimilarity rating experiments. Here we should notice that most MDS techniques suppose that the underlying space is orthogonal and metric, which means that the dimensions are independent and the notion of distance is defined. Therefore, shifting linearly from a source to a target sound in such space would correspond to a perceptually linear change across all dimensions and would also result in linear variation of the values of the correlates of each dimension.

Most perceptually salient dimensions of timbre spaces are supposed to be captured by spectral and temporal features used in this algorithm, namely, the distribution of spectral energy and the attack time.

The first step of this whole process would be then modeling perceptually relevant features, that would consist of temporal modeling, and spectral modeling.

The temporal modeling stage consists in two steps, temporal segmentation, where the duration of 4 perceptually important regions is determined (Attack, transition, steady-state or sustain, and release). And the amplitude envelope estimation. Once both sounds from source and target have been temporally aligned, an interpolated amplitude envelope will modulate the spectral frames of the morphed sound.

For the spectral modeling stage, an harmonic sinusoidal modeling plus noise residual technique is used. The morphing process then, follows the next steps: temporal alignment, spectral envelope morphing and amplitude envelope morphing.

First of all, temporal stages of the sound (attack, sustain, release) must be aligned, for a perceptually pleasant morph. Then, in the spectral envelope morphing stage, the ideal scenario would be to interpolate spectral feature values and invert the results to get the spectral envelope parameters directly related to those interpolated features. As there's no

known analytic inversion from the chosen features, instead, a study of which spectral envelope representation leads to linearly varying values of spectral shape features when its parameters are linearly interpolated is carried out.

Last step is to morphing the amplitude envelope which is similar to the spectral envelope and morphing the noise residual for what a morph of spectral envelope of the residual noise is done and then sinthesized by filtering white noise with it for mixing it into the morphed sinusoidal component.

3.3 New Approaches Currently in Development

3.3.1 Sound morphing strategies based on alterations of time-frequency represesentations by gabor multipliers.

Sound Morhping is usually based on methods that require previous analysis of the source signals. Unlike all the aglorithms and morphing strategies presented before, sound morphing based on gabor multipliers don't need any previous analysis or model for achieving sound morphing.

Representation is a key aspect for this strategy. A simultaneous temporal and sepctral representation is used for this technique and in fact, the whole approach is based on the alteration of Time-Frequency Representation.

“If we consider a piece of music as a function of time f(t), we get the temporal behavior and maybe the rhythmic patterns, but we don’t know the tone pitch that is played. If we look in contrast at the Fourier transformation f(omega), we get the frequencies of the prevailing notes, but we don’t know anything about the duration of this note. Like our ear provides

information about time and frequency at the same time, we want to obtain a function that is both dependent on time and on frequency, a function that imitates our ear" (Bammer, 2015).

Time-Frequency Representation show the evolution of the spectral content of signals over time, so we can have information about time-evolving frequencies like a spectrogram that changes over time, as our hearing system does.

This strategy focuses on invertible time-frequency representations, largely used in the context of analysis/transformation/synthesis of sounds, and exploit them to perform sounds morphing.

Works by multiplying its time-frequency representation with a time-frequency transfer function, called a Gabor mask. This approach presents a clear advantage: no need of a previous analysis. On the other hand, it only works in sounds with similar features.

Sound morphing strategies based on alterations of time-frequency represesentations by gabor multipliers is a technique still being developed, and it opens a new field of sound morphing study.

4. Theory

4.1 Introduction to Fourier

The French mathematician Joseph Fourier demonstrated that any periodic wave can be expressed and hence modeled as the sum of harmonically-related sinusoids. This simple idea hides a complex mathematical theory that's "the central tool in musical signal processing" (Roads, 1996, pg. 1075).

Fourier transform, is a mathematical process, that using Fourier Theory, transforms signals from time domain to frequency domain by decomposing a cycle of an arbitrary waveform into its sine components.

The power of Fourier Transformation and how it facilitates the capacity of spectral transformation makes it an indispensable topic which will be covered in this paper (until some extend) in the simplest way trying to make it as accessible as possible.

According to Roads (1996), Fourier theory says that any signal of infinite length can be represented with a Fourier transform (FT) spectrum that spans from 0Hz to + and – infinity. When working with sampled signals in the Digital domain though, an adaptation of the Fourier transform called Discrete Fourier Transform must be used. In order for the DFT to work accurately, it should be used in short-time slices that should ideally equal to one cylce (or period) of the signal being analysed. To perform this operation on 'real world' sounds that are almost invariably *not* strictly periodic, and of unknown frequency, one can perform the DFT on consecutive time slices to get a sense of how the spectrum changes over time. If the number of digital samples in each time slice (or frame) is a power of 2, one can use a faster version of the DFT known as the Fast Fourier Transform (Dobrian, 2001, MSP Analysis tutorial 3). The FFT is just an optimised algorithm of DFT that eliminates redundant

calculations, but unfortunately as DFT, it will deliver acceptable results only if the condition of the analysis are ideal, that's to say, if one exact period of the signal (or an exact integer) is being analysed. In most cases, one period of the signal analysed won't fit perfectly in the analysis time slice, but the FFT will still analyse the slice as if it contained a period of the signal. Such an analysis will contain many spurious frequencies not actually present in the signal. To resolve this problem, we can try to 'taper' the ends of each time slice by applying an amplitude envelope to it, and use overlapping time slices to compensate for the use of the envelope. This process using overlapped, windowed time slices is known as Short Term Fourier Transform (STFT) (Dobrian 2001, MSP Analysis tutorial 3).

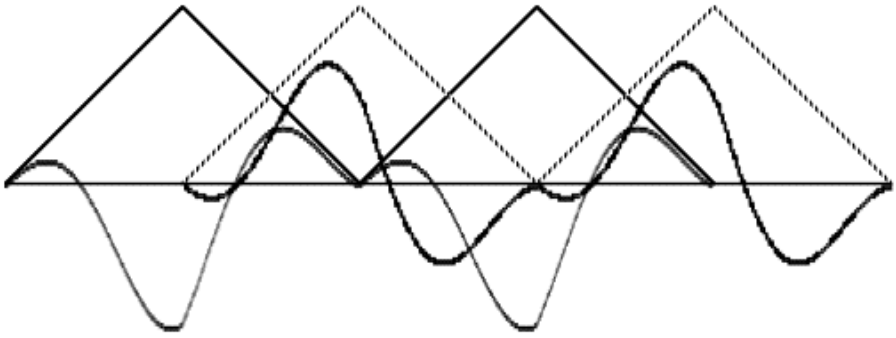

Figure 4.1 Overlapping triangular windows (envelopes) applied to a 100Hz cosine wave (Dobrian, 2001).

4.2 Fourier Analysis, Series and Transform.

As Fourier demonstrated, periodic signals can be expressed as a sum of harmonically-related sinusoids. This is how a periodic function in a form of Fourier series can be written as:

$$x(t) = C_0 + \sum_{n=1}^{\infty} C_n \cos(n\omega_0 t + \theta_n)$$

Equation 4.1 (Roads, 1996, p.1085)

A sum of harmonically-related sinusoids, with magnitude C_n, phase θ_n, frequency $\omega_n = n\omega_0 = 2\pi/T$, where T is a period of function x(t) . As *n* is an integer, the infinite sum defined in the equation will take place at integer multiples of the fundamental frequencies, or what are also known as harmonics of the signal. Acording to Smith(1997, chapter 11) periodic signals have a frequency spectrum consisting of harmonics. The first harmonic, i.e., the frequency that the time domain repeats itself, is also called the fundamental frequency. This means that the frequency spectrum can be viewed in two ways: (1) the frequency spectrum is continuous, but zero at all frequencies except the harmonics, or (2) the frequency spectrum is discrete, and only defined at the harmonic frequencies. In other words, the frequencies between the harmonics can be thought of as having a value of zero, or simply not existing. The important point is that they do not contribute to forming the time domain signal.

When working with STFT, the fundamental frequency in this process is determined by the size of the analysis window.

Fourier's theory says that *x(t)* can be accurately reconstructed with an infinite number of pure sinusoidal waves of different amplitudes, frequencies, and initial phases. These waves make up the signal's Fourier transform spectrum" (Roads, 1996, p. 1086).

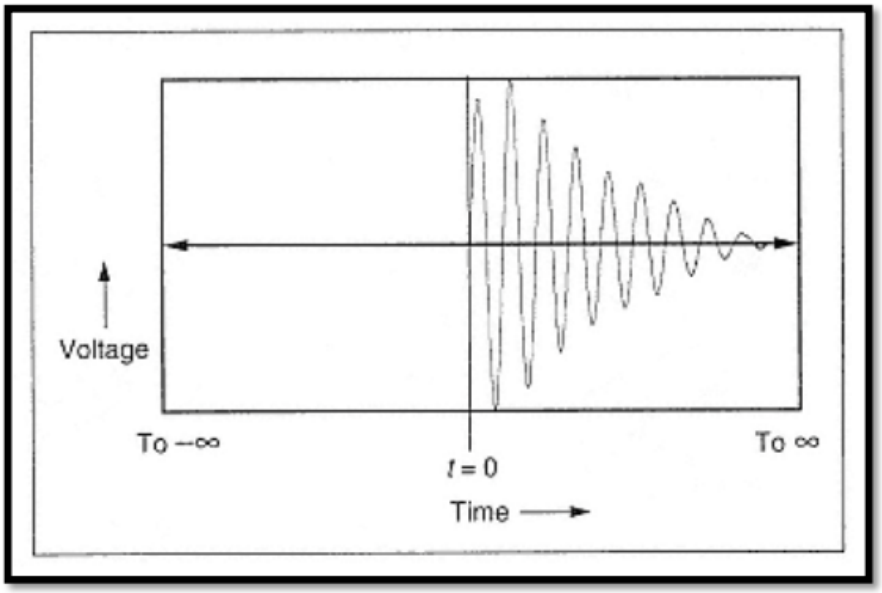

Figure 4.2 Continuous-time signal x(t) of infinite length (Rhoads, 1996, p. 1085)

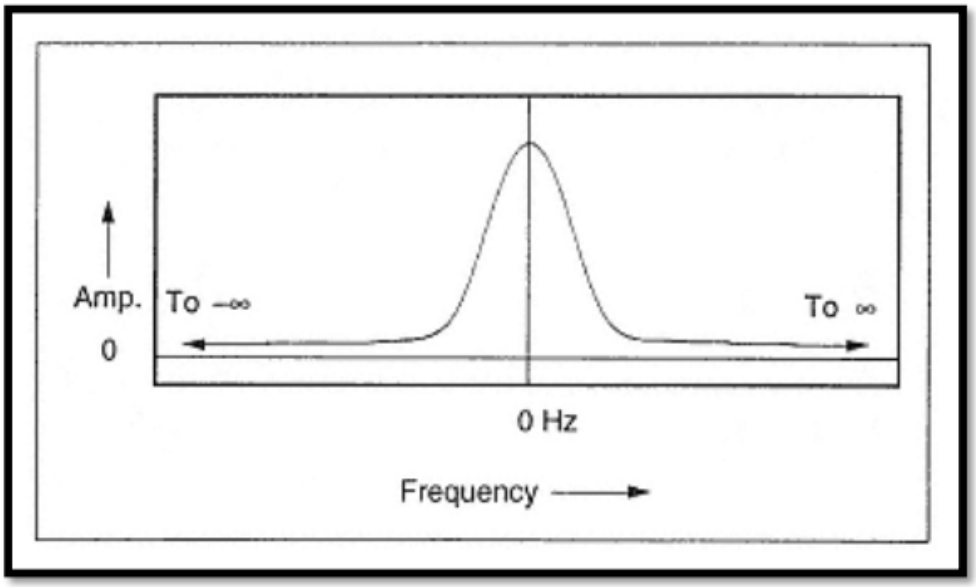

Figure 4.3 Magnitude spectrum after Fourier transform of the input signal x(t) in figure 3.3 (Rhoads, 1996, p. 1086).

The equation of the Fourier Transfor (or Fourier Integral) is the following

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

Equation 4.2 (Roads, 1996, p. 1087)

The apparently simple concept of Fourier Analysis, brings with it several layers of mathematical abstractions. A fundamental abstraction is the way sinusoidal signals can be expressed either in terms of trigonometric (circular) functions or in terms of complex numbers, vectors, and exponential functions. (Roads, 1996 ps. 1076, 1077).

For a better understanding of the Fourier Transform, an alternative way of denoting circular functions is necessary. In the next section complex numbers and it's relation with the Fourier analysis will be described.

4.3 The Fourier Integral and Complex Numbers.

When using Fourier integral, we are transforming a signal from time domain to frequency domain and vice versa (with the inverse Fourier Transform). There are two magnitudes that define a signal in the frequency domain though, Amplitude and Phase.

While the first magnitude importance is immediate, the phase is just as important.

Complex numbers are like circular functions also presented on a (2D) plane since they have a real and imaginary part. They have the unique property of representing and manipulating two variables as a single quantity, which fits perfectly with Fourier analysis and they also shorten the equations used in DSP, and enable techniques that are difficult or impossible with real numbers alone (Smith, 1997, p. 551).

As we can see from figure 4.4 a complex number in 2D space can be written in Cartesian coordinate system (x coordinate, y coordinate) or in polar coordinate system (distance from center point, angle).

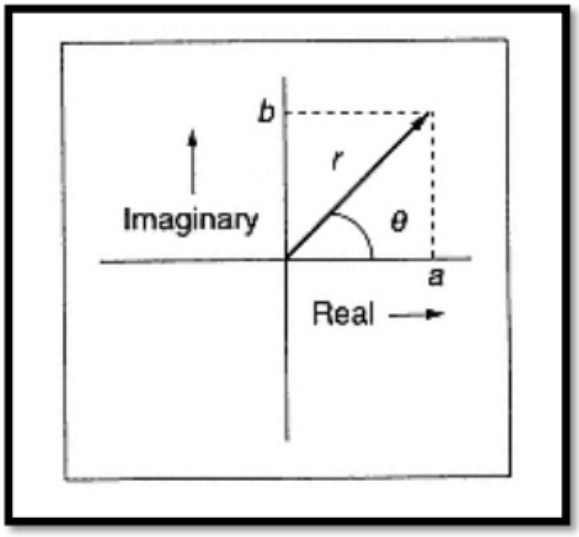

Figure 4.4 Complex number presented with Cartesian coordinates as point (a,b) and with polar coordinates as point (r, θ) where r is a distance from center and θ is the angle between vector and abscise (Roads, 1996, p. 1077).

When adding a third dimension (time) to complex numbers since audio signals exist only in time, our angle θ becomes ωt where ω is angular speed ($\omega = 2\pi f$) and t stands for time.

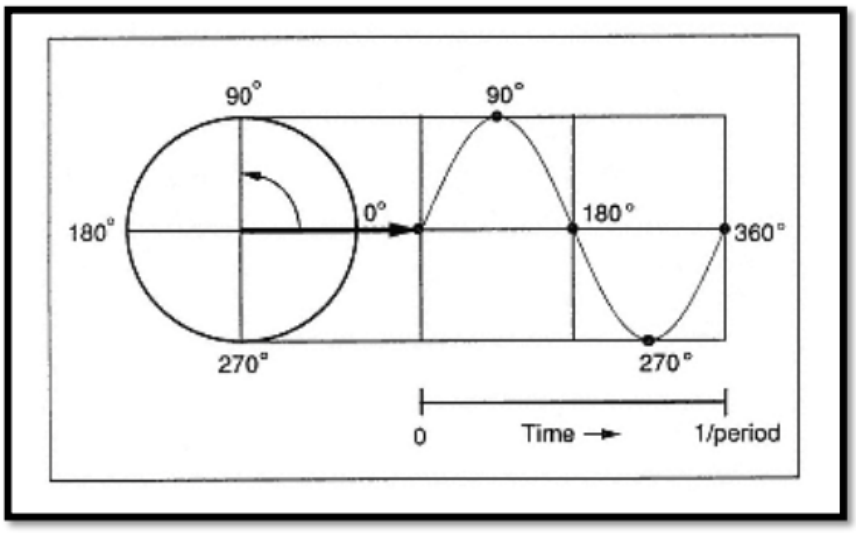

Figure 4.5 The projection of a rotating vector on a vertical axis is travelling vertically up and down between values 1 and -1.(Roads, 1996, p. 1079).

Having the equation (that can be derived from the previous representations in figures 4.5 and 4.6)

$$y(t) = r\sin(\omega t + \theta)$$

Equation 4.3

"The Fourier transform uses one of the most common tricks of engineering mathematics: the representation of a sinusoidal function as a sum of a sine and a cosine at the same frequency but with possibly different amplitudes" (Roads, 1996, p. 1081).

$$y(t) = A\sin\omega t + B\cos\omega t$$

Equation 4.4

To find the magnitude (which we have called "amplitude" until now-magnitude is the same as amplitude when we are only interested in a positive value- the absolute value):

$$magnitude = \sqrt{\mathrm{Re}^2 + \mathrm{Im}^2}$$

Equation 4.5 Redmon (2002)

and phase:

$$phase = a\tan 2(\mathrm{Im}, \mathrm{Re})$$

Equation 4.6 Redmon

Finally there's a third way of representation sine functions. According to Roads (1996, p 1082) the representation in form of complex exponential function makes the algebra manipulation much easier.

$$e^{j\omega t} = \cos \omega t + j \sin \omega t$$

Equation 4.6

After seeing the relationship between complex numbers, circular and exponential functions, we will focus in an interesting property of sinusoidal functions:
When you multiply two sine waves together, the resulting wave's average (mean) value is proportional to the sines' amplitudes if the sines' frequencies are identical, but zero for all other frequencies. And that's how sinusoidal functions act as detectors for specific harmonic content.

4.4 Sinusoidal Functions as Detectors in Fourier Analysis

As mentioned before, when multiplying sine waves (process called ring modulation) if we take a look at the average (mean) of the result, it would be zero if the sine waves are different, this means, the area above and below the abscise of the resulting wave will be identical, and the mean therefore zero. Contrarily, if we multiply two sine waves at the same frequency (and phase) the average will be proportional to the amplitudes of the sine waves. This provides a mechanism to identify matching frequencies in a signal, and that is what's constantly happening in FFT, a comparison (mutliplication) between an input and the periodic,

harmonic-rich entity that the FFT is itself, detecting that way, which of its own many virtual sine waves are also present in the input (Dobson, 1993).

Following, a demonstration of this property is performed using Max/MSP. In the first case (figure 4.6) two identical sine waves are multiplied, resulting in a signal which mean is different than zero.

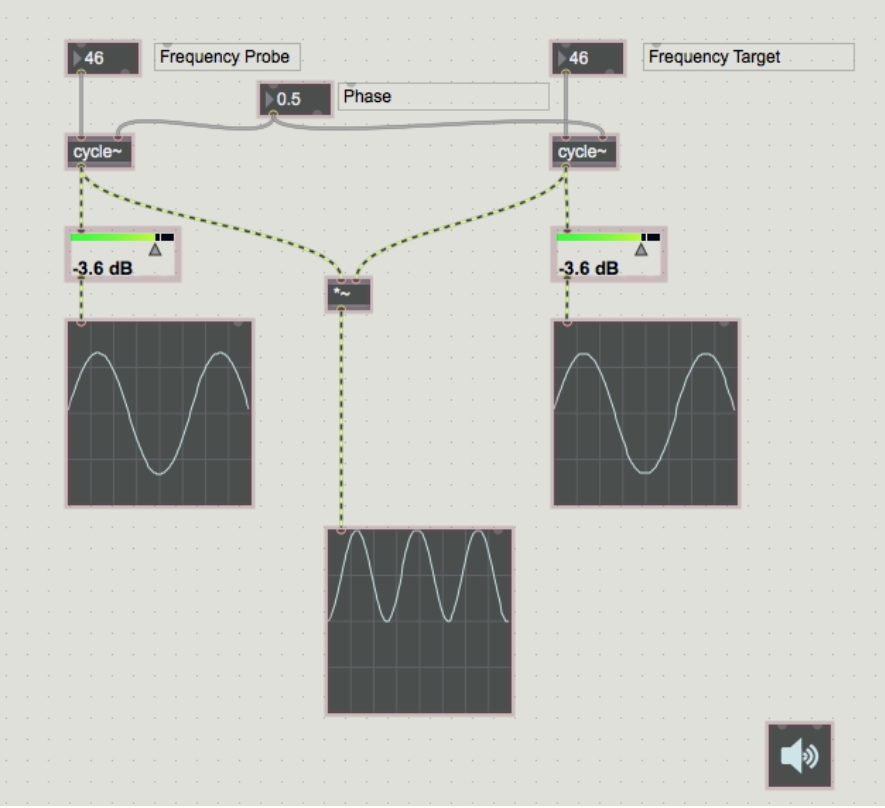

Figure 4.6 Multiplication of sine waves with same frequency.

Then, in the next patch (figure 4.7) two sine waves at different frequency are multiplied, resulting in a signal which mean is zero.

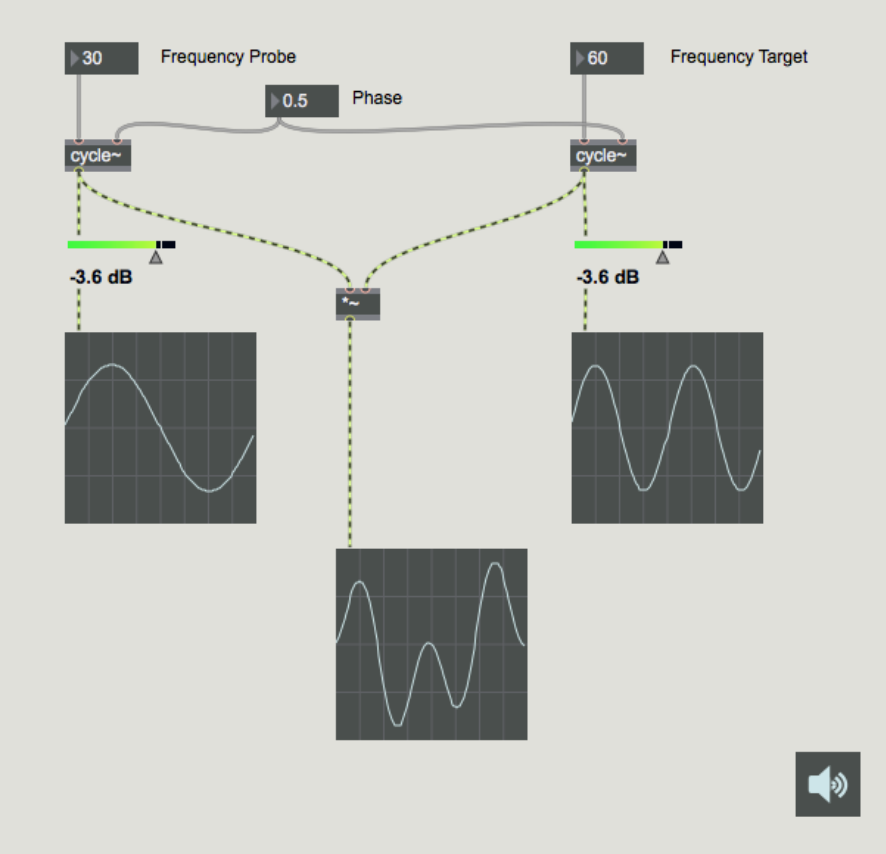

Figure 4.7 Multiplication of sine waves with different frequency.

4.5 Phase

Until now, we only considered exactly equal sine waves for fourier detection, but this will rarely happen in real case scenario. In the next example (figure 4.8), we observe how sine wave multiplication behaves when the two signals differ in phase.

As described in previous figure 4.6, the first case that we observe in fig. 4.8 is a multiplication of two sine waves at the same frequency with identical phase, so as expected, the average or offset of the result is 0.5. When phase is shifted 0.25 (pi/2) as well as when it is shifted 0.75 (3pi/2) the resulting offset is 0. And finally, when shifted 0.5 (pi) the offset results in -0.5.

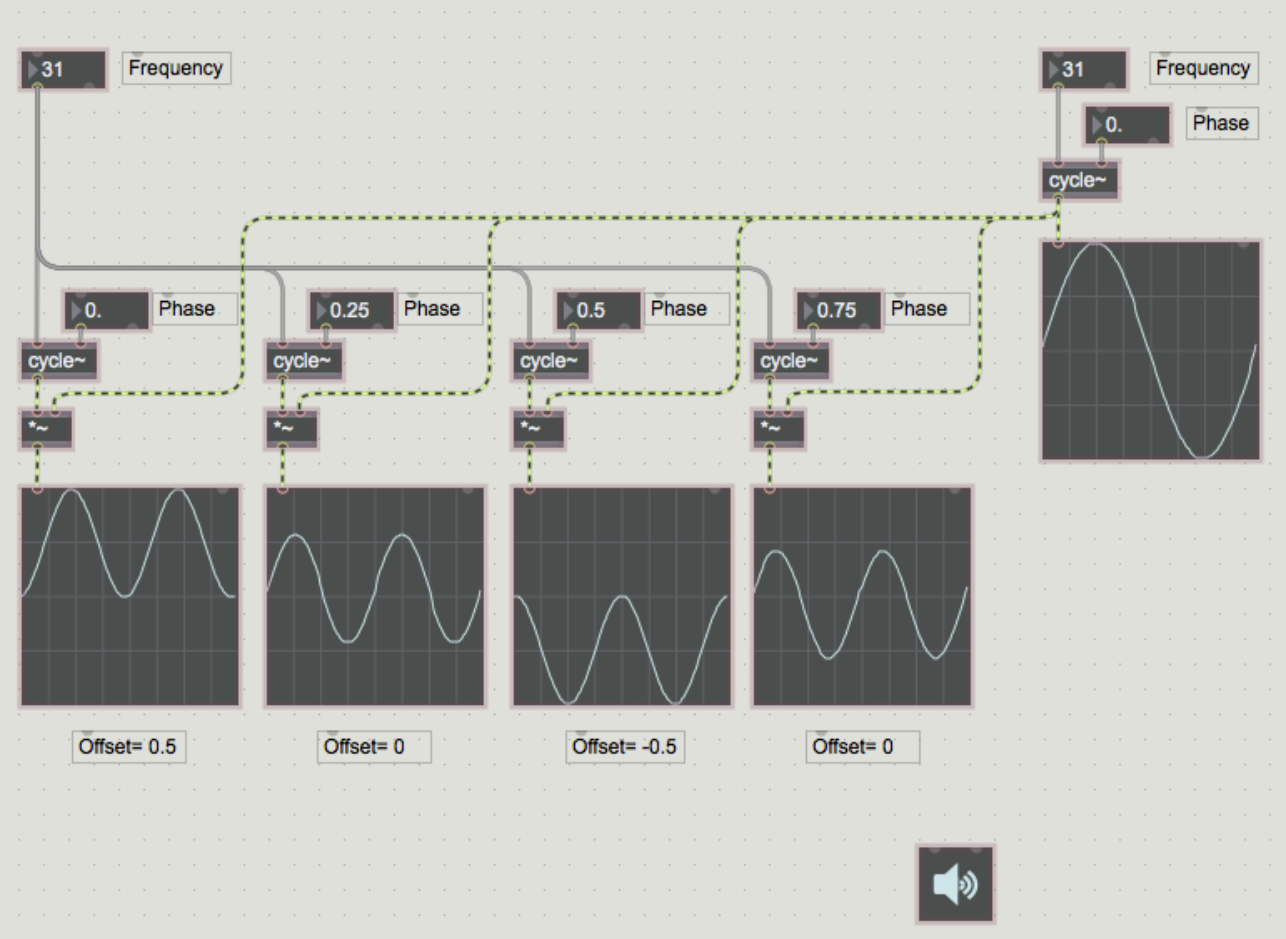

Figure 4.8 Two sine waves multiplied several times at same frequency and different phase.

The first and most obvious conclusion one comes to is that this detecting method will not work when there are phase differences between the probe and the target signal, which, in a real case, would be most of the times. So one could think that, if it's necessary for both signals to have the same phase, a solution could be probing with as many phases and take the best match. While this would work, it would also result in an extremely extensive calculation, and therefore in an Extremely Slow Fourier Transform (ESFT).

Taking a second look at figure 4.8, the worst result is obtained when the phase is shifted 0.25, which actually turns the sine into a cosine, giving a resulting offset of 0 (no match). When doing the same experiment with a cosine signal as a probe, results are exactly the opposite. perfect match (offset= 0.5) is obtained when the phase is shifted 0.25, that is, when

both signals are cosines. That gives an indicator that all possible results will be bounded in this stripe. When the target phase lies in between those extrems, both mesurements will get a partial match.

Using the identity: for any possible theta we can obtain the exact phase and amplitude using sine and cosine probes. This reduces testing with any possible phase value to only two, and it's also the basis for the DFT (Redmon, 2002).

4.6 DFT and STFT

DFT or Discrete Fourier Transform, is the adaptation of Fourier Transform to the discrete (digital) domain. Performing a DFT presents several redundancies, and by exploiting them, a faster version of FT called FFT (Fast Fourier Transform) can be used. In order to do that, a power-of-two length for the FFT must be used, meaning, the length of the analysis time slice (in samples) should be a power of 2.

As mentioned at the beginning of this chapter, Fourier Transform will work correctly only when dealing with a single cycle of a periodic waveform. Until now, tests have been run over one cycle of periodic signals, but for discrete signals, when using FFT, the interval will be an arbitrary time slice measured in samples (2^n for FFT).

Normal signals that will be used with FFT won't usually be periodic, and even if they were, by setting and arbitrary time slice to FFT will ensure that a cycle of the incoming signal will almost never fit perfectly in the FFT window. We can still get results with the transform, but there is some "spectral leakage."

When working with FFT, common values for time slices normally are 512, 1024, 2048… (always 2^n samples). By setting the amount of samples of the time slice, the lowest frequency analysable is also set. This is due to setting an amount of samples at a certain sampling rate,

will give us a fundamental probe at sampling rate/windowsize= hz, called the fundamental FFT frequency. Lower frequencies have longer periods that won't fit in the FFT time slice and therefore won't be detected.

Besides probing with the FFT fundamental, it continues with the harmonic series (2x, 3x, 4x…) through half the sample rate. At that point, there are only two sample points per probe cycle, the Nyquist limit. It also probes with 0x, which is just the average of the target and gives us the DC offset (Redmon 2002). That's why FFT spectrum is harmonic spectrum of the FFT frequency.

Another quality of FFT is that the amount of samples in the slice will also determine the amount of bands obtained in the analysis also known as frequency bins.

To clarify the FFT principles and see where its limitations are, let's think about a practical case:

Having set the FFT window at, for example, 1024 samples, gives us a fundamental FFT frequency of 44100/1025= 43.07 Hz. If the signal being analysed is a sine wave at 43.07 Hz, or a multiple, then it will fit perfectly in the FFT window, and the analysis will reveal a maximum energy in the corresponding frequency bin (second bin for the fundamental, third bin for the 2^{nd} harmonic, etc). On the other hand, if the frequency wasn't a harmonic of the fundamental, the analysis will reveal a great energy in the bin corresponding to the closest FFT harmonic of the frequency being analysed, and also smaller amount of energy in all the rest of bins. While this could lead to acceptable results, it's not the optimal result. In order to reduce such errors in the form of spectral leakage, there are ways to "force" the incoming signal to become "periodic" and fit the 2^n samples, like perform FFTs repeatedly, overlapping its windows. Those overlapped windows are called also frames (like the frames in video) and are the basis of the Short-time Fourier Transform (STFT). While using the STFT will smooth

the results, by implementing a technique known as Phase Vocoder, the exact frequency deviation from the centre bin can be calculated. Since phase is a relative measure, it's in the difference where the information relays. Phase Vocoder uses phase difference between successive FFT frames instead of phase values, allowing us to know the exact frequency of a bin, giving us the deviation from the closest harmonic.

4.7 Max and Fourier Transform

Fourier Transform, or more specific, FFT formula is encapsulated in the fft~ object in Max/MSP. This object receives a signal in its inlet and for each slice of time it receives (512 samples long by default) it sends out a signal of the same length listing the amount of energy in each frequency region (Dobrian, 2001, MSP Analysis tutorial 3). The output signal is not an audible signal though, and it's not a single stream, but two signals in parallel consisting of lists of real (left output) and imaginary (middle output) numbers which are the result of the FFT of the previous slice at the input (512 samples by default).

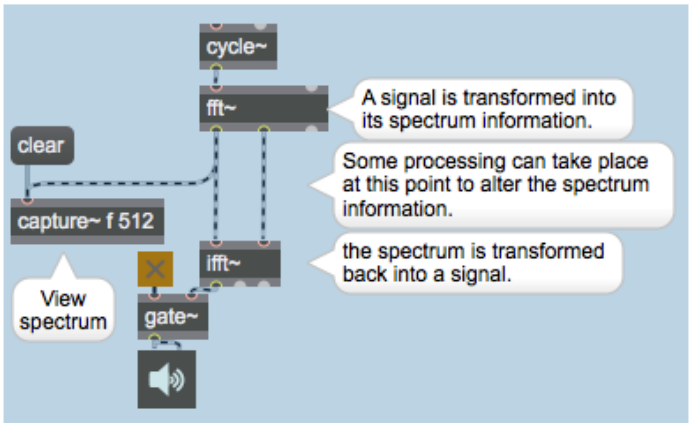

Figure 4.9 Simple example of FFT in Max (Dobrian,2001)

These (numbers) are not the amplitude and phase of each bin, but should be thought of instead as pairs of Cartesian coordinates, where *x* is the real part and *y* is the imaginary, representing points on a 2-dimensional plane. (See Figure 4.9)

The amplitude and phase of each frequency bin are the polar coordinates of these points, where the distance from the origin is the bin amplitude and the angle around the origin is the bin phase.

As mentioned before, the known problem with FFT is that when the slice being analysed doesn't comprise one, or a multiple amount of cycles, the results will contain many spurious frequencies not actually present in the signal.

This is very simple to test in Max/MSP. Using the patch from figure 4.8, and given that by default the fft~ object time slice is 512 samples long and sampling rate is 44100 Hz, the FFT fundamental frequency will be 44100/512= 83.131281 Hz, three different frequency sinusoids will be connected into fft- object, FFT fundamental, FFT fundamentalx10 (10th harmonic) and a slight variation from the 10th harmonic frequency. Next figures show a portion of the captured spectrum.

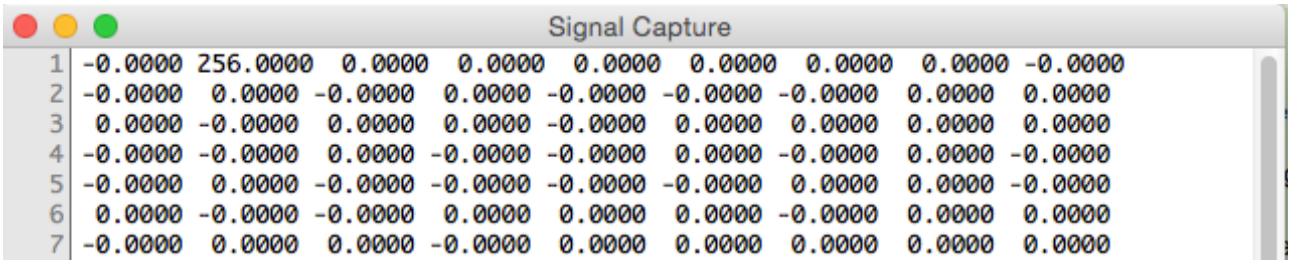

Figure 4.10 Captured spectrum when the input is the FFT fundamental

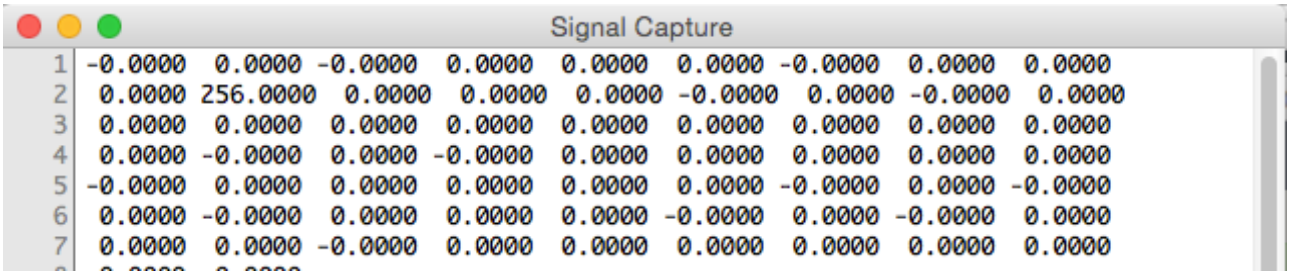

Figure 4.11 Captured spectrum when the input is 10x the FFT fundamental

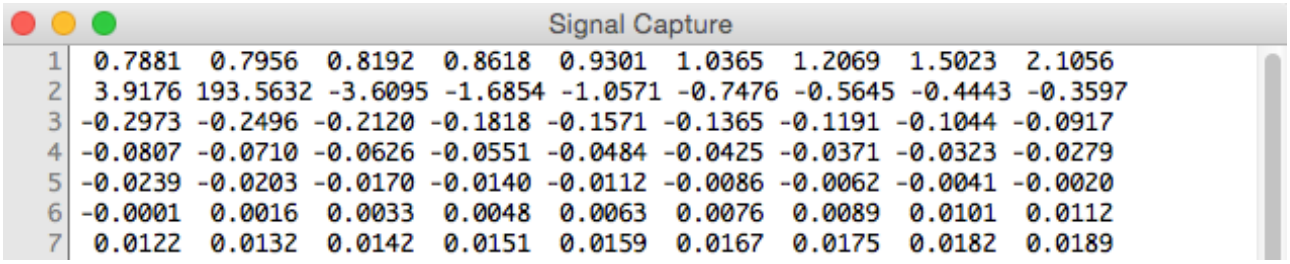

Figure 4.12 Captured spectrum when the input is a sligh variation of the 10th harmonicth of the FFT fundamental

In fig 4.10, and 4.11 the result shows perfectly the maximum amount on the correct frequency bin, 2nd bin for the fundamental, 11th bin for the 10th harmonic. On the other hand, changing slightly the frequency input form 10th harmonic a couple of Hz (from 831.31282Hz to 833) in fig 4.12 will make the input not fit perfectly in the fft window and hence, the result of the analysis shows a greater energy in the 10th harmonic and spurious frequencies present on all the rest of bins. In order to avoid that the STFT principles (perform FFTs repeatedly and overlapping its windows) can be applied. There's a patching work around to achieve STFT in Max, however, this approach can often be a challenge to program, and there is also the difficulty of generalizing the patch for multiple combinations of FFT size and overlap. Since the arguments to fft~/ifft~ for FFT frame size and overlap can't be

changed, multiple hand-tweaked versions of each subpatch must be created for different situations. For example, a percussive sound would necessitate an analysis with at least four overlaps, while a reasonably static, harmonically rich sound would call for a very large FFT size. The pfft~ object addresses many of the shortcomings of the basic fft~ and ifft~ objects, allowing you to create and load special 'spectral subpatches' that manipulate frequency-domain signal data independently of windowing, overlap and FFT size. A single sub-patch can therefore be suitable for multiple applications. Furthermore, the pfft~ object manages the overlapping of FFT frames, handles the windowing functions for you, and eliminates the redundant mirrored data in the spectrum, making it both more convenient to use and more efficient than the traditional fft~ and ifft~objects (Dobrian, 2001, MSP Analysys Tutorial 4).

5. Method

5.1 Deciding my Approach

During the process of writing this thesis, I've been trying different softwares and algorithms to perform sound morphing. The initial idea was using Native Instruments Reaktor, to develop a piece of code that could be used in the new Blocks enviroment. After realising that coding in Reaktor may not let the user go as deep as creating their own algorithm, I changed my mind and started trying Loris and the Reassigned Bandwidth-Enhanced Additive Sound model. Even if I found the approach clever and straightforward, the software wasn't as straightforward and I couldn't have any control of the algorithm itself. Then I tried to introduce Max in the ecuation, using the *sdif* data outputed by the Loris analysis with the sdif externals that cnmat developed. Noticing the versatility of Max, and after learning about it's analysis MSP objects and Jitter matrices, I decided to switch my approach completely to Max. The implementation that I present here is based on Luke Duboi's phase vocoder patch jitter_pvoc_2d.pat, distributed with Jitter and Jean-François Charles article *A Tutorial on Spectral Sound Processing Using Max/MSP and Jitter*, published in Computer Music Journal (2008)

MSP and Jitter are extensions to the Max software that were added in 1997 and 2002 respectively. MSP brought audio processing in real time to max, while Jitter allowed Max users to process video, 3D and matrices.

Note that programming in Max is somehow similar to clasisical programming where we can have several events triggered by diferent function or actions while in MSP all the events are happening continuosly in all signal paths.

5.2 Recording FFT Data in Jitter Matrices

The first problem one faces when working in frequency domain in Max is that FFT representation is two-dimensional while the audio stream and and buffers in Max are on - dimensional. The two-dimensional nature of the data and the one-dimensional framework makes coding of advanced processing patterns somewhat difficult. The introduction of Jitter, enabled manipulation of matrices in Max and hence two dimensional data is now straightforward in the Max environment.

Matrices in Jitter can have multiple planes. In this case, I'll be using two-plane matrices for storing amplitude values and phase difference values derived from the FFT analysis.

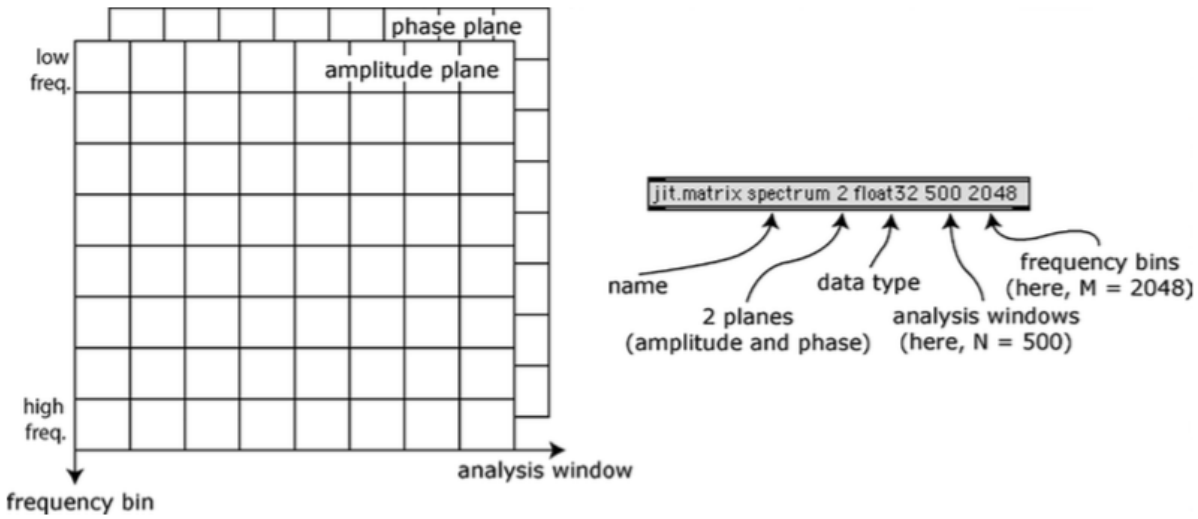

Figure 5.1 A representation of a two-plane Jitter matrix and the different arguments in the Jitter object that defines it (Charles, 2008).

In both planes of the matrix, every column will represent a frame, while the cells in that column (equal to half of the FFT size) is the number of frequency bins.

In order to record FFT spectrum into a Jitter matrix, a pfft~ object that will launch the subpatcher where the frequency domain process will be carried out must be created.

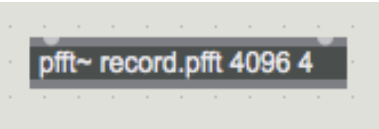

Figure 5.2 The pfft~ object in Max/MSP

This object's arguments define the name of the subpatcher (name.pfft) the FFT size (4096 samples in this case) and the overlap factor. Both inputs can handle messages or signal information. Messages will perform action in the subpatcher while audio signal will be used for the STFT.

Once created, double clicking on it will open the subpatcher where STFT will be performed and data will be recorded into the Jitter matrix.

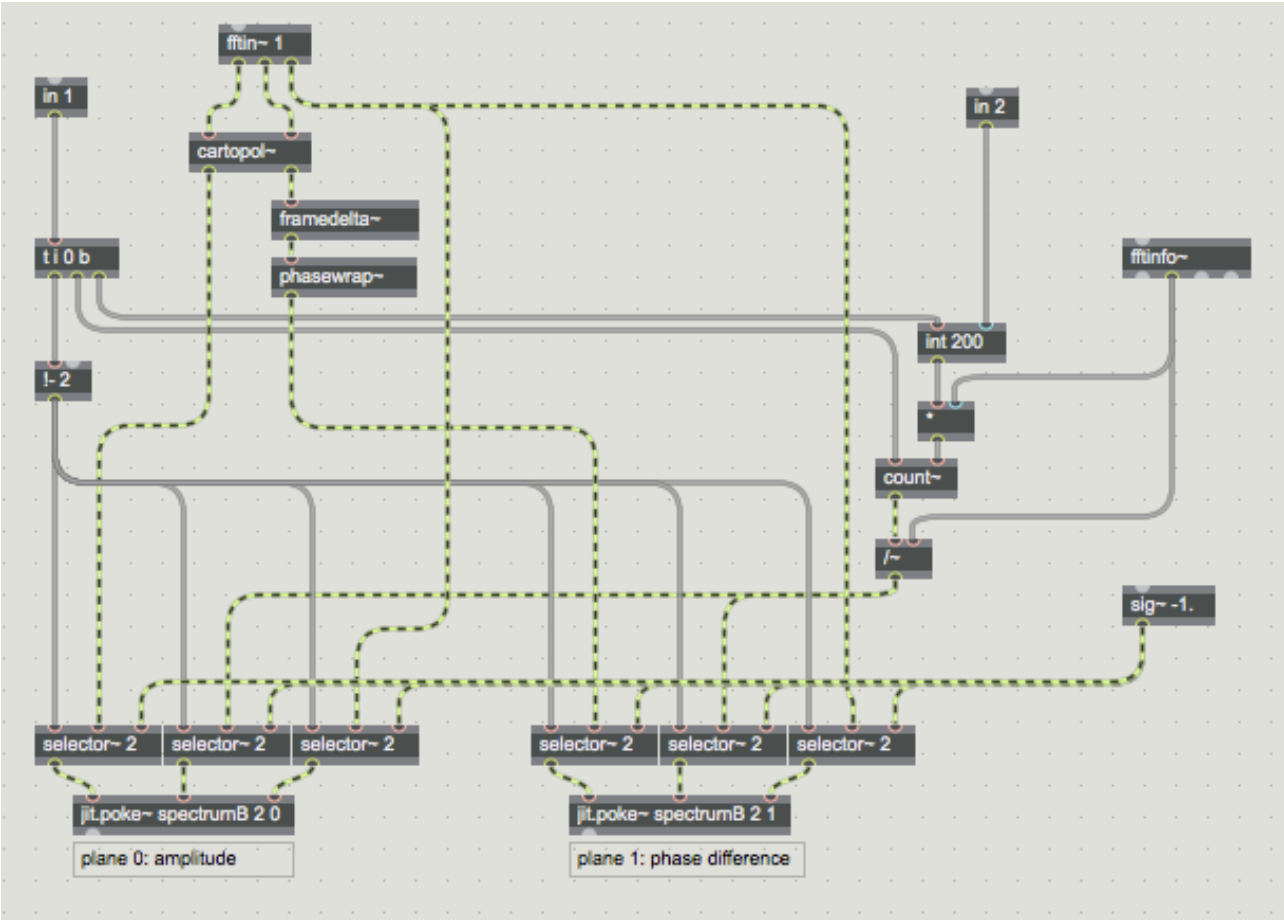

Figure 5.3 Max/MSP patch for recording spectral information into jitter matrices.

The final object on charge of writing data in the matrix is the jit.poke~. This object counts with 3 inputs. The first one is the actual data to be written in the matrix. Second and third input will set the x and y coordinates inside the matrix where the data in the first input will be written in. Directly connected to the jit.poke~ object are the selectors, the function of which is no more than selecting between actual data (when we tell the pfft~ object in the parent patch to start recording) or sending a -1 signal that will make the jit.poke~ object stop recording in the matrix.

There's one jit.poke~ object per plane, so data about amplitudes will be sent to the first one and the second one will be receiving phase deviation data. The amplitude data comes almost straight from the fftin~ object, just after being transformed into polar coordinates. For

the phase difference two more steps are necessary, first *framedelta~* object computes a running phase deviation, then the phasewarp~ object warps those values between -π to π. Both amplitude and phase values are passed to their respective jit.poke~ object that will write in the respective planes in the *spectrum* matrix.

Finally we just need to syncronise x (column) and y (row) writting position in order to increase the row value everytime we reach the end of a column. The method will be valid for both planes. To do this, the third output of fftin~, which output is the Bin index, will set y writing position. By setting a counter that goes through all the cells in the matrix (nr of frames multiplied by nr of bins) and getting just the integer part when dividing it by the bin nr again, we obtain a counter that increases 1 every time the end of a column is reached.

5.2 Controlling the Playback

Once we have all the data stored in matrices, given that we know the amount of frames and frequency bins to reproduce and the sampling rate, we can obtain the duration of the sound and adjust the a playback rate by multiplying the frequency for a normal playback by a factor that will determine the speed and direction of it.

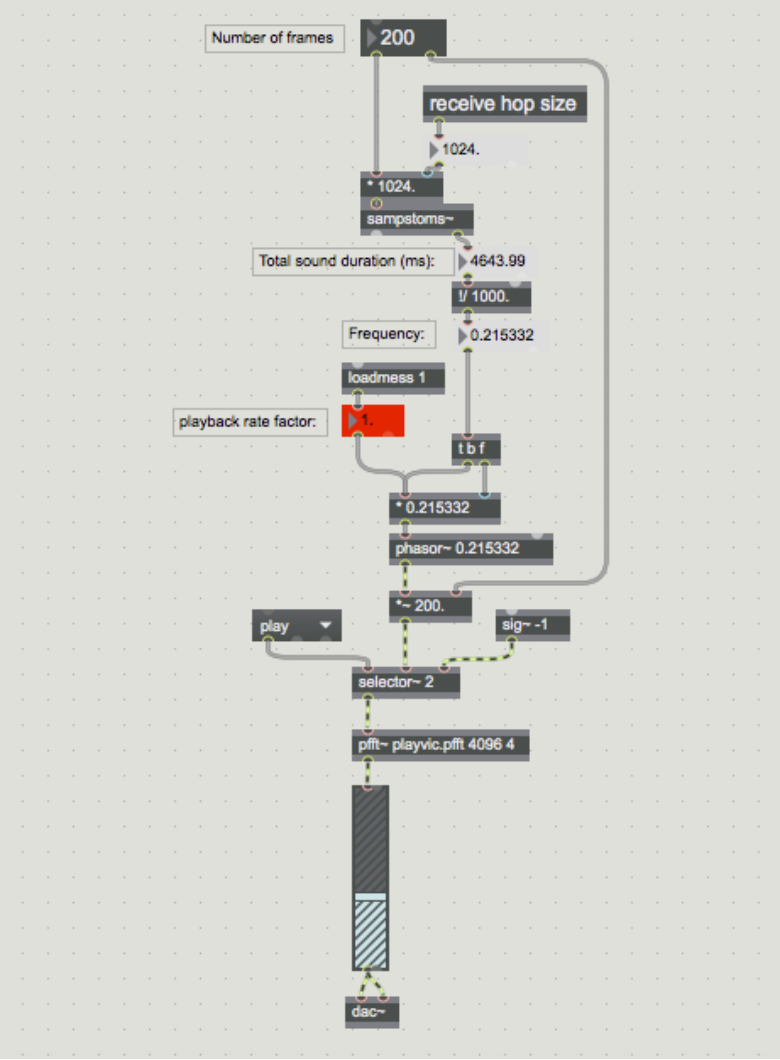

Figure 5.4 Max/MSP patch of the playback control section adapted from Charles Tutorrial

The fftinfo- object third outlet will give us the hop size in samples, by multiplying it by the number of frames we will obtain the amound of samples to be reproduced. Dividing the result by the sampling frequency give us the duration of the sound. In this particular case:

Duration of the sound= (200*1024)/44100= 4644ms

We can use a phasor- object at a frequency that will give us a 4644ms long raise signal:

$f_{phasor\text{-}}$=1000/4644= 0.215332s

The phasor- object will then output a signal that will take 0.215332s to go from 0 to 1.

Scalating it by the number of frames,200 in this case, we get a signal that goes from 0 to 200

in the desired amount of time, telling the pfft- object wich frame to reproduce. Inside the pfft- object no fft will happen thanks to the nofft argument set in the fftin- object in the subpatcher. This MSP trick allows to send control signal data from the parent patch into the spectral subpatch, hence allowing us to control the Reading speed from the matrix inside the subpatcher with the phasor- object.

Multiplying the $f_{phasor-}$ by a factor, we will change its frequency setting it's raise to different durations affecting the playback speed and direction, hence allowing us to control it.

5.3 Sound Morphing Using the jit.xfade Object

5.3.1 Basis

For my playback controllable morphing algorithm patch, I took inspiration from one of the techniques introduced by Jean-François Charles in his paper *A tutorial on Spectral Sound Processing Using Max/MSP and Jitter* (2008). In the paper, Charles describes the *Frame effect* as an artefact that appears when time-stretching a signal to the extreme, when the leap between frames at the resynthesis stage becomes audible. His technique uses the jit.xfade object to interpolate between consecutive frames in order to reduce the frame effect. In my approach, I used the jit.xfade object as the center piece for my algorithm, interpolating between frames values from two different sound sources, and controlling the evolution of the morph integrating the playback control patch from the previous subsection.

5.3.2 The Patch

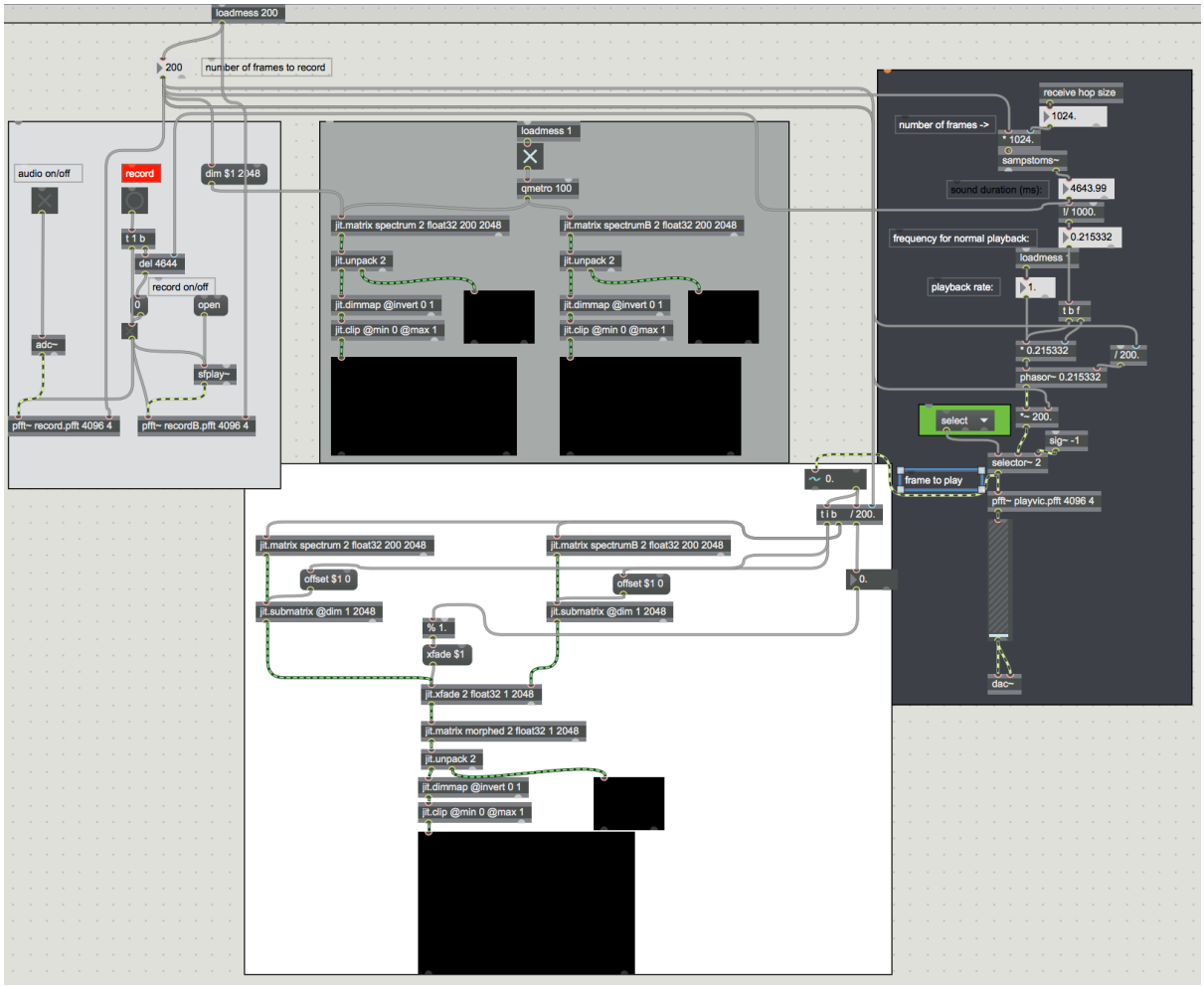

Figure 5.5 Playback Controllable Morphing Algorithm Overview

The patch is divided into 4 sections. The left section is called loader. Here is where sounds will be selected or captured and recorded into jitter matrices using pfft- objects. The right section is called the playback control and it's self-explanatory. The middle upper section is called the matrices section, and contains both matrices and their visualisations. The last section is the interlpolation, and it's were the jit.xfade object interpolates values from both matrices.

5.3.2.1 The loader section

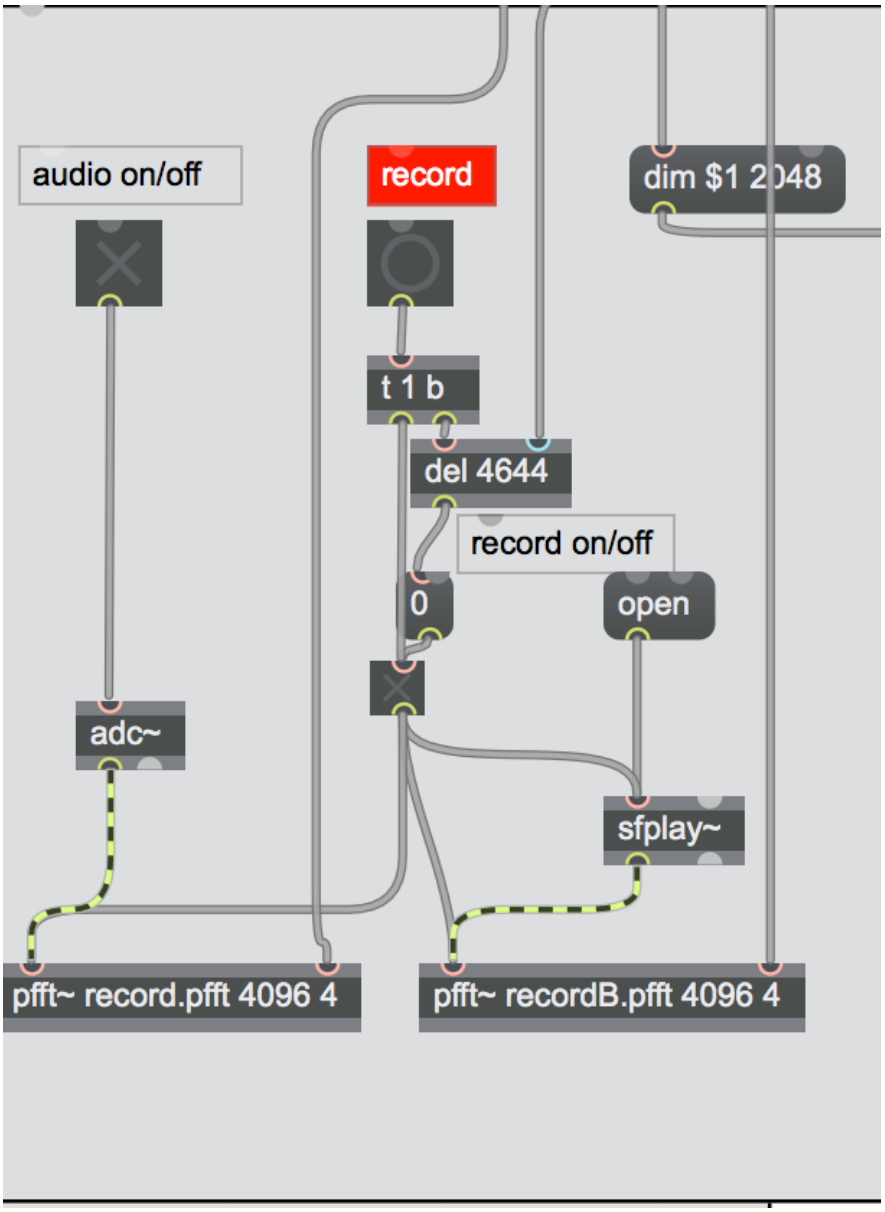

Figure 5.6 The loader section

This is the first section the user will interact with. At the top of the patch we find the audio toggle the open message the record button and a number box for setting the frames number we want to work with (200 by default). One can start by switching on the audio engine. This action will make MSP objects start to work and also will start sending the audio captured by

channel 1 of the audio device being used (laptop microphone if no external device is selected) to the first pfft- record object. Matrices won't start being written yet though. After that, clicking in the open message will open a browsing window for selecting the sound file we want to be recorded in the second matrix. At this point we are sending audio to both of the pfft- objects. The record button will send a bang setting on a toggle connected to it. This toggle will send a 1 message to pfft- objects that will cause the subpatchers linked to them to start recording the incoming audio signal in its respective matrices. In the interpolation section, both jitter displays will show the data being written in the matrices. The toggle that the record button is connected to, also has a 0 message connected to it. This 0 message will be fired at a specific time after the start of the recording, setting the toggle to 0 and causing the pfft- object to stop recording in the matrices. The time until the 0 message is fired is taken from the playback control section, and will change depending on the total amount of frames to be recorded, the sampling rate and the overlapping factor of the STFT set at the pfft- object.

5.3.2.2 The playback control section

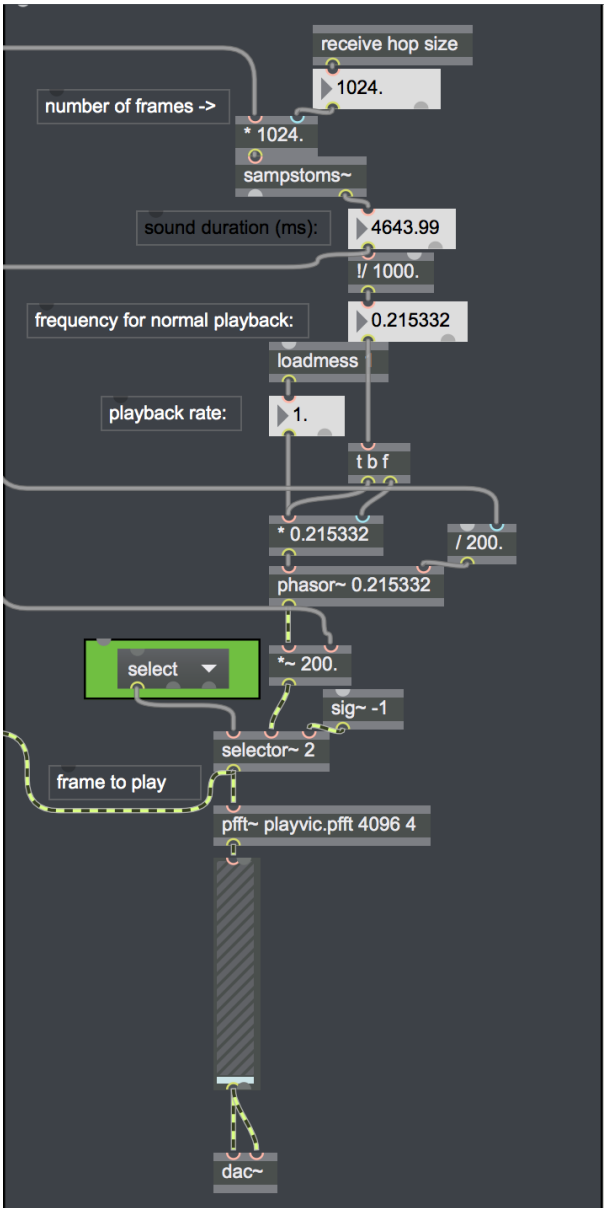

Figure 5.7 The Playback control section

This section incorporates the playback control patch described in 5.2 to the morphing algorithm. It will send the time duration to the delay object in the loader section that will stop the recording after the 200 frames (per default) have been recorded in the matrices. With the

playback rate integer- object, it's possible to control the speed and direction of the playback, where 1 will be normal speed, 0 will be a spectral freeze, and negative values will cause the playback to go backwards. This is the control the user will interact with, being able to "scroll" the sound in the morphed matrix. As well as sending the "frame index" to the pfft- object to tell it which frame to read from the morphed matrix, we send it to a number- object. This number- object will display the incoming "frame index" signal and generate constants out of it, so basically it acts as an interface between the MSP "signal world" to the Max constants values. This will allow us to divide such constant by the number of frames, scaling the 0 to 200 interval of the "frame index" signal, to a 0 to 1 interval that will last the same time as the reproduction of the 200 frames. As we'll see in next section, with this we'll be able to set the jit.xfade object interpolation progression from one matrix to another at the same speed of the playback.

5.3.2.3 The matrices section

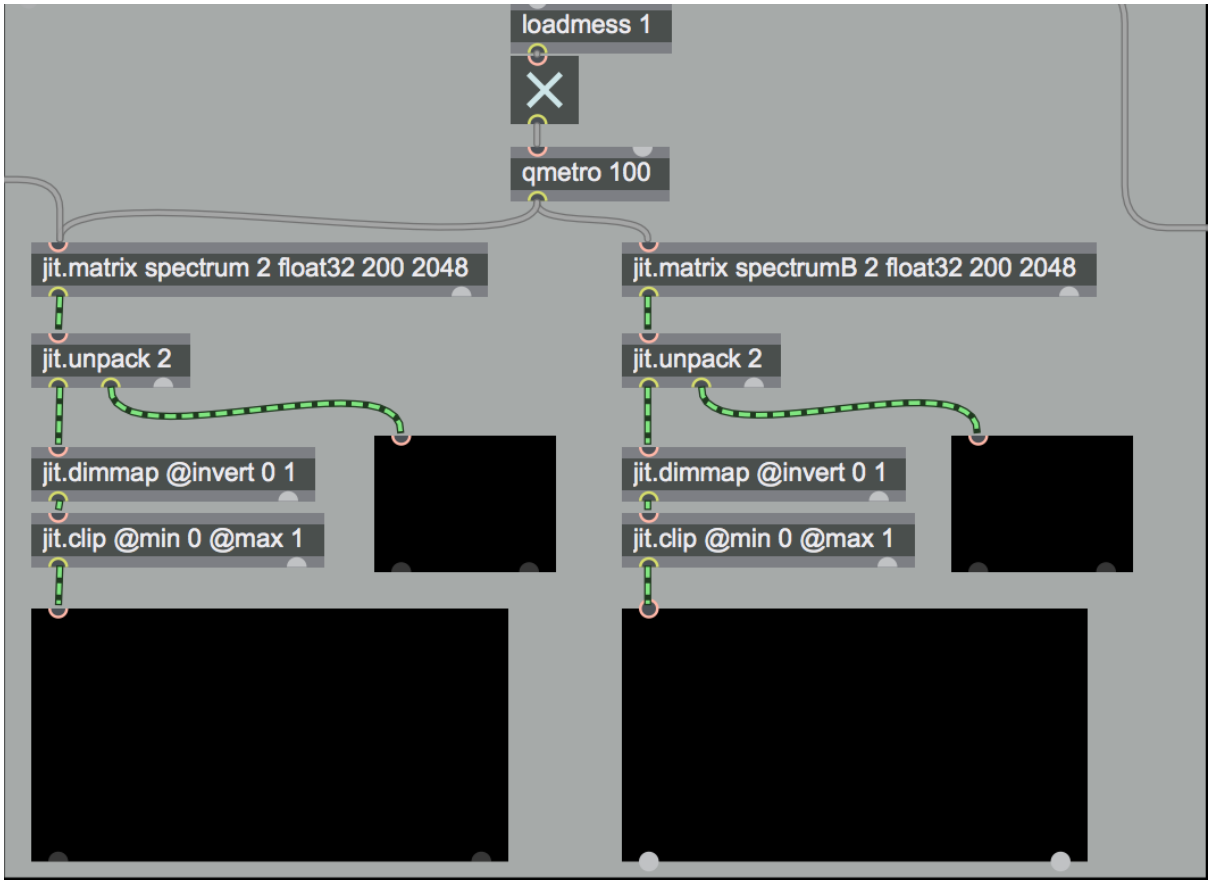

Figure 5.8 The Matrices section

In this section matrices for the two sound sources are defined. In this matrices, spectral information from the sounds to be morphed is written. A metro object is connected to both matrices to make them output their recorded value. The output signal is then splitted with the jit.unpack object, this way, we'll have amplitude and phase information in two separated channels. Those signals are then fed to jit.window objects that will plot the information contained in the matrix.

5.3.2.4 The interpolation section

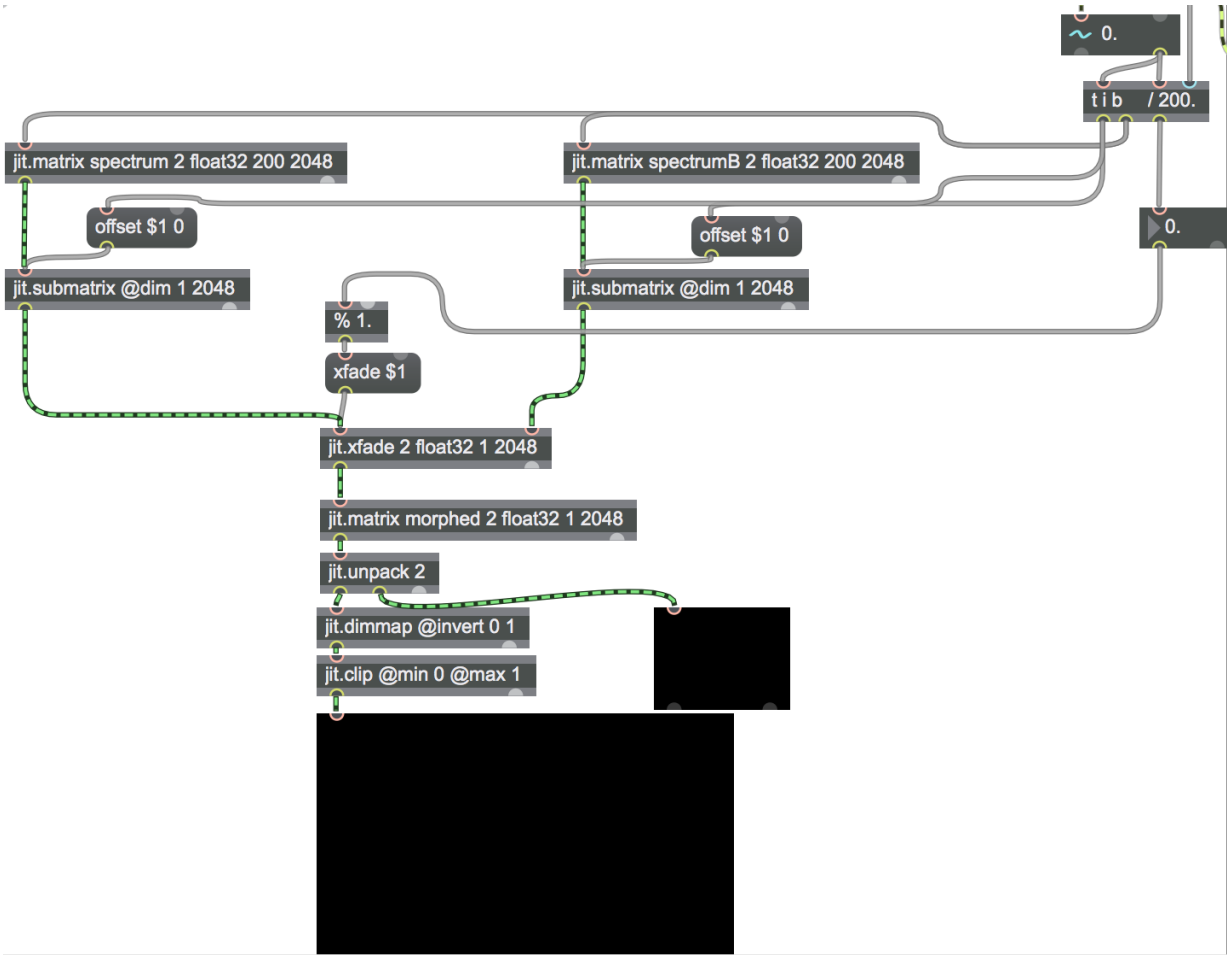

Figure 5.9 The Interpolation section

This is the section where the actual morph happens. The jit.xfade will interpolate values from the two matrices connected to it. The weight of the interpolation where 0 will be 100% sound A and 1 will be 100% sound B is set to 0.5 by default. By gradually changing the weight factor at the same time of the playback, we'll obtain the desired sound morphing, gradually evolving from sound A to sound B. This is achieved with the escalated "frame index" obtained in the playback section. The scalated "frame index" outputs float numbers from 0 to 1, during the duration of the 200 frames (or any other number of frames set by the user) playback. Sending this values to together with the correct message for changing the

weight factor in the jit.xfade object, will result in a gradual morph, that the user will be able to reproduce forward, backwards or freeze with the playback rate control.

6. Conclusion

Sound morphing it's one of the most powerfull and creative sound techniques still being developed. The hability of discovering "unexisting" intermidiate timbral spaces between different sounds is as fascinating as complex. While there's a lot of research about perceptually perfectioning the technique still going on, the idea in this thesis was to focus also in a more interactive aproach, that could make the player have some control over the morphing process.

Max, and its MSP and Jitter extensions have proved to be extremely convinient for this task. Objects like pfft- encapsulate so many functionalities for FFT, simplifing the process for the user. The way it separates time-domain and frequency-domain with its subpatchers is also very handy and facilitates the understanding of FFT processes to newcomers.

Unfortunatelly I wasn't able to get audible satisfying results with my decission of using the jit.xfade for spectral interpolation duties. This decision was made after reading Jean-François Charles *A Tutorial on Spectral Sound Processing Using Max/MSP and Jitter* (2008) where he uses it to interpolate consecutive frames of a sound. The interpolation in the patch here presented is conceptually identical to the one in Jean-François Charles paper, but for some unknown reason to me, (probably an error in my programming) the outcome is not the expected one.

On the other hand, even if the audible result is not as expected, this thesis also brought a new point of view for defining sound morphing and also an approach to Fourier Theory and spectral processing in Max/MSP and Jitter.

6.1 Futre Work

In the close future, the algorithm presented in this paper will be fixed in order to obtain the expected audio results. Furthermore, having set a solid basis for spectral transformation using Max/MSP and Jitter matrices, the algorithm can be improved by programing a new interpolating object, or by using mathematical opeartor objects in Max in order to transform the spectral content stored in the matrices to achieve a sound morph.

The integration of Max with music production DAW Ableton live, gives the posibility of adapting the algorithm to this platform in a future being able to interact with in a “programming free” enviroment for musicians and music productors.

7. Acknowledgements

En primer lloc, donar les gràcies a la meua família, pel seu suport, treball i esforç per fer possible aquesta aventura.

També agrair als meus companys, en Gandia a David i Rafa, amb els que aquestos 4 anys de sofriment hauran servit, com a poc, per a guanyar dos bones amistats, i en Alemanya, a Ana i Adriana, per oferir altruistament la seua ajuda en qualsevol moment.

En tercer lloc, agrair el increïble i incansable suport de Esra, que ha alleugerat aquesta càrrega amb la seua estima.

Finally I would like to thank Dr. Prof. Michael Oehler for the support and for giving me the oportunity of working in a field that I love.

8. References

[1]Bogaards, N. & Röbel, A. (2005). *An interface for analysis-driven sound processing*. Retrieved from: articles.ircam.fr/textes/Bogaards05b/index.pdf

[2]Dobrian, C. (2001). Max 7 Help and Documentation Cycling ’74. Available: http://www.cycling74.com/docs/max5/vignettes/intro/docintro.html

[3]Redmon, N. (2002). *A gentle Introduction to the FFT.* Retrieved from: http://www.earlevel.com/main/2002/08/31/a-gentle-introduction-to-the-fft/

[4]Roads, C. (1996). *The Computer Music Tutorial.* Cambridge, Massachusetts: The MIT Press

[5]Colasanto, F. (2010). *Max/MSP Guía de Programación para Artistas.* Mexico: Centro Mexicano para la Música y las Artes Sonoras.

[6]Olivero, A. Torrésani, B. Depalle,P. Kronland-Martinet, R. (2012) *Sound mor- phing strategies based on alterations of time-frequency representations by Gabor multipliers.* AES 45th International Conference on Applications of Time-Frequency Processing in Audio, Mar 2012, Helsinki, Finland. pp.17

[7]Caetano, M. and Rodet, X. (2011) *Sound Morphing by Feature Interpolation.* Czech Republic: IEEE International Conference on Acoustics, Speech and Signal Processing,

[8]Caetano, M. and Rodet, X. (2013) *Musical Instrument Sound Morphing Guided by Perceptually Motivated Features.* IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING, VOL. 21, NO. 8, pp. 1666-1675

[9]Hatch, W. (2005). *High-Level Audio Morphing estrategies*. Montreal, Canada: McGill University.

[10]Sibylle Bammer, R. (2015). *Signaltransformation via Gabor Multiplier.* Wien: Universität Wien.

[11]Tellman, E., Haken, L. and Holloway, B. (1995). *Timbre morphing of sounds with unequal numbers of features.* Journal of the Audio Engineering Society 43(9)

[12]Sethares, William A.; Milne, Andrew J.; Tiedje, Stefan; Prechtl, Anthony and Plamondon, James (2009). *Spectral tools for Dynamic Tonality and audio morphing.* Computer Music Journal, 33(2) pp. 71–84.

[13]Moorer, J. A. (1978). *The Use of the Phase Vocoder in Computer Music Applications.* Journal of the Audio Engineering Society. Retrieved from: www.jamminpower.com/PDF/Phase%20Vocoder.pdf

[14]Droljc, T. (2011). *STFT Analysis Driven Sonographic Sound Processing in Real-Time using Max/MSP.* Yorkshire, United Kingdom: University of Hull.

[15]Fernández-Cid, P. (2014). *Sistemas de convolución: Teoría.* Retrieved from: http://www.hispasonic.com/tutoriales/sistemas-convolucion-teoria/39446 .

[16]Smith, S.W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing.

[17]Auger, F. and Flandrin P. (1995). *Improving the readability of time-frequency and time-scale repre- sentations by the reassignment method*. IEEE

[18]Charles, J. F. (2008). *A Tutorial on Spectral Sound Processing Using Max/MSP and Jitter.* Computer Music Journal

[19]Fitz, K., Hakken, L., Lefvert, S., and O'Donnell, M. (2002) *Sound Morphing using Loris and the Reassigned Bandwidth-Enhanced Additive Sound Model.* University of Michigan Library: Michigan Publishing.

[20]Fitz, K., Hakken, L. (2003). *Current research in Real-Time Sound Morphing.* Retrieved from: http://www.hakenaudio.com/RealTimeMorph/

[21]Fitz, K., Haken, L. and Christensen, P. (2000). *Transient preservation under transformation in an additive sound model*. In Proc. ICMC, Berlin, Germany, pp. 392–395.

[22]Krumhansl, C. L. (1989). *Why is musical timbre so hard to understand?*. in Structure and perception of electroacoustic sound and music, Nielzén, S. and Olsson, O. Eds. New York, NY, USA: Excerpta Medica, pp. 43–54.

[23]Caclin, A., McAdams, S., Smith, B. K. and Winsberg, S. *Acoustic corelates of timbre space dimensions: A confirmatory study using syn- thetic tones.* J. Acoust. Soc. Amer., vol. 118, no. 1, pp. 471–482, 2005.

[24]Grey, J. M. and Moorer, J. A. (1977). *Perceptual evaluations of synthesized musical instrument tones.* J. Acoust. Soc. Amer., vol. 62, no. 2, pp. 454–462,.