UNIVERSITE PARIS 8 VINCENNES À ST-DENIS
UFR ARTS. PHILOSOPHIE, ESTHÉTIQUE
Département de Musique

Domaines fonctionelles d'un vocodeur de phase

Techniques algebriques du processus sonore

KARYSTINAIOS Emmanouil-Nikolaos

Mémoire de Master 2 réalisé sous la direction de :
BONARDI Alain

Année universitaire
2018-2019.

Resumé

...and then there was sound. Without sound music could not have been born. Understanding sound is has been the quest of many throughout the years. Playing with timbre, dynamics, register, colors, naturality and every other aspect we can think of. During the second half of the 20th century composers and researchers starting wondering about what composes sounds. What is the components that make a sound of a violin differ from a sound of a flute when they play the same note. And thus spectralism was born as a path to the true understanding of sound.

Therefore this dissertation focuses, in its core, on the exploration of the nature of sound, and to be more precise, it's an investigation on spectral sound moprhing in real time. In this paper the process of spectral processing its going to be thourally analysed and a build from scratch Phase Vocoder is going to be presented.

First is crucial to unvelop the mathematical notions behind the theory of spectral decomposition and re-synthesis. Therefore, section one deals with all the relevant math one will need to undestand the how to process spectra of sounds. Besides the mathematical formuli we set a musical context to an more musician-friendly approach.

Thereafter, we implement this mathematical context again in a musician-friendly programming environment such as the software MaxMSP. We investigate how the mathematical formuli interact with each other in a code setting. Step by step musical and programming explication build their way throught to the birth of a simple Phase Vocoder.

Finally, a combination of the knownledge we gained from the previous chapters and some basic musical programming principles we build a series of artistic implementations in MaxMSP that underline the artistic value of a phase vocoder.

One can expect this dissertation to be a musicians guide into spectral processing but also a fruitfull source of artistic examples and implementations. Its usefull for every musician or composer who seek to understand the logic behind the tools he frequently uses ; anybody who wonders how to actually implement a Phase Vocoder in MaxMSP or any artists looking for ideas for a future project. It is to be expected from this research to envellop different uses of the Phase Vocoder and give access to further parametrisation.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 About

Nowadays music has become a vast subject of research science. With the branches of music growing throughout various domains such as, musical analysis with mathematical modeling, history with ethnomusicology, acoustics with physics, composition with programming, etc. Following this interdisciplinary path we are called to bind our core, music, with the other sciences in order to advance towards a fresh point of view and present some new results.

In composition branch of music, the study on the nature of sound was especially developed producing various techniques guided by physical phenomena. A series of sound processing tools based on physical phenomena are labeled under a branch called sound analysis-synthesis [1]. In this field an ensemble of scientific disciplines come together for a single purpose, understanding music and sound. Thought analysis we proceed to Synthesis. From science we lead to art. The name of analysis-synthesis expresses exactly this principle, from observation (analysis) one can proceed to sound processing and result to a new sound-product (Synthesis). The name is literally applied on spectral processing where the analysis stage transfers a sound from the time to the frequency domain. The analysis presents the spectral information of a sound at a certain time period. The synthesis tool inverses the process transferring sound from the frequency domain

1. Freitas Caetano, 2010

to the time domains, thus ending with the conventional wave signal form.

Analysis-synthesis tools are initially created to aid composers or artists in general in contemporary composition. Some typical historic examples are : the use of computer generated series in the works of Stockhousen ; computer assisted spectral analysis for the need of spectral composers, for example Gerald Grisey. However the results or techniques are not necessarily bound to artistic use but are also applied in other fields such as communications.

In this dissertation we emphasize in the field of spectral analysis → transformation → synthesis. In particular we will investigate the functionality of Fourier analysis in discrete time with variating envelope windows for the construction of a double phase vocoder for sound morphing. We define sound morphing as the process of combining the spectra of two sounds in a certain percentage that results to an hybrid sound containing characteristic elements of the original ones. The term morphing is also known as *Cross Synthesis*. The exact mechanism of each term, such as Fourier transform, phase vocoder, etc, will be analyzed in section 2.

1.1.1 Implementation Tools

Some comments on the tools used in this research are necessary before discussing structure details. As said, an interdisciplinary research like this one demands a collaboration of fields and terms coming from mathematics, computer science and music. Thus the cadre needs to be adapted to this dissertation's needs.

For the text in place of a traditional *Word* editor, a LaTeX text editor is used. Thus this dissertation is written entirely in LaTeX code. The need for LaTeX comes from the quantity of mathematical formuli presented, as well as the code attached in some sections. Also LaTeX source code is easier to share and readable from any text editor. The use of LaTeX though renders some what different for the French standards the formulation of references and bibliography.

The programming part of this dissertation is almost entirely coded in MaxMSP and Jitter, with some small code parts in Javascript. Max is a robust tool by Cycling74 allowing real-time analysis and signal processing as well some visual processing. Max Version 7.3.2 x64 for Windows

is used with no external libraries. Audio settings are set to SR of 48000, vector size of 442 and signal vector size of 64. More information on MaxMSP can be found on "https://cycling74.com".

Last but not least, Reaper was used for sound editing and exporting the final .wav format. Reaper is Digital Audio Workstation (DAW) with multichannel controls and recording options. More information information on Reaper can be found on "www.reaper.fm".

The archiving of this research is available publicly on "Github.com" where it can be viewed for educational or research purposes. The link for the Github repository can be found on https://github.com/melkisedeath/Un-Guide-sur-l-analyse-Spectrale. In this repository one can find the code, the sound files the text and some artistic examples.

1.2 Structure

The structure is organized in three main sections. The first section presents all the technical information needed to understand the functionality of a Phase Vocoder. In the second section, we build step by step Phase Vocoder and specialize into Spectral Morphing. In the last section we propose various artistic implementations of the research's results.

More in detail, the first part concerns the basis of spectral processing, Fourier Analysis. One can revise all the mathematic context hidden behind these complicated terms and hopefully clear out this misted field. Moving up from Fourier Transformation one must understand how digitizing sound works. Therefore, windowed Fourier analysis will be thoroughly discussed. These information are crucial for the comprehension of the Phase Vocoder. The math of this concept are slightly more complicated and through a step by step ramification of the math formula one can get to conceive how it really works. Getting into such detail may not be necessary, but the knowledge is offered for other musicians with passion to mathematics towards a grasp of the nature of sound.

After all the mathematical stuff, the work of the programmer begins. MaxMSP is the key tool for real-time sound processing. Presented in Max a step by step guide to building a Phase

Vocoder and breaking down the math of a Fourier analysis we are going to penetrate the field of Spectral Morphing. The magic of seeing how a math formula comes to life is the very essence of programming and it's even more evident when working with sound. One can hear the result over and over while doing modifications leading to a deeper understanding of its functionality. Spectral morphing in this study is consisted by two simultaneous Phase Vocoders working on different inputs.

It is reasonable at this point to ask the obvious question : is it necessary for someone to be a musician to do all that. The answer is pretty straightforward : "No!". But, this is why this is the key section of this dissertation. Combining programming skills, mathematical understanding and musical creativity is what describes many of contemporary composers. And in this chapter one can seek a few model examples of creative implementations of the Phase vocoder.

1.3 MaxMSP et Jitter

MaxMSp is a software, originally created by Michael Puckette in IRCAM and thereafter purchased by the American company, Cycling74. The core of this software is build upon C++ and Javascript, translated into a graphical user environment such as box connecting.

The MaxMSP is essentially a programming language with every command being a codebox that recalls a certain function. The commands are separated into three categories : logical expression commands, sound processing commands and multidimensional matrix processing commands, each one corresponding to Max, MSP and Jitter respectively. The user can connect various commands from all three categories to create a complex patch. The most recent version of MaxMSP and Jitter is Max8, released in September 2018 [2].

In the category of sound processing a special library called spectral processing offers the user all the tools one needs to perform spectral analysis. The method MaxMSP uses is called Fast Fourier Transform (FFT). One can imagine the signal as a continuous curved line consisted

2. cycling74.com, *Max8 release date*, https://cycling74.com

by a series of points. The idea is to convert a signal which is a one dimension shape to a two-dimensional form that contains essentially more information. One can cut a fragment of this signal line and assign a value to each of its points to a vector on the Cartesian plane. From this interpretation it is possible to extract information about frequency and amplitude.

Apart from the efficiency of spectral processing and visual perceptivity of the software. MaxMSP is quite popular in the artistic community, used in plethora of projects and research groups, such as IRCAM. The vast community of Max makes it accessible even to amateur users. Last but not least, a series of technical libraries on Phase vocoders and spectral processing for Max is developed by the community projecting the leading role of this software. Some of these libraries are SuperVP, Max SoundBox, etc.

1.4 Spectral Analysis

Spectral analysis is the process of measuring the spectrum of a certain signal. The spectrum consists of information about phase and magnitude from which one can extract a diagram called frequency-magnitude diagram. In such a diagram one can visualize the frequencies and their respected magnitudes of a signal for a certain time period.

1.4.1 History

The term spectral was first introduced by Isaak Newton in the late 18th century. The Fourier transform theory was formulated in 1822 by Jean Joseph Fourier. Fourier was a physicist who reasoned on the thermodynamic properties of materials. In 1843, Georg Ohm (1789-1854) innovated in the signal processing field applying fourier transform on signals. Thereafter, H. L. F. Helmholtz (1821-1894) stated that instrumental timbre is characterized vastly by the harmonic Fourier series in 1863. In his book *The Sensation of Tone* describes mechanical tools to reproduce artificially the sinusoidal spectra of various sounds [3].

3. Curtis Roads, *The Computer Music Tutorial*, 1996, pp. 545

The first digital spectrum analyzers appear in the 1960s as a consequence to the FFT, discovered in 1965. In Fourier's theory every complex spectrum can be dissolved into a series of single frequencies [4]. Respectively every complex harmonic sound is consisted by a sum of simple sinusoids with variating amplitudes.

The Fourier transformation, meaning the passage from a continuous signal to its static frequency-magnitude graph, is a concept appreciated in most scientific fields. One must bare in mind though that the transformation is static. In sound signal the information is continuous and variating. That is the main issue of the Fourier transform. Therefore one should consider the least longest time period possible to describe a signal. The exact meaning of the previous phrase is going to be thoroughly discussed in the next chapter.

The functionality of Fourier analysis on sound does not rest only on visualization of frequencies but is used for various effects such as pitch shift with non varying frequency, time change with non varying pitch, timbre processing and more. These effects led to the Phase Vocoder, a concept based on Fourier analysis of a windowed signal (FFT). The goal is to arrive to the zenith of the Phase Vocoder, which is sound Morphing.

1.4.2 The Phase Vocoder

A phase vocoder, as the name states, is a type of vocoder which uses phase phase information to process audio signals. The core of the phase vocoder works on Short-Time Fourier Transform (STFT) such as FFT. It the typical transformation of a time domain representation of a signal to the frequency-magnitude domain.

The phase vocoder was discovered in 1966 by Flanagan [5]. However, the standard structure of the modern phase vocoder was established by by Griffin and Lim in 1984 [6]. An example of software implementation of phase vocoder based signal transformation using means similar to those described to achieve high quality signal transformation is Ircam's SuperVP [7]. Modern phase

4. Jean Joseph Baptiste Fourier. *De la Chaleur*. Paris, 1822.
5. J. L. Flanagan et R. M. Golden. *Phase Vocoder*. 1966.
6. Daniel W. Griffin et Jae S. Lim. *Signal Estimation from Modified Short-Time Fourier Transform*. 1984.
7. anasynth.ircam.fr

vocoders, such as SuperVP, use a statistical approach for signal prediction, thus eliminating any produced artifacts by unorthodox sound processing.

1.5 Morphing

Morphing is the process of interpolation between two or more objects, while the hybrid result contains recognizing elements for each of its sources. By this statement one can immediately state two concepts, sound (one dimensional) and visual morphing (multidimensional). In sound the result is achieved by combining the spectra of each source by a certain factor. In images the result is achieved by adding the pixel values of every source by a certain factor. If morphing is assumed over two objects the one is called source object and the other is called target object. Object is a term assigned to the nature of morphing, which is sound or image.

1.5.1 Sound Morphing

Sound morphing combines spectra of sounds that means the two factors, frequency and magnitude. Remember that in the phase vocoder frequency is approached via phase. The process of sound morphing can be described as a spectral interpolation between two (or more) sources resulting into a hybrid morphed sound containing elements of each source. For example, a morphing between a piano and a violin A note at 440Hz would have an percussive attack from the piano and a rich an stable tail from the violin's spectra.

In the field of sound morphing to gain a satisfactory sound manipulation it is proposed to use the concept of the Phase vocoder. The phase vocoder can be used in real-time without quality loss and with a minimum of information loss. Still, the process is quite calculative expensive as it requires a phase vocoder per source.

With a phase vocoder approach to sound morphing a minimum of two phase vocoders is required. Thusly, one can manipulate phase and frequency of the sources to match them better. In this paper we follow the guidelines of the SuperVP vocoder but in much simpler version.

Our approach idealizes the object $SuperVP.morph \sim$ and tries to get a richer parametrization through the process.

1.5.2 Visual Morphing

On the other hand visual morphing has a variety of approaches. First, one should separate 2-dimensional from 3-dimensional morphing techniques, where 2 dimensional is morphing between images and 3-dimensional is interpolation between shapes. Of course morphing can be considered for higher dimensional shapes but the result is difficult to apprehend.

Starting from 3-dimensional morphing is defined as the interpolation of the vertices' position. To visualize the effect, one should imagine two 3D shapes, for example a sphere and a cube. The shape is consisted either from a sum of points either from a sum of two dimensional simplices. Additionally, there are some other methods that are not that common thus they are omitted. The two shapes should have the same number of points/ simplices. Morphing between shapes would be the change of the position for each point/ simplex of the source shape towards the position for each point/ simplex of the target shape. Of course 2-dimensional morphing can be considered on shapes but is just a perception of orientation on 3D shapes thus the method is exactly the same.

Image morphing is more complex. Of course the simple way of just adding the values of pixels with a percentage factor is a valid method with no interesting results. The intelligent way to go considers Variational Auto-Encoders (VAE). It is a method based on machine learning using the famous Generative adversarial networks (GANs). GANs are deep neural net architectures comprised of two nets, pitting one against the other [8]. This dissertation focuses on sound thus image morphing is not going to be covered.

8. Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozeir, Courvill, Bengio. *Generative Adversarial Networks.* 2004.

Chapitre 2

Context théoritique

2.1 Introduction

In this part we will unwrap some mathematical notions of the theory behind Phase Vocoders. To embrace the theory of spectral dissolution, one has to accept that a complex signal, meaning something more than a single sine wave, can be representated through a series of overlaps of simple sinusoids with different amplitudes and frequencies.

To visualize this method the paradigm of additive synthesis should be used as an inverse construction. A classic method of synthesizing a complex time-varying sound consists of combining a number of elementary waveforms. The waveforms that are superimposed in additive synthesis are often sinusoidal. Under certain conditions, the individual sinusoids fuse together and the result is perceived as a single rich sound.

The idea behind this method is not new. Indeed, additive synthesis has been used for centuries in traditional instruments such as organ.

When an almost-periodic sound is analyzed, its spectral energy is concentrated around a few discrete frequencies (Harmoniques). These frequency lines correspond to different sunisoidal signals called partiels. The amplitude of each partial is not constant and and its time-variation is critical for timbral characterisation. Indeed, in the initial transistory phase(attack) of a note.

The frequency of each component, however, can be thought of as slowly varying. Additive synthesis consists of the sum of sinusoidal oscillators whose amplitude and frequency are time varying. The control parameters are determined through spectral analysis.

2.2 Mathematical Decoding

2.2.1 The Fourier Transform

Spectral analysis is essentially the process of passing from the temporal domain to the frequency domain for a fixed time period. From this point of view, spectral analysis, is a fourrier transformation together with some mathematical processing that allow the visualization of the frequencies that compose a complex sound wave for a fixed time period.

There are many way to compute the Discrete Fourier Transform (DFT)one of which is the FFT. Allthough all the DFT methods convert to same result, the FFT is one of the most efficient computationally. FFT is one of the most used algorithms in signal processing due to the minimal amount of code necessary to program it. Nevertheless, it's among the most difficult algorithms in the DSP domain.

FFT is a complex algorithm, in the sence that it uses complex numbers to compute. Due to its complexity the technical details are frequently omited and left to a more mathematical crowd.

The process of DFT is graphically explained in figure 2.1. In the left column, one can see the temporal representation of the signal. Accordingly, in the right column the FFT transformation can be visualized in the frequency domain. N is the window size in DSP terms or the total duration of the analysis in general. As it is shown in the figure 2.1, the signal is decomposed in 2-axe variables, a real and an imaginary.

In 1822 Jean Joseph Fourier showed that some semi-periodic functions can be formulated as a

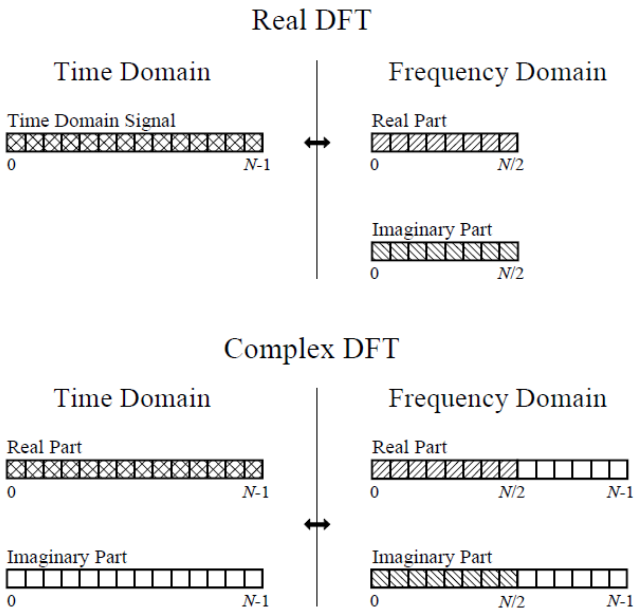


FIGURE 2.1 – Complex DFT

sum of harmonics likewise :

$$^{1}x(t) = c_0 + \sum_{n=1}^{\infty} c_n cos(\omega t + \theta_n) \tag{2.1}$$

Where t is the time period, x is the signal, c_0 is the fundamental harmonic, f the frequency, $\omega = 2\pi f$ and θ is the phase of every harmonic. In this formulation of the Fourier transform is clear that a sound x bounded of the time factor t can be described as a sum of sinusoids.

Thus, the Fourier transform for any intergatable function $f : \mathbb{R} \to \mathbb{C}$ is the written as follows :

$$^{2}F[x(t)](\omega) = \int_{-\infty}^{+\infty} e^{-j\omega t} x(t)\, dt \tag{2.2}$$

Where $F[x(t)](\omega)$ is the Fourier transformation of the signal x bounded by a frequency ω. In sound the this frequency is varied between 0 and the Sample Rate (SR). The SR is usually 44100 or 48000Hz. The Kernel of the intergatable function for digital sound signals varies into the integral $[-1, 1]$. As it is clear to see in this formula the time functor is infinite, but

1. Curtis Roads. The Computer Music Tutorial. MIT Press, Cambridge, MA, USA, 1996. ISBN 0262680823.[p. 1085]
2. Hermann L F. Helmholtz. The Sensations of Tone. New York, 1895.[p. 215]

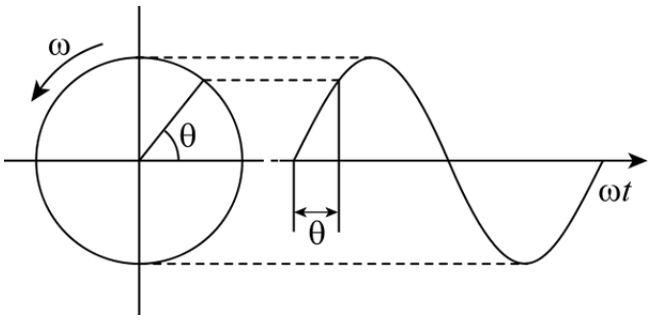


FIGURE 2.2 – Circle de la transformation Fourier

computationaly this is obviously imposible. Other variations of the Fourier transform in discrete time are actually used in computer science.

The Fourier transform is based primarily on Eulers formula where $e^{2j\pi t} = cos(2\pi t) + jsin(2\pi t)$. One can imagine this operation as drawing of a circle in the complex plane $\mathbb{C}$. During the fourier transformation a signal $x(t)$ is rendered around a circle with $(e^{-j\omega t})$, with a frequency f. Remember that $\omega = 2\pi f$. The rotation is given by the sign of j. A clockwise rotation is considered thus $-j$. This process gives us effectively two parts the real, $cos(2\pi t)$, and the imaginary, $j\ sin(2\pi t)$, part of the equation.

One would often transform the cartesian data induced by the complex exponential to a more signal familiar pollar form. After the Fourier transform, there are two factor to manipulate, phase and magnitude. Phase is induced by the angle of the two cartesian coordinates in the complex plane while magnituded is deducted by the produced vector of the two values. From the phase one can extract information about the frequency of the signal in the precise bin and, as the name implies, from magnitude information about the amplitude of the exact bin. When the time is fixed for the perform of Fourier analysis, then the sound signal is divised into frequency bins factorized by the frequency of the analysis (ω). The phase and the magnitude are calculated for each fraction of time on which Fourier analysis is performed. The magnitude is equal to : $m(x) = \sqrt{i(x)^2 + r(x)^2}$ and the phase is $\theta(x) = tan^{-1}(\frac{i(x)}{r(x)})$. 2.3

Evidently in the digital domain one cannot use an infinite time portion. Thus we introduce the notion of the window and analogically windowing. The window is a period of time expressed in frames. Frames is an equivalent notion to SR in the sense that the SR calculated an amount of

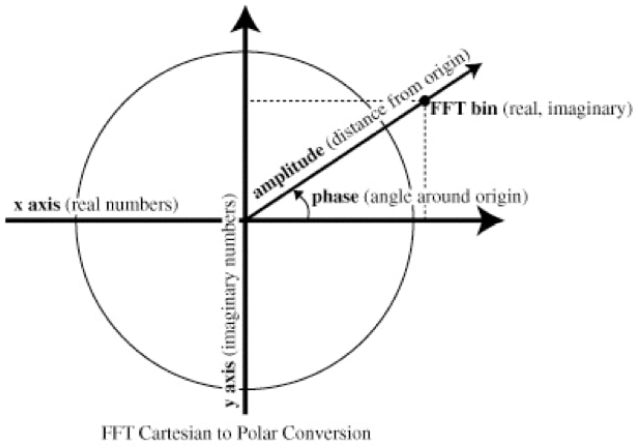


FIGURE 2.3 – Magnitude et phase

images of the sound per minute and the frames are exactly these images. A windowed analysis is usually expressed by a Short Time Fourier Transform (STFT) algorithm [3].

$$^{4}X(\omega,\tau) = \sum_{t}^{\infty} x(t)\omega(t-\tau)e^{-j\omega t} \tag{2.3}$$

Where x represents the signal, X it's Fourier transform (an abreviation of $F[x(t)]$), $\omega = 2\pi f$, t is the continuous time, τ is the instant time, c_n the harmonics. $\omega(t)$ the windowing and j a the complex constant value.

Où x est le signal, X sa transformée Fourier (une abréviation de la forme $F[x(t)]$), $\omega = 2\pi f$, t le temps continue, τ l'instant temporel, c_n l'harmoniques, $\omega(t)$ le fenêtrage, et j un nombre complexe.

Dans cette recherche, on va utiliser la forme continue TFD et puis FFT, vu que cette formule est premièrement utilisée dans MaxMSP et en plus requiert une puissance calculatrice plus efficiente que d'autres méthodes :

As mentioned above, MaxMSP uses a FFT algorithm. It is among the most efficient algorithms

3. STFT : Short Time Fourier Transform, Jown Strawn. Introduction to digital sound synthesis. Digital Audio Engineering, pages 141– 134, 1985.

4. Tadej Droljc. STFT Analysis Driven Sonographic Sound Processing in Real-Time using Max/MSP and Jitter. 2011.

for Fourier analysis.

$$^{5}X(\omega) = \frac{1}{N}\sum_{n=0}^{N-1} x(n+1)e^{-j\frac{\omega n}{N}} \tag{2.4}$$

Where N is the size of the window, which is the total amount of frames it contains.x is the signal, n enumerates every frame in N and $\omega = 2\pi f$ as usual but in this version f is varied between 0 and N. We will rather present it as $\omega = 2\pi k$ where k will represent the $k-$harmonic.

For every transition to the frequency domain, a inverse function to the temporal domain is necessary. The inverse function is caracterized as Inverse Fast Fourier Transform (IFFT) for discrete time values.

$$^{6}x(n) = \frac{1}{N}\sum_{f=0}^{N-1} X(f)e^{j\frac{2\pi f n}{N}} \tag{2.5}$$

Fast Fourier Transform

It's always fine to have a theorical basepoint but what really matters is computational formulas. By computational we mean I formula that is easy to be executed by a computer and produce an output. A solution to this statement is Fast Fourier Transform. Thus, FFT is a computable algorithm for DFT that uses a smart way to reduce the time of the computation and the complexity of the calulations.

We remind the standard DFT intepreted in computable code.

```
function DFT(x)
    N = length(x)
    # We want two vectors here for real space (n) and frequency space (k)
    n = 0:N-1
    k = n'
    transform_matrix = exp.(-2im*pi*n*k/N)
```

5. Jean-François Charles. A tutorial on spectral sound processing using maxmsp and jitter. Computer Music Journal, pages 87–102, Printemps 2008.
6. Alan V. Oppenheim et Ronald W. Schafer. Discrete-time signal processingd. Prentice Hall Press.

```
7         return transform_matrix*x
8     end.
9
```

Listing 2.1 – DFT

It's clear to see that the Fourier transform is actually a computation matrix. However, but performing that much computation one can imagine that the process is quite slow. The requirements for DFT are real-time processing and windows of minimum size 512 samples for sound data. The improvement of the algorithm was given by James Cooley and John Tukey [7]

The Cooley-Tukey algorithm uses recursion to reduce the computational complexity. In particular, the matrix produced by the reduction to frequency domain is reduced into two parts before performing the DFT calculations. We seperate the odd indices of the bins from the even and then the procedure is repeated. This process reduces the complexity to $\mathcal{O}(n\ log\ n)$ from a polynomial size. Ofcourse to perform this action due to continuous division by two we require the analysis window to be a power of two.

The core of the FFT calculation reduction are butterfly diagrams. In particular, Cooley and Tukey noticed that complex exponential is entirely repeated in the second half of the window with an opposite sign. Therefore, by constantly dividing the video the calculations of the complex exponential are vastly reducted. We can visualize the process in the figure 2.4 [8].

A code implementation of this method can be find in the Appendix A.1.

2.2.2 Windowing

To calculate the STFT, one must define the size of the window to be processed. The fourier transform in a discrete time window may produce artificacts contradicting the continum of the signal. To prevent this it is customary to multiply the window of the original sound with a default window of the same size on which a default function factorizes the amplitude of the

7. James Cooley et John Tukey, *An algorithm for the machine calculation of complex Fourier series*, 1965.
8. Image recuperé de l'article : P. G. Reshma, P. Gopi Varun, Babu V. Suresh, Wahid Khabou, *Analog CMOS implementation of FFt using cascode current mirror*, 2017.

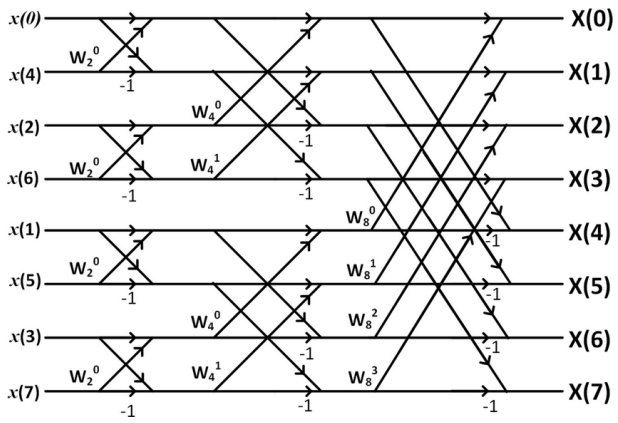


FIGURE 2.4 – Butterfly Diagrams for an 8 point FFT

sound. The Fourier transform is reproduced countably many times in relation of the size of the sound analyzed. The window is usually significantly smaller than the original sound itself, thus it repeated multiple times throughout the duration of the sound. It is important that the window is moved temporally but it is also overllaped by a fuctor to itself. The process can be visually described in the figure 2.5.

This technique, named overlapping, renders a round sound result. Due to overlapping and the envellop multiplication the listener cannot perceive any possible artifacts induced by the analysis but one hears a unified result [9].

Comme il a été déjà mentionné, pour calculer une STFT, une fenêtre d'une certaine durée est nécessaire pour être définie. Pour cette fenêtre, une certaine fonction peut être appliquée pour éviter les cliques comme (hanning, hamming, exponentielle, etc). Pendant STFT, il est habituel de chevaucher les fenêtres l'un dans l'autre. Cette procédure est expliquée graphiquement dans la figure 2.5.

Cette technique, nommée overlapping, nous permet de créer un résultat lisse au niveau sonore. Lors de changement de fenêtres l'auteur ne peut pas percevoir que le son était décomposer en morceaux mais il attend un résultat sonore unifié [10].

9. Daniel W. Griffin et Jae S. Lim. Signal estimation from modified short-time fourier transform. IEE Transaction on acoustics, speech, and signal processing, ISSE Vol. 32 :236–243, Avril 1984.
10. Daniel W. Griffin et Jae S. Lim. Signal estimation from modified short-time fourier transform. IEE Transaction on acoustics, speech, and signal processing, ISSE Vol. 32 :236–243, Avril 1984.

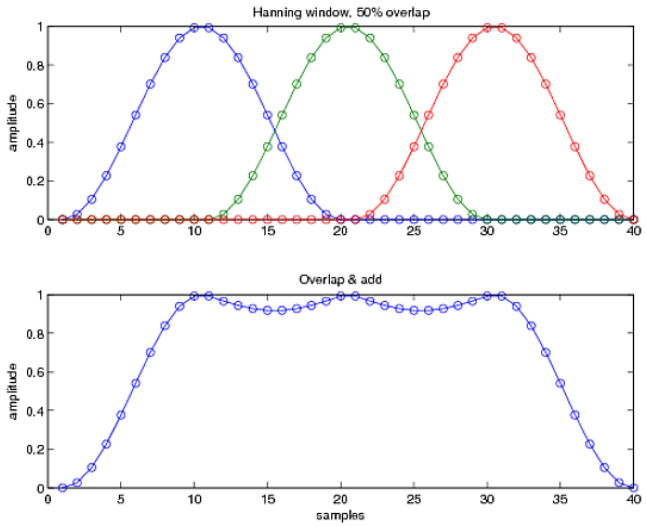


FIGURE 2.5 – Overlapping

Gabor Filters

Here we will examine the possibility of using Gabor filters to envellop the signal. Gabor filters are defined as the continuous multiplication of a Gaussian function by a complex signal bounded by time. The Gaussian filter is described in the following formula :

$$G_x(t,f) = \int_{-\infty}^{+\infty} e^{-\pi(\tau-t)^2} e^{-j\omega\tau} x(\tau)\, d\tau \tag{2.6}$$

It is possible to consider a gabor filter as two out of phase filters that occur by unvelloping th ecomplex exponential to a imaginary and a real part. Where the real part has the filter $g_{real}(t) = w(t)sin(2\pi f_k t + \theta)$ and the imaginary part $g_{ima}(t) = w(t)cos(2\pi f_0 t + \theta)$.

The two filters may be out of phase but the take phase into consideration. As a result their effect to a sinusoid is still a sinusoid.

To manipulate the curve of a Gabor filter, one is ought to change the parameters. The normalized formula transforms to :

$$G_x(t,f) = \int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\tau-t}{\sigma})^2} e^{-j\omega\tau} x(\tau)\, d\tau \tag{2.7}$$

Where σ is the parameter of the Gaussian curve. A Gabor filter will allow to create a smoother sound result after the analysis.

The same logic is valid for every type of envellop window such as hanning, hamming, etc.

2.2.3 Le vocodeur de phase

Définition

The Phase Vocoder is an analysis-synthesis tool that uses DFT. The analysis is done so that the output signal has no data loss from the transform both theoritially and practically. The output signal is identical to the input prior to the analysis if no processing is applied. The most standard uses of the phase vocoder are pitch shifting and playback speed alternation. The Phase Vocoder has no obvious restriction and ca also keep in track of the inharmonicities and the vibrato of sounds [11].

Histoire

The Phase Vocoder was introduced, in 1966, by Flanagan as an algorithm that preserves the horizontal coherence between the phases of the segments that represent the sinusoidal components. This phase vocoder didn't take into account the vertical coherence produced by the intervalles of adjacent frequencies, and consequently, the temporal stretching of the system produced sound signals with quality loss. The optimal reconstruction of the signal from the STFT after processing was proposed by Griffin and Lim [12] in 1984. This algorithm didn't consider to produce a coherent STFT, but rather find the optimal signal whose STFT is as close as possible to the modified STFT, even if the modified STFT is not coherent (does not represent any signal).

11. Johannes Grünwald. *Theory, implementation and evaluation of the digital phase vocoder in the context of audio effects*, 2010.

12. Daniel W. Griffin et Jae S. Lim. Signal estimation from modified short-time fourier transform. IEE Transaction on acoustics, speech, and signal processing, ISSE Vol. 32 :236–243, Avril 1984.

The problem of the vertical coherence was a major issue for the operational quality on temporal scale until 1999. At this time Laroche and Dolson [13] proposed a way to preserve the phse coherence of the spectral components. The develpoment of Laroche and Dolson has to be a turning point in the history of the Phase Vocoder. It has been proven that, as a result of the phase coherence,a high quality temporal transformation can be obtained.

Still, the algorithm discovered by Laroche wasn't able to preserve vertical phase on attack . A solution to this problem was proposed by Roebel [14] in 1999. A implemented example of the phase vocoder using the last version by Roebel is the Ircam's SuperVP tools.

Une approche à la description du vocodeur de phase consiste à représenter le signal via la succesion des images successives de une transformée de Fourier Discrète (TFD) de une fenêtre de longueur N. Ces images sont d'abord multipliées par une fenêtrage appropriée (telle que Hamming, Hanning, Kaiser, Blackman, etc.) puis transformé en Fourier dans le domaine fréquentiel. A ce stade, toute modification prudente du spectre peut être faite, avant la transformation inverse au domaine temporel avec la transformée de Fourier discrète inverse (TDFI). Là, les parties d'overlap et éventuellement fenêtrées sont additionnées, ce qui donne le résultat final.

La STFT (une transformation characterisée des succesions des TFD) d'un signal fenêtré est défini comme suit :

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_f(n) {W_N}^{nf}, \qquad \forall n \in SR \tag{2.8}$$

ou $W_N = e^{\frac{2\pi j}{N}}$ et $h(n)$ est une fenêtre approximativement choisie.

$$h_f(n) = \frac{1}{N} h(n) {W_N}^{nf}, \qquad k = 0, 1, \ldots, N-1 \tag{2.9}$$

13. Mark Dolson. *The phase vocoder : a tutorial. Computer Music Journal*, pages 14–27, Printemps 1986.

14. Axel Roebel. Morphing sound attractors. In 3rd. World Multiconference on Systemics, Cy- bernetics and Informatics (SCI'99) and the 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99), 1999, Florida, United States, Proc. of the 3rd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'99), 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99), 1999.

Pendant le processus de l'analyse, la succesion des images d'une STFT font parties du signal d'entrée $x(n)$, sur la position $n_a^u = uR_a$, ou R_a est nommé *input hop size* et u doit etre un entier. La fenêtre(ou la durée) de la DFT est definiée par la taille N, ou le *hop size* est forcement une sous-multiplication de N. Cet effet permettra de réaliser un *overlap* de 50%, 75%, etc.

At the analysis stage, successive DFT frames are taken from the input signal x[n] at the positions $n_a^u = uR_a$, where R_a is termed *input hop size* or analysis hop size and u being integer-valued. Given the length of the DFT as N, the input hop size is most likely (if not necessary at all) a submultiple of it. Common values are N/2, N/4 and N/8, depending on the analysis window and the required performance. These values let the frames overlap by 50 %, 75 % and 87.5 %, respectively.

$$X[n_a^u, \omega] = \sum_{n=0}^{N-1} \tilde{x}_u(n) e^{-j\omega \frac{n}{N}} \tag{2.10}$$

$$\text{ou} \qquad x_u(n) = h_a(n) x(n - n_a^u)$$

Le terme $\tilde{x}_u(n)$ propose l'estimation des fenêtres symmetriques calculées. Donc si nous supposons un overlap de fenetrage par 50 %, ou bien $N/2$, puis une maniere d'eviter les assymetrages de phase est composée ci-dessus :

$$\tilde{x}(n) = x[((n - N/2))_N] \tag{2.11}$$

ou $((.))_N$ presente l'operation du modulo et N est la durée de sequence et il devait un pair.

Donc la transformation de Fourier du signal liée au fenetrage devient :

$$X(rl, f) = \sum_{N=0}^{N-1} x(n) h(rl - n) e^{-j \frac{2\pi}{N} fn} \tag{2.12}$$

It is a function of two discrete variables, the time rl and the frequency k. The index ir is the position of the window, r being the frame number and l the step size if the analysis window.

$S(rl, k)$ can be seen as the spectrum of the multiplication $s(m)w(rl - m)$, which is the input sequence s(m) multiplied by the window shifted at position rl. It's not the exact spectrum, but its convolution with the windows Fourier transform.

C'est, donc, une fonction de deux variables discrètes, le temps rl et la fréquence f. L'index rl est la position de la fenêtre, r étant le numéro d'image (frame) et l la taille du décalage de la fenêtre d'analyse.

$X(rl, f)$ peut être vu comme le spectre de la multiplication $x(n)h(rl - n)$, qui est le signal d'entrée $x(n)$ multiplié par la fenêtre décalée à la position rl. Ce n'est pas le spectre exact, mais sa convolution avec la transformation de Fourier de la fenêtre.

The standard overlap-add procedure consists in adding together the buffers and dividing by the sum of the shifted windows. The output signal $y(m)$ is expressed as :

La procédure standard de *overlap* consiste à additionner les buffers et à les diviser par la somme des fenêtres décalés. Le signal de sortie y(n) est exprimé comme :

$$y(n) = \frac{\sum_r \bar{x}(rl, n)}{\sum_r h(rl - n)}) \tag{2.13}$$

Pour construire la reproduction du signal d'origine ou du signal transmuté après traitement, un iFFT est nécessaire. Dans le vocodeur de phase, cette procédure accède aux informations de phase et d'amplitude et se forme comme suit :

$$\bar{s}(rl, k) = \frac{1}{N} \sum_{k=0}^{N-1} |\bar{S}(rl, k| e^{j(\frac{2\pi}{N} km + \theta(r\bar{l}, k))} \tag{2.14}$$

Enfin pour calculer la frequence de chaque image instantanée il suffit de suivre la formule :

$$\bar{f}_{k,r} = \frac{\theta(rl, k) - \theta((r - l)l, k)}{l} \tag{2.15}$$

2.2.4 Applications du vocodeur de phase

Time Streching

Pour réaliser un étirement du temps d'un son, traditionellement, il suffit de baisser ou augmenter le rythme de sa lecture, cela produit par ailleurs le changement de la hauteur. Le vocodeur de phase permet d'effectuer un étirement temporel sans pour autant changer la hauteur ou la qualité sonore. Afin de mieux comprendre le fonctionnement du vocodeur il faut considérer la transformation Fourier à court terme, pour un étirement temporel. Pour visualiser le résultat on peut imaginer, que pour chaque periode de temps de la transformation Fourier appliquée, une serie des harmoniques sauvegardées dans une fenêtre, et le facteur de son overlap. Il suffit d'éloigner ou rapprocher les fenêtres, pour changer le rythme de la lecture sans affecter la hauteur sonore.

Le modele qui corresponde à l'etirement temporele est donnée par la formule suivante :

$$x(n) = \sum_{k=1}^{K(n)} A_k(n)\, e^{j\theta_k(n)} \tag{2.16}$$

$$\theta_k(n) = \theta_k(0) + \int_0^n f_k(\tau)\, d\tau \tag{2.17}$$

Ou un signal est interpreté par une somme des sinusoides $K(n)$ avec leur magnitude $A_k(n)$ et phase θ_k appropriées. Par la suite, la phase instantanée du $k-$ième sinusoide, $\theta_k(n)$, est calculée par l'addition de la phase de la première echantillion $\theta_k(0)$ avec l'integrale des frequences $f_k(n)$. Le derniere est equivalent à $\Sigma_{n=0}^{N} f_k(n)$ pour une transformation discrete.

La modification temporele est determinée par deux parametres paralleles, la modification de phase instantanée pour chaque echantillion et la modification du fenetre du decallage (hop window). Plus precisement, le *hop window* du stade de l'analyse est different du *hop window* de la synthèse.

Pour une modification temporele par un facteur constant α tel que $n_k^u = \alpha n_\alpha^u$ la phase d'un

etirement temporel devient [15] :

$$\theta_k^(n_k^u) = \theta_k(0) + \alpha \int_0^{n_k^u} f_k(\tau)\, d\tau \tag{2.18}$$

Transposition de l'hauteur

Comme le vocodeur de phase peut être utilisé pour réaliser un étirement temporel d'un son sans affecter sa hauteur, il devrait également être possible de faire l'inverse, c'est-à-dire changer la hauteur sans changer le rythme de la lecture. En effet, cette opération est facilement accomplie. La procédure est de changer le rythme de la lecture par le facteur de changement de la hauteur souhaitée, puis de jouer le résultat sonore produit à la fréquence d'échantillonnage «incorrecte ». Par exemple, pour augmenter la hauteur d'une octave, le son est d'abord agrandi d'un facteur de deux, et il est ensuite joué à deux fois l'original taux d'échantillonnage. [16]

Freeze

À cet effet, nous prenons un certain fenêtre d'analyse à partir du son sélectionné et nous «gèlons» ce son dans le temps. Pour achever cet effet il s'agit de rendre le rhythme de la lecture de notre vocodeur de phase au zero. Cela peut se comparer à un etirement sonore infini. On peut ainsi appliquer les mêmes principes d'un étirement sonore normal.

Robotisation

Pour faire une robotisation d'un signal il faut mettre la phase de chaque echantillon à zero. Cet effet resulte à un son robotisé et métalique.

15. Jean Laroche et Mark Dolson, *Improved Phase Vocoder*, 1999, p. 324
16. Mark Dolson. The phase vocoder : a tutorial. Computer Music Journal, pages 14–27, Printemps 1986.

Harmonisation - chuchotement

Pour réaliser cet effet on doit donner une valeur aléatoire soit à la phase, soit à la magnitude de chaque échantillon de la fenêtre de la FFT [17].

Morphing

La définition générale du morphing consiste à combiner deux (ou plusieurs) éléments distincts en une seule entité qui contient les deux éléments. Le processus de morphing dépend généralement d'une seule variable, appelée un facteur de morphing ou une interpolation. Ce processus dépend, par ailleurs, du facteur temporel puisque le morphing est un phénomène dynamique.

[18]

$$M(\alpha, t) = \alpha(t)\widehat{S}_1 + [1 - \alpha(t)]\widehat{S}_2 \qquad \text{Ou } \alpha(t) \in [0 : 1] \tag{2.19}$$

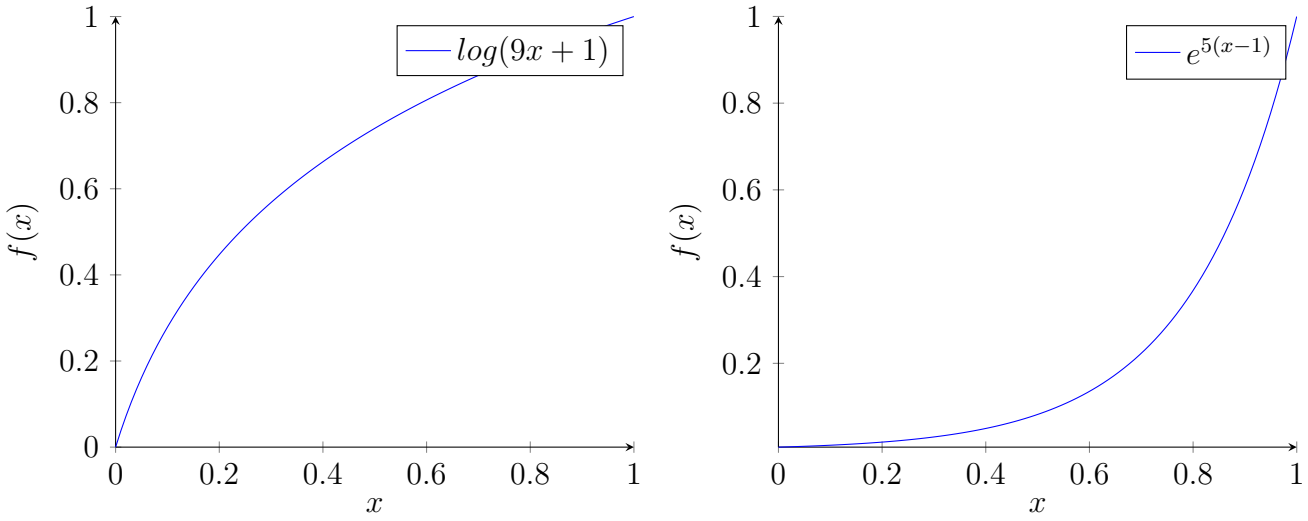
La manipulation du paramètre α nous permettra de changer la dynamique du morphing. La question qui se pose c'est pourquoi se contenter d'une manipulation linaire ? Nous proposons alors d'effectuer une manipulation de α sur une courbe exponentielle ou logarithmique(ex. sur la figure 2.6).

La différence fondamentale entre le morphing d'une image, est le rapport a la variable temporelle. Le son est un phénomène dynamique, donc il ne peut pas être interpolé linéairement. Le terrain du morphing sonore nécessite des fonctions multiples pour atteindre un résultat satisfaisant. Pour obtenir un morphing sonore il faut calculer l'enveloppe spectrale de notre FFT.

17. Johannes Grünwald. Theory, implementation and evaluation of the digital phase vocoder in the context of audio effects, 2010.

18. Axel Roebel. Morphing sound attractors. In 3rd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99), 1999, Florida, United States, Proc. of the 3rd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'99), 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99), 1999.

FIGURE 2.6 – Interpolation du facteur α



Noise modeling

The Phase Vocoder as presented until now is a fine model but nevertheless is not the complete model. The model here is proposed by Xavier Serra [19]. In this model the sound components are idealized as deterministic. This suggests that every sound component is a sinusoid or a slowly varying sinusoid. The non-deterministic part implies that sound is modelled with an additional noise component as is shown in the formula below.

$$^{20}s(t) = \sum_{n=0}^{N} A_n(t)cos(\theta_n(t)) + \epsilon(t) \tag{2.20}$$

Where A_n and θ_n are the magnitude and phase respectively of the n-th frequency bin. Under the assumption that $\epsilon(t)$ is a stochastic component it can be considered as a filtered white noise.

$$\epsilon(t) = \int_0^t h(t,\tau)u(\tau)\ d\tau \tag{2.21}$$

Where $u(\tau)$ is white noise and $h(t,\tau)$ us the response of a time-varying filter that pops a value at time t.

19. Curtis Roads et al., *Musical Signal Processing*, 1997, pp. 91 - 122
20. Curtis Roads, *Musical Signal Processing*, 1997, p. 94

Stochastic Modeling

Some other approaches suggest a simultaneous stochastic approach to the traditional Phase Vocoder. This principle is also based on the assumption that sound is composed by semi-sinusoidal components. In a way that a periodicity is expected to be preserved. Based on the first analysis we suppose a periodicity of signals and during every new analysis the frequency of it's bin is computed based on an Error factor.

$$Err_{p\to q} = \sum_{n=1}^{N} E_\omega(\Delta f_n, f_n a_n, A_{max})$$

Where Δf_n is the difference between a predicted and its closest measured peak. f_n is the frequency of the bin and a_n is the magnitude of the predicted peaks. A_{max} is the maximum recorded magnitude.

$$Err_{q\to p} = \sum_{k=1}^{K} E_\omega(\Delta f_k, f_k a_k, A_{max})$$

Now Δf_k is the difference between a mesured and its closest predicted peak. f_n is the frequency of the bin and a_n is the magnitude of the recorded peaks.

The total error is :

$$^{21}Err_{total} = Err_{p\to q}/N + Err_{q\to p}/K \tag{2.22}$$

21. Curtis Roads, *Musical Signal Processing*, 1997, p. 103

Chapitre 3

Design

In this chapter we are going to implement the formuli presented in the previous chapter in the MaxMSP software. Additionally, we are going to structure some other effects and variation on the context of the Phase Vocoder.

We will present the MaxMSP function that encapsulate mathematical functions like FFT, IFFT and other transformations. One can expect to fully understand the majority of the spectral tools that MaxMSP offers to the user. At the same time we will construct an efficient series of tools to build a simple Phase Vocoder.

Dans ce chapitre nous allons réaliser les théories discutées, dans le chapitre précédent (No. 2). La réalisation des notions théoriques vont être implémentées dans le logiciel MaxMSP. L'implémentation de la théorie nous permettra de la comprendre profondément et aussi de la découvrir

3.1 MaxMSP Objects

In this section we will mention some of the basic MaxMSP spectral objects that are delivered directly by the programm and we will analyze their functionality.

In MaxMSP exist 3 categories of objects : 1) the logical objects that logical expressions and

calculi, 2) the signal objects, for signal processing, that are usually followed be a $\sim$ indication after their name, 3) and the Jitter objects that are used for multidimensional data.

The way the spectral object work in MaxMSP is somewhat different from the rest signal objects. They are deployed in a special environment called $PFFT\sim$. In this environment as the name suggest an FFT is performed and because the objects deal with 2-dimensional data they are treated in a different patch [1].

$FFT\sim$

The $FFT\sim$ object belong to the signal family as the $\sim$ indicator suggests. It's the object that performs Fast Fourier Transform. It has no inputs and three outputs one for the real part of the exponential, one for the imaginary part and one counter that keep in track of the frequency bins.

In the usual windowed Fourier transformation :

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi kn}{N}} \tag{3.1}$$

Where k is the index of the bin and the corresponding complex coordinates are $\sum_{n=0}^{k} x_n + cos(2\pi t)$ for the real part and $\sum_{n=0}^{k} x_n + jsin(2\pi t)$ for the imaginary. Of course the corresponding reduction of calculative ramification is applied by the FFT algorithm discussed in previous chapter.

$IFFT\sim$

Similarly the inverse FFT is treated by the appropriate object function with the inverse inputs and outputs.

1. In MaxMSP a code file is called a patch among users

cartopol ~

Usually is not that helpful to use this Cartesian coordinates, produced bu the FFT ~ corresponding to the real and the imaginary part in the $\mathbb{C}^2$plane. Therefore an object that transforms Cartesian coordinates to a polar form called *cartopol* ~ is frequently deployed. This object takes of course two inputs(real and imaginary) and outputs a phase and a magnitude.

poltocar ~

Exactly the inverse function from *cartopol* ~.

$FFTinfo$ ~

The $FFTinfo$ ~ is very frequently used and it reports information about the FFT parameters such as the size of the window, etc.

$framedelta$ ~

A very important object as it computes the phase derivation between successive FFT frames.

gen ~

An object that creates a new patch window. In the *gen* ~ object the user can use a new set of functions that specialize single sample treatment. The functions are relatively simpler than the usual coding environment but one can do a more precise work on details. Is frequently used to perform one sample delay creating a low-pass filter or similar effects

3.2 Pitch tracking

Using some of these MaxMSP spectral objects we are going to start by building a simple pitch tracker :

> " Frequency domain pitch detection methods dissect the input signal into frequencies that make up the overall spectrum. The spectrum shows the strength of the various frequency components contained in the signal. The goal is to isolate the dominant frequency, or "pitch" out of the spectrum ". [2]

Likewise we implemented a MaxMSP patch to find the dominant frequency of the FFT window. The patch can be find in figure 3.1.

As expected, we use the object $fftin\sim$ to transform in to the frequency domain. In the FFT we have to declare two things. First the number of frames objected to FFT and second the size of the window. We remind that the first two outputs yield the real and the imaginary components respectively. The third output yields the indexed frequency bin taking corresponding values from 0 to N, where N is the size of the FFT window.

To continue the simple pitch tracking patch, we normalize the real and the imaginary components to restrain the data flow within computable limits. In more detail, $fft.normalize\sim$ is a simple Max external, created in C++ code using the Max SDK, that divides the number of each output by half the size of the window.

Thereafter, we transform cartesian coordinates to polar using *cartopol* $\sim$. The process is explained in section 2 under the terms phase and magnitude. We are only interested on the phase of each bin. Using phase we can calculate the exact frequency of each harmonic. We remind that the terms harmonic, bin and Fourier frequency are the same in this context. First we delay the phase of every bin by an entire window to compute the change amount. Then we substract the current phase of every bin by the corresponding phase of the previous window. Hereafter,

2. Curtis Roads, *The Computer Music Tutorial, 1996, p. 513, op. cit*,

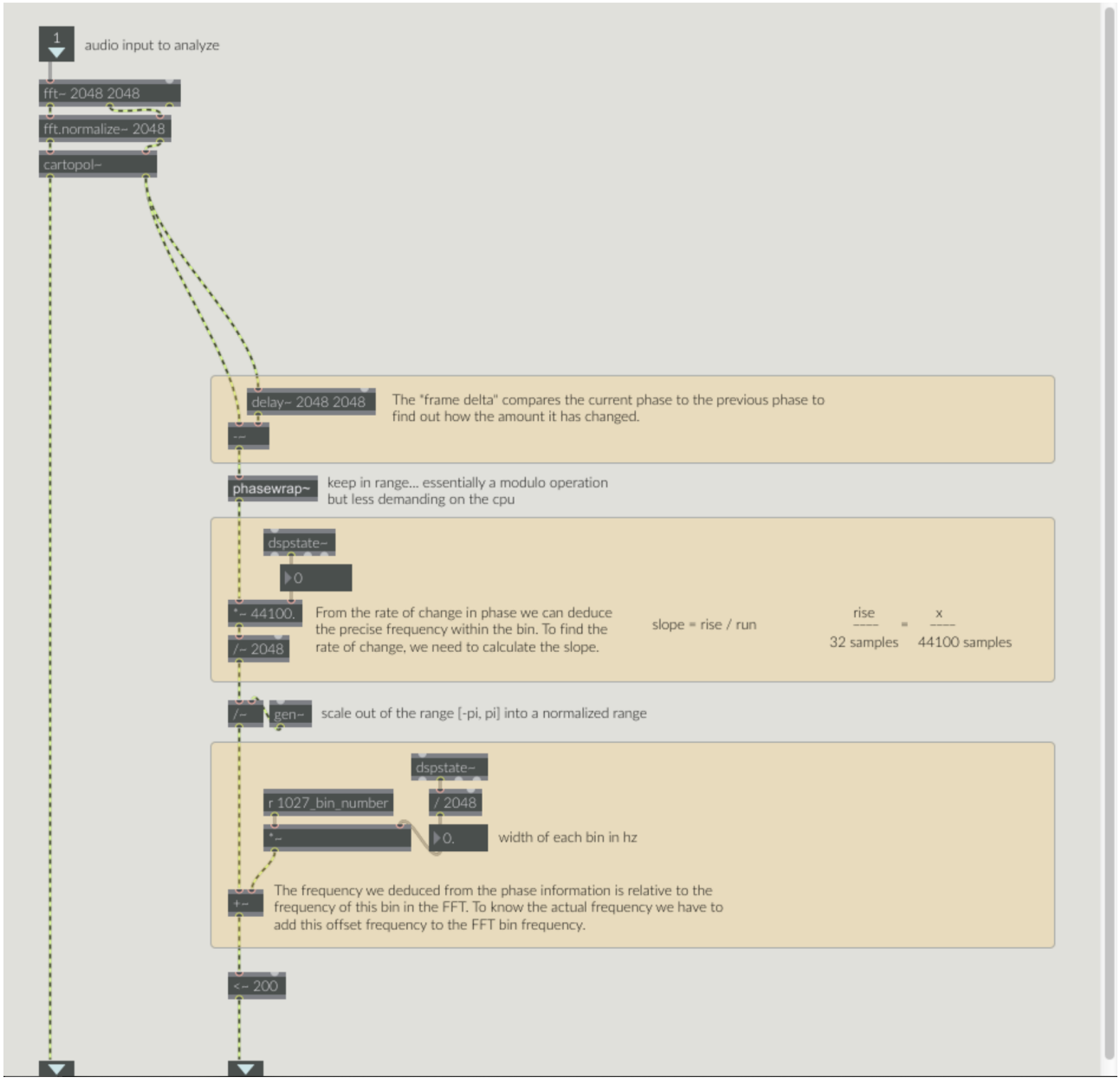


FIGURE 3.1 – Pitch Tracking

we use the object *phasewrap* ~ to clip the value range between $-\pi$ and π. It's essentially a modulo function to keep the values within a computable range.

At this point we have to advise our sound settings. An object in Max called *dspstate* ~ retrieves all the necessary data we can find in $/Options/Audio_Status$ within the Max Application. We only need the sample rate frequency. This value is usually $44100Hz$ thus we already have stored the value and if different we change it automatically. The signal obtained from *phasewrap* ~ is multiplied with the sampling frequency and consequently devided by the size of the window. These operations, dividing the SR by the size of the window, deduct weighted partitions of the spectrum. The value of the frequency subtraction ranks the deviation of the frequency partition. After, we normalize by multiplying with 2π stored in a *gen* ~ object. We find the position of the partition by multiplying the value of the spectrum division with the index of the bin and adding it to the deviation yields the frequency of the bin.

The spectrum devision or the partition is given by :

$$x = \frac{SR}{\text{window size}}$$

Then, the frequency is simply : $index * x+$ the phase deviation.

This procedure generates all the frequencies of all the bins of the FFT window.

The next step is to keep the frequency with the highest magnitude. We are going to use a *gen* ~ object. First we declare the inputs. We only need three variables. The magnitude of each bin, the index of each bin and the size of the FFT window. We declare one single output for the strongest bin of each window. The patch is shown in figure **??**.

To compute the strongest bin, we are going to use an object called *codebox*. *Codebox* is a object that allows the user to input code in the "traditional" form. Just like javascript code, we have to declare the variables we are going to use but first the inputs of the object. The variables in this case are : magnitude, index, frameSize and last. Where frameSize is the FFT window size and last is just a variable that store the index of the strongest bin for the previous window.

The process of defining the strongest bin is implementated by two nested if functions. For every window this process is going to be repeated and it will determine eventually the frequencies of the strongest bins across all windows. In the nested if loop we determine if the current index is stronger than the previous one and we return the result. This means if the magnitude of the $n-$th index is a bigger value than the magnitude of the $(n-1)-$th bin, we store its index value. The last step is to restrict the number of the indexes within the limit of the window size, in order to terminate the loop and re-initialize the variables.

To track the stronger pitch a few more functions are required. The output of the *gen* $\sim$ object is filtered by a *sah* $\sim$ object. Before that, the current index is compared with the index value outputted by the *gen* $\sim$ object. *Sah* $\sim$ stands for "sample and hold". This function filters the first input of the object by a factor similarly to the *gate* $\sim$ object. Every time the factor value changes it allows the first input to go thought and outputs its value. We use it twice, once for the magnitude and once for the phase. This way only the index of the bin with the strongest magnitude goes thought along with the corresponding magnitude and phase. We use the system provided earlier to find the exact frequency and we translate the magnitude's polar coordinates to dB. Last but not least we use a *slide* $\sim$ object to smooth the result. The final patch is shown in the figure 3.2. This method avoids computing every time the frequency if the sound is periodic and thus the dominant frequency does not change.

3.2.1 Multiple Pitch Tracking

To build a multiple pitch tracking patch we use the single pitch tracking method with a few modifications. The majority of the computations occurs within the *gen* $\sim$ object as the *codebox* implementation is copied accordingly to the number of pitches one wishes to calculate.

The figure 3.3 allows to visualize how one can compute a series of the strongest harmonics of each window. The goal is to give sufficient amount of information about the basic spectral components of sound.

As the figure 3.3 suggests, the output of each *codebox* is the input of the next one thus computing

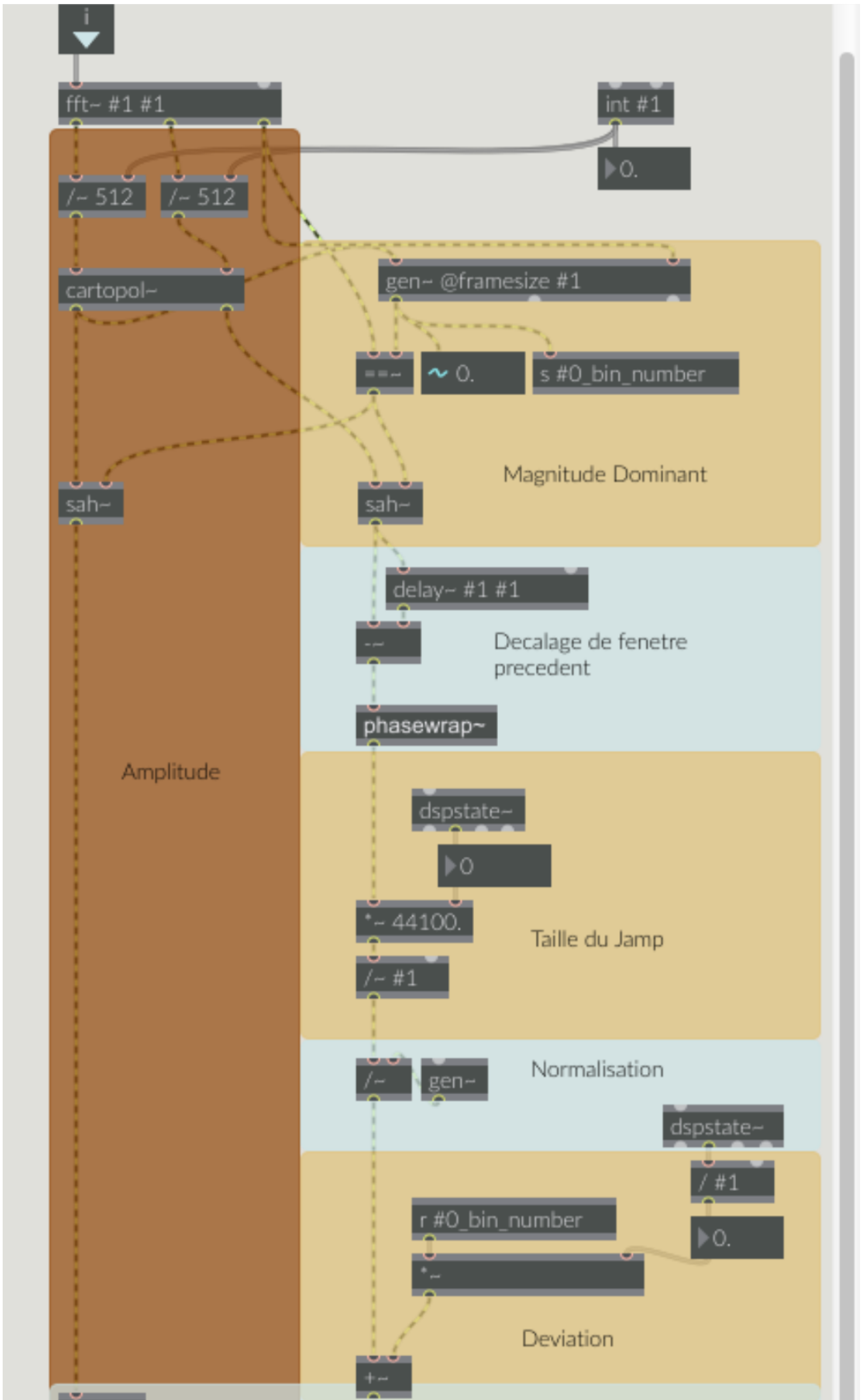

i
fft~ #1 #1
int #1
0.
/~ 512
/~ 512
cartopol~
gen~ @framesize #1
==~
0.
s #0_bin_number
Magnitude Dominant
sah~
sah~
delay~ #1 #1
-~
Decalage de fenetre precedent
phasewrap~
Amplitude
dspstate~
0
*~ 44100.
Taille du Jamp
/~ #1
/~
gen~
Normalisation
dspstate~
/ #1
r #0_bin_number
0.
*~
Deviation
+~

FIGURE 3.3 – Gen multiple

the next loudest index of the window. The quantity of *codebox* objects used relates to the number of pitches that we are going to output.

An implementation of the multiple pitch tracking method can be shown in figure 3.4. Essentially we reproduce the frequency computation for each index. The result can be outputted either as a list or in different outlets. Then one can use the *mc.cycle* ~ object, simple oscillators or resonators.

An important notice for this patch construction but also for every pitch tracking via spectral methods is that the size of the window affects the frequency tracking. The window size influences the temporal or frequency resolution of the representation of the signal. The frequency resolution can be increased changing the FFT size, that is, the number of bins of the analysis window. In particular, the size of the bins is simply the half of the analysis window. The frequency range is defined as the division of the SR to the window size [3].

$$Frequency\ Range = \frac{SR}{Window\ Size}$$

3. Coralie Diatkine, *AudioSculpt 3.0 User Manual*, 2011

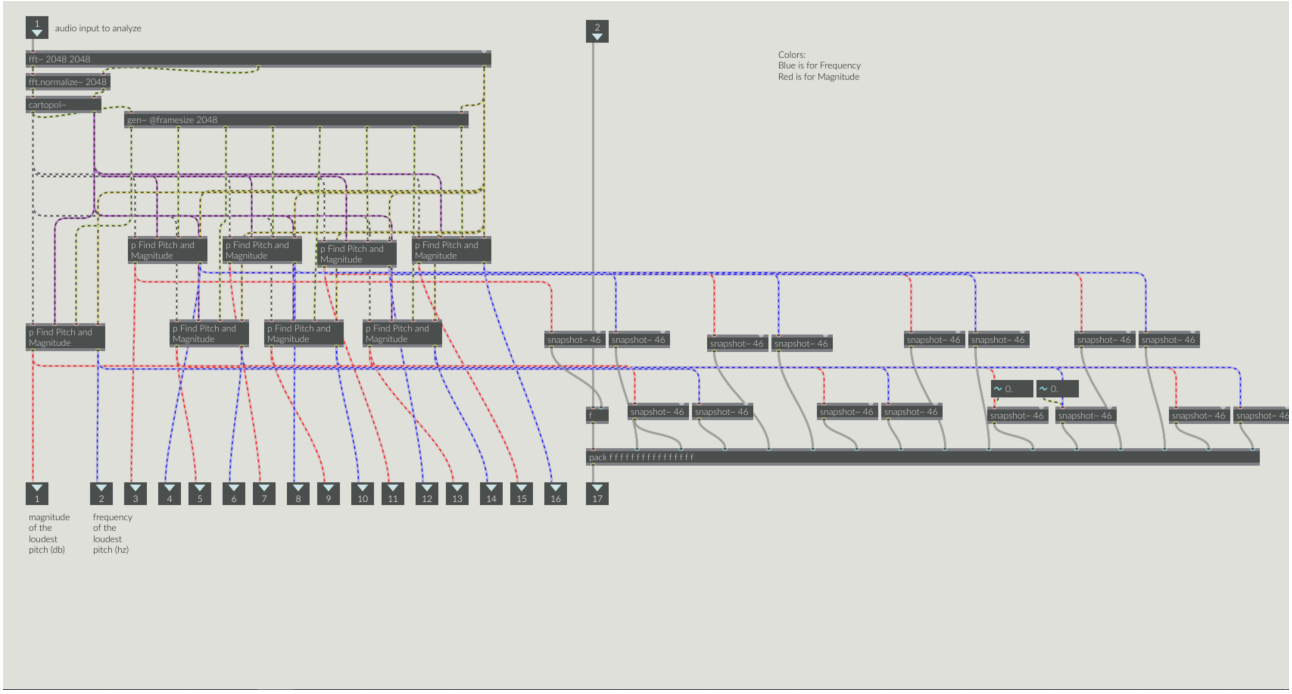


FIGURE 3.4 – Multiple Pitch tracking

3.3 Le vocodeur de phase

The phase vocoder was used originally for pitch transposition and changing playback speed. Likewise, we are going to build our phase vocoder on the basic model and then we are going to propose some modifications. The phase vocoder is labeled under spectral processing thus is going to be stored in a $pfft\sim$ object.

In order to access a sound directly into the pfft buffer one must avoid using the presets of the $pfft\sim$. Therefore we are going to avoid using the object $fftin\sim$ instead we are going to an object $fft\sim$ in a subpatch. The only differences is that the parameters of the $fftin\sim$ and $fftout\sim$ objects are controlled directly from the $pfft\sim$ object and that the in and out are the first and last objects. To build a phase vocoder we need to process sound before applying the Fourier transform. One, should follow the patch provided in the figure 3.5 to understand the procedure.

Le raisonnement derrière telle logique repose sur le coeur du vocodeur de phase pour l'étirement sonore et le changement de l'hauteur. Le vocodeur de phase nécessite une lecture indépendante du son à transformer pour chaque superposition de la FFT. Chaque image sonore de la lecture

doit être synchronisée avec son image sonore respective de la FFT. Par conséquent, une seule copie du son ne peut pas être lancée dans un objet $fftin\sim$, mais plutôt dans un objet $fft\sim$. L'objet $fft\sim$ exécute une FFT à spectre complet (c'est-à-dire en miroir), donc $fft\sim$ peut fonctionner en synchronisation avec les images consecutives de la FFT traitées dans l'objet $pfft\sim$ mais c'est nécessaire de faire quelques modifications sur l'objet $pfft\sim$ pour que il se comporte de la même manière.

Tout d'abord, l'objet $pfft\sim$ doit traiter des images sonore de la FFT à spectre complet, au lieu de l'image spectrale par défaut qui correspond à la moitié de la taille FFT (jusqu'à la moitié de la frequence de Nyquist). Cela se fait facilement en ajoutant un cinquième argument non nul à l'objet $pfft\sim$. Comme l'argument du spectre complet est le cinquième argument, nous devons fournir tous les autres arguments avant lui, y compris le quatrième argument, le début, qui sera défini sur la valeur par défaut de zéro.

Ensuite, puisque les objets $fftin\sim$ et $fftout\sim$ effectuent le calcul de la FFT à la phase zéro par rapport à la FFT (le premier échantillon de la fenêtre envoyée au FFT est le milieu de la fenêtre), et les $fft\sim$ et $ifft\sim$ objets exécutent la FFT déphasée de 180 degrés, il faut assurer que tous les objets $fftin\sim$ et $fftout\sim$ du patch ont le même décalage de phase FFT utilisé dans les objets $fft\sim$.

Cela peut être accompli en spécifiant un déphasage par rapport aux objets $fftin\sim$ et $fftout\sim$. Une valeur de phase de 0.5 signifie un déphasage de 180 degrés, donc c'est la valeur préférable dans ce cas. Bien que, le objet $fftin\sim$ ne soit pas volu, le objet $fftout\sim$ peut pratiquement être utilisé comme sortie pour l'objet $pfft\sim$. Le fenêtrage automatique dans l'objet $fftout\sim$ devrait se comporter comme le fenêtrage manuel avec les objets $fft\sim$.

Dans cette version du vocodeur de phase on va utiliser un son pre-determiné. Ca veut dire que on utilisera un enregistrement et on le stora dans un objet $buffer\sim$. Le $buffer\sim$ doit être accessible à deux endroits. Premierement à l'emplacement de la image sonore de la FFT actuelle et, en deuxième lieu, à l'emplacement de la image FFT précédente du $buffer\sim$. C'est possible d'utiliser soit l'objet $index\sim$ ou l'objet $play\sim$[4] pour accéder au $buffer\sim$. Lorce

4. L'objet $play\sim$ est utilisé comme interface de lecture de l'objet $buffer\sim$. $play\sim$ joue des samples en

que on precise la position de la transformation Fourier en durée courte manuellement, on doit trouver un systeme pour preciser le decalage du fenetre dans notre buffer.

La solution dans cette problematique et de utiliser deux objets $fft\sim$ parallels avec un decalage d'un quart. Un tel systeme permetra de calculer l'image actuelle de chaque index du fenetre en comparaison de l'index correspondant de la fentre precendant comme etait fait dans le patch du pitch tracking. Le decalage est calculé pour un distance d'un quart puisque on prend en compte le parametre d'overlap. L'objet $frameaccum\sim$ dans ce cas remplace la sequence des objets *phasewrap* $\sim$, et normalisation entre π et $-\pi$ que on a construit dans le patch du pitch tracking.

Pour la synchronisation du buffer on utilise un objet *count* $\sim$ au parametres 0, la taille du fenetrage, 1 et 1. Comme ca on peut trouver l'echantillon au quelle la FFT commence à la fois, ainsi que la position actuelle en durée totale du son. Chaque fois que le compter ce re-initialize au zero, un objet *sah* $\sim$ permet au valeur de passer, dont on deduit le premier sample de la FFT. en ajoutant les valuers du compter on peut deduire la position actuelle sur le fichier sonore. Le dernier nous permet de jouer le fichier à partir de un objet *play* $\sim$.

On rappelle la formule du vocodeur de phase :

$$Phase\ Vocoder\ =\ \sum_{n=0}^{N-1} h_a(n)\ x(n+uR_a)\ e^{-j\omega\frac{n}{N}}$$

Dans la formule ci-dessus on peut voir que il y a un fenetrage correspondant, marqué $h_a(n)$. Pour simplifier on va substituer avec quelques parametres réels. On remplace N par une taille de la fenetre à 1024 echantillons. On etablit on decallage de 4 fois par fenetre, donc $uR_a = (\frac{1}{4} * 1024)_a$ pour l'index a de la fenetre precedente.

$$Phase\ Vocoder\ =\ \sum_{n=0}^{1023} h_a(n)\ x(n+256_a)\ e^{-j\omega\frac{n}{1024}}$$

À partir de cette formule on peut identifier les elements qui la reproduce dans le patch Max.

accord d'un offset parmi le buffer.

On identifie les elements de la formule dans notre patch. La fonction, correspondant à la phase de l'analyse, contient la somme des echantillions du signal qui appartiennes à la taille de la fenetre de la FFT plus des echantillions qui appartiennes $\Sigma_n(x_n)$ à la fennetres decallée par la taille *hop*, uR_a. Les echantillions sont multipliées, d'abord, par une fenétrage $h_a(n)$ et ensuite par la formule de Euler qui equivaute à la transformation de Fourier.

We can see that a windowing feature is including both in the formula and in the patch presented. Instead of using a default windowing from the $fft\sim$ object, we are going to use a fixed buffer of N size and using the *count* $\sim$ object we are going to access each sample with the assistance of the *index* $\sim$[5] object and multiply it with the sound we want to process before performing the FFT.

Filtre Gabor

In section 2 we presented Gabor filtering. In this section, we are going to create a personalized Gabor filter. The normalized Gabor formula allows to modify the curve of the gaussian function while keeping the maximum and minimum values constants. This custom made gaussian windowing is presented in figure 3.6.

We use an *uzi*[6] object with argument the size of the FFT window, followed by the expression which performs a normalized gaussian curve. Then we send it to the buffer named "windowing" with the help of the object *peek* $\sim$[7] Of course in this case the Gabor filter is not directly performed during the FFT but exactly before. Using this filtering we can freely overlap windows and avoid artifacts.

Pitch Shifting

To apply pitch shifting with the Phase Vocoder one should change the pitch factor while at the same time change the sample rate. For example, in order to transpose a sound an octave lower

5. The object *index* $\sim$ is used to read from a $buffer\sim$ object at a signal-driven sample index with no interpolation on the output.
6. sends the number of banks that have been set in the argument position at once.
7. The object *peek* $\sim$ is used to read and write sample values to a named $buffer\sim$.

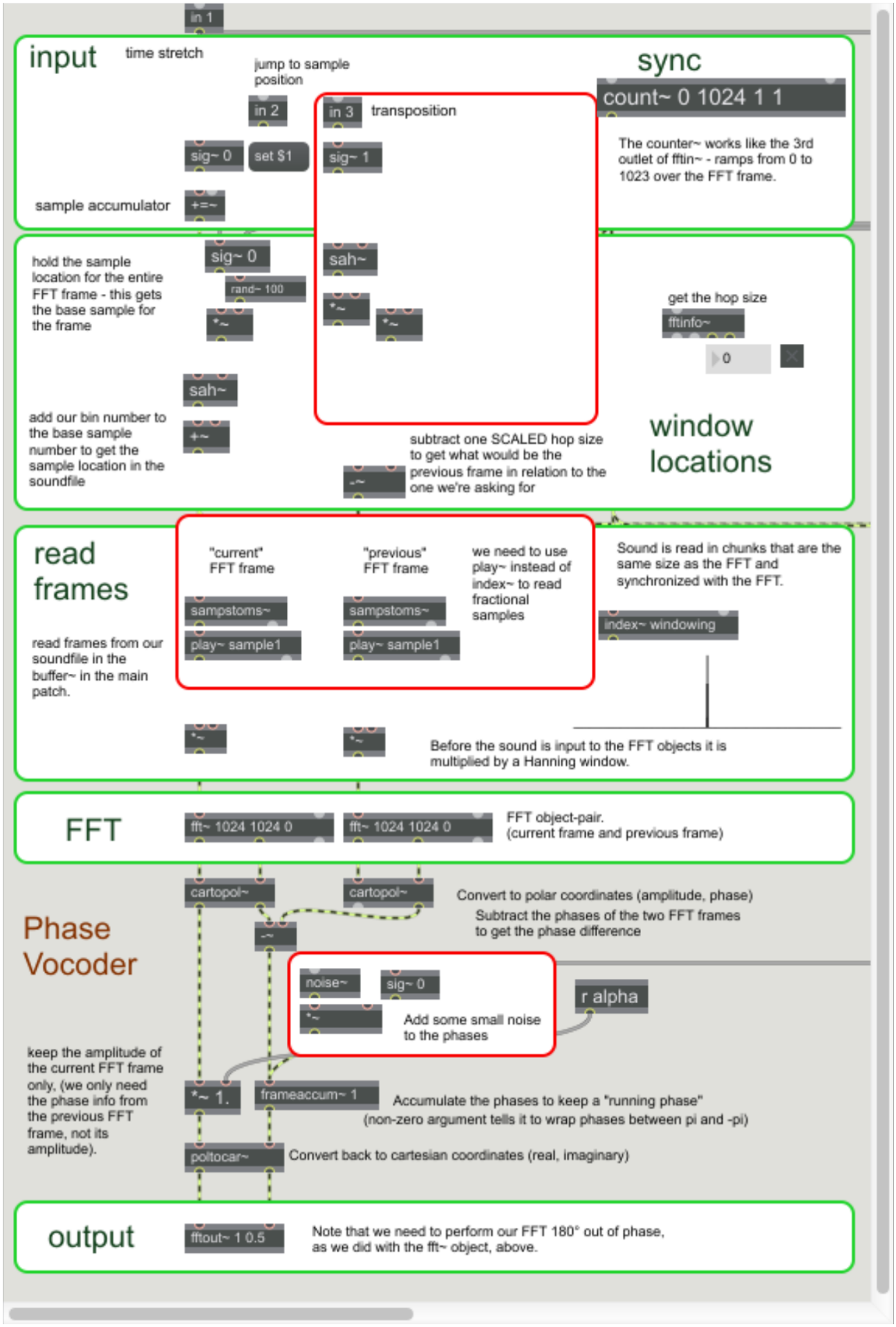


FIGURE 3.5 – Le vocodeur de phase

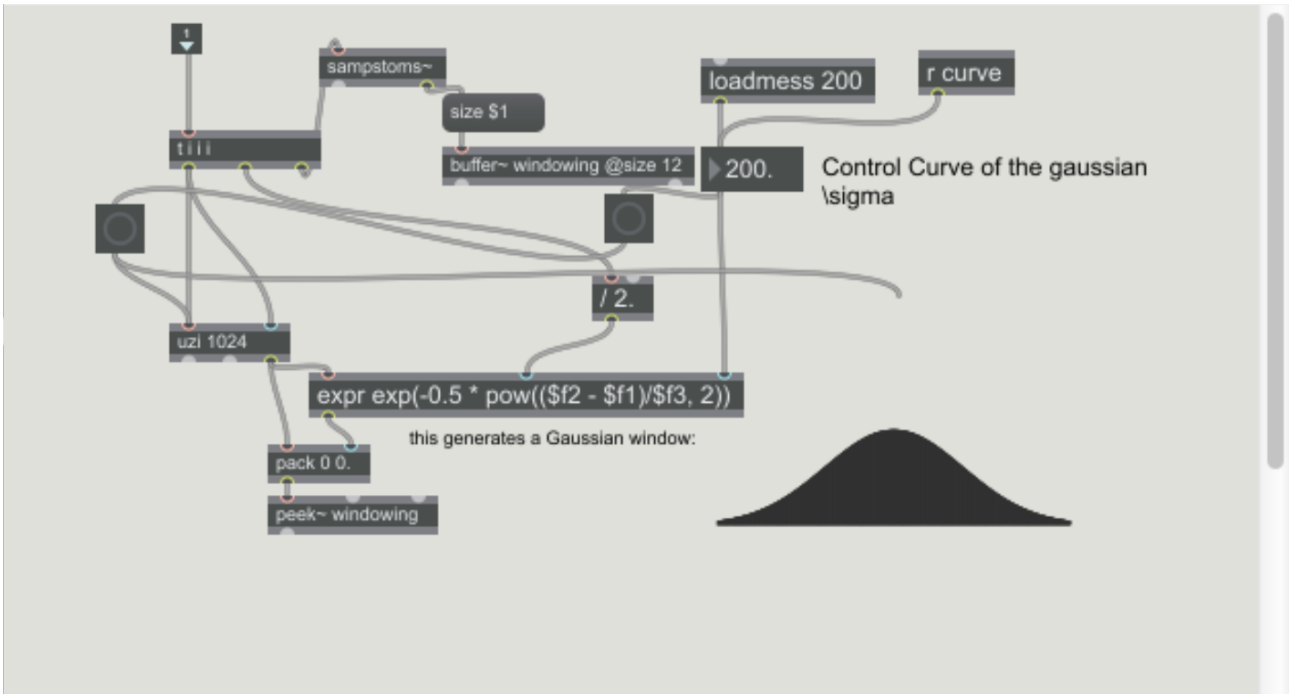


FIGURE 3.6 – Fenetrage gaussien

we should playback the sound at half the speed while doubling the SR. By using this method it is possible to change the pitch with affecting the playback speed.

To use this method we use standard midi values translated to frequency and adapted to the window overlap. The user then can input the value in a virtual midi piano thus facilitating the manipulation. By default the midi note Do with value 60 is the standard pitch. Then the pitch changes accordingly to the interval accordingly to the default value. Therefore, a Do an octave lower in the virtual midi piano, corresponds to an octave down of transposition from the original pitch of the sound.

Inside the phase Vocoder, the pitch shifting is translated as skipping samples or oversampling the *count* $\sim$ object. Of course to increase the SR is not a valid solution thus we choose to take a smaller or larger portion from the $buffer \sim$ corresponding to the window size. A transposition up corresponds to taking a larger window from the $buffer \sim$ and reading it in a faster speed, while a pitch shift down corresponds to cutting a smaller portion of the $buffer \sim$ and reading it slower.

For this example we use a *sah* $\sim$ object to make sure the transposition value is held constant

for all bins during the FFT. Then we multiply with the current value of the counter and we add the output value to the number of samples played so far to get the desired location of the file. In respect of the original frequency ω, the individual shift amounts to :

$$\Delta\omega(\omega) = \omega(\alpha - 1)$$

Where α is the transposition factor and $\Delta\omega$ the frequency difference. This technique imposes a constant frequency transposition $\Delta\omega$ to all the detected frequencies of the FFT.

The Pitch shifting implementation can be viewed in the figure . It takes only a few objects to implement a pitch shifting on the basic body of the phase vocoder.

Playback speed

As it was investigated in section 2, changing the playback speed while keeping the pitch stable is the inverse operation of pitch shifting. Playback can be controlled by taking into consideration the overlapping factor of the FFT windows. One can simply divide by the overlapping factor and add the percentage of the desired playback. Inside the vocoder the factorized value is added to the sample accumulator. On this wise, the playback speed value is added in the sample position interpretation. A faster playback skips samples where a slower playback speed read samples multiple times. The subtraction from the previous position of each sample (la window of the FFT), fixes the frequency deviations.

3.4 Morphing spectrale

The final demonstration for this chapter is an implementation of spectral morphing or spectral cross synthesis. To accomplish this technique we use two parallel phase vocoders and we interpolate phase and amplitude of each bin index. To understand the computational complexity of spectral morphing we will remind that a simple phase vocoder uses two parallel FFTs shifted by the overlap window factor. So if we use a window of 1024 samples and a overlap factor of

4, then the first FFT starts in position 0 and parallel window at sample position 256. Now for spectral morphing we use 4 parallel FFTs. That's a lot of computation, but still is possible with current computers.

In this double phase vocoder the outputs of the object *cartopol* ~, the magnitude and the phase after the Fourier analysis, are multiplied by the morphing interpolation factor. The phase is corrected beforehand by an accumulation with the previous frame and an smalled amount noise is added for a more natural result. We separate the magnitude and phase interpolation channels. Thusly, one can prefer to output, for example, the magnitude of the source sound but the phase of the target sound.

The output of each sound after the interpolation is transformed into Cartesian form and a inverse FFT is applied. Thus we obtain a seamless sound result. At this point a number of editing operation can be added, such as noise modeling or varying interpolation per frame, etc. But before diving into advanced modifications let us first investigate the interpolations techniques.

Interpolation is set by default to linear. A simple Javascript code we implemented creates a linear curve. The interpolation values vary between $[0, 1]$. The value 0 corresponds to zero interpolation, thus the magnitude or phase of the source sound is outputted. A value of 1 outputs entirely the characteristics of the target sound and omits the source sound. A linear curve sets a linear way to interpolate between sounds. But we go a step further and implement a number of curves such as exponential, logarithmic and, of course, linear. Therefore, one has different ways to interpolate between sounds based on their frequency components or the magnitude of their frequency components. All interpolations vary between the same integral $[0, 1]$. The Javascript code can be viewed in the listing bellow.

```
// autowatch 1;

// global variables
var s = 1;

function bang(){
  if (myFunc == "Expondential")
  {
    x = Math.log(1.7182*x+1);
    post("Expondential Interpolation");
  }
  else if (myFunc == "Logarithmic")
  {
    x = (x-1)*10;
    x = Math.pow(2, x);
    post("Logarithmic Interpolation");
  }
  else {
    x = x;
    post("Linear Interpolation");
  }
  outlet(0,x);
}

function msg_float(v){
  post("interpolation value " + v + "\n");
  x = v;
  bang();
}

  function anything(){
  var a = arrayfromargs(messagename, arguments);
  post("received" + a + "\n");
  myFunc = a;
  bang();
}
```

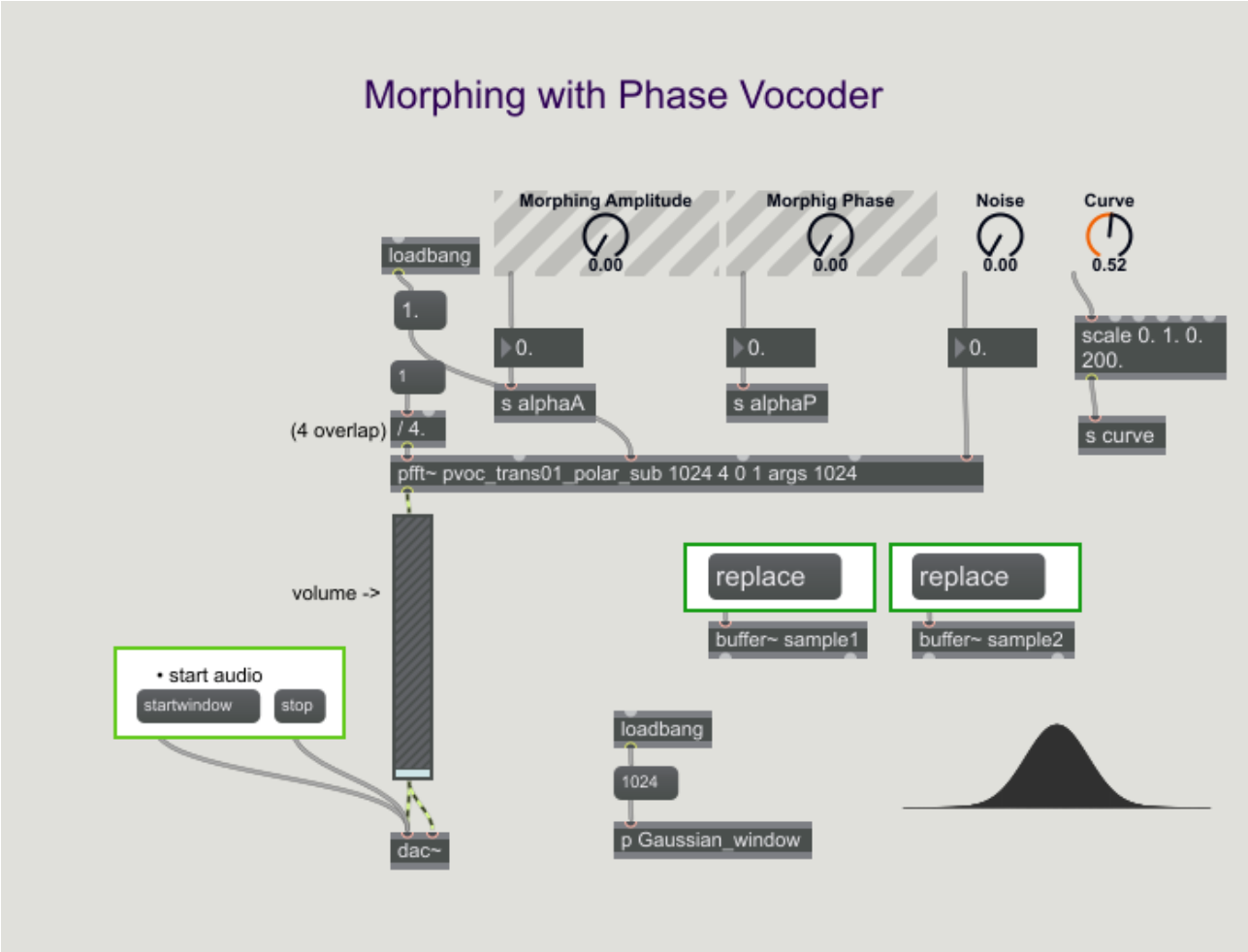


FIGURE 3.7 – Morphing en temps réel

The version of the phase vocoder for spectral morphing is shown in the figure 3.7. There are two buffers one for the source and one for the target sound. The Gaussian windowing for Gabor filtering with an remote normalized curve. An interpolation knob for phase and an knob for magnitude interpolation control the interpolation factors for morphing and a noise factorizing for naturalizing the phase are implemented.

Chapitre 4

Implémentations artistiques

4.1 *Introduction*

This section concludes an artistic point of view of the aspects discussed in the previous chapters. Starting from the Fourier transform to the phase vocoder and ending to morphing artistic interpretations of the implemented technical Max patches.

The goal of this chapter is to support the idea that spectral analysis and resynthesis and in expansion musical signal processing is useful both to scientifics and artists. Indeed the previous extensively technical details come to use for artistic and aesthetical purposes. The following examples consist of personal experimentation and interpretation of the technical material already presented.

Cette section conclut un point de vue artistique sur les aspects abordés dans les chapitres précédents. À partir de la transformation de Fourier en vocodeur de phase et en passant au morphing par des interprétations artistiques des patchs techniques mis en œuvre.

L'objectif de ce chapitre est de soutenir l'idée que l'analyse et la re-synthèse spectrales, et donc en expansion, le développement du signal musical sont utiles aux scientifiques et aux artistes. En effet, les détails techniques antérieurs sont utilisés à des fins artistiques et esthétiques. Les exemples suivants consistent en une expérimentation personnelle et une interprétation du

matériel technique déjà présenté.

4.2 Le vocodeur de phase - *Une capacité sans fin*

4.2.1 Super Phase Vocoder

For the purpose of combining mulptiple techniques of the phase vocodeur we implementated a Super Phase Vocoder. This spectral tool includes time streching, time freeze, pitch shifting, morphing and other effects beneficiating from the basic function of the phase vocoder.

Dans le but de combiner plusieurs techniques du vocodeur de phase, nous avons implémenté un «*super* vocodeur de phase». Cet outil spectral comprend l'étirement temporel, le freeze temporel, le décalage de hauteur, le morphing et d'autres effets bénéficiant de la fonction élémentaire du vocodeur de phase.

The graph 4.1 displays the control window of all parameters for the phase vocoder. This version of the phase vocoder contains every effect that was investigated in the previous section. There is not something particularly different from the basic phase vocoder for morphing but some more functionalities. It is only natural to seek for a larger functionality for the same tool. Thus, we implemented every basic effect into a single patch. The math here are not interesting to analyse as they were all seen before. The most crucial details are in the sequencing of all effects that can be visualized in figure 4.2.

4.2.2 Phase interference

In this MaxMSP patch we investigate a modification to the polar coordinates after the analysis stade. We simply add a phasor oscillator after the FFT and apply it to each magnitude and corrected phase of the window. This way the identity of the signal is not altered but an oscilating frequency interferes with every index of the FFT window. The patch can be viewed in the figure

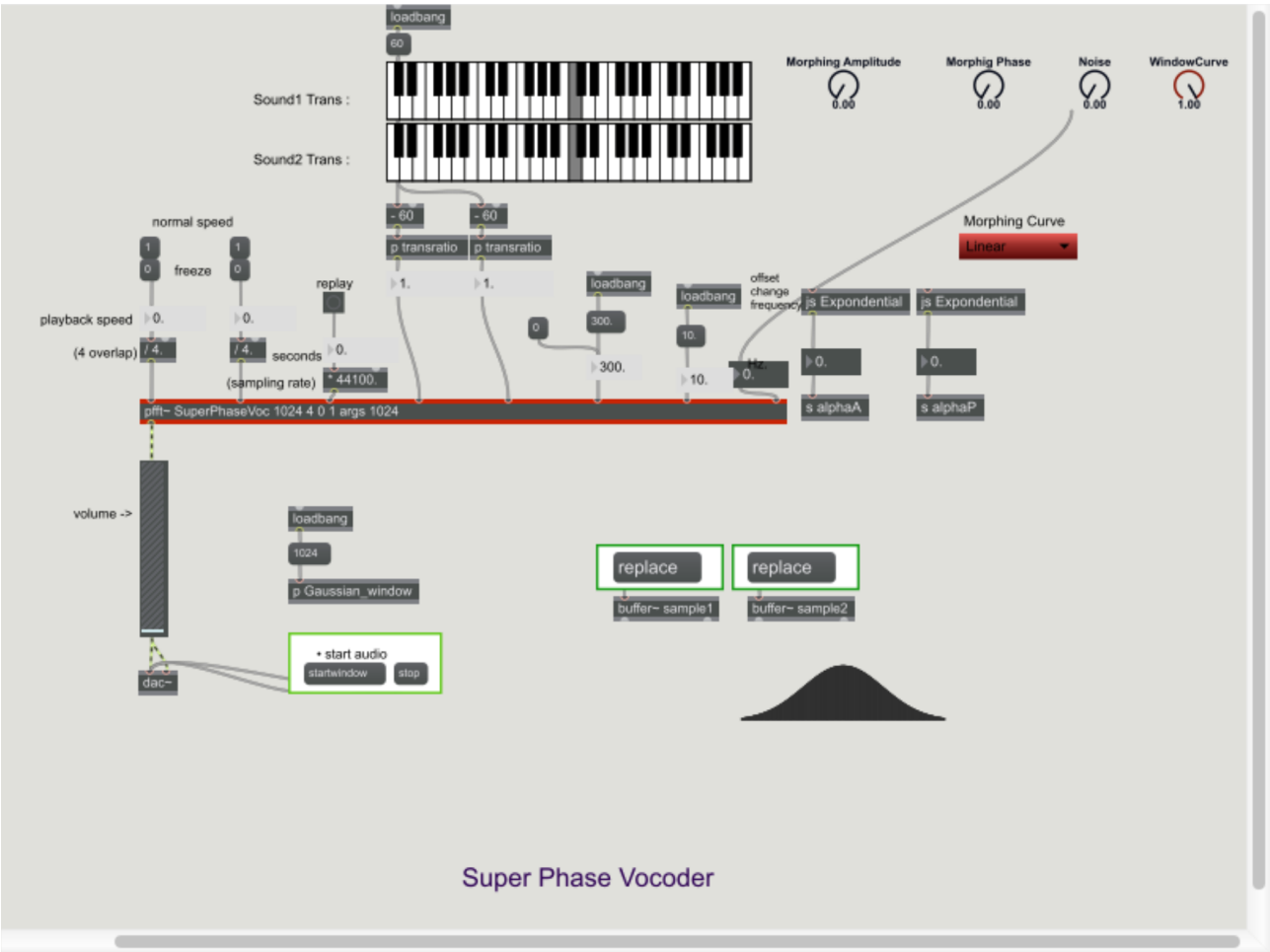


FIGURE 4.1 – Super Phase Vocoder I

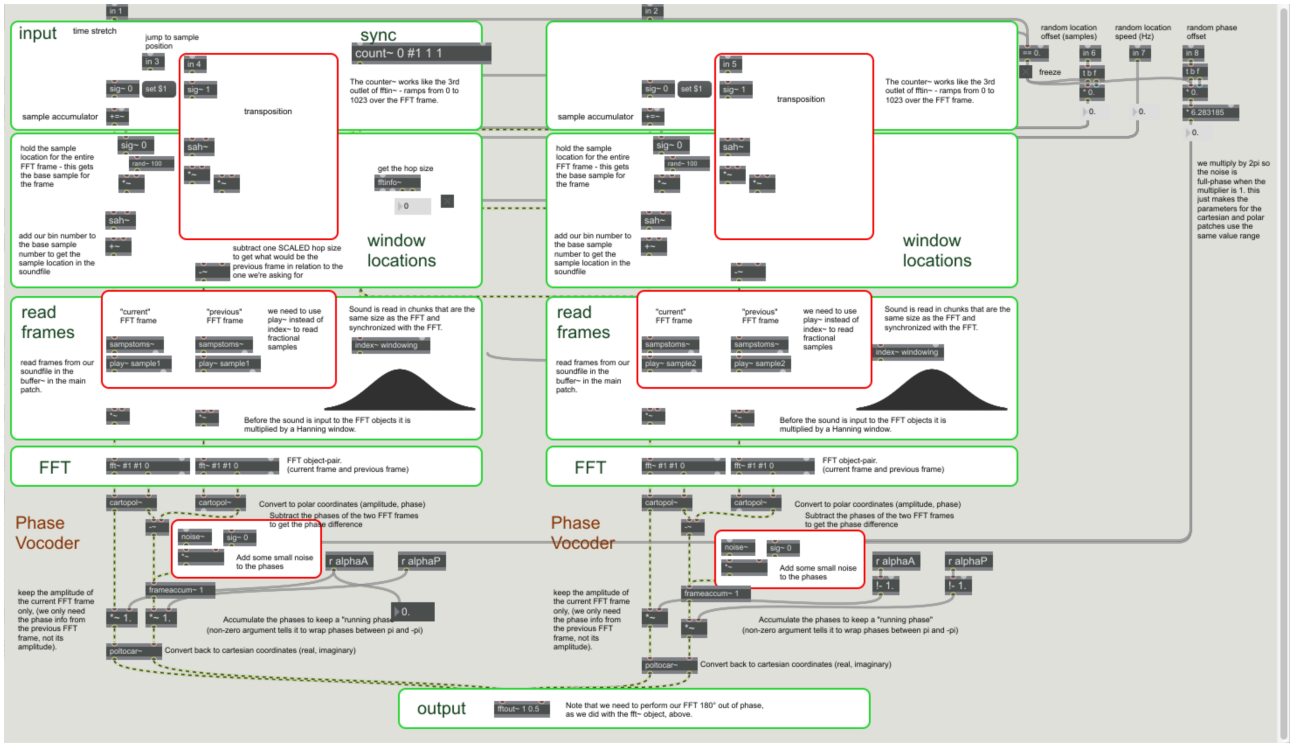


FIGURE 4.2 – Super Phase Vocoder II

. The formula of this modification is presented bellow by the synthesis equation.

$$y(n,k) = \sum_{k=1}^{K} (A[n] + phi(n))\ e^{j(\theta_k(n)+\phi(n)} \tag{4.1}$$

Where y(n, k) is the windowed signal after the analysis. K is the number of sinisuoids, $A[n]$ is the instantaneous magnitude, θ is the instantaneous phase and $\phi(n)$ is the instantaneous phasor given by the formula :

$$\phi(n) = n - floor(n)$$

We can see that, the underlying model of the FFT is used to represent this modification. Other models could perfectly describe this effect but in this version it is more direct.

4.2.3 Modulation au bruit

In this Phase Vocoder we investigate a noise modulation on the frequency component. We add the noise component and we tried some different noise generators such as *rand* ∼, pink-noise, white-noise and simultaneously we controlled the amplitude factor of this noise component.

4.2.4 Filtre aletoire sur la position du buffer∼

In this patch we modulate the position of the buffer through a random signal generator that passes it's values to the reader of the file position. The random generator is created by the object *random* ∼ with two filters modulating the frequency of the random value generation and the amplitude of its values. The values that passes into the reader of the *play* ∼ object is modulated by a sample holder that takes as inputs the output of a counter and the random generated value. In this manner we hold and release abstractly values within the FFT window creating a hybrid of pitch and playback modulation.

4.3 Morphing visuel

En avançant sur le morphing visuel, nous avons implémenté le patch suivant avec l'aide du Jitter, pour une interpolation linéaire entre deux dessins.

En répétant la formule de base de morphing $M(\alpha) = \alpha\widehat{S_1} + [1-\alpha]\widehat{S_2}$ une patch sur le morphing en 3D était implémenté avec l'aide de l'objet jit.gen (figure 4.3).

Donc, fondamentalement, un morphing visuel est facile à faire avec les fonctions 3D primordiaux de Jitter telles que jit.gl.gridshape et jit.gen pour une personnalisation de la procédure de morphing. L'objet jit.gl.mesh est utilisé pour combiner le résultat du morphing alors que l'objet gen est contrôlé par un facteur de fondu.

À l'intérieur de gen, une procédure assez simple se produit. Les données multidimensionnelles provenant des matrices de localisation sont utilisées séparément pour chaque forme et leur

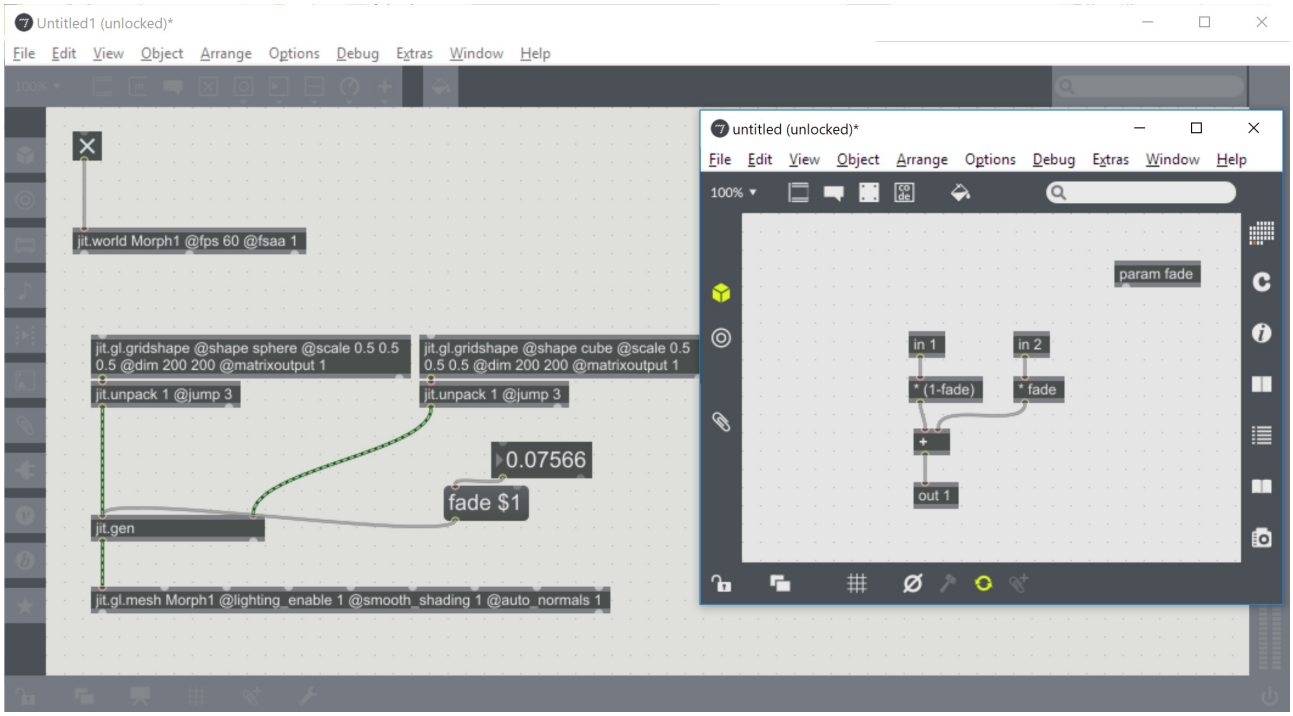


FIGURE 4.3 – Visual Morphing

amplitude est multipliée par le facteur α comme dans le morphing audio.

Bien entendu, nous pourrions également implémenter le script expondential.js pour une courbe de morphing différente sur le visuel.

4.3.1 Visualization du spectre

In this patch a lot of Jitter work was implemented along with a Phase Vocoder. This code allows to visualize the spectral information of a sound as a packet of two dimensional lines mimicking the harmonics found from a Fourier Analysis.

This patch used in unison with a phase Vocoder for morphing or even simple pitch shifting one can visualize the change of the spectrum in a real-time rendering environment. In this code we use an additional jitter library to produce multiple jitter objects rendered in virtual three dimensional space. The objective of this dissertation is not image oriented thus we won't unveil a lot of processing information.

Here we will attribute briefly the most iconic jitter objects used in this patch. The *jit.world*

object creates a new window and a three dimensional virtual rendering space that also could contain physics, planes, three dimensional objects, control the camera position, the background color and a lot more. The *jit.gridshape* renders a three dimensional object into a window. The *jit.mo* library replicates copies of jitter objects in a smooth manner. And the *jit.catch* takes a signal and translates it into a jitter matrix for potential visualization.

For transforming the spectral information into three dimensional data we use of course an FFT transform and a buffer that reads the polar components of the FFT (magnitude and phase) by a *peek* ~ object sends them into a buffer. An object *lookup* ~ recuperates the contents of the buffer and inputs them into the *jit.catch* object.

We can regulate the dimension of the *jit.mo* to output more harmonics. The output sound of the vocoder is sent to this harmonic visualization patch to grasp the result in a 3D space.

Chapitre 5

Conclusion

5.1 Résumé de la recherche

This dissertation served documentary means in order to inform musicians and composers of non-everyday notions and facts. This work was also fruitful in a personal level, as it helped me furthen my knownledge in the domain. Spectral processing is frequently very hard to understand for musicians. Maybe the work of a musician on the field will help the community to understand better through this series of implementations and artistic uses.

Cette recherche sert à des fins documentaires, le but est d'informer les musiciens et les compositeurs des notions complexes. Ce travail me sera aussi enrichissant au niveau personnel, il me permettra d'approfondir mes connaissances dans ce domaine. Le spectre est une terminologie difficile à comprendre pour les musiciens et un point de vue plus musical va faciliter sa compréhension. En approfondissant sur le terrain du morphing, on découvrira de nouvelles manières pour manipuler cet effet. Cette problématique, constituera le but final de la recherche.

5.2 Applications

The applications of the Phase Vocoders are pretty much endless as seen in the implementation chapter. A few propositions will follow to display the capacities of spectral analysis and the phase vocoder.

The phase vocoder can be used to analyse voice and transform it to text. The picks and the frequencies can be deducted to certain vowels and consonants allowing to predict the exact letters used and produce the written form of the sound.

The phase vocoder human voice-wised can also transform a certain type of voice, for example a man's voice to a female or any other type of voice by morphing between the characteristics that determine a voice as male, female or any other.

The spectral tools and specially those of analysis-synthesis are used frequently by composers vis DAW programs. A famous library would TRAX by Ircam tools.

5.3 Discussion

The phase vocoder today is more than just a spectral analysis tool. Methods develloped to stachastically forsee the most suitable wave resynthesis.

These formuli consider propabilistic methods for reconstructing signals, for better frequency prediction techniques. For more information on this subject, the book of Roads and al. [1] was pretty enlighting.

5.4 Recherche pour l'avenir

En outre, je propose une suite de ma recherche sur les processus spectraux en utilisant les principes tels que Deep Learning. Spécifiquement un approche sur le Morphing spectrale en

1. Curtis Roads, Stephen Travis Pope, Aldo Piccialli, Giovanni De Poli, *Musical Signal Processing*, 1997

temps réel connecte aux networks neurals [2].

Je suis particulierment interessé par le fait que un vocoder de phase puisse detecter les frequences dominantes d'un son. Ce fait pourrait essentiallement servir à une method intelligente de presiser les differentes notes contenue dans une information sonore. L'extraction des frequence en forme de notes pourrait assister à l'analyse automatique de la musique à partir des fichiers sonores et, par suite, transformer le son à une forme symbolique. Dans une recherche de futur, je souhaiterais m'orienter dans le domaine de l'analyse Topologic de la musique. Une interpretation de la musique symbolique comme des structures topologiques bidimentionelles, c'est-à-dire des graphes orrientés, peut assister à extracter informations du style musical ou de la structure. La théorie de graphes contiente plusieurs methodes spectrales telles que la transformation Fourier des graphes et toute une infra-structure mathématiques basée sur les concepts investigués dans cet recherche.

I am interested in using Variational Autoencoders (VAE) to identify the probability distribution of an ensemble of musical pieces. VAEs are autoencoders that parameterize the probability distribution of the latent variables using a neural network and proceed to maximize the likelihood of the data distribution subject to constraints in the latent variable distribution. Numerical experiments show it's capable of identifying that data distribution to the extent that sampling from the latent variable distribution generates realistic datapoints. I aspire to use this representation to produce new music by combining other musical pieces in the VAE embedding.

I believe that through the process of musical analysis on harmony and melody blended with unsupervised learning one can advance to enhanced models of musical prediction, vari- ation and creativity. To reach this goal, I visualize first algebraic modeling formalization on vertical (chords) and linear (melodies) data such as transformational music theory approaches. Proceeding to building a style database and applying an algorithm for prediction, I believe that through VAE this can be advanced and produce groundbreaking results. Hitherto VAE presented limited application to sequential data, and existing recurrent VAE models have difficulty modeling sequences with long-term structure. To address this issue, I propose the use

2. Jesse Engel. Making a neural synthesizer instrument, 2008.

of a hierarchical decoder, which first outputs embeddings for subsequences of the input and then uses these embeddings to generate each subsequence independently. It is also possible to model multitrack polyphonic music as vectors in a latent space. Via chord conditioning, which allows deeper understanding of melody while keeping harmony fixed, and further allows chords to be changed while maintaining musical style. This approach was first presented in the project Magenta of Google AI and in my opinion presents great potential and could advance on simultaneous chord and melody generation.

Je suis intéressé à utiliser des autoencodeurs variationnels (VAE) pour identifier la distribution de probabilité. création d'un ensemble de pièces musicales. Les VAE sont des auto-encodeurs qui paramètrent le pro- distribution de capacité des variables latentes en utilisant un réseau de neurones et procéder à maximiser la probabilité que la distribution des données soit soumise à des contraintes dans la distribution des variables latentes. Des expériences numériques montrent qu'il est capable d'identifier cette distribution de données dans la mesure cet échantillonnage de la distribution des variables latentes génère des points de données réalistes. J'aspire à utiliser cette représentation pour produire une nouvelle musique en combinant d'autres pièces musicales dans le VAE intégration.

Je crois que, grâce au processus d'analyse musicale sur l'harmonie et la mélodie, avec l'apprentissage non supervisé, on peut passer à des modèles améliorés de prédiction musicale, de ation et créativité. Pour atteindre cet objectif, je visualise la première formalisation de la modélisation algébrique sur données verticales (accords) et linéaires (mélodies) telles que les approches de la théorie de la musique transformationnelle. Pour construire une base de données de styles et appliquer un algorithme de prédiction, je crois que Grâce à VAE, cela peut être avancé et produire des résultats révolutionnaires. Jusqu'à présent, VAE présentait une application limitée aux données séquentielles et une VAE récurrente existante. les modèles ont du mal à modéliser des séquences avec une structure à long terme. Pour résoudre ce problème, je proposer l'utilisation d'un décodeur hiérarchique, qui produit d'abord des imbrications pour les sous-séquences de l'entrée et utilise ensuite ces imbrications pour générer chaque sous-séquence indépendamment. Il Il est également possible de modéliser de la musique polyphonique multipiste en tant que vecteurs dans un espace latent. Via accord conditionnant, qui permet une compréhension

plus profonde de la mélodie tout en maintenant l'harmonie, et permet en outre de modifier les accords tout en maintenant le style musical. Cette approche était présenté pour la première fois dans le projet Magenta de Google AI et présente à mon avis un grand potentiel et pourrait avancer sur la génération simultanée d'accords et de mélodies.

Appendix A

Appendix

A.1 FFT Cooley - Tukey algorithm

```
using FFTW
#simple DFT function
function DFT(x)s
    N = length(x)
    # We want two vectors here for real space (n) and frequency space (k)
    n = 0:N-1
    k = n'
    transform_matrix = exp.(-2im*pi*n*k/N)
    return transform_matrix*x
end
# Implementing the Cooley-Tukey Algorithm
function cooley_tukey(x)
    N = length(x)
    if (N > 2)
        x_odd = cooley_tukey(x[1:2:N])
        x_even = cooley_tukey(x[2:2:N])
    else
        x_odd = x[1]
        x_even = x[2]
    end
```

```
21          n = 0:N-1
22          half = div(N,2)
23          factor = exp.(-2im*pi*n/N)
24          return vcat(x_odd .+ x_even .* factor[1:half],
25                      x_odd .- x_even .* factor[1:half])
26      end
27      function bitreverse(a::Array)
28          # First, we need to find the necessary number of bits
29          digits = convert(Int,ceil(log2(length(a))))
30
31          indices = [i for i = 0:length(a)-1]
32
33          bit_indices = []
34          for i = 1:length(indices)
35              push!(bit_indices, bitstring(indices[i]))
36          end
37          # Now stripping the unnecessary numbers
38          for i = 1:length(bit_indices)
39              bit_indices[i] = bit_indices[i][end-digits:end]
40          end
41          # Flipping the bits
42          for i =1:length(bit_indices)
43              bit_indices[i] = reverse(bit_indices[i])
44          end
45          # Replacing indices
46          for i = 1:length(indices)
47              indices[i] = 0
48              for j = 1:digits
49                  indices[i] += 2^(j-1) * parse(Int, string(bit_indices[i][end-j])
    )
50              end
51             indices[i] += 1
52          end
53          b = [float(i) for i = 1:length(a)]
54          for i = 1:length(indices)
```

```
55          b[i] = a[indices[i]]
56      end
57      return b
58  end
59  function iterative_cooley_tukey(x)
60      N = length(x)
61      logN = convert(Int,ceil(log2(length(x))))
62      bnum = div(N,2)
63      stride = 0;
64      x = bitreverse(x)
65      z = [Complex(x[i]) for i = 1:length(x)]
66      for i = 1:logN
67          stride = div(N, bnum)
68          for j = 0:bnum-1
69              start_index = j*stride + 1
70              y = butterfly(z[start_index:start_index + stride - 1])
71              for k = 1:length(y)
72                  z[start_index+k-1] = y[k]
73              end
74          end
75          bnum = div(bnum,2)
76      end
77      return z
78  end
79  function butterfly(x)
80      N = length(x)
81      half = div(N,2)
82      n = [i for i = 0:N-1]
83      half = div(N,2)
84      factor = exp.(-2im*pi*n/N)
85
86      y = [0 + 0.0im for i = 1:length(x)]
87
88      for i = 1:half
89          y[i] = x[i] + x[half+i]*factor[i]
```

```
90              y[half+i] = x[i] - x[half+i]*factor[i]
91          end
92          return y
93      end
94      function approx(x, y)
95          val = true
96          for i = 1:length(x)
97              if (abs(x[i]) - abs(y[i]) > 1e-5)
98                  val = false
99              end
100         end
101         println(val)
102     end
103     function main()
104         x = rand(128)
105         y = cooley_tukey(x)
106         z = iterative_cooley_tukey(x)
107         w = fft(x)
108         approx(y, w)
109         approx(z, w)
110     end
111     main()
112
```

Listing A.1 – Cooley-Tukey with Butterfly Diagrams

[1].

A.2 Espaces L^p et STFT

La STFT est une version locale de la transformée de Fourier. Normalement, pour effectuer la transformation de Fourier, on part d'une fonction $f(x)$ qui appartient à $L^2(\mathbb{R}^k)$ soit à $L^1(\mathbb{R}^k)$ et on obtient une fonction $\hat{f}(\omega)$ qui appartient respectivement à $L^2(\mathbb{R}^k)$ ou à $C_0(\mathbb{R}^k)$ [2].

1. Le code est recuperé sur le lien www.algorithm-archive.org distribué sous la licence *MIT*
2. L'espace des fonctions continues qui tendent vers 0 à l'infini

Pour effectuer la STFT on doit d'abord fixer une fonction qui nous servira de fenêtre. Dans une approche plus practique, on utilisera généralement des fonctions avec une formte décroissance à l'infini (par exemple une fonction gaussienne). De cette manière on pourra appliquer la STFT à des fonctions qui n'ont peut-être pas de transformée de Fourier ordinaire. On laissera les détails de quelles fonctions sont celles qui admettent une STFT (en fonction de la fenêtre fixée) pour plus tard et on se contentera d'abord de voir que si $g \in L^p(\mathbb{R}^k)$ alors toute $f \in L^{p'}(\mathbb{R}^k)$ admet une STFT.

Pour appliquer la STFT à une fonction il faut être sur que le résultat sera bien défini ; cela nous rapporte au problème de choisir un domaine de définition pour la STFT. Dans la définition de la STFT on a choisit comme domaines pour f et pour g respectivement $L^{p'}(\mathbb{R}^k)$ et $L^p(\mathbb{R}^k)$. Ce choix a été fait pour la première approche mais le fait est que la STFT peut être étendue à bien d'autres espaces, notament des espaces de distributions.

Dans cette section on va proposer quelques uns (ceux qui sont présentés en Gröchenig [2001]) ; loin d'être une exposition exhaustive, on ne présentera que les espaces les plus importants dans le cadre de l'analyse de Fourier. L'étude détaillé du cas des espaces de modulation sera laissé pour quand ceux-là seront présentés.

On rappelle qu'on utilise $\mathscr{D}(\mathbb{R}^k)$ pour noter l'espace des distributions ordinaires et $\mathscr{T}(\mathbb{R}^k)$ pour noter l'espace des distributions tempérées.

Definition A.1 *Soit E indifféremment $C_c^\infty(\mathbb{R}^k)$ (l'espaces des fonctions lisses à support compact) ou $\mathscr{T}(\mathbb{R}^k)$ et soit $\langle \cdot, \cdot \rangle$ son crochet de dualité.*

$$\begin{array}{rccccc} V_g\sigma: & \mathbb{R}^d \times \mathbb{R}^k & \to & \mathbb{C} & & \\ & (t,\omega) & \mapsto & V_g\sigma(t,\omega) & = & \sigma(M_\omega T_t g) := \langle \sigma, M_\omega T_t g \rangle \end{array}$$

et on obtient une extension de la STFT.

Cette définition n'a pas besoin de justification ; en effet, l'évaluation de la distribution dans un élément de son domaine est bien évidemment définie. On pourrait se demander si les opérateurs

M_ω et T_t laissent invariant les espaces $C_c^\infty(\mathbb{R}^d)$ et $\mathscr{T}(\mathbb{R}^k)$. De plus, la justification de que ce soit une extension est faite par l'interpretation de toute fonction de E comme élément de E' par le crochet de dualité.

Cela nous dit de plus que si B est un espace de Banach contenu dans $\mathscr{T}(\mathbb{R}^k)$ qui est invariant par des traslations et modulations alors la STFT est bien définie pour $f \in B'$ et $g \in B$.

Listings

Bibliographie

Xavier Marcelo et Xavier Rodet Freitas Caetano. Automatic timbral Morphing of Musical Instrument Sounds by High-Level Descriptors. pages pp. 11 – 21, United States, juin 2010. URL `https://hal.archives-ouvertes.fr/hal-00604390`.

cycling74.com. Max8 release date, 2018. URL `https://cycling74.com`.

Curtis Roads. *The Computer Music Tutorial*. MIT Press, Cambridge, MA, USA, 1996. ISBN 0262680823.

J. L. Flanagan et R. M. Golden. Phase vocoder. *The Bell System Technical Journal*, 45(9) : pp. 1493–1509, Nov 1966.

Daniel W. Griffin et Jae S. Lim. Signal estimation from modified short-time fourier transform. *IEE Transactions on acoustics, speech, and signal processing*, ISSE Vol. 32 :pp. 236–243, Avril 1984.

Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian Goodfellow, Jean Pouget-Abadie. Generative adversarial networks. page pp. 2672–2680, 2004.

An algorithm for the machine calculation of complex fourier series. *JSTOR*, 1965.

Mark Dolson et Jean Laroche. Impoved Phase Vocoder Time-Scale Modification of Audio. *IEEE Transactions on Speech and Audio Processing*, Vol. 7, mai 1999.

Aldo Piccialli Giovanni De Poli Curtis Roads, Stephen Travis Pope. *Musical Signal Processing*. Londres et New York, 1997.

Coralie Diatkine. Audiosculpt 3.0 user manual.

Karlheinz Gröchenig. *Foundations of Time-Frequency Analysis*. Birkhäuser Boston, 2001.

William Fulton. Introduction to intersection theory in algebraic geometry. In *Regional Conference Series in Mathematics*, number 54, 1983.

Axel Roebel. Morphing sound attractors. In *3rd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99), 1999, Florida, United States, Proc. of the 3rd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'99), 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99)*, 1999a.

Axel Roebel et Xavier Rodet. Efficient spectral envelope estimation and its application to pitch shifting and envelope preservation. URL `https://hal.archives-ouvertes.fr/hal-01161334`. cote interne IRCAM : Roebel05b, Journal = "International Conference on Digital Audio Effects", ADDRESS = "Madrid", PAGES = "pp. 30 – 35", YEAR = 2005, Month = septembre.

Mark Goresky et Robert MacPherson. On the topology of complex algebraic maps. In *Algebraic Geometry Proceedings, La Rábida, Lecture Notes in Mathematics*, volume 961, 1981.

Axel Roebel. Morphing sound attractors. *3rd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the 5th. Int'l Conference on Information Systems Analysis and Synthesis (ISAS'99)*, 1999b. URL `https://hal.archives-ouvertes.fr/hal-01253228`.

Hermann L F. Helmholtz. *The Sensations of Tone*. New York, 1895.

Michael Kateley Klingbeil. *Spectral Analysis, Editing and Resynthesis : Methods and Applications*. Université de Columbia, 2009.

Tadej Droljc. *STFT Analysis Driven Sonographic Sound Processing in Real-Time using Max/MSP and Jitter*. 2011.

Gérald Grisey. Écrits : ou l'invention de la musique spectrale. MF Éditions, 2008.

Javier R. Movellan. A tutorial on gabor filters, 2008. URL `http://mplab.ucsd.edu/tutorials/gabor.pdf`.

Jesse Engel. Making a neural synthesizer instrument, 2008. URL `https://magenta.tensorflow.org/nsynth-instrument`.

Alex Hofmann Iain McCurdy et Alexandre Abrioux Joachim Heinz, Andes Cabrerra. Fourier transformation / spectral processing.

Richard Ducas et Cort Lippe. The phase vocoder - part i, 2 novembre 2006. URL `https://cycling74.com/tutorials/the-phase-vocoder-%E2%80%93-part-i`.

Richard Ducas et Cort Lippe. The phase vocoder - part ii, 2 juillet 2007. URL `https://cycling74.com/tutorials/the-phase-vocoder-part-ii`.

Emmanouil-Nikolaos Karystinaios. An investigation into the spectral music idiom and timbral analysis functionality with max smp jitter. Master's thesis, Septembre 2017. preprint.

Cavin Wu. How computer technology influences art and design programs in higher education. Master's thesis, La Sierra University, 2006.

Jonathan Boley. *Auditory Component analysis using perceptual pattern recognition to identify and extract independent components from an auditory scenee.* PhD thesis, Université de Miami, 2005.

Johannes Grünwald. Theory, implementation and evaluation of the digital phase vocoder in the context of audio effects, 2010.

Jean-François Charles. A tutorial on spectral sound processing using maxmsp and jitter. *Computer Music Journal*, pages pp. 87 – 102, Printemps 2008.

Mark Dolson. The phase vocoder : a tutorial. *Computer Music Journal*, pages pp. 14 – 27, Printemps 1986.

Alan V. Oppenheim et Ronald W. Schafer. Discrete-time signal processing. *Prentice Hall Press*, pages pp. 822 – 835, 2009.

Steven W. Smith. The scientist and engineer's guide to digital signal processing. *California Technical Publishing*, 1997.

Jown Strawn. Introduction to digital sound synthesis. *Digital Audio Engineering*, pages pp. 141 – 134, 1985.

Solvi Ystad et Kristoffer Jensen Richard Kronalnd-Martinet, Thierry Voinier. Computer music modeling and retrieval. *4th international Symposium CMMR*, 2007.

Joshua Fineberg. Guide to the basic concepts and techniques of spectral music. *Contemporary Music Review*, pages pp. 81 – 113, 2000.

Dennis Gabor. Theory of communication. *ICMC*, 1944.

Bruno Depalle et Richard Kronland-Martinet Olivero Anaïk, Olivero Torrésani. Sound morphing strategies based on alterations of time-frequency representations by gabor multipliers. page pp. 17, Helsinki, mars 2012. URL `https://hal.archives-ouvertes.fr/hal-00682959`.

Axel Roebel. A new approach to transient processing in the phase vocoder. pages pp. 344 – 349, Londrais, septebre 2003a. URL `https://hal.archives-ouvertes.fr/hal-01161124`.

Axel Roebel. Transient detection and preservation in the phase vocoder. pages pp. 247 – 250, Singapore, octobre 2003b. URL `https://hal.archives-ouvertes.fr/hal-01161125`.

Geoffroy Peeters. A large set of audio features for sound description in the cuidado project. *A large set of Audio Feautures for Sound Description*, 2004.

Anne Sédes. Spectralisme français, du domaine fréquenctiel au domaine temporel, analyse, modélisation, synthèse. . . et prospectives. *The Foundations of Contemporary Composing*, 2002.

Fernando Villavicencio et Xavier Rodet Axel Roebel. On cesptral and all-pole based spectral envelope modeling witg unknown model order. *Pattern Recognition Letters*, (28) :pp. 1343 – 1350, 2007.