

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

Domaines fonctionnels du vocodeur de phase

Méthodes algébriques et réalisation des processus sonores

KARYSTINAIOS Emmanouil-Nikolaos

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

**Année universitaire
2018-2019.**

Resumé

... et puis il y avait du son. Sans le son, la musique n'aurait pu naître. Comprendre le son a été la quête de beaucoup de gens au fil des ans. Jouer avec le timbre, la dynamique, le registre, les couleurs, la naturalité et tous les autres aspects auxquels nous pouvons penser. Au cours de la seconde moitié du XXe siècle, les compositeurs et les chercheurs se sont interrogés sur ce qui compose les sons. Quels sont les composants qui font qu'un son de violon est différent d'un son de flûte quand ils jouent la même note ? C'est ainsi que le spectralisme est né comme voie menant à la véritable compréhension du son.

Par conséquent, cette thèse se concentre essentiellement sur l'exploration de la nature du son et, pour être plus précis, sur une étude du son spectral en temps réel. Dans cet article, le processus de traitement spectral va être analysé en profondeur et un build de phase Vocoder va être présenté.

Premièrement, il est crucial d'envelopper les notions mathématiques qui sous-tendent la théorie de la décomposition spectrale et de la resynthèse. Par conséquent, la première section traite de toutes les mathématiques pertinentes dont on aura besoin pour comprendre comment traiter les spectres des sons. Outre les formules mathématiques, nous plaçons un contexte musical dans une approche plus conviviale pour les musiciens.

Ensuite, nous implémentons à nouveau ce contexte mathématique dans un environnement de programmation convivial pour les musiciens, tel que le logiciel MaxMSP. Nous étudions comment les formules mathématiques interagissent les unes avec les autres dans un code. Des explications musicales et de programmation pas à pas se construisent tout au long de la naissance d'un simple Vocoder de phase.

Enfin, en combinant les connaissances acquises au cours des chapitres précédents et quelques principes de programmation musicale de base, nous construisons une série d'applications artistiques dans MaxMSP qui soulignent la valeur artistique d'un vocodeur de phase.

On peut s'attendre à ce que cette thèse soit un guide des musiciens pour le traitement spectral, mais également une source fructueuse d'exemples et d'applications artistiques. Son utilité pour tout musicien ou compositeur qui cherche à comprendre la logique des outils qu'il utilise fréquemment ; Quiconque se demande comment implémenter réellement un Vocoder de phase

dans MaxMSP ou tout artiste à la recherche d'idées pour un futur projet. Il faut s'attendre à ce que ces recherches englobent différentes utilisations du vocodeur de phase et donnent accès à un paramétrage plus poussé.

Table des matières

| | |
|--|-----------|
| Abstract | i |
| 1 Introduction | 1 |
| 1.1 À propos | 1 |
| 1.1.1 Outils de la réalisation | 2 |
| 1.2 Structure | 3 |
| 1.3 MaxMSP et Jitter | 5 |
| 1.4 Analyse spectrale | 6 |
| 1.4.1 Histoire | 6 |
| 1.4.2 Le vocodeur de phase | 7 |
| 1.5 Morphing | 8 |
| 1.5.1 Morphing sonore | 8 |
| 1.5.2 Morphing visuel | 9 |
| 2 Context théoritique | 10 |
| 2.1 Introduction | 10 |
| 2.2 Décodage mathématique | 11 |

| | | |
|----------|---|-----------|
| 2.2.1 | The Fourier Transform | 11 |
| 2.2.2 | Fenetrage | 17 |
| 2.2.3 | Le vocodeur de phase | 19 |
| 2.2.4 | Applications du vocodeur de phase | 23 |
| 3 | Design | 28 |
| 3.1 | MaxMSP Objects | 28 |
| 3.2 | Pitch tracking | 31 |
| 3.2.1 | Multiple Pitch Tracking | 36 |
| 3.3 | Le vocodeur de phase | 37 |
| 3.4 | Morphing spectrale | 44 |
| 4 | Implémentations artistiques | 48 |
| 4.1 | Introduction | 48 |
| 4.2 | Le vocodeur de phase - <i>Une capacité sans fin</i> | 48 |
| 4.2.1 | Super Phase Vocoder | 48 |
| 4.2.2 | Phase interference | 49 |
| 4.2.3 | Modulation au bruit | 51 |
| 4.2.4 | Filtre aleatoire sur la position du buffer~ | 52 |
| 4.3 | Morphing visuel | 53 |
| 4.3.1 | Visualization du spectre | 54 |

| | |
|---|-----------|
| 5 Conclusion | 58 |
| 5.1 Résumé de la recherche | 58 |
| 5.2 Applications | 58 |
| 5.3 Discussion | 59 |
| 5.4 Recherche pour l'avenir | 59 |
| .1 FFT Cooley - Tukey algorithm | 62 |
| .2 Espace L^p et STFT | 65 |

Table des figures

| | | |
|-----|---|----|
| 2.1 | Complex DFT | 12 |
| 2.2 | Circle de la transformation Fourier | 13 |
| 2.3 | Magnitude et phase | 14 |
| 2.4 | Butterfly Diagrams for an 8 point FFT | 17 |
| 2.5 | Overlapping | 18 |
| 2.6 | Interpolation du facteur α | 26 |
| 3.1 | Pitch Tracking | 32 |
| 3.2 | Pitch Tracker | 35 |
| 3.3 | Gen multiple | 36 |
| 3.4 | Multiple Pitch tracking | 37 |
| 3.5 | Le vocodeur de phase | 41 |
| 3.6 | Fenetrage gaussien | 42 |
| 3.7 | Morphing en temps réel | 47 |
| 4.1 | Super Phase Vocoder | 50 |
| 4.2 | modulation des coordonées polaires | 51 |

| | | |
|-----|--|----|
| 4.3 | Selection du bruit | 51 |
| 4.4 | Filtre aleatoire | 52 |
| 4.5 | Visual Morphing | 53 |
| 4.6 | Visual Morphing 2^{me} version | 54 |
| 4.7 | Stades du morphing visuel | 55 |
| 4.8 | Harmoniques dans l'espace | 57 |
| 4.9 | Le patch qui control les objets jitter | 57 |

Chapitre 1

Introduction

1.1 À propos

De nos jours, la musique est devenue un vaste sujet de recherche scientifique. Les branches de la musique se développent à travers divers domaines tels que, l'analyse musicale mêlée avec la modélisation mathématique, l'histoire avec l'ethnomusicologie, l'acoustique avec la physique, la composition avec la programmation, etc. En empruntant cette voie interdisciplinaire, on va lier la musique aux autres sciences, afin ainsi de progresser vers un nouveau point de vue et de présenter de nouveaux résultats.

Dans la branche de la composition musicale, l'étude de la nature du son a été spécialement développée en produisant diverses techniques guidées par des phénomènes physiques. Une série d'outils de traitement du son basés sur des phénomènes physiques sont étiquetés sous une branche appelée analyse-synthèse du son. Dans ce domaine, un ensemble de disciplines scientifiques se rencontrent dans un seul but, de comprendre la musique et la nature sonore. À travers l'analyse sonore, nous procédons à la synthèse. De la science nous nous menons à l'art. Le nom d'analyse-synthèse exprime ce principe, d'observation (analyse). Cela permet ainsi de procéder au traitement du son et aboutir à un nouveau produit sonore (synthèse). Le nom, est traduit littérairement au traitement spectral, où l'étape d'analyse transfère un son du temps au domaine fréquentiel. L'analyse présente les informations spectrales d'un son à une certaine période du

temps. L'outil de synthèse inverse le processus de transfert du son du domaine fréquentiel vers le domaine temporel ce qui produit comme résultat la forme de signal d'onde conventionnelle.

Les outils d'analyse-synthèse sont initialement conçus pour aider les compositeurs ou les artistes en général dans la composition contemporaine. Ils peuvent servir notamment, pour l'utilisation de séquences générées par l'ordinateur sur les œuvres de Stockhausen ou bien pour l'analyse spectrale assistée par ordinateur pour répondre aux besoins des compositeurs spectraux, tel que Gerard Grisey. Pour autant, les résultats ou les techniques ne sont pas nécessairement bornés à un usage artistique, mais sont également appliqués dans d'autres domaines tels que les communications, quelques hardwares, etc.

Dans ce dissertation, on souligne le processus dans le domaine de l'analyse spectrale → transformation → synthèse. On examine, en particulier, la fonctionnalité de l'analyse de Fourier en temps discret avec des fenêtres d'enveloppe variés pour la construction d'un double vocodeur de phase pour réaliser du morphing sonore. Nous définissons le morphing sonore comme le processus consistant à combiner les spectres de deux sons dans un certain pourcentage, ce qui donne un son hybride contenant des éléments caractéristiques de ceux d'origine. Le terme morphing est également appelé, synthèse croisée, par la suite, les deux termes sont devenus équivalents. Le mécanisme exact de chaque terme, tel que transformée de Fourier, fenêtres d'enveloppe, vocodeur de phase, etc., sera analysé à la section 2.

1.1.1 Outils de la réalisation

Certains commentaires sur les outils utilisés dans cette recherche sont nécessaires avant de discuter des détails de la structure. Comme précédemment mentionné, une recherche interdisciplinaire comme celle-ci exige une collaboration de domaines et de termes issus des mathématiques, de l'informatique et de la musique. Par conséquent, le cadre doit être adapté aux besoins de cette dissertation.

Pour le texte à la place d'un éditeur *WORD* traditionnel, un éditeur de texte *LATEX* est utilisé. Ainsi, cette thèse est entièrement écrite en code *LATEX*. La nécessité de *LATEX* provient

de la quantité de formules mathématiques présentées, ainsi que du code joint dans certaines sections. De plus, le code source L^AT_EX est relativement plus facile à partager et lisible à partir de n'importe quel éditeur de texte. Cependant, l'utilisation de L^AT_EX rend différemment la formulation de références et la bibliographie en comparaison des normes françaises.

La partie programmation de ce dissertation est presque entièrement codée en MaxMSP et Jitter, avec quelques petites parties de code en Javascript. Max est un outil robuste de Cycling74 permettant une analyse et un traitement du signal en temps réel, ainsi que certains traitements visuels. Max Version 8.0.3 est utilisé sans bibliothèques externes. Les paramètres audio sont définis sur SR de 48.000, la taille du vecteur 442 et le vecteur du signal 64. Plus d'informations sur MaxMSP sont disponibles à l'adresse suivante : "<https://cyclisme74.com>".

Dernièrement, Reaper a été utilisé pour l'édition du son et l'exportation du format final .wav. Reaper est une station de travail audio numérique (DAW) avec des commandes multicanaux et des options d'enregistrement. Plus d'informations sur Reaper sont disponibles sur "www.reaper.fm".

L'archivage de cette recherche est disponible publiquement sur "Github.com" où il peut être consulté à des fins d'enseignement ou de recherche. Le lien vers le référentiel Github est disponible sur <https://github.com/melkisedeath/Un-Guide- sur-l-analyse-Spectrale> sous une licence *MIT*. Dans ce référentiel, on peut trouver le code, les fichiers sonores, le texte et quelques exemples artistiques.

1.2 Structure

La structure est organisée en trois sections principales. La première section présente toutes les informations techniques nécessaires à la compréhension des fonctionnalités d'un vocodeur de phase. Dans la deuxième section, on construit étape par étape un vocodeur de phase et on se focalise sur le morphing spectral. Dans la dernière section, nous proposons diverses applications artistiques comme suite des résultats de la recherche.

Plus en détail, la première partie concerne les bases du traitement spectral, tels que l'analyse de Fourier. L'objectif est de faciliter la compréhension des termes complexes du contexte mathématique, afin d'aboutir à une meilleure clarté pour les lecteurs. La transformation de Fourier va nous permettre de comprendre, comment fonctionne la transformation numérique du son. Par conséquent, l'analyse de Fourier fenêtrée sera amplement investiguée. Ces informations sont cruciales pour la compréhension du vocodeur de phase. Les maths derrière le vocodeur de phase sont relativement plus complexes, ainsi donc pour arriver à concevoir comment il fonctionne, il faut utiliser une ramifications pas à pas de la formule mathématique principale. Il n'est peut-être pas nécessaire d'entrer dans de tels détails, mais cette connaissance est offerte à d'autres musiciens passionnés de mathématiques qui veulent comprendre la nature sonore.

Après une introduction, plutôt mathématique, l'épreuve de la programmation commence. MaxMSP est l'outil clé du traitement du son en temps réel. Présenté dans Max, un guide étape par étape pour la construction d'un vocodeur de phase et l'analyse mathématique d'une analyse de Fourier, nous allons pénétrer le domaine du morphing spectral. La magie de voir comment une formule mathématique prend vie est l'essence même de la programmation, on peut le constater dès lors qu'on travaille avec des données sonores. On peut entendre le résultat plusieurs fois tout en faisant des modifications menant à une compréhension plus profonde de ses fonctionnalités. Le morphing spectral dans cette étude est constitué de deux vocodeurs de phase simultanés travaillant sur des entrées différentes.

Il est raisonnable à ce stade de poser la question évidente : est-il nécessaire que quelqu'un soit musicien pour réaliser tout ça ? La réponse sera sûrement négative. Mais, c'est la raison que ça soit la section clé de ce dissertation. La combinaison de compétences en programmation, de compréhension mathématique et de créativité musicale est ce qui décrit de nombreux compositeurs contemporains. Et dans ce chapitre, on peut chercher quelques exemples de modèles d'implémentations créatives du vocodeur Phase. Cette section, donc, sera une investigation sur l'extension des possibilités artistiques d'un vocodeur de phase.

1.3 MaxMSP et Jitter

MaxMSP est un logiciel créé à l'origine par Michael Puckette dans l'IRCAM puis acheté par l'entreprise américaine, Cycling74. Le noyau de ce logiciel est construit sur C ++ et Javascript, traduit dans un environnement d'utilisateur graphique tel que *box connecting*.

MaxMSP est essentiellement un langage de programmation, avec chaque rappelle de fonction étant une boîte-code correspondant à une certaine action. Les commandes sont séparées en trois catégories : les commandes d'expression logique, les commandes de traitement sonore et les commandes de traitement de matrice multidimensionnelle, chacune correspondant respectivement à Max, MSP et Jitter. L'utilisateur peut connecter diverses commandes des trois catégories pour créer un patch complexe. La version la plus récente de MaxMSP and Jitter est Max8, publiée en septembre 2018¹.

Dans la catégorie du traitement sonore, une bibliothèque spéciale appelée traitement spectral offre à l'utilisateur tous les outils nécessaires pour effectuer une analyse spectrale. La méthode utilisée par MaxMSP est appelée transformation rapide de Fourier (FFT). On peut imaginer le signal comme une ligne courbée continue constituée d'une série de points. L'idée est de convertir un signal qui est une forme à une dimension en une forme à deux dimensions qui contient essentiellement plus d'informations. On peut couper un fragment de cette ligne du signal et attribuer une valeur à chacun de ses points qui correspondrait à un vecteur dans le plan cartésien. À partir de cette interprétation, il est possible d'extraire des informations de la fréquence et de l'amplitude d'un son.

Outre l'efficacité du traitement spectral et de la perceptivité visuelle du logiciel. MaxMSP est très populaire dans la communauté artistique. Il est utilisé dans une multitude de projets et de groupes de recherche, tels que l'IRCAM, Paris8, etc. La vaste communauté de Max le rend accessible même aux utilisateurs amateurs. Dernier point, mais non des moindres, une série de bibliothèques techniques sur les vocodeurs de phase et le traitement spectral pour Max est développée par la communauté, projetant ainsi le rôle principal de ce logiciel. Certaines de ces

1. cycling74.com, *Max8 release date*, <https://cycling74.com>

bibliothèques sont SuperVP, Max SoundBox, etc.

1.4 Analyse spectrale

L'analyse spectrale est le processus de mesure du spectre d'un certain signal. Le spectre contient des informations sur la phase et la magnitude à partir desquelles on peut extraire un diagramme appelé diagramme fréquence-magnitude. Dans un tel diagramme, on peut visualiser les fréquences et leurs amplitudes respectées d'un signal pendant une certaine période du temps.

1.4.1 Histoire

Le terme “spectral” a été introduit par Isaak Newton à la fin du 18ème siècle. La théorie de la transformation de Fourier a été formulée en 1822 par Jean Joseph Fourier. Fourier était un physicien qui avait raisonné sur les propriétés thermodynamiques des matériaux. En 1843, Georg Ohm (1789-1854) innova dans le domaine du traitement du signal en appliquant une transformée de Fourier sur des signaux. Par la suite, H. L. F. Helmholtz (1821-1894) déclara que le timbre instrumental était largement caractérisé par la série harmonique de Fourier en 1863. Dans son livre *The Sensation of Tone* décrit des outils mécaniques permettant de reproduire artificiellement le spectre sinusoïdal de divers sons².

Les premiers analyseurs de spectre numériques apparaissent dans les années 1960 à la suite de la FFT, découverte en 1965. Dans la théorie de Fourier, tout spectre complexe peut être dissous en une série de fréquences simples³. Respectivement, chaque son harmonique complexe est constitué d'une somme de simples sinusoïdes d'amplitudes variées.

La transformation de Fourier, signifiant le passage d'un signal continu à son graphe statique fréquence-magnitude, est un concept apprécié dans la plupart des domaines scientifiques. Cependant, il ne faut pas oublier que la transformation est statique. Dans le signal sonore, l'information est continue et variable. C'est l'enjeu principal de la transformation de Fourier. Par

2. Curtis Roads, *Le didacticiel d'informatique musicale*, 1996, p. 545

3. Jean Joseph Baptiste Fourier. *De la Chaleur*. Paris, 1822.

conséquent, il convient de considérer supremum de temps nécessaire pour décrire un signal. Le sens exact de la phrase précédente fera l'objet d'une discussion approfondie au chapitre suivant.

La fonctionnalité de l'analyse de Fourier sur le son ne repose pas uniquement sur la visualisation des fréquences, mais est utilisée pour divers effets tels que le décalage de hauteur variable, le changement de temps avec une hauteur variable, le traitement du timbre, etc. Ces effets ont conduit au Phase Vocoder, un concept basé sur l'analyse de Fourier d'un signal en fenêtre (FFT). Le but est d'arriver au zénith du vocodeur de phase, qui est du morphing sonore.

1.4.2 Le vocodeur de phase

Le vocodeur de phase, comme son nom l'indique, est un type de vocodeur qui utilise l'information de phase pour traiter les signaux audio. Le coeur du vocodeur de phase fonctionne sur la transformation de Fourier à court terme (STFT) telle que la FFT. C'est la transformation typique d'une représentation dans le domaine temporel d'un signal dans le domaine fréquence-magnitude.

Le vocodeur de phase a été découvert en 1966 par Flanagan⁴. Cependant, la structure standard du vocodeur de phase moderne a été établie par Griffin et Lim en 1984⁵. Un exemple d'implémentation logicielle de la transformation du signal basée sur un vocodeur de phase utilisant des moyens similaires à ceux décrits pour obtenir une transformation du signal de haute qualité est le logiciel SuperVP de l'Ircam⁶. Les vocodeurs de phase modernes, tels que SuperVP, utilisent une approche statistique pour la prédiction de signal, éliminant ainsi tout artefact produit par un traitement sonore peu orthodoxe.

4. J. L. Flanagan et R. M. Golden. *Phase Vocoder*. 1966.

5. Daniel W. Griffin et Jae S. Lim. *Signal Estimation from Modified Short-Time Fourier Transform*. 1984.

6. ??anasynt.ircam.fr

1.5 Morphing

Le morphing est le processus d'interpolation entre deux objets ou plus, tandis que le résultat hybride contient des éléments reconnaissants pour chacune de ses sources. Par cette affirmation, on peut immédiatement énoncer deux concepts : morphing sonore (unidimensionnel) et morphing visuel (multidimensionnel). Dans le son, le résultat est obtenu en combinant les spectres de chaque source avec un certain facteur. Pour les images, le résultat est obtenu en ajoutant les valeurs de pixel de chaque source d'un facteur donné. Si le morphing est assumé sur deux objets, l'un s'appelle objet source et l'autre s'appelle objet target. Objet est un terme attribué à la nature du morphing, quoi que ce soit sonore ou visuelle.

1.5.1 Morphing sonore

Le morphing sonore associe des spectres de sons qui associent les deux facteurs, fréquence et magnitude. On rappelle que dans le vocodeur de phase, la fréquence est calculée par rapport à la phase. Le processus de morphing du son peut être décrit comme une interpolation spectrale entre deux (ou plus) sources aboutissant à un son hybride morphé contenant des éléments de chaque source. On peut citer comme exemple, un morphing entre un piano et un violon qui jouent tout les deux une note à 440Hz aurait une attaque percutante du piano et une queue riche et stable correspondante au spectre du violon.

Dans le domaine du morphing sonore, il est proposé d'utiliser le concept de vocodeur de phase pour obtenir une manipulation du son satisfaisante. Le vocodeur de phase peut être utilisé en temps réel sans perte de qualité et avec un minimum de perte d'informations. Néanmoins, le processus est assez coûteux en calcul, car il nécessite un vocodeur de phase par source.

Avec une approche de un vocodeur de phase pour réaliser du morphing sonore, un minimum de deux vocodeurs de phase est requis. Également, on peut manipuler la phase et la fréquence des sources pour mieux les faire correspondre. Dans cet article, nous suivons les directives du vocodeur SuperVP mais dans une version beaucoup plus simple. Notre approche idéalise l'objet

SuperVP.morph ~ et tente d'obtenir une paramétrisation plus riche tout au long du processus de sa création.

1.5.2 Morphing visuel

D'autre part, le morphing visuel a une variété d'approches. Premièrement, il convient de séparer les techniques de morphing bidimensionnel de tridimensionnel, où le morphing bidimensionnel est utilisé entre les images et l'interpolation tridimensionnelle entre les formes. Bien sûr, le morphing peut être envisagé pour des formes de dimensions supérieures, mais le résultat est mentalement un peu plus difficile à appréhender.

À partir du morphing en 3 dimensions, on entend l'interpolation de la position des sommets. Pour visualiser l'effet, imaginons deux formes 3D, par exemple une sphère et un cube. La forme est composée soit d'une somme de points, soit d'une somme de simplices à deux dimensions. De plus, il existe d'autres méthodes qui ne sont pas si courantes et, par conséquent, elles sont omises. Les deux formes doivent avoir le même nombre de points / simplices. Le morphing entre les formes correspond au changement de la position de chaque point / simplex de la forme source vers la position de chaque point / simplex de la forme target. Bien sûr, le morphing bidimensionnel peut être envisagé sur des formes, mais il ne s'agit que d'une perception de l'orientation sur des formes 3D. La méthode est donc exactement la même.

Le morphing de l'image est plus complexe. Bien sûr, le simple moyen d'ajouter les valeurs de pixels avec un facteur de pourcentage est une méthode valide sans résultats intéressants. La solution intelligente consiste à utiliser des VAE (Variational Auto-Encoders). C'est une méthode basée sur l'apprentissage automatique utilisant les fameux réseaux génératifs adversariaux (GANs). Les GANs sont des architectures de réseaux neuronaux composées de deux réseaux l'un opposé à l'autre⁷. Ce dissertation se concentre sur le son, donc le morphing de l'image ne sera pas examiné.

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

Chapitre 2

Context théoritique

2.1 Introduction

Dans cette section, on va décrire quelques notions mathématiques de la théorie derrière le vocodeur de phase. Pour embrasser la théorie de la dissolution spectrale, il faut accepter qu'un signal complexe, signifiant autre chose qu'une simple onde sinusoïdale, puisse être représenté par une série de chevauchements de simples sinusoïdes d'amplitudes et de fréquences différentes.

Pour visualiser cette méthode, le paradigme de la synthèse additive peut être utilisé comme une construction inverse. Une méthode classique de synthèse d'un son complexe variant dans le temps consiste à combiner plusieurs formes d'ondes élémentaires. Les formes d'ondes qui se superposent à la synthèse additive sont souvent sinusoïdales. Dans certaines conditions, les sinusoïdes individuels fusionnent et le résultat est perçu comme un son riche et unique.

L'idée derrière cette méthode n'est pas nouvelle. En effet, la synthèse additive est utilisée depuis des siècles dans des instruments traditionnels tels que l'orgue.

Lorsqu'un son presque périodique est analysé, son énergie spectrale est concentrée autour de quelques fréquences discrètes (harmoniques). Ces fréquences correspondent à différents signaux sunisoïdaux appelés partiels. L'amplitude de chaque partiel n'est pas constante et sa variation dans le temps est cruciale pour la caractérisation du timbre, spécialement dans la phase

transitoire initiale d'une note (attaque). On peut toutefois penser que la fréquence de chaque composante varie lentement. La synthèse additive consiste de la somme d'oscillateurs sinusoïdaux dont l'amplitude et la fréquence varient dans le temps. Les paramètres de contrôle sont déterminés par une analyse spectrale.

2.2 Décodage mathématique

2.2.1 The Fourier Transform

L'analyse spectrale consiste essentiellement à passer du domaine temporel au domaine fréquentiel pendant une période temporelle fixée. De ce point de vue, l'analyse spectrale est une transformation de Fourier associée à un processus mathématique permettant de visualiser les fréquences qui composent une onde sonore complexe pendant une période temporelle fixée.

Il existe de nombreuses façons de calculer la transformation de Fourier discrète (DFT), dont la FFT. Bien que toutes les méthodes DFT soient converties en un même résultat, la FFT est l'une des méthodes de calcul les plus efficaces. La FFT est l'un des algorithmes les plus utilisés en traitement du signal en raison de la quantité minimale de code nécessaire à sa programmation. Néanmoins, il fait partie des algorithmes les plus difficiles du domaine DSP.

La FFT est un algorithme complexe, dans le sens où il utilise des nombres complexes pour s'exécuter. En raison de sa complexité, les détails techniques sont fréquemment omis et laissé pour un contexte peut-être pure mathématique.

Le processus de DFT est expliqué graphiquement dans la figure 2.1. Dans la colonne de gauche, on peut voir la représentation temporelle du signal. En conséquence, dans la colonne de droite, la transformation FFT peut être visualisée dans le domaine fréquentiel. N est la taille de la fenêtre en termes de DSP ou la durée totale de l'analyse en général. Comme le montre la figure 2.1, le signal est décomposé en variables à 2 axes, un réel et un imaginaire.

En 1822, Jean Joseph Fourier montra que certaines fonctions semi-périodiques peuvent égale-

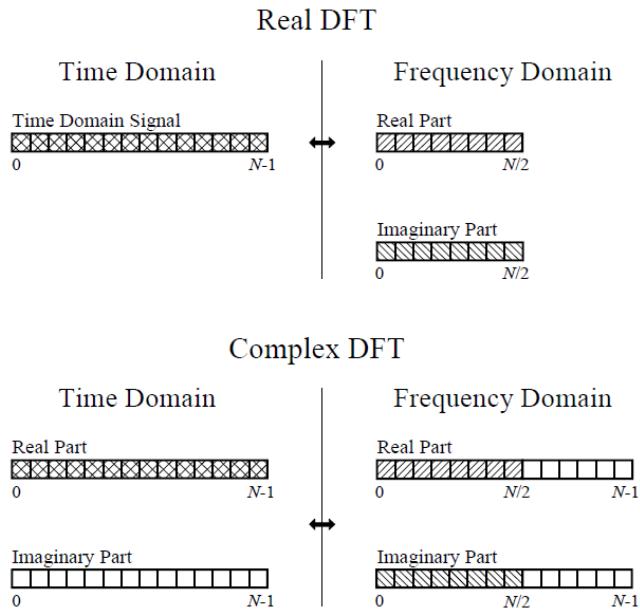


FIGURE 2.1 – Complex DFT

ment être formulées comme une somme d'harmoniques :

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

Où t est la période, x est le signal, c_0 est l'harmonique fondamentale, f la fréquence, $\omega = 2\pi f$ et θ est la phase de chaque harmonique . Dans cette formulation de la transformation de Fourier, il est clair qu'un son x lié au facteur de temps t peut être décrit comme une somme de sinusoïdes.

Pour cette raison, la transformation de Fourier pour toute fonction intégrable $f : \mathbb{R} \rightarrow \mathbb{C}$ est la suivante :

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

Où $F[x(t)](\omega)$ est la transformation de Fourier du signal x bornée par sa fréquence ω correspondante. Dans le domaine sonore numérique, cette fréquence varie entre 0 et la fréquence d'échantillonnage (SR). Le SR est généralement de 44100 ou 48000Hz. Le domaine de cette

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]

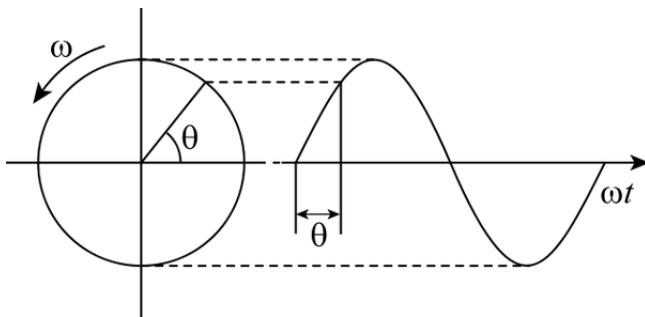


FIGURE 2.2 – Circle de la transformation Fourier

fonction pouvant être intégrée aux signaux sonores numériques varie dans l'intégrale $[-1, 1]$. Comme on le voit clairement dans cette formule, le foncteur temporel est infini, mais un tel calcul est évidemment impossible. D'autres variations de la transformation de Fourier en temps discret sont en réalité utilisées en informatique.

La transformation de Fourier est basée principalement sur la formule d'Euler où $e^{2j\pi t} = \cos(2\pi t) + j\sin(2\pi t)$. On peut imaginer cette opération comme dessiner un cercle dans le plan complexe \mathbb{C} . Au cours de la transformation de Fourier, un signal $x(t)$ est rendu autour d'un cercle avec $(e^{-j\omega t})$, avec une fréquence f . Rappelons que $\omega = 2\pi f$. La rotation est donnée par le signe de j (nombre imaginaire). Une rotation dans le sens des aiguilles d'une montre est donc considérée comme $-j$. Ce processus nous donne effectivement deux parties, la vraie, $\cos(2\pi)$, et l'imaginaire, $j \sin(2\pi t)$, partie de l'équation.

On transformerait souvent les données cartésiennes induites par l'exponentielle complexe en une forme polaire plus familière. Après la transformation de Fourier, il existe deux facteurs à manipuler, la phase et la magnitude. La phase est induite par l'angle des deux coordonnées cartésiennes dans le plan complexe tandis que la magnitude est déduite par le vecteur produit des deux valeurs. De la phase, on peut extraire des informations sur la fréquence du signal sur l'échantillon précise et, comme son nom l'indique, des informations de magnitude sur l'amplitude de l'échantillon exacte. Lorsque le temps est fixé pour l'exécution de l'analyse de Fourier, le signal sonore est divisé en intervalles de fréquence factorisés par la fréquence de l'analyse (ω). La phase et la magnitude sont calculées pour chaque fraction de temps sur laquelle l'analyse de Fourier est effectuée. La magnitude est égale à : $m(x) = \sqrt{i(x)^2 + r(x)^2}$ et la phase est $\theta(x) = \tan^{-1}\left(\frac{i(x)}{r(x)}\right)$.

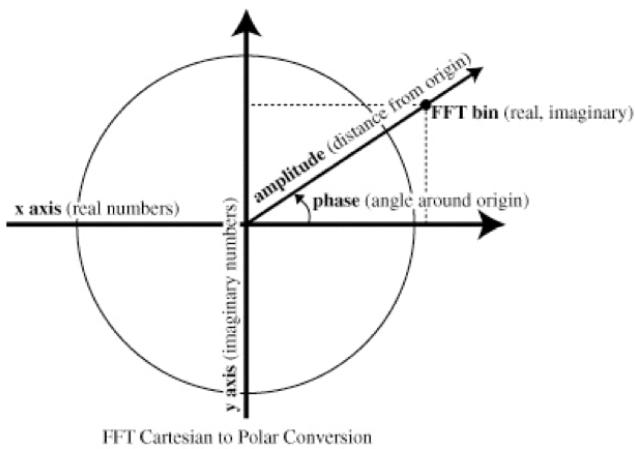


FIGURE 2.3 – Magnitude et phase

Évidemment, dans le domaine numérique, on ne peut pas utiliser une partie de temps infinie. Nous introduisons, donc, la notion de la fenêtre et, ainsi, de fenêtrage. La fenêtre est une période de temps exprimée en images. Les images sont une notion équivalente du SR en ce sens que le SR calcule une quantité d'images du son par minute et que les images sont exactement ces images. Une analyse fenêtrée est généralement exprimée par un algorithme de transformation à court terme de Fourier (STFT)³.

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

Où x est le signal, X sa transformation Fourier (une abréviation de la forme $F[x(t)]$), $\omega = 2\pi f$, t le temps continu, τ l'instant temporel, c_n les 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 :

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

3. Jown Strawn, *STFT : Short Time Fourier Transform*, 1985, pp. 141– 134.

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

5. Jean-François Charles. A tutorial on spectral sound processing using maxmsp and jitter. Computer Music

Où N est la taille de la fenêtre, qui correspond au nombre total d'images qu'il contient. X est le signal, n énumère chaque image dans N et $\omega = 2\pi f$ comme d'habitude mais dans cette version, f varie entre 0 et N . Nous le présenterons plutôt comme $\omega = 2\pi k$ où k représentera la k -ième harmonique.

Pour chaque transition vers le domaine fréquentiel, une fonction inverse du domaine temporel est nécessaire. La fonction inverse se caractérise par la transformation de Fourier rapide inverse (IFFT) pour des valeurs de temps discrets.

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

Transformation Fourier rapide (FFT)

C'est toujours bien d'avoir un point de base théorique, mais ce qui compte vraiment, ce sont les formules computationnelles. Par computation, nous entendons une formule qui est facile à exécuter par un ordinateur en temps raisonnable et à produire une sortie. Une solution à cette affirmation est la transformation de Fourier rapide. Ainsi, FFT est un algorithme calculable pour DFT qui utilise un moyen intelligent pour réduire le temps de calcul et la complexité des calculs.

Nous rappelons la DFT standard interprétée dans du code informatique.

Journal, pages 87–102, Printemps 2008.

6. Alan V. Oppenheim et Ronald W. Schafer. Discrete-time signal processingd. Prentice Hall Press.

```

1   function DFT(x)
2       N = length(x)
3       # We want two vectors here for real space (n) and frequency space (k)
4       n = 0:N-1
5       k = n'
6       transform_matrix = exp.(-2im*pi*n*k/N)
7       return transform_matrix*x
8   end.
9

```

Listing 2.1 – DFT

Il est clair que la transformation de Fourier est en réalité une matrice de calcul. Cependant, en effectuant autant de calculs, on peut imaginer que le processus est assez cher computationnellement. Les conditions requises pour la DFT sont le traitement en temps réel et des fenêtres d'une taille minimale de 512 échantillons pour les données sonores. L'amélioration de l'algorithme a été donnée par James Cooley et John Tukey⁷

L'algorithme de Cooley-Tukey utilise la récursivité pour réduire la complexité de calcul. En particulier, la matrice produite par la réduction dans le domaine fréquentiel est réduite en deux parties avant d'effectuer les calculs DFT. Nous séparons les indices impairs des casiers du même, puis la procédure est répétée. Ce processus réduit la complexité à $\mathcal{O}(n \log n)$ à partir d'une taille polynomiale. Bien sûr, pour effectuer cette action en raison de la division continue par deux, nous demandons que la fenêtre d'analyse soit une puissance de deux.

Le diagrammes au forme de “Butterfly” est l'idée principale de la réduction du calcul FFT. En particulier, Cooley et Tukey ont remarqué que les exponentielles complexes sont entièrement répétées dans la seconde moitié de la fenêtre avec un signe opposé. Par conséquent, en divisant constamment la vidéo, les calculs de l'exponentielle complexe sont considérablement réduits. Nous pouvons visualiser le processus dans la figure 2.4⁸.

7. James Cooley et John Tukey, *Un algorithme pour le calcul automatique de séries de Fourier complexes*, 1965.

8. Image récupérée de l'article : P. G. Reshma, P. Gopi Varun, Babu V. Suresh, Wahid Khabou, *Analog implementation of FFT using cascade current mirror*, 2017.

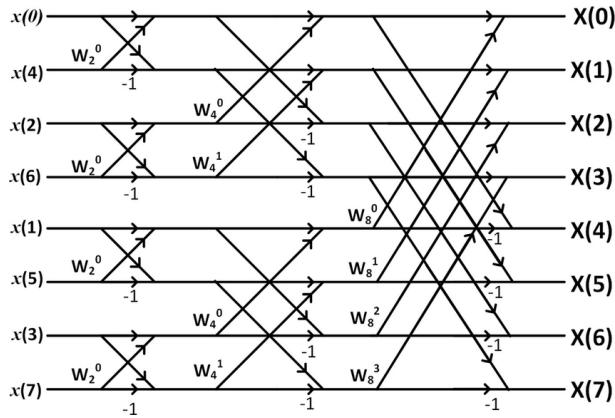


FIGURE 2.4 – Butterfly Diagrams for an 8 point FFT

Une mise en oeuvre du code correspondant est en disposition dans l'appendix .1.

2.2.2 Fenetrage

Pour calculer le STFT, il faut définir la taille de la fenêtre à traiter. La transformation de Fourier dans une fenêtre temporelle discrète peut produire des artefacts contredisant le continuum du signal. Pour éviter cet effet, il est habituel de multiplier la fenêtre du son d'origine par une fenêtre, qui ne contient pas des informations sonores, de la même taille sur laquelle une fonction factorise l'amplitude du son. La transformation de Fourier est reproduite un nombre dénombrable de fois en fonction de la taille du son analysé. La fenêtre est généralement beaucoup plus petite que le son d'origine, ce qui la répète plusieurs fois pendant toute la durée du son. Il est important que la fenêtre soit déplacée dans le temps, mais elle est également se chevaucher par un facteur. Le processus peut être décrit visuellement dans la figure 2.5.

Cette technique, appelée chevauchement où superposition, donne un résultat sonore plus arrondi. En raison du chevauchement et de la multiplication enveloppe, l'auditeur ne peut percevoir aucun des artefacts possibles induits par l'analyse mais on entend un résultat unifié⁹.

9. Daniel W. Griffin et Jae S. Lim. Estimation du signal à partir de la transformation de Fourier à court terme modifiée. IEE Transaction sur l'acoustique, la parole et le traitement du signal, ISSE Vol. 32 : 236-243, avril 1984.

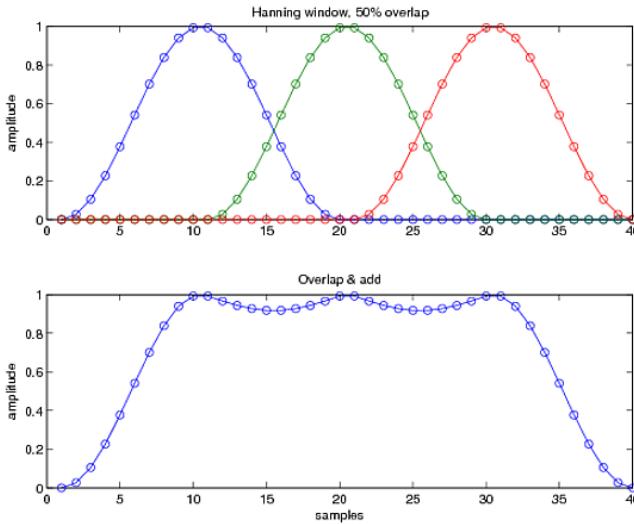


FIGURE 2.5 – Overlapping

Filtres Gabor

Ici, nous examinerons la possibilité d'utiliser des filtres de Gabor pour appliquer un enveloppe au signal sonore. Les filtres de Gabor sont définis comme la multiplication continue d'une fonction gaussienne par un signal complexe bornée par un facteur temporel. Le filtre gaussien est décrit dans la formule suivante :

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

Il est possible de considérer un filtre de gabor comme deux filtres déphasés se produisant en décomposant le complexe exponentiel par rapport à une partie imaginaire et réelle. Où la partie réelle a le filtre $g_{real}(t) = w(t)pch(2\pi f_k t + \theta)$ et la partie imaginaire $g_{ima}(t) = w(t)\cos(2\pi f_0 t + \theta)$.

Les deux filtres peuvent être déphasés mais la phase de prise en compte. Par conséquent, leur effet sur une sinusoïde est toujours une sinusoïde.

Pour manipuler la courbe d'un filtre de Gabor, il suffit de changer les paramètres. La formule normalisée se transforme en :

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

Where σ is the parameter of the Gaussian curve. A Gabor filter will allow to create a smoother sound result after Où σ est le paramètre de la courbe gaussienne. Un filtre de Gabor permettra de créer un résultat sonore plus lisse après l'analyse.

La même logique est valable pour tous les types d'un enveloppe de fenêtre tels que le hanning, le hamming, etc.

2.2.3 Le vocodeur de phase

Définition

Le vocodeur de phase est un outil d'analyse-synthèse qui utilise la DFT. L'analyse est effectuée de sorte que le signal de sortie ne subisse aucune perte de données de la transformation, que ce soit théoriquement ou concrètement. Le signal de sortie est identique à l'entrée avant l'analyse si aucun traitement n'est appliqué. Les utilisations les plus courantes du vocodeur de phase sont le décalage de hauteur de ton et l'alternance de la vitesse de lecture. Le vocodeur de phase n'a pas de restriction évidente et peut également garder une trace des inharmonicités et du vibrato des sons¹⁰.

Histoire

Le vocodeur de phase a été introduit, en 1966, par Flanagan en tant qu'un algorithme préservant la cohérence horizontale entre les phases des segments représentant les composants sinusoïdaux. Le vocodeur de phase n'a pas pris en compte la cohérence verticale produite par les intervalles de fréquences adjacentes et, par conséquent, l'étirement temporel du système a produit des signaux sonores avec une perte de qualité. La reconstruction optimale du signal de la STFT après traitement a été proposée par Griffin et Lim¹¹ en 1984. Cet algorithme ne considérait pas produire un STFT cohérent, mais plutôt rechercher le signal optimal dont le STFT est

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

11. Daniel W. Griffin et Jae S. Lim, *Signal estimation from modified short-time fourier transform*, 198, pp. 236–243.

aussi proche que possible du STFT modifié, même si le STFT modifié n'est pas cohérent (ne représente aucun signal).

Le problème de la cohérence verticale était un problème majeur pour la qualité opérationnelle à l'échelle temporelle jusqu'en 1999. A cette époque, Laroche et Dolson¹² ont proposé un moyen de préserver la cohérence phase des composantes spectrales. Le développement de Laroche et de Dolson doit marquer un tournant dans l'histoire du vocodeur de phase. Il a été prouvé que, grâce à la cohérence de phase, une transformation temporelle de haute qualité peut être obtenue.

Néanmoins, l'algorithme découvert par Laroche et Dolson n'a pas pu conserver la phase verticale à l'attaque d'un son. Roebel a proposé une solution à ce problème¹³ en 1999. Un exemple mis en oeuvre du vocodeur de phase utilisant la dernière version de Roebel est constitué par les outils SuperVP de l'Ircam.

Une approche à la description du vocodeur de phase consiste à représenter le signal une séquence des images successives d'une transformation de Fourier Discrète (TFD) d'une fenêtre de longueur N . Ces images sont d'abord multipliées par une fenêtrage appropriée (telle que Hamming, Hanning, Kaiser, Blackman, etc.) puis transformés dans le domaine fréquentiel. A ce stade, toute modification prudente du spectre peut être faite, avant la transformation inverse au domaine temporel avec la transformation 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 caractérisée des successions des TFD) d'un signal fenêtré est défini comme suit :

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

12. Mark Dolson, *The phase vocoder : a tutorial*, 1986, pp. 14-27.

13. Axel Roebel, *Morphing sound attractors*, 1999.

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

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

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

Le terme $\tilde{x}_u(n)$ propose l'estimation des fenêtres symétriques calculées. Donc si nous supposons un chevauchement du fenêtrage par 50 %, ou bien $N/2$, on propose une manière d'éviter les asymétries de phase ci-dessus :

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

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

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

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

ou $((.))_N$ présente l'opération du modulo, et N est la durée de la fenêtre et il devait être un pair.

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

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

C'est une fonction de deux variables discrètes, le temps rl et la fréquence k . L'indice ir est la position de la fenêtre, r étant le numéro de l'échantillon et l la taille du pas de chevauchement de la fenêtre d'analyse. $S(rl, k)$ peut être vu comme le spectre de la multiplication $s(m)w(rl - m)$, qui est la séquence en entrée s (m) multipliée par la fenêtre décalée à la position rl . Ce n'est pas le spectre exact, mais sa convolution avec la transformation de Fourier des fenêtres correspondantes.

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.

La procédure standard de chevauchement (*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 :

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

Pour construire une 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 :

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

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

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

2.2.4 Applications du vocodeur de phase

Time Stretching

Pour réaliser un étirement du temps d'un son, traditionnellement, il suffit de baisser ou augmenter le rythme de sa lecture, mais cela produit également un 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 période de temps de la transformation Fourier appliquée, une série 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 modèle qui correspond à l'étirement temporelle est donnée par la formule suivante :

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

$$\theta_k(n) = \theta_k(0) + \int_0^n f_k(\tau) d\tau \quad (2.17)$$

Où un signal est interprété par une somme des sinusoïdes $K(n)$ avec leur magnitude $A_k(n)$ et phase θ_k appropriées. Par la suite, la phase instantanée du k -ième sinusoïde, $\theta_k(n)$, est calculée par l'addition de la phase de la première échantillon $\theta_k(0)$ avec l'intégrale des fréquences $f_k(n)$. La dernière est équivalente à $\sum_{n=0}^N f_k(n)$ pour une transformation discrète.

La modification temporelle est déterminée par deux paramètres parallèles, c'est à dire la modification de phase instantanée pour chaque échantillon et la modification du fenêtre du décalage (*hop window*). Plus précisément, le *hop window* du stade de l'analyse est différent du *hop window* de la synthèse.

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

étirement temporel devient¹⁴ :

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

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.¹⁵

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 rythme de la lecture de notre vocodeur de phase au zéro. Cela peut se comparer à un étirement 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 échantillon à zéro. Cet effet résulte à un son robotisé et métallique.

14. Jean Laroche et Mark Dolson, *Improved Phase Vocoder*, 1999, p. 324

15. Mark Dolson, *The phase vocoder*, 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 FFT¹⁶.

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.

$$M(\alpha, t) = \alpha(t)\widehat{S}_1 + [1 - \alpha(t)]\widehat{S}_2 \quad (2.19)$$

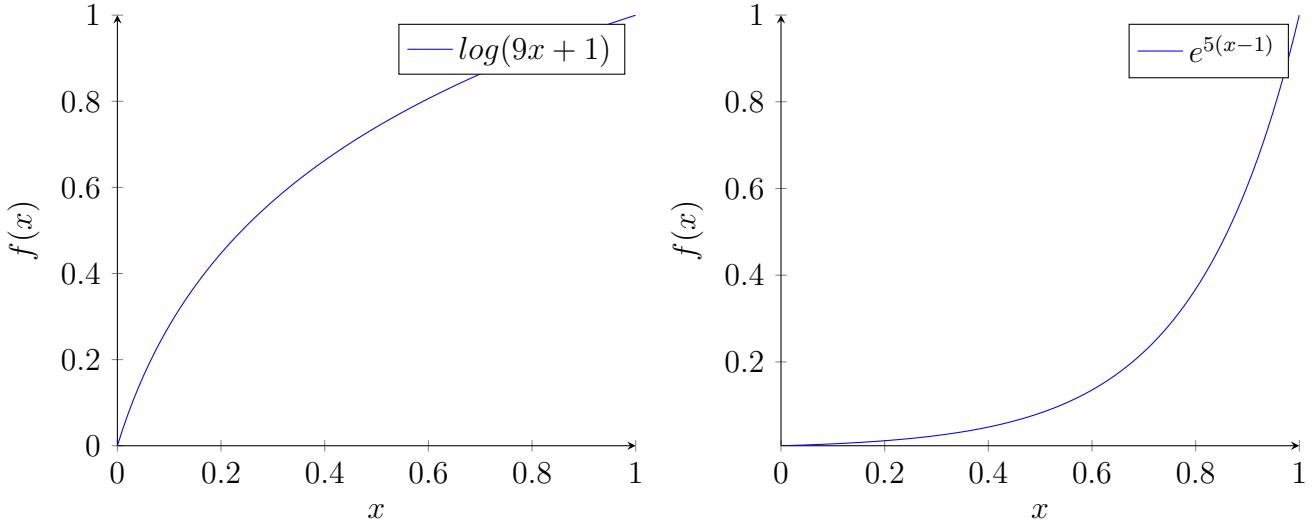
Ou $\alpha(t) \in [0 : 1]$

La manipulation du paramètre α nous permettra de changer la dynamique du morphing. Bien évidemment, on peut s'attendre d'un autre fonction que d'une manipulation linéaire. 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.

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

17. Axel Roebel, *Morphing sound attractors*, 1999.

FIGURE 2.6 – Interpolation du facteur α 

Noise modeling

Le Vocodeur de Phase tel qu'il est présenté jusqu'à présent est un modèle fin mais il ne s'agit toutefois pas du modèle complet. Le modèle présenté ici est celui de Xavier Serra¹⁸. Dans ce modèle, les composantes sonores sont idéalisées comme déterministes. Cela suggère que chaque composant sonore est une sinusoïde ou une sinusoïde à variation lente. La partie non déterministe implique que le son est modélisé avec une composante de bruit supplémentaire, comme le montre la formule ci-dessous.

$$^{19}s(t) = \sum_{n=0}^N A_n(t) \cos(\theta_n(t)) + \epsilon(t) \quad (2.20)$$

Où A_n et θ_n sont respectivement l'amplitude et la phase de la n-ième fréquence. En supposant que $\epsilon(t)$ soit un composant stochastique, il peut être considéré comme un bruit blanc filtré.

$$\epsilon(t) = \int_0^t h(t, \tau) u(\tau) d\tau \quad (2.21)$$

Où $u(\tau)$ est un bruit blanc et $h(t, \tau)$ nous la réponse d'un filtre variant dans le temps qui affiche une valeur au temps t .

18. Curtis Roads et autres, *Traitemet du signal musical*, 1997, p. 91 - 122

19. Curtis Roads, *Musical Signal Processing*, 1997, p. 94

Stochastic Modeling

Certaines autres approches suggèrent une approche stochastique simultanée au vocodeur de phase traditionnel. Ce principe repose également sur l'hypothèse que le son est composé de composants semi-sinusoidaux. De manière à ce qu'une périodicité soit préservée. Sur la base de la première analyse, nous supposons une périodicité de signaux et lors de chaque nouvelle analyse, la fréquence de sa corbeille est calculée en fonction d'un facteur d'erreur.

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

Où Δf_n est la différence entre un pic mesuré et le plus proche mesuré. f_n est la fréquence de la corbeille et a_n est la magnitude des pics prédicts. A_{max} est la magnitude maximale enregistrée.

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

Maintenant, Δf_k est la différence entre un pic mesuré et son pic prédict le plus proche. f_n est la fréquence de la corbeille et a_n est la magnitude des pics enregistrés.

L'erreur totale est :

$$^{20} Err_{total} = Err_{p \rightarrow q}/N + Err_{q \rightarrow p}/K \quad (2.22)$$

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

Chapitre 3

Design

Dans ce chapitre, nous allons implémenter les formules mathématiques présentées dans le chapitre précédent dans la langue du logiciel MaxMSP. De plus, nous allons structurer d'autres effets et variations sur le contexte du vocodeur de phase.

On va présenter, à la suite, les objets MaxMSP qui encapsulent¹ des fonctions mathématiques telles que FFT, IFFT et d'autres transformations. L'objectif de ce partie est de permettre de comprendre la plupart des outils spectraux que MaxMSP offre à l'utilisateur. Parallèlement, on va construire une série d'outils efficaces pour structurer un simple vocodeur de phase.

3.1 MaxMSP Objects

Dans cette section, nous allons mentionner quelques-uns des objets spectraux de base de MaxMSP fournis directement par la librairie standard, et nous analyserons leurs fonctionnalités.

Dans MaxMSP, il existe 3 catégories d'objets :

1. Les objets logiques utilisés pour les expressions et les calculs logiques ;

1. encapsuler : créer un sub-patch qui contient un ensemble d'objets / fonctions

2. Les objets signal, pour le traitement du signal, suivis généralement par une indication ~ après leur nom ;
3. Les objets Jitter qui sont utilisés pour les données multidimensionnelles, tels que images, formes 3D, etc.

La manière dont un objet spectral fonctionne dans MaxMSP est quelque peu différente de celle des autres objets signal. Ils sont déployés dans un environnement spécial appelé *PFFT* ~. Dans cet environnement, comme son nom l'indique, une FFT est effectuée et, comme les objets traitent des données bidimensionnelles, ils sont traités dans un patch différent.².

FFT ~

Les objets *FFT* ~ appartiennent à la famille de signaux, comme l'assume l'indicateur ~. C'est l'objet qui effectue la transformation rapide de Fourier. Il n'a pas d'entrées mais il y trois sorties, une pour la partie réelle de l'exponentielle complexe, une pour la partie imaginaire et un compteur qui garde l'index des corbeilles de fréquence.

Dans la transformation de Fourier fenêtrée habituelle :

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j \frac{2\pi k n}{N}} \quad (3.1)$$

Où k est l'indice de la corbeille et les coordonnées complexes correspondantes sont $\sum_{n=0}^k x_n + \cos(2\pi t)$ pour la partie réelle et $\sum_{n=0}^k x_n + j \sin(2\pi t)$ pour l'imaginaire. Bien entendu, la réduction correspondante de la ramifications calculée est appliquée par l'algorithme FFT présenté au chapitre précédent.

IFFT ~

De manière similaire, la FFT inverse est traitée par la fonction d'objet appropriée avec les entrées et les sorties inverses.

2. Dans MaxMSP, un fichier de code est appelé un patch entre utilisateurs

cartopol ~

Il n'est généralement pas utile d'utiliser ces coordonnées cartésiennes, produites par le *FFT* ~ correspondant aux parties réelle et imaginaire du plan \mathbb{C}^2 . Par conséquent, un objet transformant les coordonnées cartésiennes en une forme polaire appelée *cartopol* ~ est fréquemment déployé. Cet objet prend bien sûr deux entrées (réelle et imaginaire) et génère une phase et une magnitude pour chaque index.

poltocar ~

Exactement la fonction inverse de *cartopol* ~. L'objet *poltocar* ~ transforme des coordonnées polaires (deux entrées) aux coordonnées Cartésiennes (deux sorties).

FFTinfo ~

FFTinfo ~ est très fréquemment utilisé et fournit des informations sur les paramètres de la FFT, tels que la taille de la fenêtre, etc.

framedelta ~

Un objet très important car il calcule la dérivation de phase entre des images successives de la FFT.

gen ~

Un objet qui crée une nouvelle fenêtre d'un patch. Dans l'objet *gen* ~, l'utilisateur peut utiliser un nouvel ensemble de fonctions spécialisées dans le traitement d'un seul échantillon. Les fonctions sont relativement plus simples que l'environnement de codage habituel, mais on peut effectuer un travail plus précis sur les détails. L'environnement *gen* ~ est fréquemment utilisé pour effectuer un délai à l'échelle d'échantillons en créant un filtre passe-bas ou des effets similaires

3.2 Pitch tracking

En utilisant certains de ces objets spectraux MaxMSP, nous allons commencer par construire un simple patch pour trouver la hauteur de la fréquence dominante :

'Les méthodes de détection du pitch dans le domaine fréquentiel dissèquent le signal d'entrée en fréquences constituant le spectre global. Le spectre montre la force des différentes composantes des fréquences contenues dans le signal. Le but est d'isoler la fréquence dominante, ou "pitch", hors du spectre'.³

De même, nous avons implémenté un patch MaxMSP pour trouver la fréquence dominante de la fenêtre FFT. Le patch peut être trouvé dans la figure 3.1.

Bien entendu, nous utilisons l'objet *fftin* ~ pour transformer le son d'entrée dans le domaine fréquentiel. Dans la FFT, nous devons déclarer deux choses. Tout d'abord, le nombre d'images faisant l'objet d'une opposition à la FFT et, deuxièmement, la taille de la fenêtre. Nous rappelons que les deux premières sorties donnent respectivement les composantes réelles et imaginaires. La troisième sortie donne le corbeille de la fréquence indexée prenant les valeurs correspondantes de 0 à N , où N est la taille de la fenêtre FFT.

Pour continuer le patch du simple suivi de la hauteur, nous normalisons les composants réels et imaginaires pour restreindre le flux de données dans des limites calculables. De manière plus détaillée, *fft.normalize* ~ est un simple Max externe, créé en code C ++ à l'aide du Max SDK, qui divise le nombre de chaque sortie par la moitié de la taille de la fenêtre.

Ensuite, nous transformons les coordonnées cartésiennes en coordonnées polaires en utilisant *cartopol* ~. Le processus est expliqué à la section 2 sous les termes phase et magnitude. Nous ne sommes intéressés que par la phase de chaque bin. En utilisant la phase, nous pouvons calculer la fréquence exacte de chaque harmonique. Nous rappelons que les termes harmonique, bin et fréquence de Fourier sont les mêmes dans ce contexte. Premièrement, nous retardons la phase de chaque bin d'une fenêtre entière pour calculer le montant de sa modification. Ensuite,

3. Curtis Roads, *A tutorial in musical signal processing*, 1996, p. 513, op. cit,

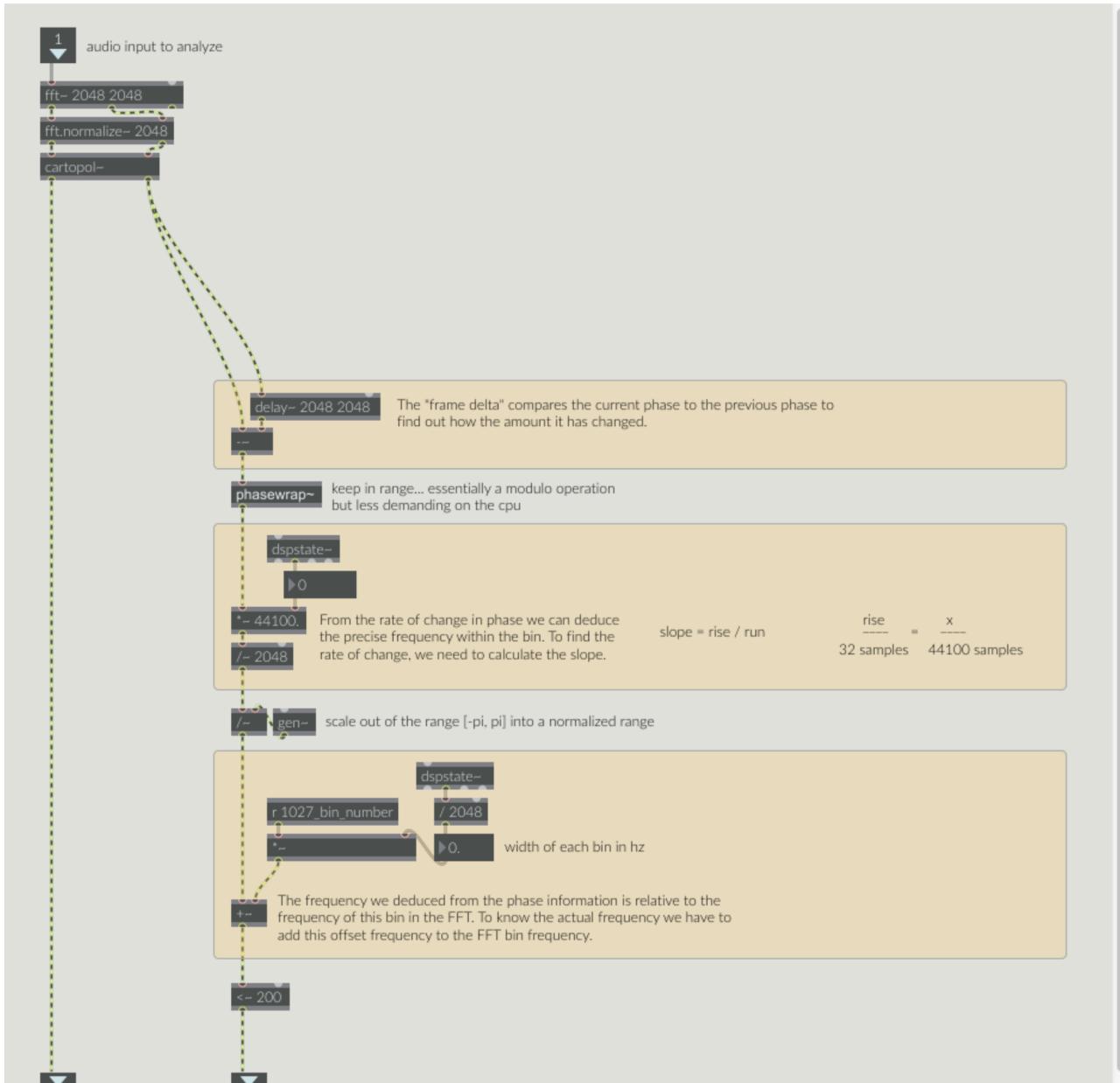


FIGURE 3.1 – Pitch Tracking

nous soustrayons la phase en cours de chaque bin par la phase correspondante de la fenêtre précédente. Dans la suite, nous utilisons l'objet *phasewrap* ~ pour découper la valeur comprise entre $-\pi$ et π . Il s'agit essentiellement d'une fonction modulo qui permet de conserver les valeurs dans un intégral computable.

À ce stade, nous devons conseiller nos paramètres sonores. Un objet dans Max appelé *dspstate* ~ récupère toutes les données nécessaires que nous pouvons trouver dans */Options/Audiostatus* dans l'application Max. Nous avons seulement besoin de la fréquence d'échantillonnage. Cette valeur est généralement 44100Hz donc nous avons déjà stocké la valeur et si différente nous la changeons automatiquement. Le signal obtenu à partir de *phasewrap* ~ est multiplié par la fréquence d'échantillonnage et par conséquent divisé par la taille de la fenêtre. Ces opérations, qui divisent le SR par la taille de la fenêtre, déduisent des partitions pondérées du spectre. La valeur de la soustraction de fréquence classe l'écart de la partition de fréquence. Ensuite, nous normalisons en multipliant par 2π stocké dans un objet *gen* ~, pour manipuler la précision sur le nombre π . Nous trouvons la position de la partition en multipliant la valeur de la division du spectre par l'indice de l'harmonique correspondante et en l'ajoutant à la déviation pour obtenir la fréquence de cette harmonique.

La division du spectre ou la partition est donnée par :

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

Ensuite, la fréquence est simplement : *index * x + la déviation de phase*.

Cette procédure génère toutes les fréquences de toutes les cases de la fenêtre FFT.

L'étape suivante consiste à conserver la fréquence avec la magnitude la plus forte. Nous allons utiliser un objet *gen* ~. Nous déclarons d'abord les entrées. Nous n'avons besoin que de trois variables. La magnitude de chaque bin, l'index de chaque bin et la taille de la fenêtre FFT. Nous déclarons une seule sortie pour le bin dominant de chaque fenêtre. Le patch est montré dans la figure ??.

Pour calculer le bin le plus fort, nous allons utiliser un objet appelé *codebox*. *Codebox* est un objet qui permet à l'utilisateur de saisir du code sous la forme "traditionnelle". Tout comme le code javascript, nous devons déclarer les variables que nous allons utiliser mais d'abord les entrées de l'objet. Les variables dans ce cas sont : magnitude, index, frameSize et last. Où frameSize est la taille de la fenêtre FFT et last est juste une variable qui stocke l'index du bin le plus fort pour la fenêtre précédente.

Le processus de définition du bin le plus fort est mis en œuvre par deux fonctions *if* imbriquées. Ce processus sera répété pour chaque fenêtre et il déterminera éventuellement les fréquences des cellules les plus puissantes dans toutes les fenêtres. Dans la boucle *if* imbriquée, nous déterminons si l'index actuel est plus fort que le précédent et nous retournons le résultat. Cela signifie que si la magnitude de l'index $n - \text{th}$ est supérieure à celle du bin $(n - 1) - \text{th}$, nous stockons sa valeur d'index. La dernière étape consiste à limiter le nombre d'index dans la limite de la taille de la fenêtre afin de terminer la boucle et de réinitialiser les variables.

Pour localiser l'harmonique le plus fort en précisant la fréquence, quelques fonctions supplémentaires sont nécessaires. La sortie de l'objet *gen ~* est filtrée par un objet *sah ~*. Avant cela, l'index actuel est comparé à la valeur d'index générée par l'objet *gen ~*. *Sah ~* signifie "échantillonneur bloqueur". Cette fonction filtre la première entrée de l'objet par un facteur similaire à l'objet *gate ~*. Chaque fois que la valeur du facteur change, elle permet à la première entrée d'aller de l'avant et de sortir sa valeur. Nous l'utilisons deux fois, une fois pour la magnitude et une fois pour la phase. De cette façon, seul l'indice de la harmonique avec la plus grande magnitude va de pair avec la magnitude et la phase correspondantes. Nous utilisons le système fourni précédemment pour trouver la fréquence exacte et nous traduisons les coordonnées polaires de l'amplitude en dB. Enfin, nous utilisons un objet *slide ~* pour lisser le résultat. Le patch final est montré dans la figure 3.2. Cette méthode évite de calculer chaque fois la fréquence si le son est périodique et donc la fréquence dominante ne change pas.

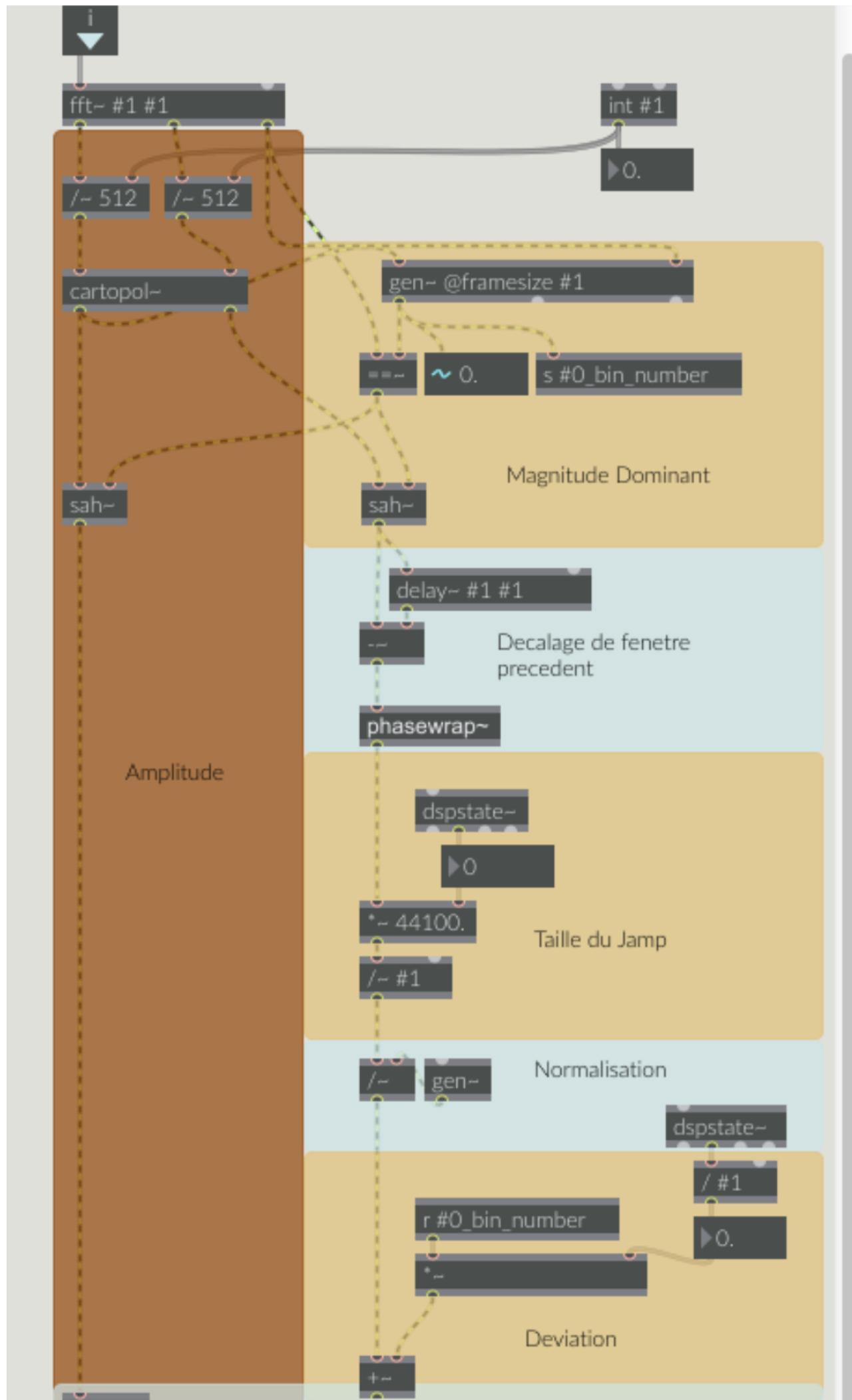




FIGURE 3.3 – Gen multiple

3.2.1 Multiple Pitch Tracking

Pour créer un patch de suivi des hauteurs multiples, nous utilisons la méthode de la localisation de la hauteur simple unique avec quelques modifications. La majorité des calculs ont lieu dans l'objet *gen ~* car l'implémentation de *codebox* est copiée en fonction du nombre de hauteurs que l'on souhaite calculer.

La figure 3.3 permet de visualiser comment on peut calculer une série des harmoniques les plus fortes de chaque fenêtre. L'objectif est de fournir une quantité suffisante d'informations sur les composantes spectrales de base du son.

Comme le suggère la figure 3.3, la sortie de chaque *codebox* correspond à l'entrée de la suivante, calculant ainsi le prochain indice le plus fort de la fenêtre. La quantité d'objets *codebox* utilisée correspond au nombre de pas que nous allons afficher.

La mise en oeuvre de la méthode de la localisation des plusieurs hauteurs peut être montrée à la figure ???. Nous reproduisons essentiellement le calcul de fréquence pour chaque indice. Le résultat peut être affiché sous forme de liste ou dans différents points de vente. Ensuite, on peut utiliser l'objet *mc.cycle ~* dans la version Max8, de simples oscillateurs ou résonateurs dans les

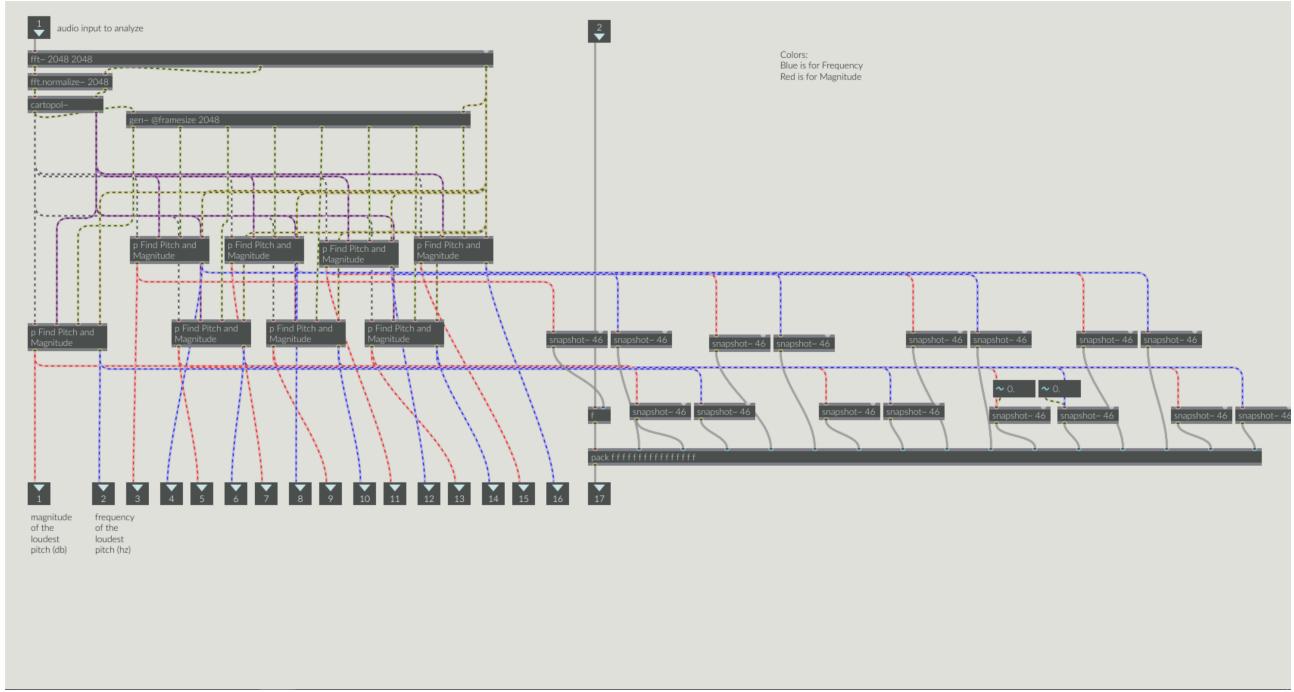


FIGURE 3.4 – Multiple Pitch tracking

autres cas.

Un avis important pour cette construction de patch, mais aussi pour chaque suivi de hauteur via les méthodes spectrales, est que la taille de la fenêtre affecte le suivi de fréquence. La taille de la fenêtre influence la résolution temporelle ou fréquentielle de la représentation du signal. La résolution en fréquence peut être augmentée en modifiant la taille de la FFT, c'est-à-dire le nombre de segments de la fenêtre d'analyse. En particulier, la taille des bins correspond simplement à la moitié de la fenêtre d'analyse. La plage de fréquences est définie comme la division du SR en taille de fenêtre⁴.

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

3.3 Le vocodeur de phase

Le vocodeur de phase était utilisé à l'origine pour la transposition de la hauteur et la modification de la vitesse de lecture. De même, nous allons construire notre vocodeur de phase sur

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

le modèle de base, puis nous proposerons quelques modifications. Le vocodeur de phase est étiqueté sous traitement spectral et va donc être stocké dans un objet $pfft \sim$.

Pour accéder à un son directement dans l'objet $pfft$, il faut éviter d'utiliser les prérglages du $pfft \sim$. Par conséquent, nous allons éviter d'utiliser l'objet $fftin \sim$ à la place, nous allons à un objet $fft \sim$ dans un sous-patch. La seule différence est que les paramètres des objets $fftin \sim$ et $fftout \sim$ sont contrôlés directement à partir de l'objet $pfft \sim$ et que les entrées et sorties sont les premier et dernier objets. Pour construire un vocodeur de phase, nous devons traiter le son avant d'appliquer la transformation de Fourier. L'un devrait suivre le patch fourni dans la figure 3.5 pour comprendre la procédure.

Le raisonnement derrière cette logique se repose sur le coeur du vocodeur de phase pour l'éti-rement 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 consécutives 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 fréquence 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* ~.

Cela peut être accompli en spécifiant un déphasage par rapport aux objets *fftin* ~ et *fftout* ~. 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* ~ ne soit pas voulu, l'objet *fftout* ~ peut pratiquement être utilisé comme sortie pour l'objet *pfft* ~. Le fenêtrage automatique dans l'objet *fftout* ~ devrait se comporter comme le fenêtrage manuel avec les objets *fft* ~.

Dans cette version du vocodeur de phase on va utiliser un son pré-déterminé. Ca veut dire que on utilisera un enregistrement et on le stora dans un objet *buffer* ~. Le *buffer* ~ 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* ~. C'est possible d'utiliser soit l'objet *index* ~ ou l'objet *play* ~⁵ pour accéder au *buffer* ~. Lorce que on précise la position de la transformation Fourier en durée courte manuellement, on doit trouver un système pour préciser le décalage de la fenêtre dans notre *buffer* ~.

La solution dans cette problématique et d'utiliser deux objets *fft* ~ parallels avec un décalage d'un quart, pour un chevauchement de 25%. Un tel système permettra de calculer l'image actuelle de chaque index du fenêtre en comparaison de l'index correspondant de la fenêtre précédent comme était fait dans le patch du pitch tracking. Le décalage est calculé pour une distance d'un quart puisqu'on prend en compte le paramètre du chevauchement . L'objet *frameaccum* ~ dans ce cas remplace la séquence des objets *phasewrap* ~, 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* ~ au paramètre 0, la taille du fenêtrage, 1 et 1. On peut ainsi stocker les valeurs de l'échantillon exact au quelle la fenêtre de la FFT commence à la fois, ainsi que la position actuelle en durée totale du son. Chaque fois que le compteur se re-initialise au zéro, un objet *sah* ~ permet au valeur de passer, dont on déduit le premier sample de la FFT. en ajoutant les valeurs du compteur on peut déduire la position actuelle sur le fichier sonore. Le dernier nous permet de jouer le fichier à partir de un

5. L'objet *play* ~ est utilisé comme interface de lecture de l'objet *buffer* ~. *play* ~ joue des samples en accord d'un offset parmi le buffer.

objet *play* ~.

On rappelle la formule du vocodeur de phase :

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

Dans la formule ci-dessus on peut voir qu'il y a un fenêtrage correspondant, marqué $h_a(n)$. Pour simplifier on va substituer avec quelques paramètres réels. On remplace N par une taille de la fenêtre à 1024 échantillons. On établit un décalage de 4 fois par fenêtre, donc $uR_a = (\frac{1}{4} * 1024)_a$ pour l'index a de la fenêtre précédente.

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

À partir de cette formule on peut identifier les éléments qui la reproduisent dans le patch Max. On identifie les éléments de la formule dans notre patch. La fonction, correspondant à la phase de l'analyse, contient la somme des échantillons du signal qui appartiennent à la taille de la fenêtre de la FFT, plus des échantillons qui appartiennent $\Sigma_n(x_n)$ à la fenêtre décalée par la taille *hop*, uR_a . Les échantillons sont multipliés, par une fenêtrage $h_a(n)$ et ensuite par la formule d'Euler qui équivaut à la transformation de Fourier.

Nous pouvons voir qu'une fonctionnalité de fenêtrage inclut à la fois dans la formule et dans le patch présenté. Au lieu d'utiliser une fenêtre par défaut de l'objet *fft* ~, nous allons utiliser un tampon fixe de taille N et utiliser l'objet *count* ~ pour accéder à chaque échantillon avec l'aide de *index* ~⁶ et le multiplier par le son souhaité et à traiter avant d'effectuer la FFT.

Filtre Gabor

Dans la section 2, nous avons présenté le filtrage de Gabor. Dans cette section, nous allons créer un filtre Gabor personnalisé. La formule de Gabor normalisée permet de modifier la courbe de

6. L'objet *index* ~ est utilisé pour lire un objet *buffer* ~ dans un exemple d'index piloté par signal sans interpolation sur la sortie.

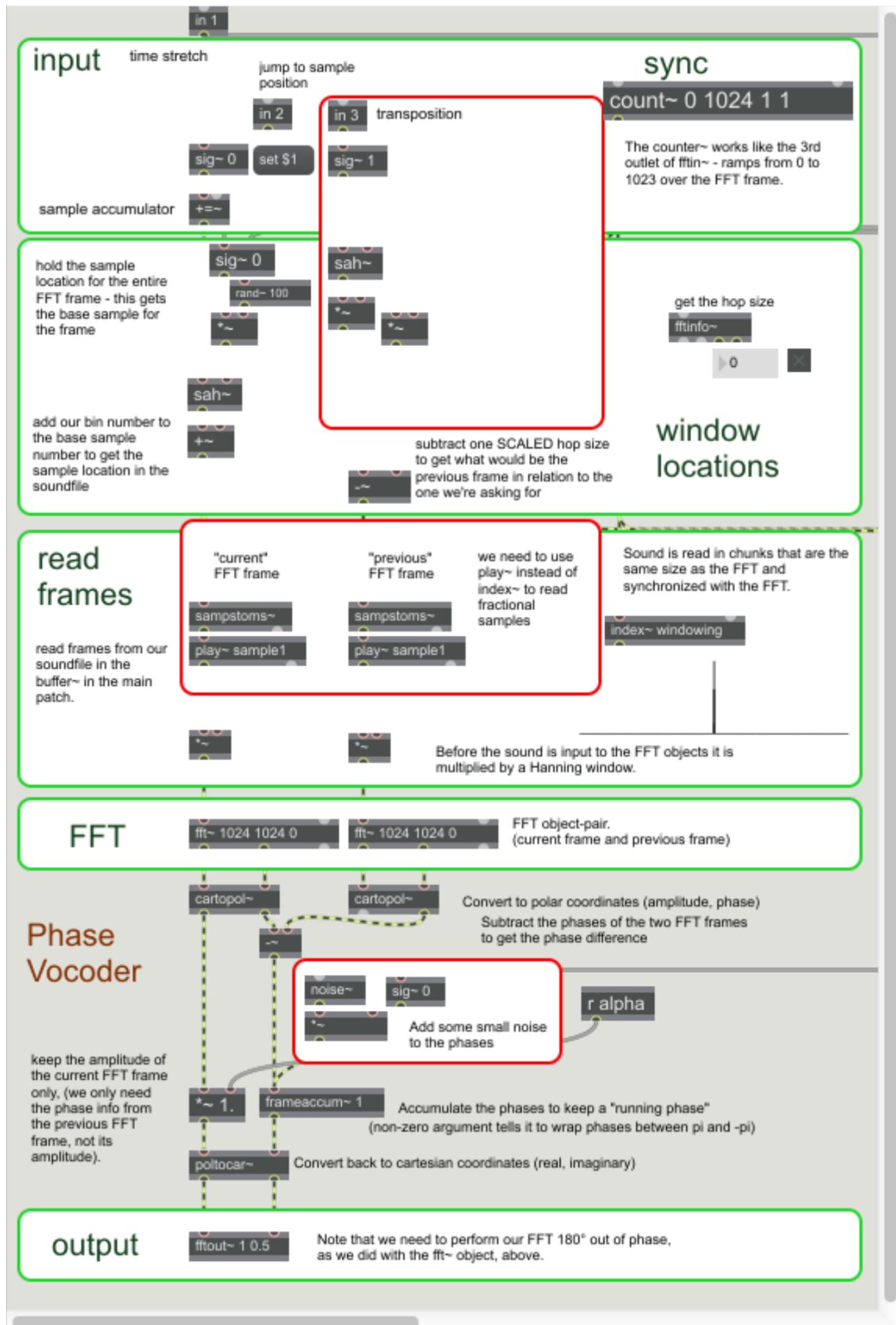


FIGURE 3.5 – Le vocodeur de phase

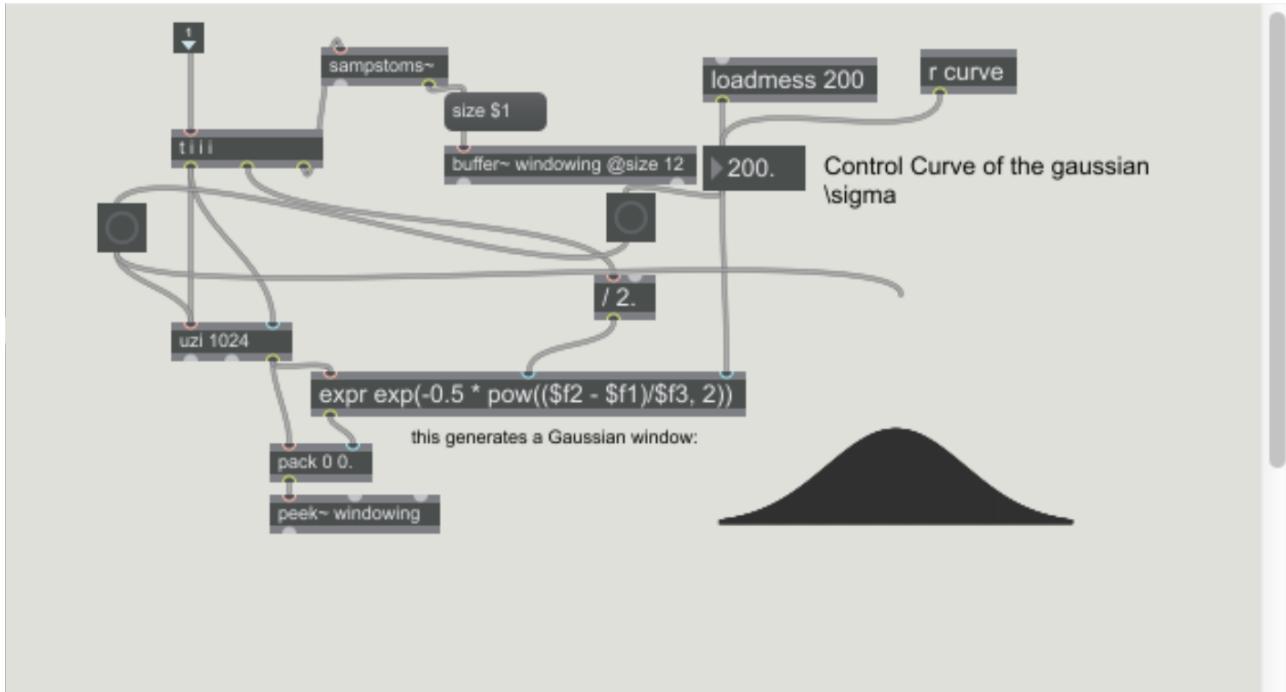


FIGURE 3.6 – Fenétrage gaussien

la fonction gaussienne tout en conservant les constantes de valeurs maximales et minimales. Cette fenêtre gaussienne faite sur mesure est présentée en figure ??.

Nous utilisons une *uzi*⁷ objet avec argument la taille de la fenêtre FFT, suivie de l'expression effectuant une courbe gaussienne normalisée. Ensuite, nous l'envoyons dans le tampon nommé "windowing" à l'aide de l'objet *peek ~*⁸. Bien sûr, dans ce cas, le filtre de Gabor n'est pas directement exécuté pendant la FFT, mais juste avant. En utilisant ce filtrage, nous pouvons librement chevaucher des fenêtres et éviter des artefacts.

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 we should playback the sound at half the speed while doubling the SR. By using this method it is possible to change the pitch without affecting the playback speed.

To use this method we use standard midi values translated to frequency and adapted to the

7. envoie le nombre de banques qui ont été définies dans la position de l'argument en même temps.

8. L'objet *peek ~* est utilisé pour lire et écrire des exemples de valeurs dans un *buffer ~*.

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 ~* 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 ~* corresponding to the window size. A transposition up corresponds to taking a larger window from the *buffer ~* and reading it in a faster speed, while a pitch shift down corresponds to cutting a smaller portion of the *buffer ~* and reading it slower.

For this example we use a *sah ~* 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 :

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

Où α est le facteur de transposition et $\Delta\omega$ la différence de fréquence. Cette technique impose une transposition de fréquence constante $\Delta\omega$ à toutes les fréquences détectées de la FFT.

L'implémentation Pitch Shifting peut être vue dans la figure. Quelques objets suffisent pour implémenter un décalage de hauteur sur le corps de base du vocodeur de phase.

Playback speed

Comme il a été étudié à la section 2, modifier la vitesse de lecture tout en maintenant la hauteur tonale stable est l'opération inverse du décalage de hauteur tonale. La lecture peut être contrôlée en prenant en compte le facteur de chevauchement des fenêtres FFT. On peut

diviser par le facteur du chevauchement et ajouter le pourcentage de la lecture souhaitée. À l'intérieur du vocodeur, la valeur factorisée est ajoutée à l'accumulateur d'échantillons. Dans ce cas, la valeur de la vitesse de lecture est ajoutée dans l'interprétation de position d'échantillon. Une lecture plus rapide ignore les échantillons où une vitesse de lecture plus lente lit plusieurs fois les échantillons. La soustraction de la position précédente de chaque échantillon (la fenêtre de la FFT), fixe les déviations de fréquence.

3.4 Morphing spectrale

La démonstration finale de ce chapitre est une implémentation du morphing spectral ou de la synthèse croisée spectrale. Pour accomplir cette technique, nous utilisons deux vocodeurs en phase parallèle et nous interpolons la phase et l'amplitude de chaque indice du bin. Pour comprendre la complexité de calcul du morphing spectral, rappelons-nous qu'un vocodeur à phase simple utilise deux FFT parallèles décalées du facteur de fenêtre de recouvrement. Donc, si nous utilisons une fenêtre de 1024 échantillons et un facteur de chevauchement de 4, la première FFT commence en position 0 et la fenêtre parallèle en position d'échantillon 256. Nous utilisons maintenant 4 FFT parallèles pour le morphing spectral. Cela fait beaucoup de calculs, mais reste possible avec les ordinateurs actuels.

Dans ce vocodeur à double phase, les sorties de l'objet *cartopol ~*, la magnitude et la phase après l'analyse de Fourier, sont multipliées par le facteur d'interpolation de morphing. La phase est préalablement corrigée par une accumulation avec la trame précédente et une petite quantité de bruit est ajoutée pour un résultat plus naturel. Nous séparons les canaux d'interpolation de magnitude et de phase. Ainsi, on peut préférer émettre, par exemple, l'amplitude du son source mais la phase du son cible.

La sortie de chaque son après l'interpolation est transformée en forme cartésienne et une FFT inverse est appliquée. Ainsi, nous obtenons un résultat sonore homogène. À ce stade, un certain nombre d'opérations d'édition peuvent être ajoutées, telles que la modélisation du bruit ou une interpolation variable par image, etc. Mais avant de plonger dans des modifications avancées,

examinons d'abord les techniques d'interpolation.

L'interpolation est définie par défaut sur linéaire. Un simple code Javascript que nous avons implémenté crée une courbe linéaire. Les valeurs d'interpolation varient entre $[0, 1]$. La valeur 0 correspond à une interpolation nulle, et par conséquent la magnitude ou la phase du son source est sortie. Une valeur de 1 restitue entièrement les caractéristiques du son target et omet le son source. Une courbe linéaire définit une méthode linéaire pour interpoler les sons. Mais nous allons plus loin et implémentons un certain nombre de courbes telles que exponentielle, logarithmique et, bien entendu, linéaire. Par conséquent, il existe différentes manières d'interpoler des sons en fonction de leurs composantes de fréquence ou de la magnitude de leurs composantes de fréquence. Toutes les interpolations varient entre la même intégrale $[0, 1]$. Le code Javascript peut être consulté dans le bout de code ci-dessous.

```
1 // autowatch 1;
2
3 // global variables
4 var s = 1;
5
6 function bang(){
7     if (myFunc == "Exponential")
8     {
9         x = Math.log(1.7182*x+1);
10        post("Exponential Interpolation");
11    }
12    else if (myFunc == "Logarithmic")
13    {
14        x = (x-1)*10;
15        x = Math.pow(2, x);
16        post("Logarithmic Interpolation");
17    }
18    else {
19        x = x;
20        post("Linear Interpolation");
21    }
22    outlet(0,x);
23}
24
25 function msg_float(v){
26    post("interpolation value " + v + "\n");
27    x = v;
28    bang();
29}
30
31 function anything(){
32    var a = arrayfromargs(messagename, arguments);
33    post("received" + a + "\n");
34    myFunc = a;
35    bang();
36}
```

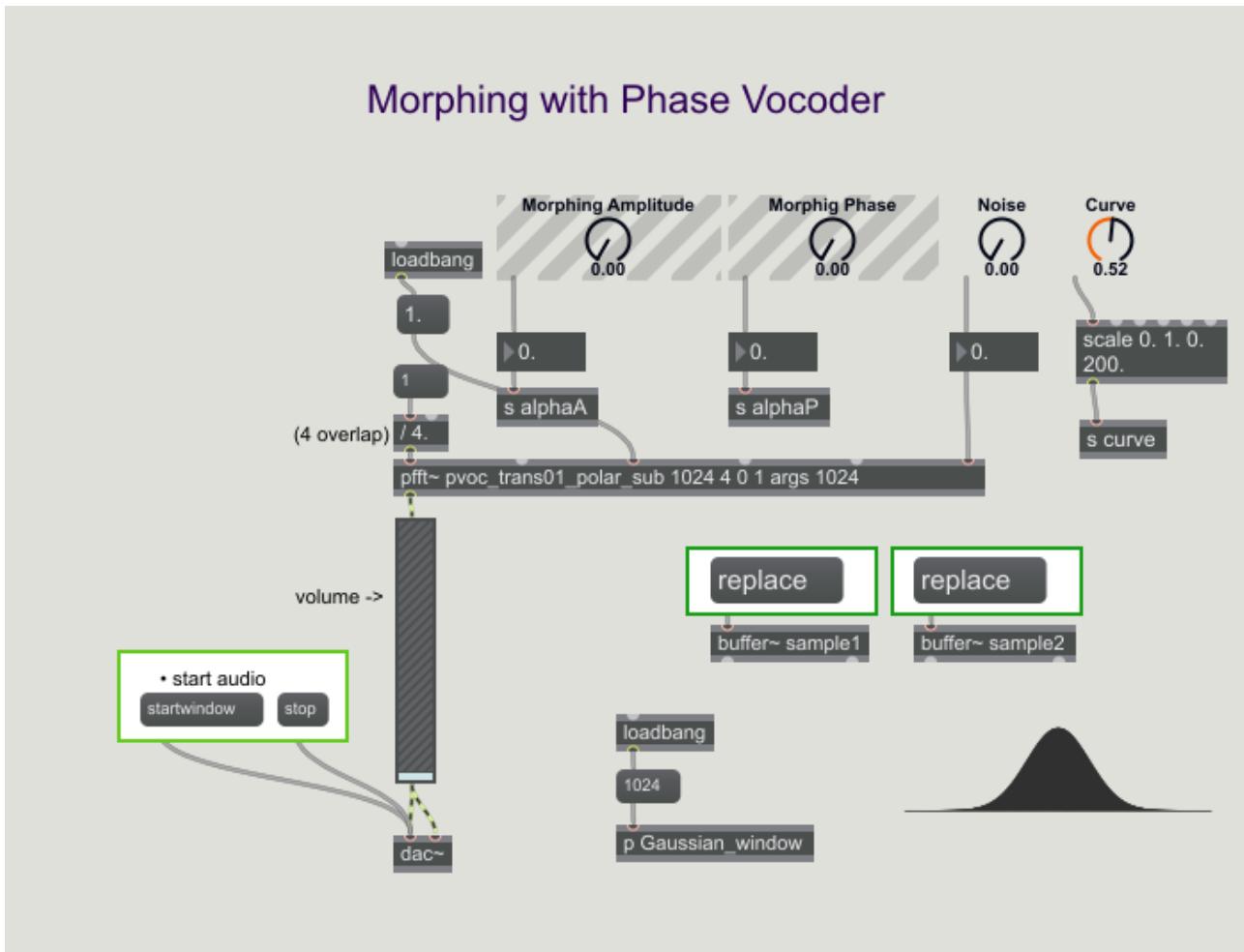


FIGURE 3.7 – Morphing en temps réel

La version du vocodeur de phase pour le morphing spectral est montrée dans la figure 3.7. Il existe deux buffers, un pour la source et un pour le son target. Le fenêtrage gaussien pour du filtrage Gabor, avec une courbe normalisée à distance, est aussi paramétrable dans la plateforme. Un bouton d’interpolation pour la phase et un bouton pour l’interpolation de la magnitude contrôlent les facteurs d’interpolation pour le morphing et une factorisation du bruit pour la naturalisation de la phase sont mis en oeuvre.

Chapitre 4

Implémentations artistiques

4.1 Introduction

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

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

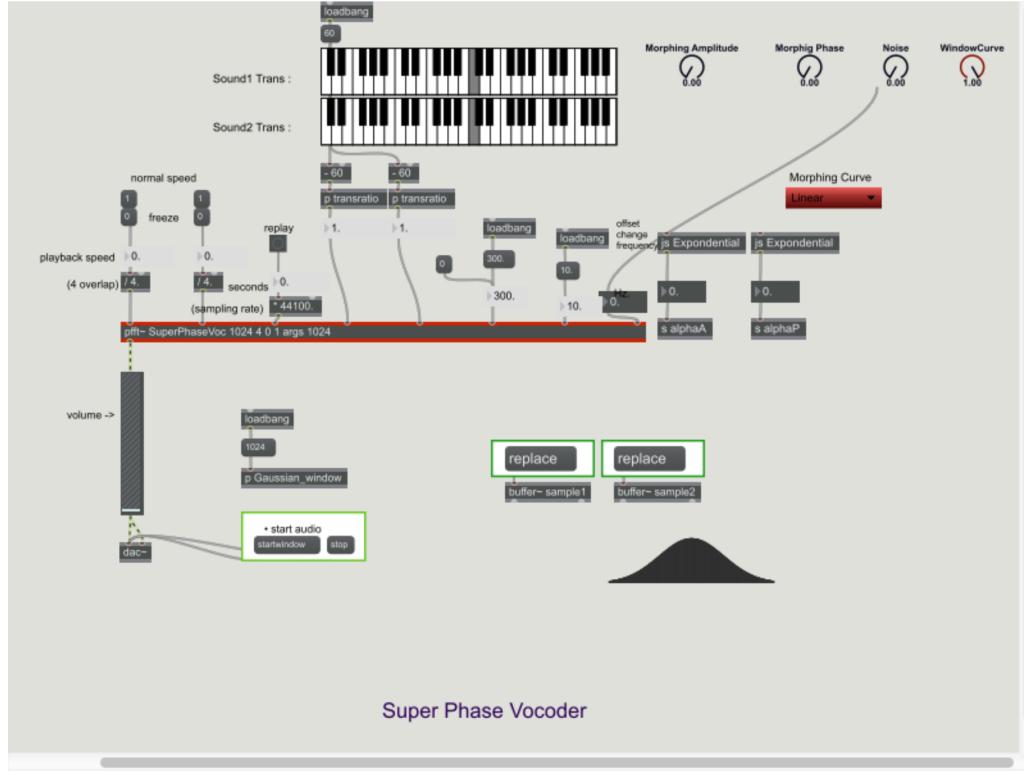
temporel, le décalage de hauteur, le morphing et d'autres effets bénéficiant de la fonction élémentaire du vocodeur de phase.

Le graphe 4.1a présente le patch de contrôle de tous les paramètres du vocodeur de phase. Cette version du vocodeur de phase contient tous les effets étudiés dans les sections précédentes. Il n'y a pas quelque chose de particulièrement différent du vocodeur de phase de base pour le morphing apart quelques fonctionnalités de plus. Il est naturel de rechercher une fonctionnalité plus large pour le même outil. En suivant cette raisonnement, nous avons implémenté tous les effets de base dans un seul patch. Les maths ici ne sont pas intéressants à analyser car ils ont tous été vus auparavant. Les détails les plus cruciaux se trouvent dans la séquence de tous les effets visibles dans figure 4.1b.

Dans le patch principal, vous pouvez voir deux pianos virtuels de contrôle permettant de modifier la hauteur de chaque son séparément. Idéalement, ce système devrait être automatique. Il faut estimer la fréquence de chaque son et corriger à mi-chemin la hauteur du son pour qu'il en soit de même. Une correction de hauteur intervient avant tout effet de morphing et n'affecte donc pas le résultat. Dans le patch principal, nous pouvons également trouver des commandes permettant de manipuler le filtre de Gabor, en ajustant les composantes de bruit de la phase qui, dans ce patch, sont identiques pour les deux sons. On peut également trouver des facteurs d'interpolation de magnitude et de phase pour le morphing, ainsi que la factorisation de la courbe, comme indiqué dans la section précédente. La dernière modification présentée dans ce patch est l'effet de *freeze* qui contient les valeurs spectrales d'une seule fenêtre. Pour cet effet, deux types de filtrage sont également ajoutés pour modifier le timbre des spectres gelés.

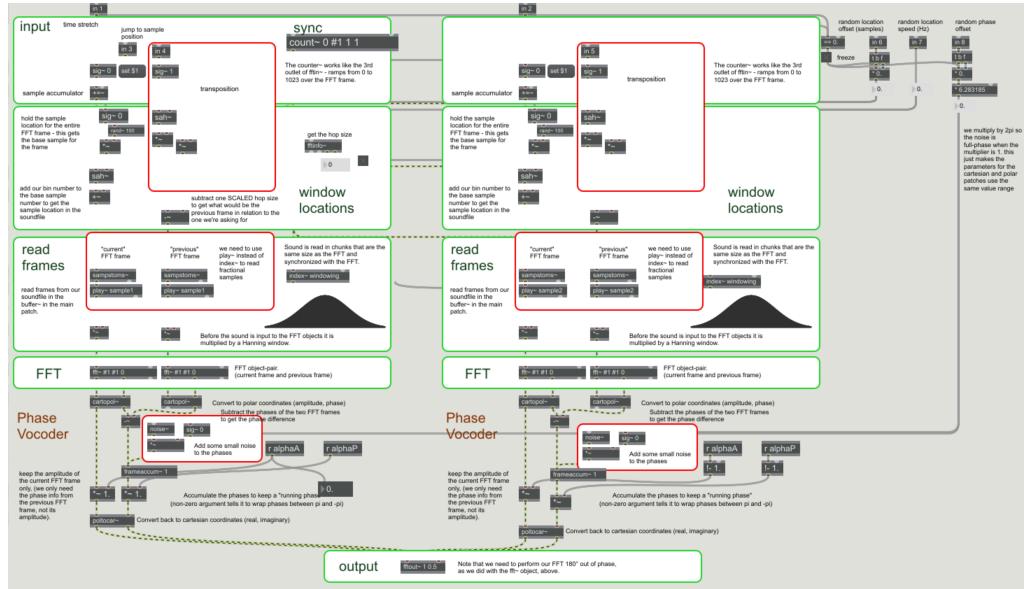
4.2.2 Phase interference

Dans ce patch MaxMSP, nous étudions une modification des coordonnées polaires après l'étape de l'analyse. Nous ajoutons simplement un oscillateur de phaseur après la FFT et l'appliquons à chaque amplitude et phase corrigée de la fenêtre. De cette façon, l'identité du signal n'est pas modifiée, mais une fréquence oscillante interfère avec chaque index de la fenêtre FFT. La formule de cette modification est présentée ci-dessous par l'équation de synthèse.



Super Phase Vocoder

(a) Super Phase Vocoder I



(b) Super Phase Vocoder II

FIGURE 4.1 – Super Phase Vocoder

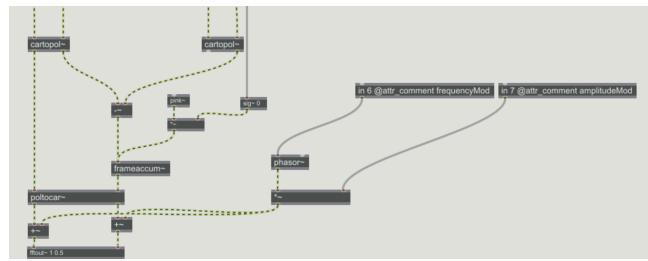


FIGURE 4.2 – modulation des coordonées polaires

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

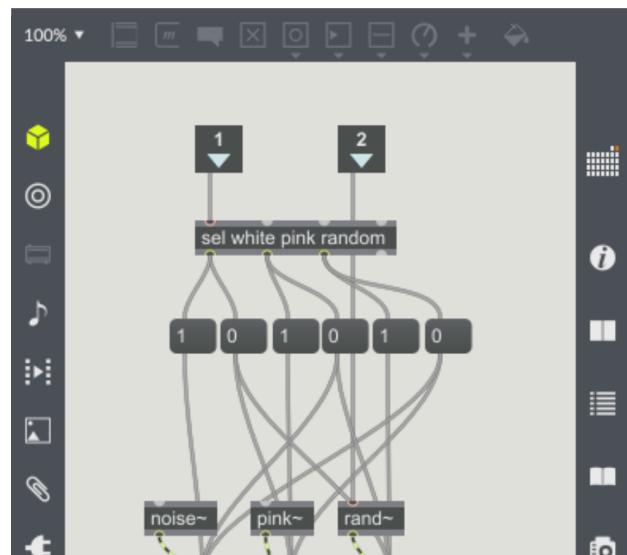
Où $y(n, k)$ est le signal fenêtré après l'analyse. K est le nombre de sinusoïdes, $A[n]$ est la magnitude instantanée, θ est la phase instantanée et $\phi(n)$ est le phasor instantané donné par la formule :

$$\phi(n) = n - \text{floor}(n)$$

Nous pouvons voir que le modèle sous-jacent de la FFT est utilisé pour représenter cette modification. D'autres modèles pourraient parfaitement décrire cet effet, mais dans cette version, il est plus direct. Dans la figure correspondante 4.2, nous pouvons voir le phasor moduler la magnitude et la phase. La fréquence et l'amplitude du phasor sont contrôlées dans le patch principal contenant l'objet $pfft \sim$.

4.2.3 Modulation au bruit

Dans ce vocodeur de phase, nous étudions une modulation de bruit sur la composante de fréquence. Nous avons ajouté la composante de bruit et essayé différents générateurs de bruit tels que $rand \sim$, pink-noise, white-noise et en même temps nous avons contrôlé le facteur



d'amplitude de cette composante. Le sous-patch pour la sélection du bruit est visible dans la figure ???. Un objet umenu fonctionne comme un stade de sélection pour l'utilisateur du maître transmettant ses valeurs au sous-patch pfft.

La phase corrigée, par la fenêtre FFT de chevauchement, est multipliée par le facteur du bruit. Dans les dossiers sonores de cette thèse, on entend les différentes modulations de bruit, mais le résultat sonore n'est pas évident. Le facteur de bruit est utilisé pour la naturalisation du son car les fréquences continues pures ne sont pas produites dans les sons réels. Par conséquent, une petite correction de l'oscillateur de bruit ne donne pas un résultat strictement différent.

4.2.4 Filtre aleatoire sur la position du buffer~

Dans ce patch, nous modulons la position du tampon via un générateur de signaux aléatoires qui transmet ses valeurs au lecteur de la position du fichier. Pour être plus précis, nous filtrons les harmoniques de la FFT en produisant des nombres aléatoires pour chaque harmonique et en ne calculant l'harmonique k -ième que si le nombre aléatoire correspondant est inférieur à sa valeur d'index.

Le générateur aléatoire est créé par l'objet *random ~* avec un filtre modulant la fréquence de la génération de valeur aléatoire. La valeur transmise au lecteur de l'objet *play ~* est modulée par un détenteur d'échantillon qui prend en entrée la sortie d'un compteur et la valeur générée de manière aléatoire. Le porte-échantillon est créé manuellement par

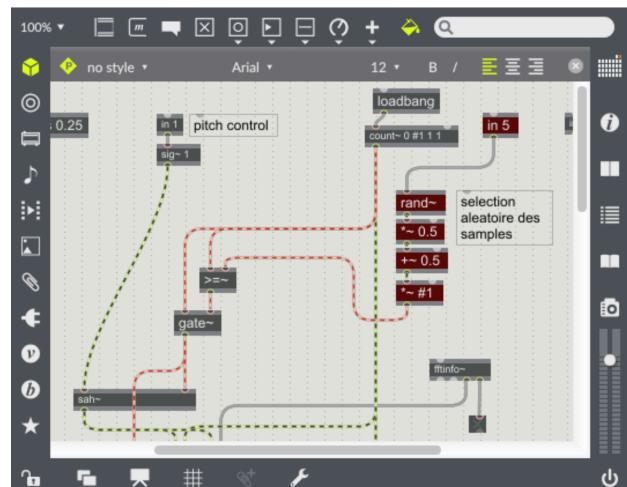


FIGURE 4.4 – Filtre aleatoire

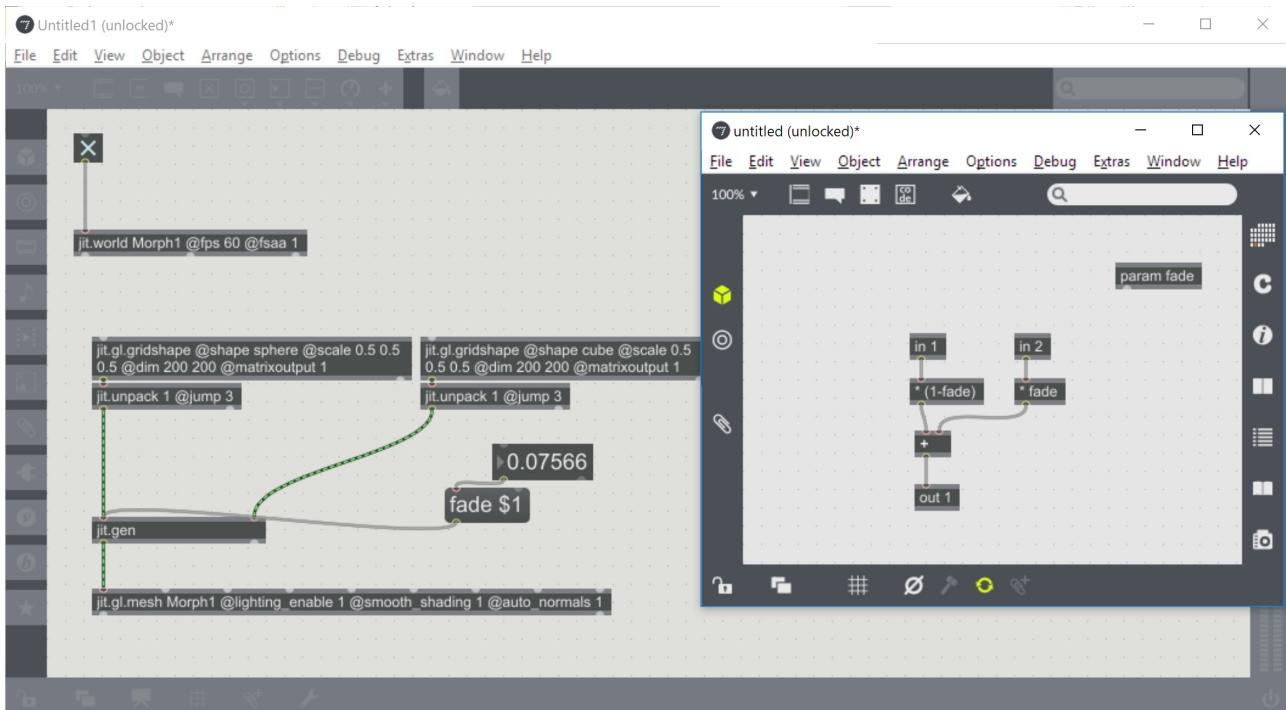


FIGURE 4.5 – Visual Morphing

un objet supérieur à un objet et une porte. De cette manière, nous conservons et publions de manière abstraite des valeurs dans la fenêtre FFT, créant ainsi un système hybride de modulation de la hauteur et de la lecture. La modulation peut être trouvée dans la figure.

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 $M(\alpha) = \alpha\hat{S}_1 + [1 - \alpha]\hat{S}_2$ une patch sur le morphing en 3D était implanté avec l'aide de l'objet jit.gen (figure 4.5).

Donc, fondamentalement, un morphing visuel est facile à faire avec les fonctions 3D primordiales 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.

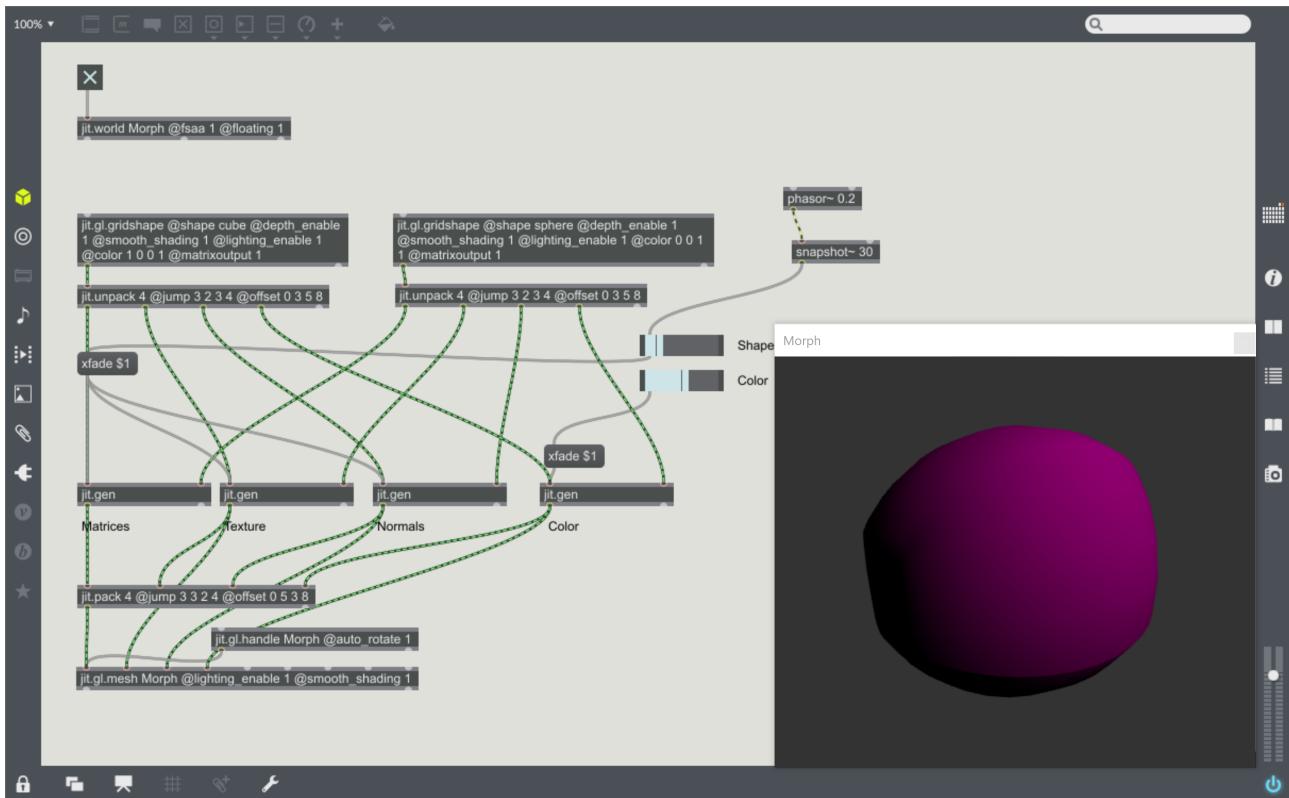


FIGURE 4.6 – Visual Morphing 2^{me} version

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

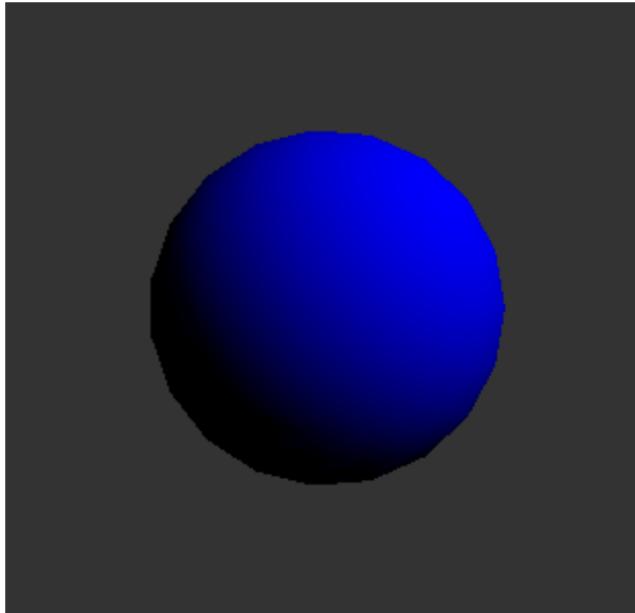
Bien entendu, nous pourrions également implémenter le script exponential.js pour une courbe de morphing différente sur le visuel.

Dans le patch suivant nous sommes allés un étape plus loin en séparant tout composant d’un modèle tri-dimensionnel dans jitter. Nous distinguons les valeurs des matrices, des normaux et de la texture. Nous avons ajoutés également une interpolation du couleur. Le patch est en disposition en figure 4.6.

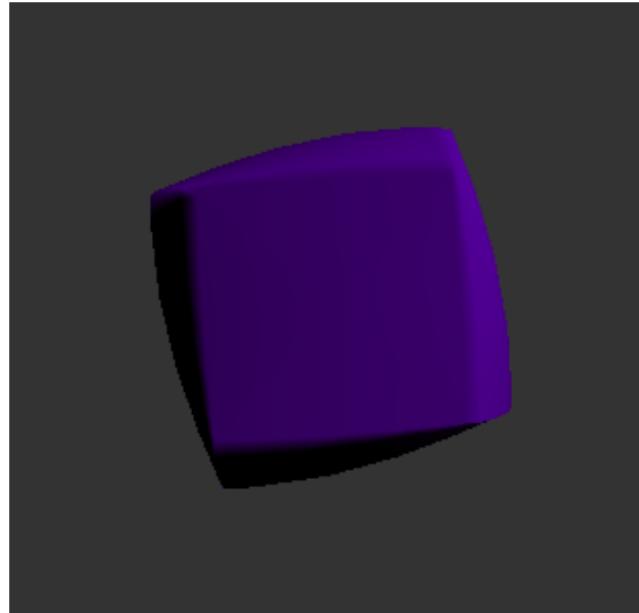
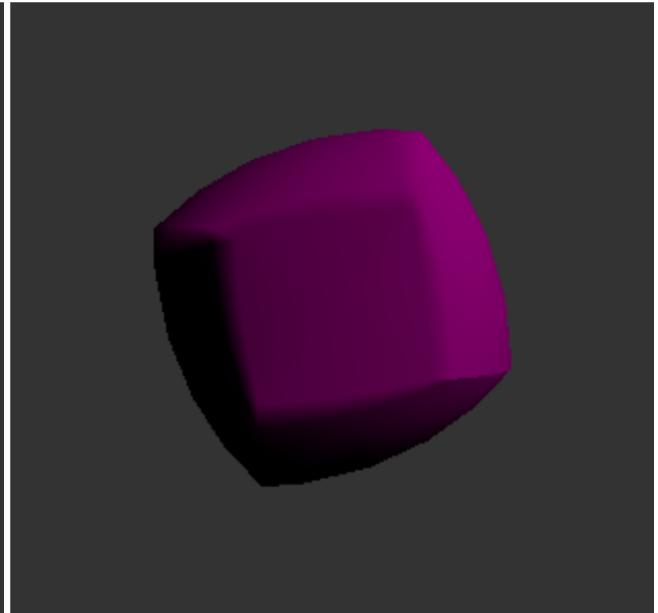
4.3.1 Visualization du spectre

Dans ce patch, beaucoup de travail sur la gigue a été implémenté avec un vocodeur de phase. Ce code permet de visualiser les informations spectrales d’un son sous la forme d’un paquet de

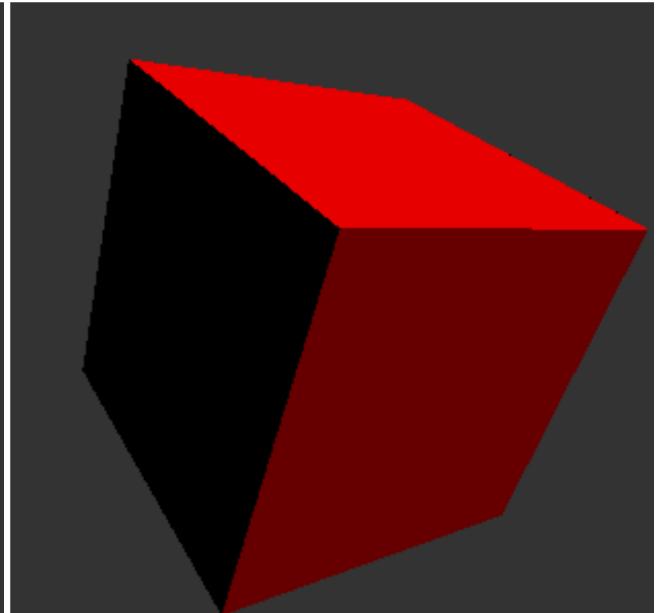
(a) Interpolation de la forme : 0.0
Interpolation couleur : 0.0



(b) Interpolation de la forme : 0.4
Interpolation couleur : 0.8



(c) Interpolation de la forme : 0.8
Interpolation couleur : 0.4



(d) Interpolation de la forme : 1.0
Interpolation couleur : 1.0

FIGURE 4.7 – Stades du morphing visuel

lignes bidimensionnelles imitant les harmoniques trouvées lors d'une analyse de Fourier.

Ce patch utilisé à l'unisson avec un vocodeur de phase pour le morphing ou même le simple décalage de pitch permet de visualiser l'évolution du spectre dans un environnement de rendu en temps réel. Dans ce code, nous utilisons une bibliothèque de gigue supplémentaire pour produire plusieurs objets de gigue rendus dans un espace tridimensionnel virtuel. L'objectif de cette thèse n'étant pas orienté image, nous ne dévoilerons pas beaucoup d'informations sur le traitement.

Nous allons attribuer ici brièvement les objets les plus emblématiques de la gigue utilisés dans ce patch. L'objet *jit.world* crée une nouvelle fenêtre et un espace de rendu virtuel tridimensionnel pouvant également contenir de la physique, des plans, des objets tridimensionnels, contrôler la position de la caméra, la couleur d'arrière-plan, etc. Le *jit.gridshape* rend un objet tridimensionnel dans une fenêtre. La bibliothèque *jit.mo* réplique de manière fluide les copies d'objets de gigue. Et le *jit.catch* prend un signal et le traduit en matrice de gigue pour la visualisation potentielle.

Pour transformer les informations spectrales en données tridimensionnelles, nous utilisons bien sûr une transformation FFT et un tampon qui lit les composantes polaires de la FFT (magnitude et phase) avec un objet *peek ~* les envoie dans un tampon. Un objet *lookup ~* récupère le contenu du tampon et les entre dans l'objet *jit.catch*.

Nous pouvons régler la dimension de *jit.mo* pour générer plus d'harmoniques. Le son de sortie du vocodeur est envoyé à ce patch de visualisation harmonique pour saisir le résultat dans un espace 3D.

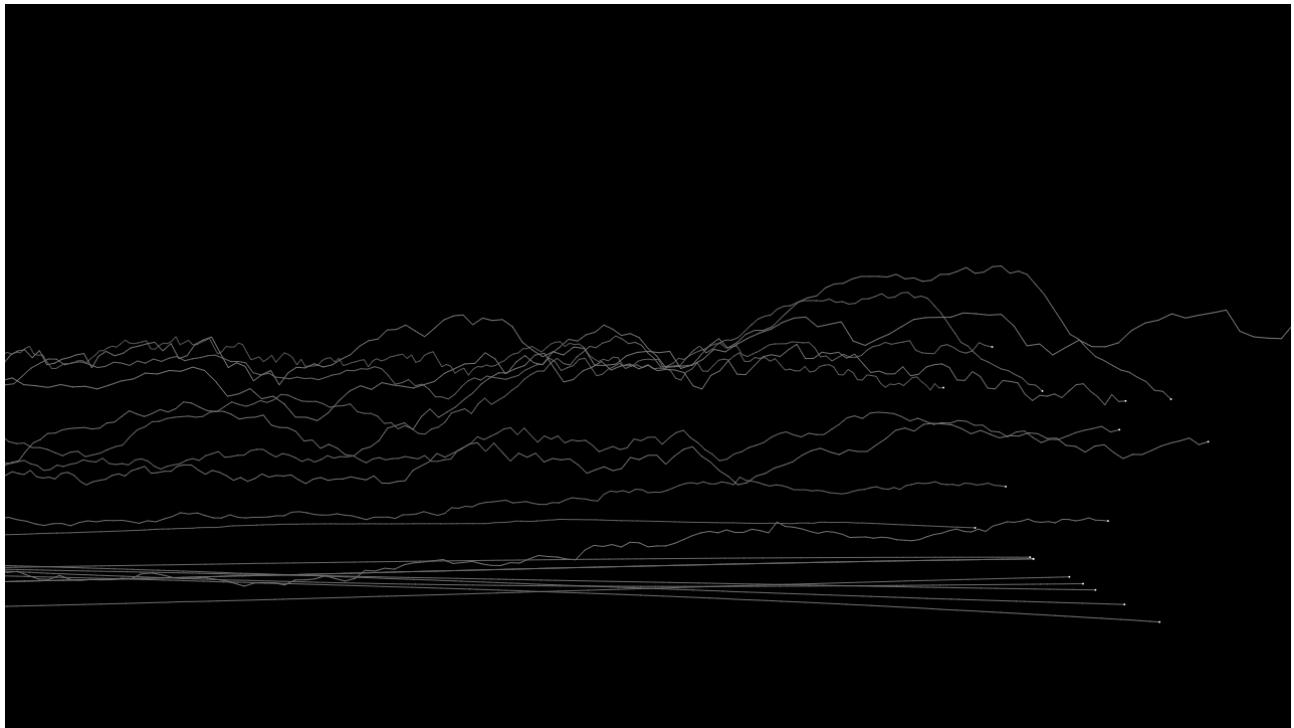


FIGURE 4.8 – Harmoniques dans l'espace

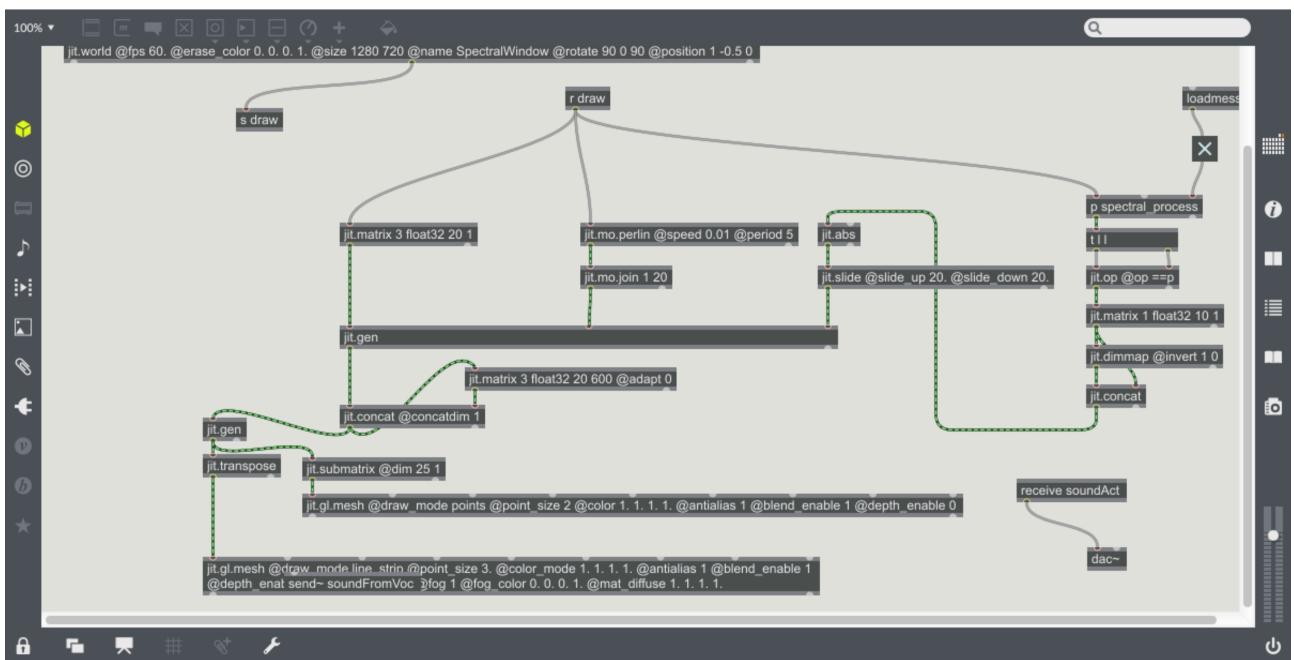


FIGURE 4.9 – Le patch qui control les objets jitter

Chapitre 5

Conclusion

5.1 Résumé de la recherche

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

Les applications des vocodeurs de phase sont quasiment infinies, comme quelq'un peut envisager par le chapitre sur la mise en œuvre. Quelques propositions suivront pour afficher les capacités d'analyse spectrale et de vocodeur de phase.

Le vocodeur de phase peut être utilisé pour analyser la voix et la transformer en texte. Les sélections et les fréquences peuvent être déduites de certaines voyelles et consonnes, ce qui permet de prédire les lettres exactes utilisées et de produire la forme écrite du son.

Le vocodeur de phase humain doté de la voix peut également transformer un certain type de voix, par exemple une voix d'homme en une voix féminine ou tout autre type de voix, en se transformant entre les caractéristiques qui déterminent une voix en tant qu'homme, femme ou autre.

Les outils spectraux et spécialement ceux d'analyse-synthèse sont fréquemment utilisés par les compositeurs vis-à-vis des programmes DAW. Une bibliothèque célèbre serait TRAX par les outils Ircam.

5.3 Discussion

Le vocodeur de phase est aujourd'hui plus qu'un simple outil d'analyse spectrale. Des méthodes conçues pour prévoir de manière stochastique la resynthèse des vagues la mieux adaptée.

Ces formules considèrent des méthodes probabilistes pour reconstruire les signaux, pour de meilleures techniques de prédiction de fréquence. Pour plus d'informations à ce sujet, le livre de Roads and al.¹ était plutôt éclairant.

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 une approche sur le Morphing spectrale en temps réel connecte aux networks neurals².

Je suis particulièrement intéressé par le fait qu'un vocoder de phase puisse détecter les fréquences dominantes d'un son. Ce fait pourrait essentiellement servir à une méthode intelligente de préciser les différentes notes contenue dans une information sonore. L'extraction des fréquences 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

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

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

futur, je souhaiterais m'orienter dans le domaine de l'analyse Topologique de la musique. Une interprétation de la musique symbolique comme des structures topologiques bidimensionnelles, c'est-à-dire des graphes orientés, peut assister à extraire informations du style musical ou de la structure. La théorie de graphes contient plusieurs méthodes spectrales telles que la transformation Fourier des graphes et toute une infrastructure mathématiques basée sur les concepts investigués dans cet recherche.

Une piste alternative très intrigante serait pour moi l'utilisation de VAE (Variational Autoencoders) pour identifier la distribution de probabilité d'un ensemble de pièces musicales. Les VAE sont des auto-encodeurs qui paramètrent la distribution de probabilité des variables latentes à l'aide d'un réseau neuronal et maximisent la probabilité que la distribution de données soit soumise à des contraintes de la distribution de variables latentes. Des expériences numériques montrent qu'il est capable d'identifier cette distribution de données dans la mesure où l'échantillonnage à partir de la distribution de 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 l'intégration VAE.

Je crois que le processus d'analyse musicale sur l'harmonie et la mélodie, associé à un apprentissage non supervisé, permet de passer à de meilleurs modèles de prédiction musicale, de variation et de créativité. Pour atteindre cet objectif, je visualise la première formalisation de modélisation algébrique sur des données verticales (accords) et linéaires (mélodies) telles que les approches de la théorie de la musique transformationnelle. Avant de construire une base de données de styles et d'appliquer un algorithme de prédiction, j'estime 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 les modèles existants de VAE récurrents rencontraient des difficultés pour modéliser des séquences avec une structure à long terme. Pour résoudre ce problème, je propose l'utilisation d'un décodeur hiérarchique, qui génère d'abord des intégrations pour les sous-séquences de l'entrée, puis utilise ces intégrations pour générer chaque sous-séquence de manière indépendante. Il est également possible de modéliser de la musique polyphonique multipiste en tant que vecteurs dans un espace latent via le conditionnement d'accords. En effet celui-ci permet une compréhension plus profonde de la mélodie tout en

maintenant une harmonie fixe, et permet en outre de modifier les accords tout en maintenant le style musical. Cette approche a été présentée pour la première fois dans le projet Magenta de Google AI et présente, à mon avis, un potentiel important et pourrait progresser dans la génération simultanée d'accords et de mélodies.

Appendix

.1 FFT Cooley - Tukey algorithm

```
1 using FFTW
2 #simple DFT function
3 function DFT(x)s
4     N = length(x)
5     # We want two vectors here for real space (n) and frequency space (k)
6     n = 0:N-1
7     k = n'
8     transform_matrix = exp.(-2im*pi*n*k/N)
9     return transform_matrix*x
10 end
11 # Implementing the Cooley-Tukey Algorithm
12 function cooley_tukey(x)
13     N = length(x)
14     if (N > 2)
15         x_odd = cooley_tukey(x[1:2:N])
16         x_even = cooley_tukey(x[2:2:N])
17     else
18         x_odd = x[1]
19         x_even = x[2]
20     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
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             )
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 1 – Cooley-Tukey with Butterfly Diagrams

3.

.2 Espace 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'espace Lebesgue $L^2(\mathbb{R}^k)$ 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 $C_0(\mathbb{R}^k)$ ⁴.

Soit $f : \mathbb{R}^k \rightarrow \mathbb{C}$, $f \in L^1(\mathbb{R}^k)$.

3. Le code est recuperé sur le lien www.algorithm-archive.org distribué sous la licence *MIT*

4. L'espace des fonctions continues qui tendent vers 0 à l'infini

On définit la transformée de Fourier de f et on la note $\mathcal{F}f$ ou $\mathcal{F}[f]$ ou encore \hat{f} par

$$\begin{aligned}\hat{f} &: \mathbb{R}^k \rightarrow \mathbb{C} \\ \omega &\mapsto \hat{f}(\omega) := \int_{\mathbb{R}^k} f(x)e^{-2\pi i x \cdot \omega} dx\end{aligned}$$

En effet, la transformée de Fourier peut être définie pour des fonctions de $L^2(\mathbb{R}^k)$ et par le théorème de Plancherel :

Si $f \in L^1(\mathbb{R}^k) \cap L^2(\mathbb{R}^k)$ alors

$$\|f\|_2 = \|\hat{f}\|_2$$

et donc l'opérateur \mathcal{F} peut être étendu à $L^2(\mathbb{R}^k)$. De plus, on a ce qu'on appelle l'identité de Parseval

$$\langle f, g \rangle = \langle \hat{f}, \hat{g} \rangle, \quad \forall f, g \in L^2(\mathbb{R}^k)$$

En somme, la transformée de Fourier est un isomorphisme isométrique de $L^2(\mathbb{R}^k)$ en $L^2(\mathbb{R}^k)$, ce qui de manière symbolique est traduit par

$$\begin{aligned}\mathcal{F} : L^2(\mathbb{R}^k) &\rightarrow L^2(\mathbb{R}^k) \\ f &\mapsto \hat{f}\end{aligned}$$

et

$$\mathcal{F} \circ \mathcal{F}^{-1} = \mathcal{F}^{-1} \circ \mathcal{F} = Id$$

Pour effectuer la STFT on doit d'abord fixer une fonction qui nous servira de fenêtre. Dans une approche plus pratique, on utilisera généralement des fonctions avec une forme 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 choisi 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, notamment 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 $\mathcal{D}(\mathbb{R}^k)$ pour noter l'espace des distributions ordinaires et $\mathcal{T}(\mathbb{R}^k)$ pour noter l'espace des distributions tempérées.

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

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

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 $\mathcal{T}(\mathbb{R}^k)$. De plus, la justification de que ce soit une extension est faite par l'interprétation 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 $\mathcal{T}(\mathbb{R}^k)$ qui est invariant par des translations et modulations alors la STFT est bien définie pour $f \in B'$ et $g \in B$.

Listings

| | | |
|-----|--------------------------------------|----|
| 2.1 | DFT | 16 |
| | Exponential.js | 46 |
| 1 | Cooley-Tukey with Butterfly Diagrams | 62 |

Bibliographie

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

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.

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.

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

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

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://mplat.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 scene*. PhD thesis, Université de Miami, 2005.

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

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>.

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 Anaik, 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 cespral and all-pole based spectral envelope modeling witg unknown model order. *Pattern Recognition Letters*, (28) :pp. 1343 – 1350, 2007.