

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

Le compositeur Gérard Grisey dans son texte *Écrits ou l'invention de la musique spectrale* disait : "L'architecture magnifie l'Espace disait Le Corbusier. Aujourd'hui comme jadis la musique transfigure le Temps"¹. Par cette citation Gérard Grisey, démontre que sa musique, ou bien la musique spectrale, donne une autre dimension au temps. Il met ici en évidence l'importance de la temporalité dans la musique spectrale. En essayant de comprendre l'essentialité de la musique spectrale on est venu se confronter à la notion du spectre. Dès lors, les compositeurs spectrales ont expérimenté avec le timbre, la dynamique, le registre, les couleurs, le rythme et tous les autres aspects sonores auxquels on pourrait penser. 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 ?

Cette question a défini la trajectoire de ce mémoire. L'objectif est d'explorer la nature du son et plus précisément l'étude de l'analyse spectrale en temps réel. Dans ce mémoire, le processus de traitement spectral va être analysé en profondeur et une construction d'une vocodeur de phase va être présenté pour le but d'une projection artistique.

Dans un premier temps, 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 décoder comment il est possible de traiter les spectres des sons. D'autre part, afin de faciliter la compréhension du cadre mathématique par des musiciens, nous intégrons un contexte musical.

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. Les explications musicales et de programmation se construisent pas à pas 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 afin de composer une pièce acousmatique qui va souligner la valeur artistique d'un vocodeur de phase.

1. Gérard Grisey, *Écrits ou l'invention de la musique spectrale*, 2008

On peut s'attendre à ce que ce mémoire soit un guide pour les musiciens sur le traitement spectral, mais également une source fructueuse d'exemples et d'applications artistiques. Ce mémoire sera utile pour tout musicien ou compositeur qui cherche à comprendre la logique des outils qu'il utilise fréquemment mais aussi pour toute personne qui 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 ce recherche englobe différentes utilisations du vocodeur de phase et donne d'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	4
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 Contexte théorique	11
2.1 Introduction	11
2.2 Décodage mathématique	12

2.2.1	The Fourier Transform	12
2.2.2	Fenêtrage	18
2.2.3	Le vocodeur de phase	20
2.2.4	Applications du vocodeur de phase	24
3	Design	29
3.1	Objets MaxMSP	29
3.2	Détection du pitch	32
3.2.1	Détection des pitchs multiples	35
3.3	Le vocodeur de phase	38
3.4	Morphing spectrale	45
4	Implémentations artistiques	50
4.1	Introduction	50
4.2	Le vocodeur de phase - <i>Une capacité sans fin</i>	51
4.2.1	Super Phase Vocoder	51
4.2.2	Phase interference	53
4.2.3	Modulation au bruit	54
4.2.4	Filtre aléatoire sur la position du buffer~	54
4.2.5	Robotisation	55
4.3	Morphing visuel	56
4.3.1	Visualisation du spectre	57

4.4	La mise en œuvre compositionnelle	60
4.4.1	Évaluation Artistique	62
5	Conclusion	64
5.1	Résumé de la recherche	64
5.2	Applications	65
5.3	Discussion	65
5.4	Recherche pour l'avenir	66
A	Appendix	68
A.1	FFT Cooley - Tukey algorithm	68

Table des figures

2.1	Complex DFT	13
2.2	Circle de la transformation Fourier	14
2.3	Magnitude et phase	15
2.4	Diagramme papillon pour une FFT à 8 points	18
2.5	Overlapping	19
2.6	Interpolation du facteur α	27
3.1	Pitch Tracking	33
3.2	Codebox~	34
3.3	Pitch Tracker	36
3.4	Gen multiple	37
3.5	Multiple Pitch tracking	38
3.6	Le vocodeur de phase	42
3.7	Fenêtrage gaussien	43
3.8	Morphing en temps réel	47
4.1	Super Phase Vocoder	52

4.2	modulation des coordonées polaires	53
4.3	Selection du bruit	54
4.4	Filtre aléatoire	55
4.5	Robotisation	55
4.6	Visual Morphing	56
4.7	Visual Morphing <i>2^{me}</i> version	57
4.8	Stades du morphing visuel	58
4.9	Harmoniques dans l'espace	59
4.10	Le patch qui control les objets jitter	60
4.11	Partie du mixage de Diakrotima sur Reaper	61

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 des résultats innovants.

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 cette branche, un ensemble de disciplines scientifiques se contracte 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 alons vers l'art. Le nom d'analyse-synthèse exprime ce principe, d'observation (analyse). Cela permet 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 domaine temporel 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, il transfère le 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 particulier de 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, Tristan Murail, etc. 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 cette 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. Le but artistique consiste à créer un ou plusieurs objets informatiques qui vont servir à la composition d'une pièce acousmatique.

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 *LATEX* est relativement plus facile à partager et lisible à partir de n'importe quel éditeur de texte. Cependant, l'utilisation de *LATEX* rend différemment la formulation de références et la bibliographie en comparaison des normes françaises.

La partie programmation de cette 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 7.3.2 est utilisé sans bibliothèques externes. Les paramètres audio sont définis sur SR de 44.100, 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, également, utilisé pour le mixage des enregistrements faits dans Max pour orchestrer une composition de courte durée. Reaper est une plateforme de travail d'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 «github.com/melkisedeath/Memoire-du-Master-Paris8—Domaines-fondtionnels-du-vocodeur-de-phase » sous une licence *MIT*. Dans ce référentiel, on peut trouver le code, les fichiers sonores, la pièce acousmatique en format *wav*, 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 pour arriver, en fin, à la composition d'une pièce inspiré par le vocodeur de phase.

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 progressionnelle 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. On va présenter dans Max un guide de construction, étape par étape, d'un vocodeur de phase et en passant par l'exploration mathématique de l'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 un son à plusieurs reprises tout en faisant des modifications. Ce processus mène à une compréhension plus profonde de la fonctionnalité d'un outil informatique sonore. Le morphing spectral, étant l'outil informatique en question, dans cette étude est constitué de deux vocodeurs de phase simultanés qui travaillent sur des entrées différentes.

Il est raisonnable à ce stade de poser la question : est-il nécessaire que quelqu'un soit musicien pour réaliser tout cela ? La réponse repose sur l'implementation artistique. Elle sera, donc, la section clé de cette dissertation. En combinant de compétences en programmation, en compréhension mathématique et en créativité musicale, on se rapproche du profile de nombreux compositeurs contemporains. Dans le dernier chapitre, on peut chercher quelques exemples de modèles d'implémentations créatives du vocodeur de phase et, également, une composition à partir le vocodeur. Cette section sera consacrée à une investigation sur les différents univers artistiques possibles d'un vocodeur de phase.

1.3 MaxMSP et Jitter

MaxMSP est un logiciel développé à l'origine par Michael Puckette sous l'étiquette de l'IRCAM puis acheté par l'entreprise américaine, Cycling74. Le cœur 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 :

1. les commandes d'expression logique
2. les commandes du traitement sonore
3. les commandes du traitement des matrices multidimensionnelles

Chacune correspondant respectivement à Max, MSP et Jitter. L'utilisateur peut connecter diverses commandes des trois catégories pour créer un patch¹ complexe. La version la plus récente de MaxMSP et Jitter est Max8, publiée en septembre 2018².

Dans la catégorie du traitement sonore, une librairie 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

1. Dans Max un fichier code est suivant appelé «patch »parmi utilisateurs

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

signal comme une ligne courbée continue qui est 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 aussi par des instituts de recherche, tels que l'IRCAM, Paris8, etc. La vaste communauté de Max le rend accessible même aux utilisateurs amateurs. Dernier point, mais non pas des moindres, une série de bibliothèques techniques pour Max, sur les vocodeurs de phase et sur le traitement spectral, est développée par la communauté, démontrant ainsi le rôle principal de ce logiciel. Certaines de ces 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 graphe fréquence-magnitude. Dans un tel diagramme, on peut visualiser les fréquences d'un signal et leurs amplitudes correspondantes, pendant une certaine période temporelle.

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*³ il décrit des outils mécaniques permettant de reproduire artificiellement le spectre sinusoïdal de divers sons⁴.

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 simples⁵. 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 conséquent, il convient de considérer la borne supérieure (supremum) de temps nécessaire pour décrire un signal. Ce point précisément 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 vocodeur de phase, 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 le 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 cœur 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

3. H. L. F. Helmholtz, *The Sensation of Tone*, 1863

4. Curtis Roads, *The Computer Music Tutorial*, 1996. [p. 545]

5. Jean Joseph Baptiste Fourier, *De la Chaleur*, 1822.

fréquence-magnitude⁶.

Le vocodeur de phase a été découvert en 1966 par Flanagan⁷. Cependant, la structure standard du vocodeur de phase moderne a été établie par Griffin et Lim en 1984⁸. 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'Ircam⁹. 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.

1.5 Morphing

Le morphing est le processus d'interpolation entre deux objets ou plus, qui donne un résultat hybride contenant 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, quelle soit sonore ou visuelle.

1.5.1 Morphing sonore

Le morphing sonore combine des spectres de sons ce qui entraîne une interpolation des 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 plusieurs sources aboutissant à un son hybride morphé

6. Joshua Fineberg, *Guide to the Basic Concepts and Techniques of Spectral Music*, 2000.

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

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

9. ??www.anasynth.ircam.fr

contenant des éléments de chaque source. On peut citer comme exemple, un morphing entre un piano et un violon qui jouent tout le 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 du vocodeur de phase afin d'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 sonore.

Avec une approche d'un vocodeur de phase pour réaliser du morphing sonore, un minimum de deux vocodeurs de phase est requis. On peut, également, manipuler l'amplitude 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 semble relativement 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. 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'autre¹⁰. Ce mémoire se concentre sur le son, et par conséquent, le morphing de l'image ne sera pas examiné.

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

Chapitre 2

Contexte théorique

2.1 Introduction

Dans cette section, on va décrire quelques notions mathématiques de la théorie qui est la base du vocodeur de phase. Le but est de développer des outils qui vont servir à la composition d'une pièce musicale. 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.

Afin de visualiser cette méthode on peut utiliser le paradigme de la synthèse additive 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 varia-

tion 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 donnée. 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 pour sa programmation. Néanmoins, il fait partie des algorithmes les plus difficiles du domaine DSP¹.

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

Le processus de la 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.

1. DSP : Digital Signal Processing (Processus du signal digital)

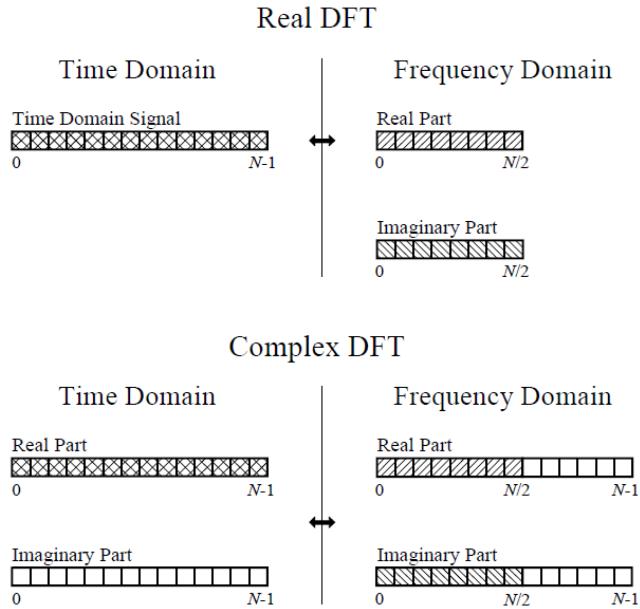


FIGURE 2.1 – Complex DFT

En 1822, Jean Joseph Fourier montra que certaines fonctions semi-périodiques peuvent également être formulées comme une somme d'harmoniques :

$$^2x(t) = c_0 + \sum_{n=1}^{\infty} c_n \cos(\omega t + \theta_n) \quad (2.1)$$

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 :

$$^3F[x(t)](\omega) = \int_{-\infty}^{+\infty} e^{-j\omega t} x(t) dt \quad (2.2)$$

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

2. Curtis Roads, *The Computer Music Tutorial*, 1996. [p. 1085]

3. Hermann L F. Helmholtz, *The Sensations of Tone*, 1895. [p. 215]

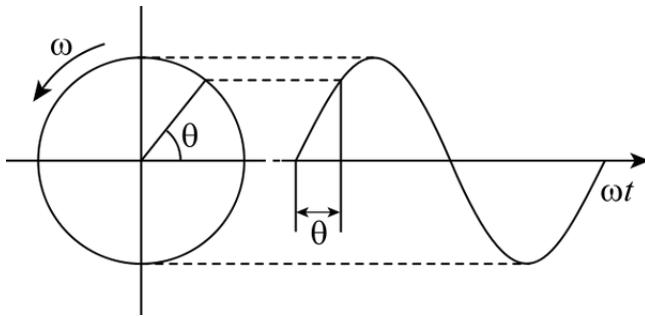


FIGURE 2.2 – Circle de la transformation Fourier

d'échantillonnage (SR). Le SR est généralement 44100 ou 48000Hz. Le domaine de cette fonction pouvant être intégrée aux signaux sonores numériques varie dans l'intégrale $[-1, 1]$. Comme on le voit clairement dans cette formule, le facteur 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 transforme souvent les données cartésiennes induites par l'exponentielle complexe en une forme polaire plus manipulable. Après la transformation de Fourier, il existe deux facteurs à manipuler, la phase et la magnitude. La phase est déduite 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 précis 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

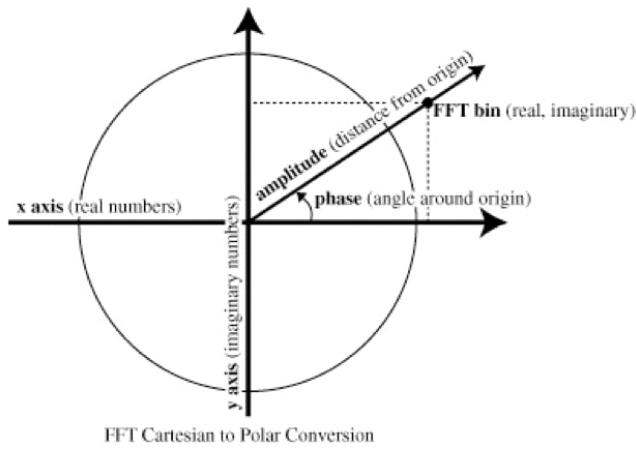


FIGURE 2.3 – Magnitude et phase

$$\theta(x) = \tan^{-1}\left(\frac{i(x)}{r(x)}\right). \quad 2.3$$

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, du fenêtrage. La fenêtre est une période de temps exprimée en images. Les images sont une notion équivalente du SR en le sens que le SR calcule une quantité des trames du son par minute et que les images sont exactement ces trames. Une analyse fenestrée est généralement exprimée par un algorithme de transformation à court terme de Fourier (STFT)⁴.

$$^5 X(\omega, \tau) = \sum_t^{\infty} x(t) \omega(t - \tau) e^{-j\omega t} \quad (2.3)$$

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

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

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

efficiente que d'autres méthodes :

$$^6X(\omega) = \frac{1}{N} \sum_{n=0}^{N-1} x(n+1)e^{-j\frac{\omega n}{N}} \quad (2.4)$$

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.

$$^7x(n) = \frac{1}{N} \sum_{f=0}^{N-1} X(f)e^{j\frac{2\pi fn}{N}} \quad (2.5)$$

Transformation Fourier rapide (FFT)

Or, la formule de Fourier générale n'est pas représentative dans le monde informatique. Pour autant, un point de vue théorique reste essentiel mais l'objectif est de se concentrer sur 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 un code informatique.

6. Jean-François Charles, *A tutorial on spectral sound processing using maxmsp and jitter*, 2008. [pp. 87–102.]
 7. Alan V. Oppenheim et Ronald W. Schafer, *Discrete-time signal processing*, 2009. [pp. 822 - 835]

```

1   function DFT(x)
2       N = length(x)
3       # On voit deux vecteurs , un pour l'espace reel (n) et un pour l'espace
4       de frequence (k)
5       n = 0:N-1
6       k = n'
7       transform_matrix = exp.(-2im*pi*n*k/N)
8       return transform_matrix*x
9   end.

```

Listing 2.1 – DFT

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 Tukey⁸

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 “Papillon” 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 fenetre, les calculs de l'exponentielle complexe sont considérablement réduits. Nous pouvons visualiser le processus dans la figure 2.4⁹.

8. James Cooley et John Tukey, *An algorithm for the machine calculation of complex Fourier series*, 1965.

9. 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.

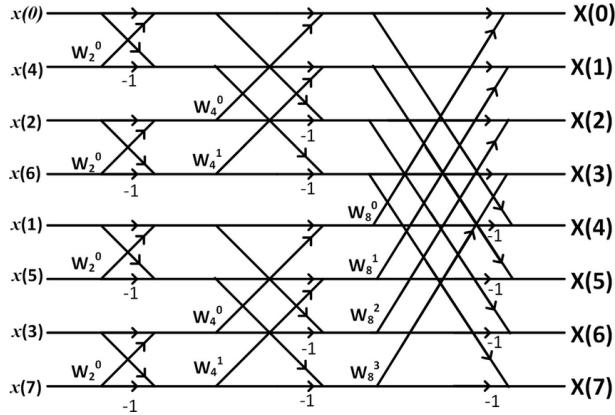


FIGURE 2.4 – Diagramme papillon pour une FFT à 8 points

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

2.2.2 Fenêtrage

Pour calculer la STFT, il faut définir la taille de la fenêtre à traiter. La transformation de Fourier dans une fenêtre temporelle discrète peut produire des artéfacts 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 entraîne la répétition à plusieurs reprises de la fenêtre pendant la durée totale du son. Il est important que la fenêtre soit déplacée dans le temps, mais d'être également chevaucher sur lui même par un facteur de décalage. 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 de l'enveloppe, l'auditeur ne peut percevoir aucun des artéfacts possibles induits par l'analyse mais on entend un résultat unifié¹⁰.

10. Daniel W. Griffin et Jae S. Lim, *Signal estimation from modified short-time fourier transform*, 1984.

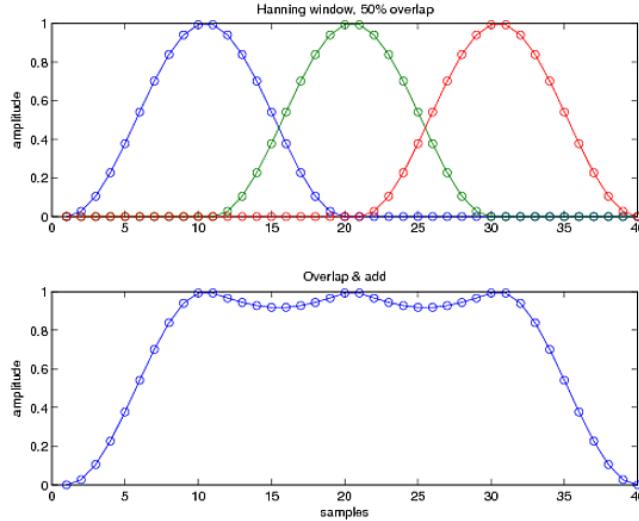


FIGURE 2.5 – Overlapping

Filtres Gabor

Nous examinerons, ici, la possibilité d'utiliser des filtres de Gabor pour appliquer une 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 suivante¹¹ :

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

Il est possible de considérer un filtre de Gabor comme deux filtres déphasés qui sont un produit direct de la décomposition de l'exponentiel complexe. L'un correspond à la partie imaginaire et l'autre à la partie réelle. Où la partie réelle a le filtre $g_{real}(t) = w(t)\sin(2\pi f_k t + \theta)$ et la partie imaginaire $g_{imag}(t) = w(t)\cos(2\pi f_0 t + \theta)$.

Les deux filtres peuvent être déphasés mais la phase est 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

11. Dennis Gabor, *Theory of Communication*, 1994

normalisée se transforme en¹² :

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

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'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 sons¹³.

Histoire

Le vocodeur de phase a été introduit, en 1966, par Flanagan comme 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

12. Javier R. Movellan, *A tutorial on gabor filters*, 2008.

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

après traitement a été proposée par Griffin et Lim¹⁴ en 1984. Cet algorithme n'avait pas comme objectif de produire une STFT cohérente, mais plutôt de rechercher le signal optimal afin que la STFT soit aussi proche que possible de la STFT modifiée, même si la STFT modifiée n'est pas cohérente (ne représente aucun signal).

Jusqu'en 1999, la question de la cohérence verticale était un problème majeur pour la qualité opérationnelle à l'échelle temporelle. A cette époque, Laroche et Dolson¹⁵ ont proposé un moyen de préserver la cohérence de la phase des composantes spectrales. Le développement de Laroche et de Dolson a marqué 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. En 1999, Roebel a proposé une solution à ce problème¹⁶. Un exemple, de mis en oeuvre du vocodeur de phase utilisant la dernière version de Roebel, est constitué par les outils SuperVP de l'Ircam.

Une manière pour décrire le vocodeur de phase consiste à représenter le signal par une séquence d'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). Les parties d'overlap et éventuellement un fenêtrage sont additionnées, ce qui donne le résultat final.

La STFT (une transformation caractérisée des successions des TFD) d'un signal fenêtré est

14. Daniel W. Griffin et Jae S. Lim, *Signal estimation from modified short-time fourier transform*, 1998. [pp. 236–243]

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

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

défini comme suit :

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

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

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

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 fenêtrage 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.

Par conséquent, 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 Streching

Afin de 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. Afin de visualiser le résultat, on peut imaginer, pour chaque période de temps de la transformation Fourier appliquée, qu'une série d'harmoniques seront sauvegardées dans une fenêtre, ainsi que son facteur d'overlap. Pour changer le rythme de la lecture sans affecter la hauteur sonore, Il suffit d'éloigner ou de rapprocher les fenêtres.

Le modèle qui correspond à l'étirement temporelle est donnée par la formule suivante :

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

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

Ou 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 $f_k(n)$.

Le dernière est équivalent 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 devient¹⁷ :

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

Transposition de la 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 une 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.¹⁸

Freeze

À cet effet, nous prenons une certaine 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 à zéro. Cela peut se comparer à un étirement sonore infini. On peut ainsi appliquer les mêmes principes d'un étirement sonore normal.

17. Jean Laroche et Mark Dolson, *Improved Phase Vocoder*, 1999. [p. 324]

18. Mark Dolson, *The phase vocoder*, 1986.

Robotisation

Pour faire une robotisation d'un signal, il faut mettre la phase de chaque échantillon à zéro. Cet effet résulte d'un son robotisé et métallique.

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 FFT¹⁹.

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, qui est appelée, un facteur de morphing ou une interpolation. Ce processus dépend, par ailleurs, du facteur temporel puisque le morphing est un phénomène dynamique.

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

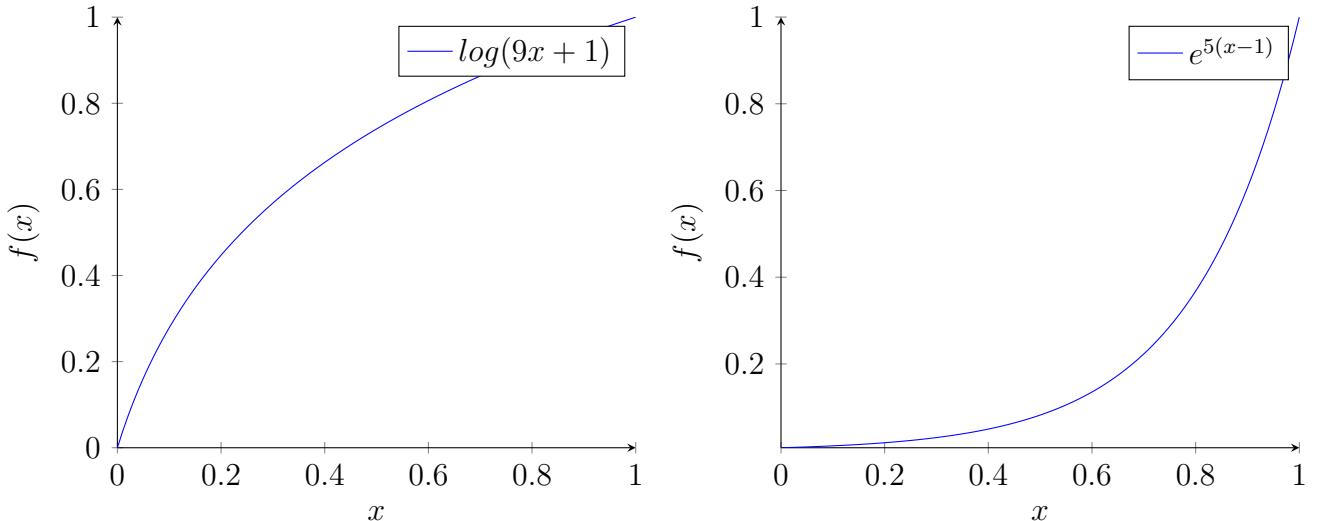
Ou $\alpha(t) \in [0 : 1]$

La manipulation du paramètre α nous permettra de changer la dynamique du morphing. Traditionnellement, la fonction de manipulation agit d'une fonction linéaire. En extension, on peut s'attendre d'un autre type de fonction de manipulation que d'une fonction 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 à la variable temporelle. Le son est un phénomène dynamique, donc il ne peut pas être interpolé linéairement.

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

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

FIGURE 2.6 – Interpolation du facteur α 

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.

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 Serra²¹. 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.

$$^{22}s(t) = \sum_{n=0}^N A_n(t) \cos(\theta_n(t)) + \epsilon(t) \quad (2.20)$$

Où A_n et θ_n sont respectivement l'amplitude et la phase de la n-ième fréquence. En supposant

21. Curtis Roads et autres, *Traitements du signal musical*, 1997. [p. 91 - 122]

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

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 .

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 prédis. 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 :

$$^{23} Err_{total} = Err_{p \rightarrow q}/N + Err_{q \rightarrow p}/K \quad (2.22)$$

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

Chapitre 3

Design

Dans ce chapitre, nous allons implémenter, dans le language du logiciel MaxMSP, les formules mathématiques présentées précédemment. De plus, nous allons introduire d'autres effets spectraux et des variations dans le contexte du vocodeur de phase.

Par ailleurs, nous allons présenter les objets MaxMSP qui encapsulent¹ des fonctions mathématiques telles que FFT, IFFT et d'autres transformations. On peut attendre de ce partie qu'il permet 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 Objets MaxMSP

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 signaux, 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 signaux. Ils sont déployés dans un environnement spécial appelé *PFFT~*. Le PFFT contient, fréquemment, une FFT et une IFFT. La majorité des objets dans le PFFT traitent des données bidimensionnelles, par conséquent, ils sont traités dans un patch différent.

FFT~

L'objet *fft~* appartient à la famille de signaux, comme le précise l'indicateur ~. C'est l'objet qui effectue la transformation rapide de Fourier. Il n'existe aucune entrée mais il y a trois sorties, une pour la partie réelle de l'exponentielle complexe, une pour la partie imaginaire et un compteur qui garde l'index des harmoniques de la transformation de Fourier.

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 l'harmonique 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.

cartopol~

Il n'est généralement pas utile d'utiliser ces coordonnées cartésiennes, produites par le *fft~* correspondant aux parties réels et imaginaires du plan \mathbb{C}^2 . Par conséquent, un objet transformant les coordonnées cartésiennes en une forme polaire appelée *cartopol~* est fréquemment déployé. Cet objet a donc 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 il fournit des informations sur les paramètres de la FFT, tels que la taille de la fenêtre, etc.

framedelta~

framedelta~ est un objet très important car il calcule la dérivation de phase entre des images successives de la FFT. Également l'objet *phaseaccum~* peut être utilisé à la place de *framedelta~*. *phaseaccum~* qui calcule directement la phase après la derivation.

gen~

gen~ est un objet qui permet de créer une nouvelle fenêtre spéciale d'un patch. Dans cette fenêtre, l'utilisateur peut utiliser un nouvel ensemble de fonctions spécialisées sur 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 Détection du pitch

²

En utilisant certains de ces objets spectraux MaxMSP, nous allons commencer par construire un simple patch afin de trouver la hauteur de la fréquence dominante³ :

«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 ». ⁴

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 l'harmonique 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 entre 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.

2. Pitch signifie un terme équivalent à la hauteur ou bien à la fréquence

3. Emmanouil-Nikolaos Karystinaios, *An investigation into the spectral music idiom and timbral analysis functionality with Max SMP Jitter*, 2017.

4. Curtis Roads, *A tutorial in musical signal processing*, 1996. [p. 513],

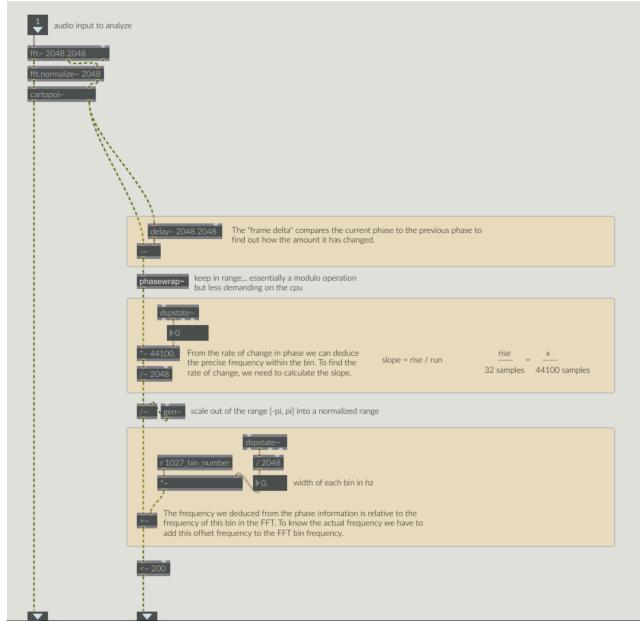


FIGURE 3.1 – Pitch Tracking

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 de chaque bin, 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.

Tout d'abord, nous retardons la phase de chaque bin d'une fenêtre entière pour calculer le montant de sa dérivation. Ensuite, 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 consulter nos paramètres sonores. Un objet dans Max appelé *dspstate~* récupère toutes les données nécessaires que nous pouvons trouver dans */Options/Audio Status* dans l'application Max. Nous avons seulement besoin de la fréquence d'échantillonnage. Cette valeur est généralement 44100Hz donc on la stocke par avance et si elle différente nous la changerions automatiquement. Le signal obtenu à partir de *phasewrap~* est multiplié par la fréquence d'échantillonnage et par la suite 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 la fréquence classe la deviation des partitions fréquencantiels. Plus précisement, on divise le spectre audible dans des parties isometriques qui sont determinés par la division du SR par la taille de la fenêtre. 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 bins 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 ci-dessous.

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 un code sous la forme «traditionnelle ». Tout comme le code javascript, nous devons déclarer les variables que nous allons utiliser, qui sont suivant aussi les entrées de cet 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

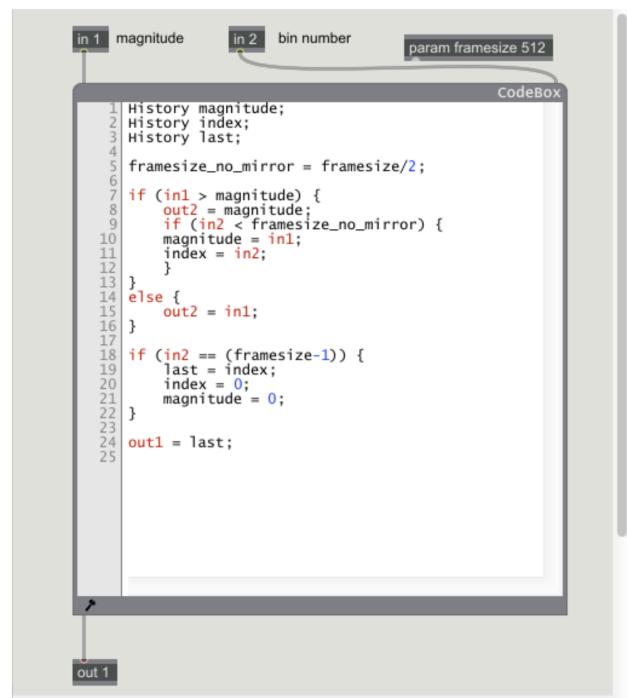


FIGURE 3.2 – Codebox~

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 du n -ième bin est supérieure à celle du $(n - 1)$ -ième bin, 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~*. *Sah~* signifie «sample holder» (stockage d'un échantillon). Avant cela, l'index actuel est comparé à la valeur d'index générée par l'objet *gen~*. 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 l'harmonique avec la plus grande magnitude va de paire avec la magnitude et la phase correspondante. 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 3.3. Cette méthode évite de calculer chaque fois la fréquence si le son est périodique et donc la fréquence dominante ne change pas.

3.2.1 Détection des pitchs multiples

Pour créer un patch de suivi des hauteurs multiples, nous utilisons la méthode du suivi 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.

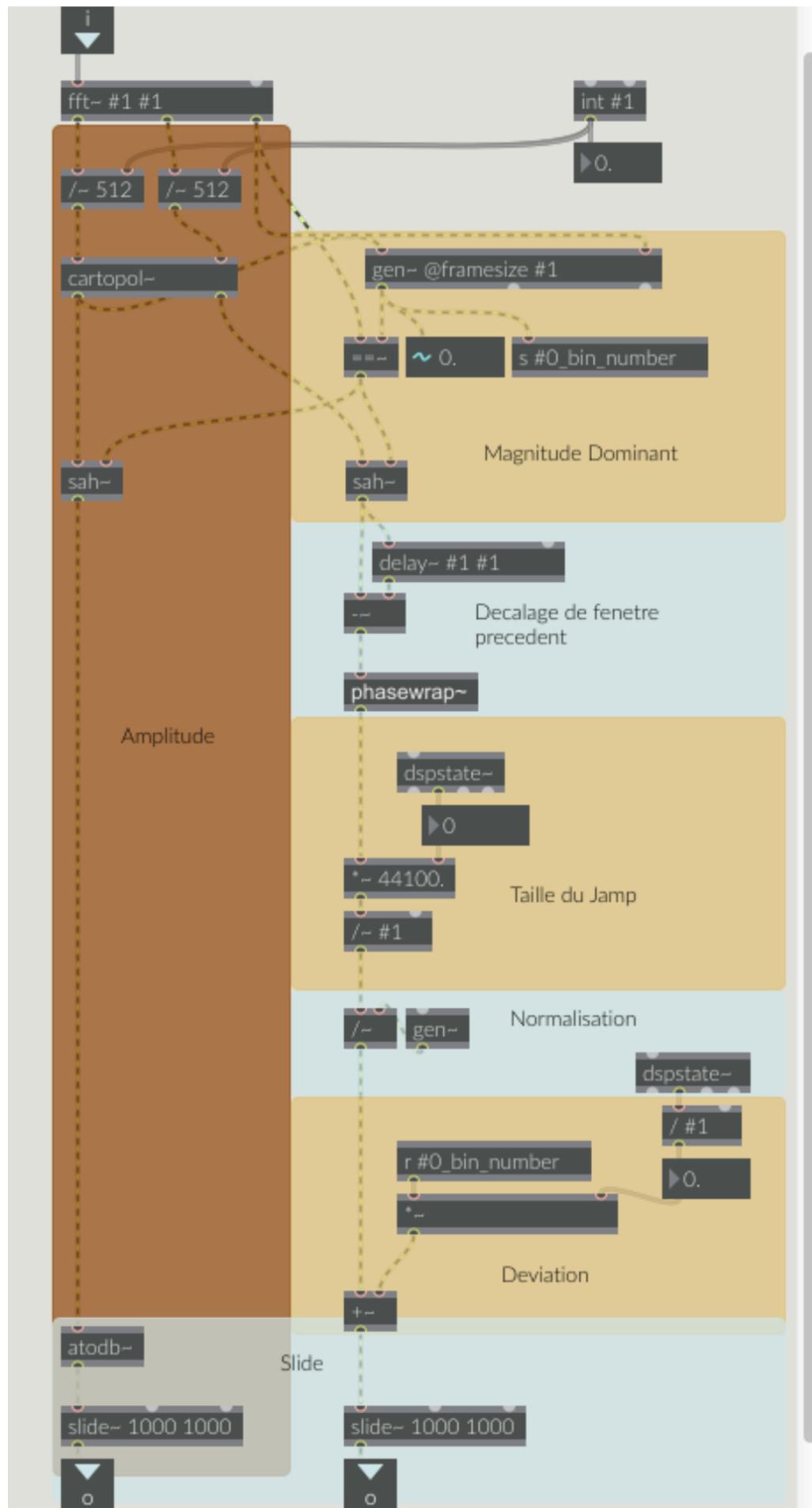


FIGURE 3.3 – Pitch Tracker



FIGURE 3.4 – Gen multiple

La figure 3.4 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 3.4, 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 des harmoniques que nous allons afficher.

La mise en oeuvre de la méthode de la localisation des plusieurs hauteurs peut être montrée à la figure 3.5. Nous reproduisons essentiellement le calcul de fréquence pour chaque indice. Le résultat peut être affiché sous forme de liste ou dans différents sorties. Ensuite, on peut utiliser l'objet *mc.cycle~* dans la version Max8, de simples oscillateurs ou résonateurs dans les autres cas.

Afin de construire au mieux le patch, il est important de noter que la taille de la fenêtre affecte la qualité de la détection de la fréquence. La taille de la fenêtre influence la résolution temporelle ou fréquentielle de la représentation du signal. La résolution de 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

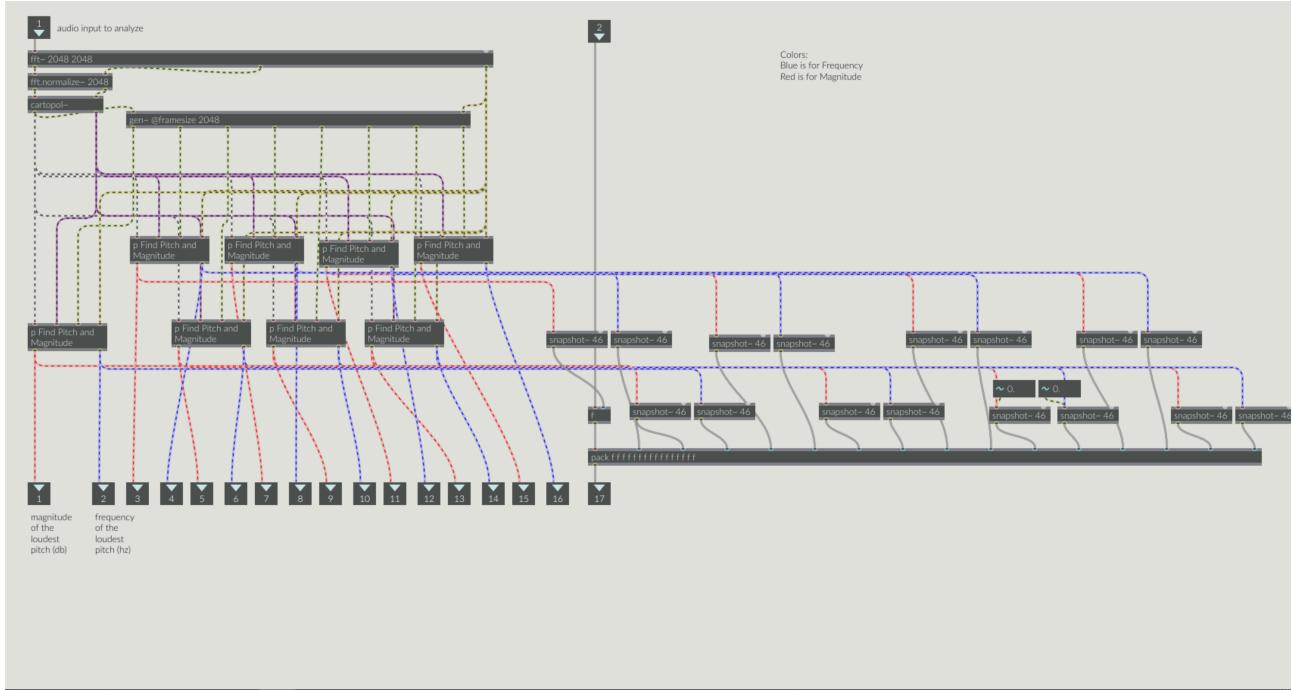


FIGURE 3.5 – Multiple Pitch tracking

dispersion des fréquences est définie comme la division du SR par la taille de la fenêtre⁵.

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

3.3 Le vocodeur de phase

Le vocodeur de phase est utilisé à l'origine pour la transposition de la hauteur et la modification de la vitesse de lecture. Pareillement, nous allons construire notre vocodeur de phase sur 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~*. Notre modèle est basé sur le vocodeur de phase par Dudas et Lippe⁶.

Pour accéder à un son directement dans l'objet *pfft~*, il faut éviter d'utiliser les prérglages du *pfft~*. Par conséquent, nous allons éviter d'utiliser l'objet *fftin~* à la place, nous allons utiliser un objet *fft~* dans un sous-patch. La seule différence est que les paramètres des objets *fftin~*

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

6. Richard Dudas et Cort Lippe, *The Phase Vocoder*, 2006.

et *fftout~* sont contrôlés directement à partir de l'objet *pfft~* 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. Pour comprendre la procédure on pourrait suivre le patch fourni dans la figure 3.6.

Le raisonnement derrière cette logique se repose sur le cœur du vocodeur de phase pour l'éti-rement sonore et le changement de la 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~*, mais plutôt dans un objet *fft~*. L'objet *fft~* exécute une FFT à spectre complet (c'est-à-dire en miroir), donc *fft~* peut fonctionner en synchronisation avec les images consécutives de la FFT traitées dans l'objet *pfft~* mais il est nécessaire de faire quelques modifications sur l'objet *pfft~* pour qu'il se comporte de la même manière.

Tout d'abord, l'objet *pfft~* 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 de la FFT (jusqu'à la moitié de la fréquence de Nyquist). Cela peut être réalisé en ajoutant un cinquième argument non nul à l'objet *pfft~*. Comme l'argument du spectre complet est le cinquième argument, nous devons fournir tous les autres arguments avant lui, y compris le quatrième argument (la position où la FFT s'execute) qui sera défini par défaut par zéro.

Ensuite, puisque les objets *fftin~* et *fftout~* 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~* et *ifft~* objets exécutent la FFT déphasée de 180 degrés, il faut assurer que tous les objets *fftin~* et *fftout~* du patch ont le même décalage de phase FFT utilisé dans les objets *fft~*.

Cela peut être accomplie 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, l'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. Cela veut dire qu'on utilisera un enregistrement et on le stockera dans un objet *buffer~*. Le *buffer~* doit être accessible à deux endroits. Premièrement, à l'emplacement de l'image sonore de la FFT actuelle et, en deuxième lieu, à l'emplacement de l'image FFT précédente du *buffer~*. Il est possible d'utiliser soit l'objet *index~* ou l'objet *play~*⁷ pour accéder au *buffer~*. Lorsqu'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 est d'utiliser deux objets *fft~* parallèles avec un décalage d'un quart, pour un chevauchement de 25%. Un tel système permettra de calculer l'image actuelle de chaque index de la fenêtre en comparaison de l'index correspondant de la fenêtre précédente, comme cela a pu être 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 $-\pi$ qu'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 la 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 réinitialise à zéro, un objet *sah~* permet à la valeur de passer, dont on en déduit le premier sample de la FFT. En ajoutant les valeurs du compteur, on peut en déduire la position actuelle sur le fichier sonore. La valeur de la position sur le fichier nous permet de jouer le fichier à partir d'un objet *play~*.

On rappelle la formule du vocodeur de phase :

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

7. 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.

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. 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 du *hop*, uR_a . Les échantillons sont multipliées, 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 est incluse 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 buffer fixé de taille N . En utilisant l'objet *count~* avec l'aide de l'objet *index~*⁸, pour accéder à chaque échantillon, on multiplie le son de l'entre par les valeurs du fenêtrage avant d'effectuer de 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 la fonction gaussienne tout en conservant les constantes de valeurs maximales et minimales. Cette fenêtre gaussienne faite sur mesure est présentée en figure 3.7.

Nous utilisons un objet *uzi*⁹ avec argument la taille de la fenêtre FFT, suivie de l'expression effectuant une courbe gaussienne normalisée. Ensuite, nous l'envoyons dans le buffer nommé

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

9. Envoyer instantanément le nombre de «bangs» qui ont été définies dans la position de l'argument de l'objet.

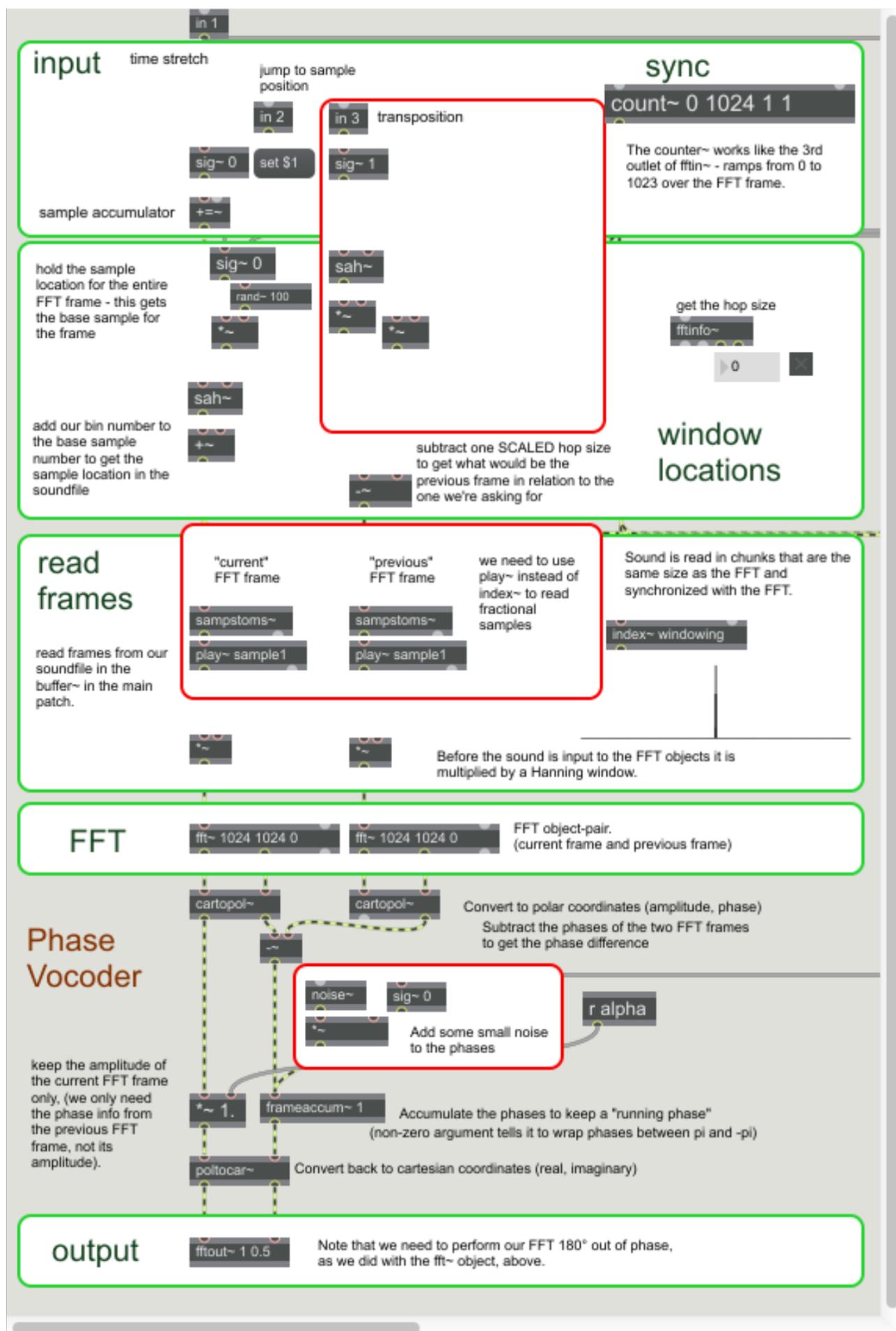


FIGURE 3.6 – Le vocodeur de phase

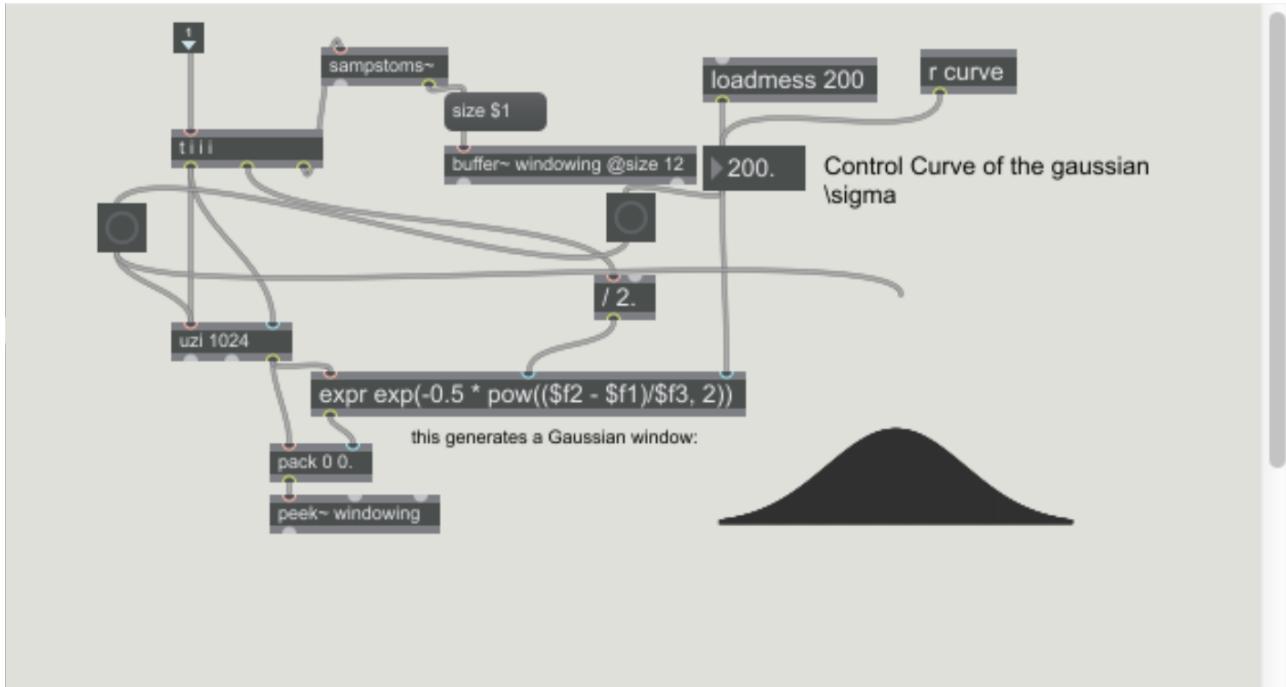


FIGURE 3.7 – Fenêtrage gaussien

«windowing» à l'aide de l'objet *peek~*¹⁰. 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.

Trasposition de la hauteur

Pour appliquer la trasposition de la hauteur avec le vocodeur de phase, il convient de modifier le facteur de la hauteur tout en modifiant simultanément la fréquence d'échantillonnage. Par exemple, afin de transposer un son d'une octave plus basse, nous devons reproduire le son à une vitesse deux fois moins rapide, tout en doublant le SR. En utilisant cette méthode, il est possible de changer la hauteur en modifiant la vitesse de lecture.

Pour utiliser cette méthode, nous utilisons des valeurs MIDI standard traduites en fréquence et adaptées au chevauchement des fenêtres. L'utilisateur peut ensuite saisir la valeur dans un piano midi virtuel, facilitant ainsi la manipulation. Par défaut, la note midi Do avec la valeur 60 est la hauteur standard. Ensuite, la hauteur change en fonction de l'intervalle en fonction

10. L'objet *peek~* est utilisé pour lire et écrire des valeurs des échantillons dans un *buffer~*

de la valeur par défaut. Par conséquent, la valeur Do une octave inférieure dans le piano midi virtuel correspond à une octave de transposition inférieure à la hauteur originale du son.

À l'intérieur du vocodeur de phase, le décalage de hauteur est traduit par le saut d'échantillons ou le suréchantillonnage de l'objet *count~*. Bien sûr, augmenter le SR n'est pas une solution valable, nous avons donc choisi de prendre une portion plus petite ou plus grande du *buffer~* correspondant à la taille de la fenêtre. Une transposition vers le haut correspond au fait de prendre une fenêtre plus grande dans *buffer~* et de la lire à une vitesse plus rapide, tandis qu'un décalage de hauteur correspond à couper une plus petite partie de la variable *buffer~* et à la lire plus lentement.

Pour cet exemple, nous utilisons un objet *sah~* pour nous assurer que la valeur de transposition est maintenue constante pour tous les bins pendant la FFT. Ensuite, nous multiplions par la valeur actuelle du compteur et nous ajoutons la valeur de sortie au nombre d'échantillons déjà lus pour obtenir l'emplacement souhaité du fichier. En ce qui concerne la fréquence initiale ω , le décalage individuel s'élève à :

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

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.

Modulation Temporelle

Comme il a été étudié à la section 2, modifier la vitesse de lecture tout en gardant la hauteur du pitch stable est l'opération inverse de la transposition de la hauteur. 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 sur la position de l'interprétation de chaque échantillon. Une lecture plus rapide peut enjamber des échantillons où une vitesse de lecture plus lente lit plusieurs fois chaque échantillon. 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 parallèles et nous interpolons la phase et l'amplitude entre toute indice du bin. Pour comprendre la complexité computationnelle 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 une prix computationnelle relativement haute, mais il reste possible de le calculer avec les ordinateurs actuels.

Dans ce double vocodeur de 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 l'harmonique précédente et une petite quantité de bruit est ajoutée pour la naturalisation du résultat sonore. 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. En allons plus loin, nous implementons 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.

La version du vocodeur de phase pour le morphing spectral est montrée dans la figure 3.8. 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.

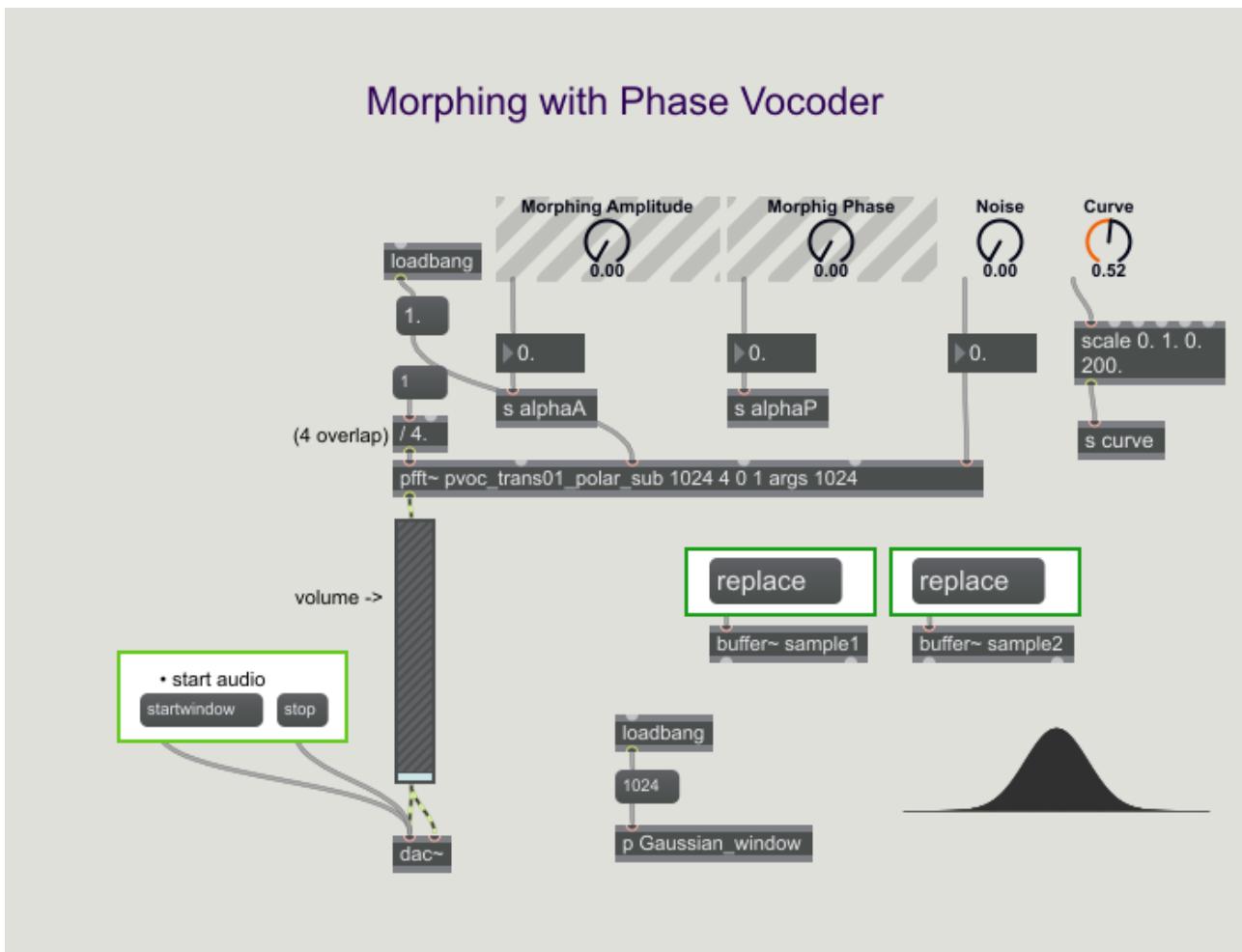


FIGURE 3.8 – Morphing en temps réel

```
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}
```


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. Pour finir, on va utiliser un sous-ensemble du code écrit pour composer une pièce démonstrative de l'univers sonore abordée par le vocodeur de phase.

L'objectif de ce chapitre est de soutenir l'idée que l'analyse spectrale et la re-synthèse spectrale, sont utiles aux scientifiques et aux artistes. Donc, par extension, le développement du processus informatique du signal musical peut l'être aussi. 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.

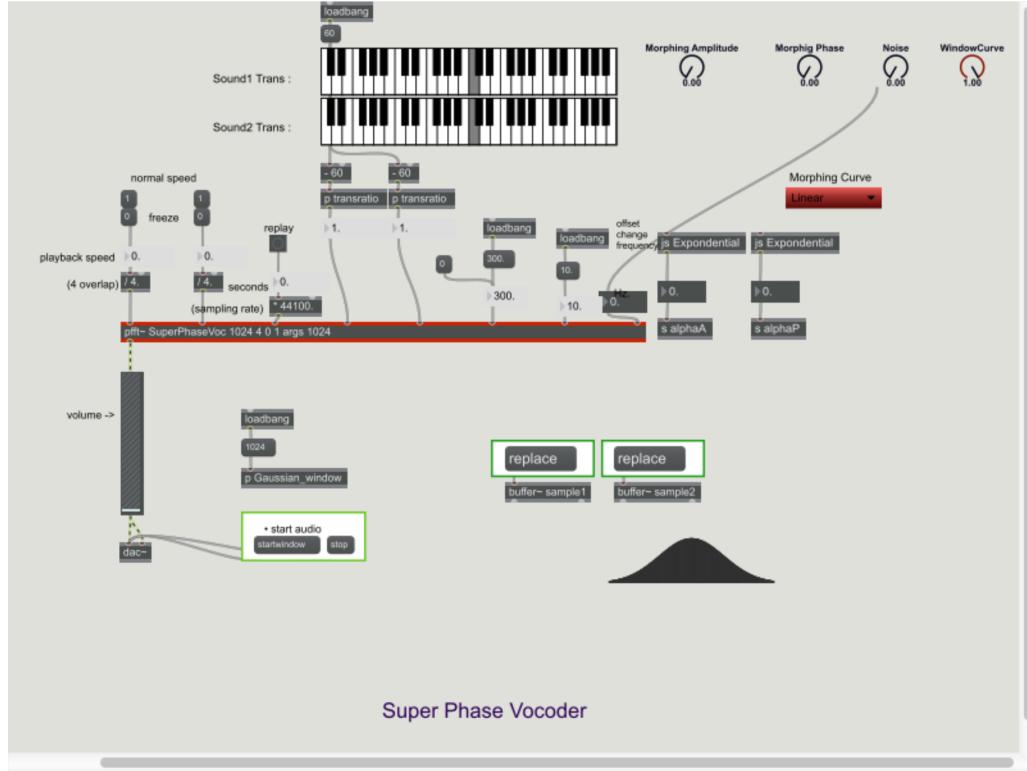
4.2 Le vocodeur de phase - *Une capacité sans fin*

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.

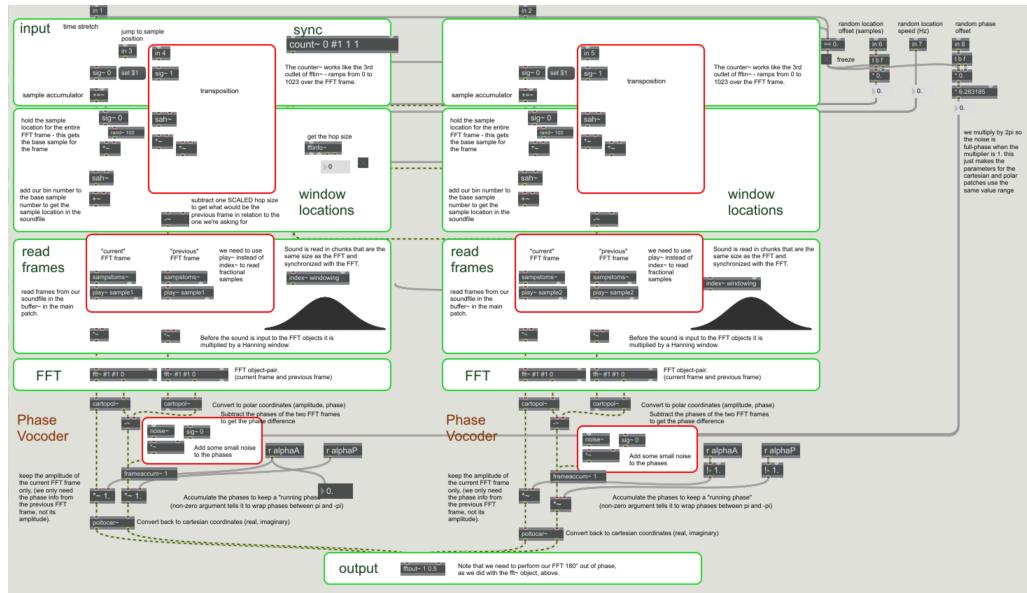
Le graphe 4.1a présente le patch de contrôle de tous les paramètres du vocodeur de phase. Cette version du vocodeur de phase contient tous les effets étudiés dans les sections précédentes. La différence entre le patch du vocodeur de phase de base pour le morphing est minime. Essentiellement c'est le même code avec quelques fonctionnalités supplémentaires. 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 formules mathématiques correspondantes à ce partie sont omises car nous les avons déjà balayées 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.



Super Phase Vocoder

(a) Super Phase Vocoder I



(b) Super Phase Vocoder II

FIGURE 4.1 – Super Phase Vocoder

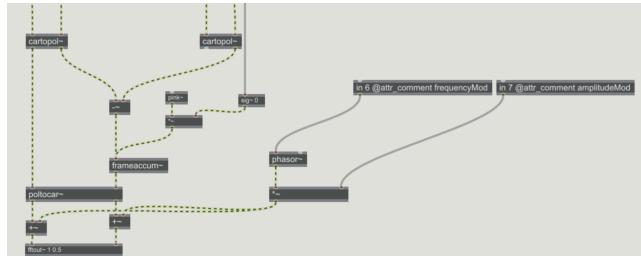


FIGURE 4.2 – modulation
des coordonées polaires

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.

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

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 phaseur 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 phaseur moduler la magnitude et la phase. La fréquence et l'amplitude du phaseur sont contrôlées dans le patch principal contenant l'objet *pfft~*.

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~*, pink-noise, white-noise et au même moment 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 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 aléatoire 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

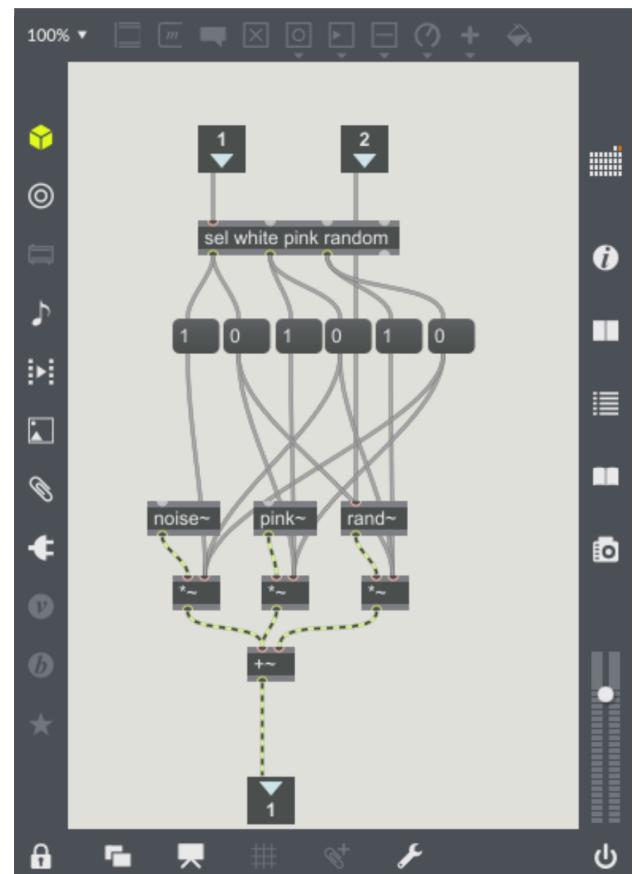


FIGURE 4.3 – Selection du bruit

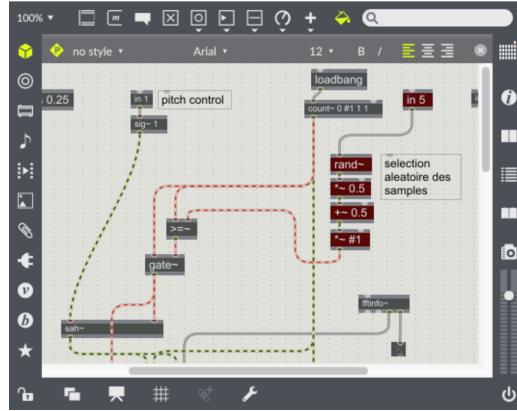


FIGURE 4.4 – Filtre aléatoire

générée de manière aléatoire. Le porte-échantillon est créé manuellement par 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 4.4.

4.2.5 Robotisation

La robotisation est achevée par une modification sur la phase de chaque harmonique du vocodeur de phase pendant l'analyse. En effet, en mettant la phase de tout harmonique à zéro on garde que la magnitude. Cette technique permet d'avoir une phase incorrecte constante pour les partiels du son d'entrée mais avec une amplitude correspondante variée. L'implémentation est montré dans la figure.

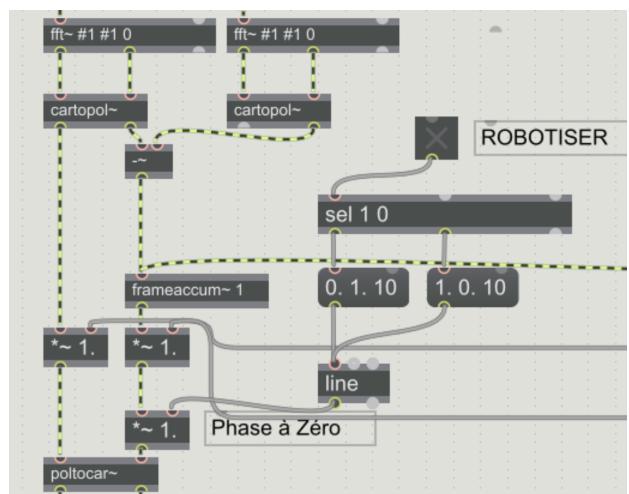


FIGURE 4.5 – Robotisation

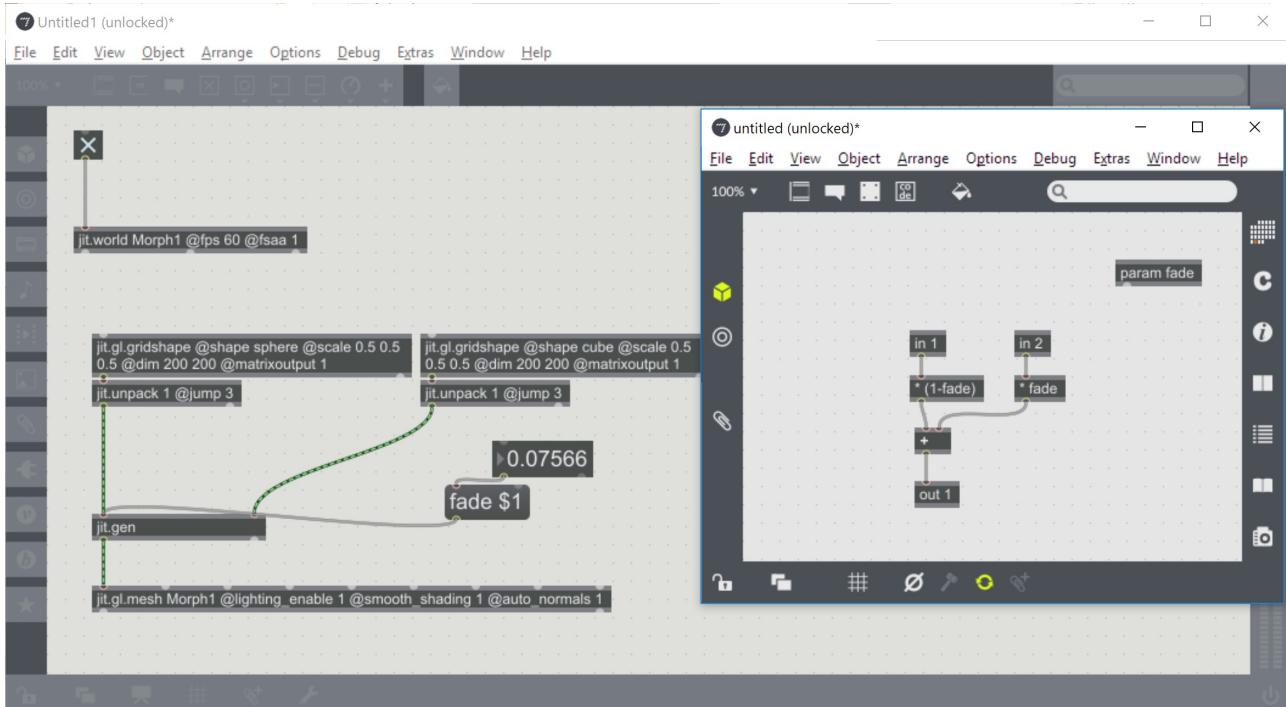


FIGURE 4.6 – Visual Morphing

4.3 Morphing visuel

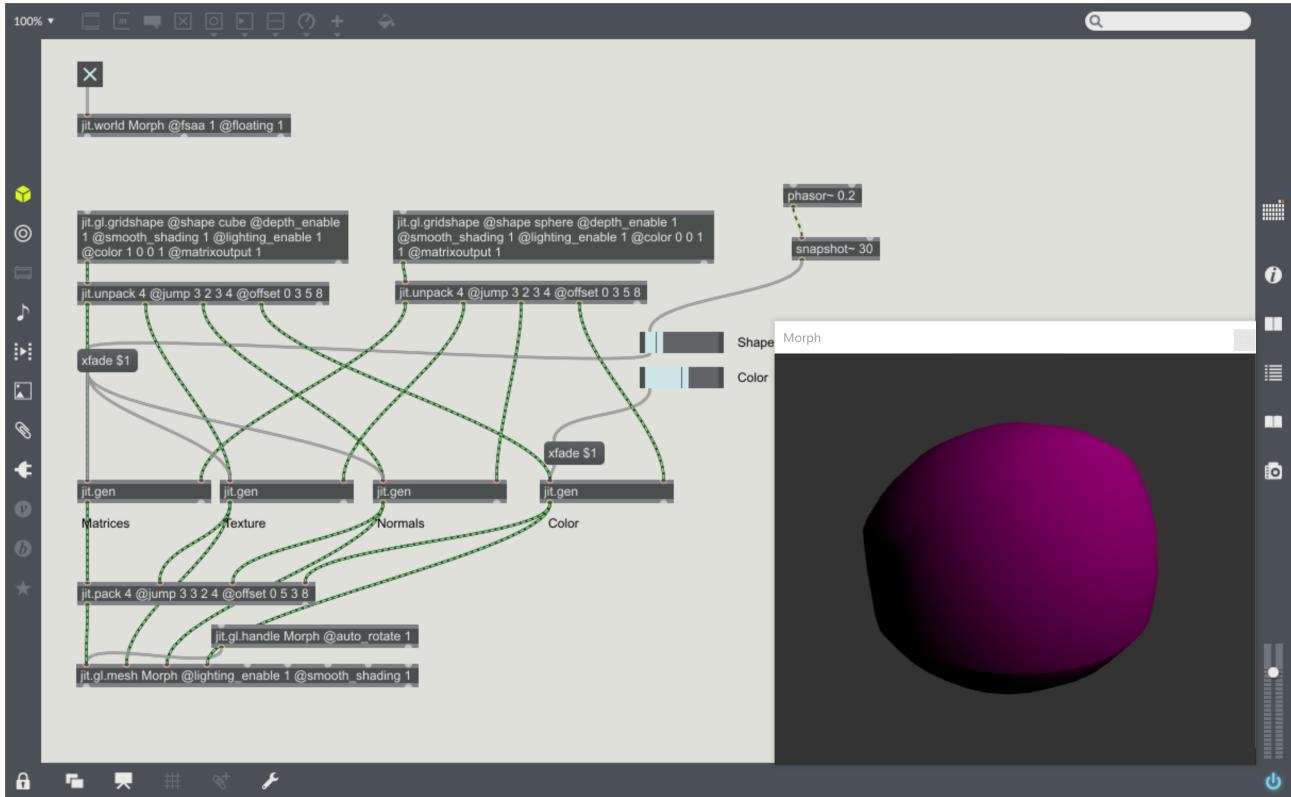
En avançant sur le morphing visuel, nous avons implémenté le patch suivant avec l'aide du Jitter, pour une interpolation linéaire entre deux dessins.

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 4.6).

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

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

FIGURE 4.7 – Visual Morphing 2^{me} version

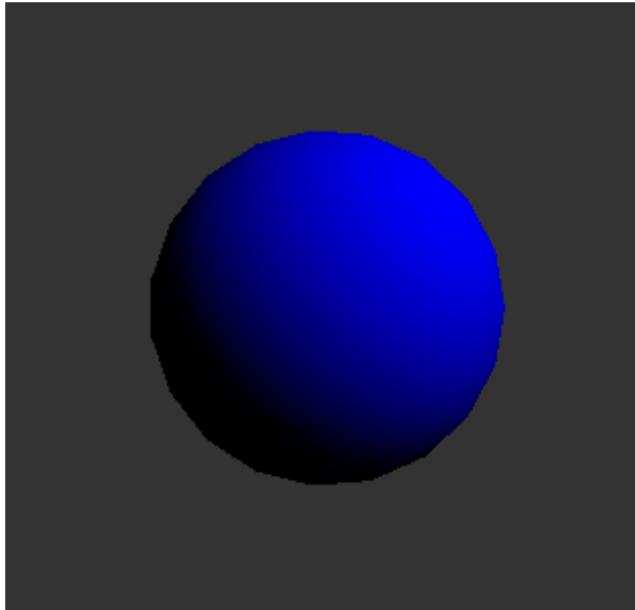
Dans le patch suivant, nous avons voulu aller plus loin, en séparant tout composant d'un modèle tri-dimensionnel dans jitter. Nous distinguons les valeurs des matrices, des normaux et de la texture. Nous avons ajoutés également une interpolation du couleur. Le patch est en disposition en figure 4.7.

4.3.1 Visualisation du spectre

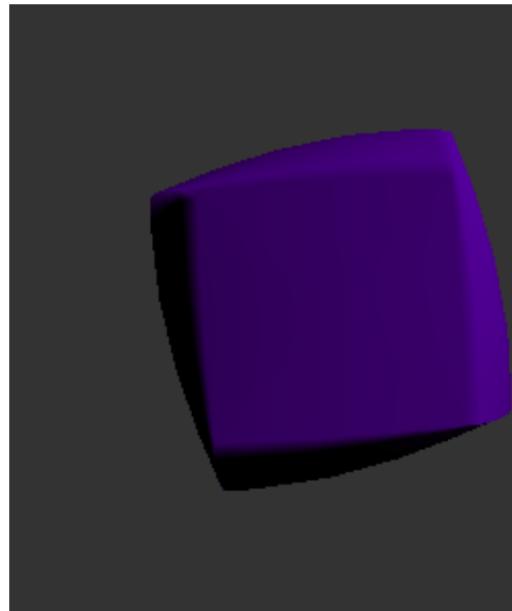
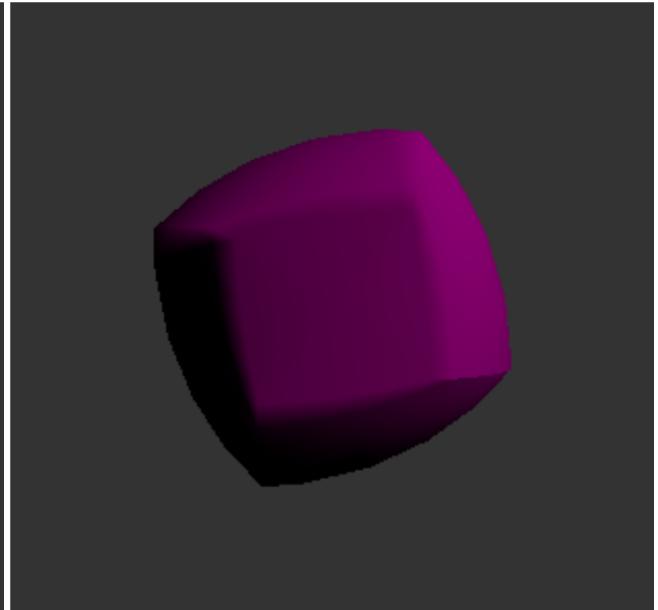
Dans ce patch, beaucoup de travail de Jitter a été implémenté à l'unisson d'un vocodeur de phase. Ce code permet de visualiser les informations spectrales d'un son sous la forme d'un paquet de lignes bidimensionnelles imitant les harmoniques trouvées lors d'une analyse de Fourier.

Ce patch est utilisé en parallèle d'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 jitter supplémentaire pour produire plusieurs objets jitter rendus dans un espace tridimensionnel virtuel. L'objectif de ce

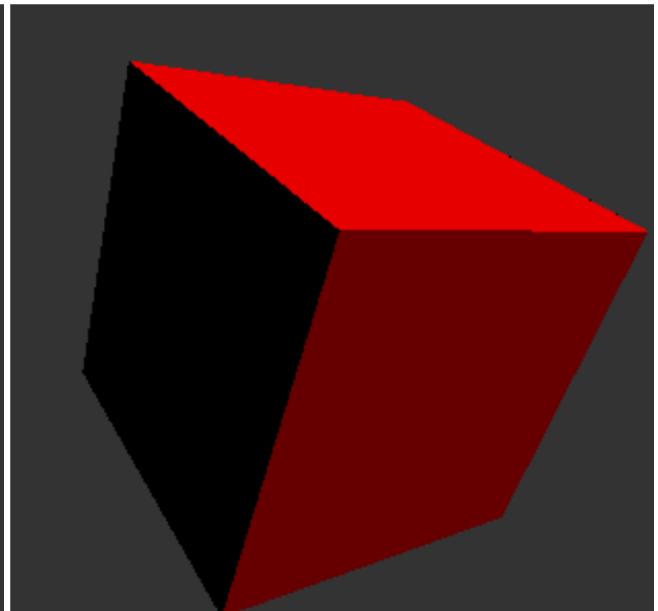
(a) Interpolation de la forme : 0.0
Interpolation couleur : 0.0



(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 4.8 – Stades du morphing visuel

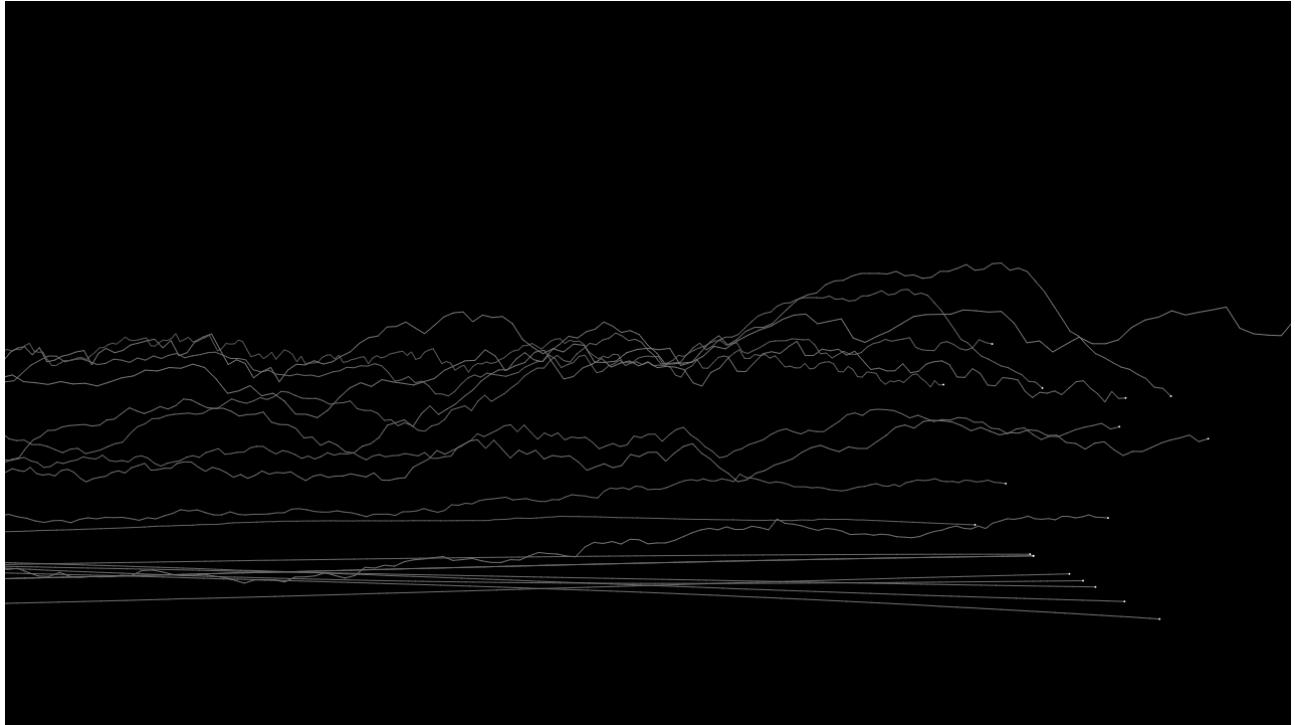


FIGURE 4.9 – Harmoniques dans l'espace

mémoire n'étant pas orienté sur l'image, nous ne dévoilerons, donc, que très peu d'informations sur le traitement présenté.

Nous allons attribuer ici brièvement les objets les plus emblématiques du Jitter 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 jitter. Et le *jit.catch* prend un signal et le traduit en matrice jitter 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.

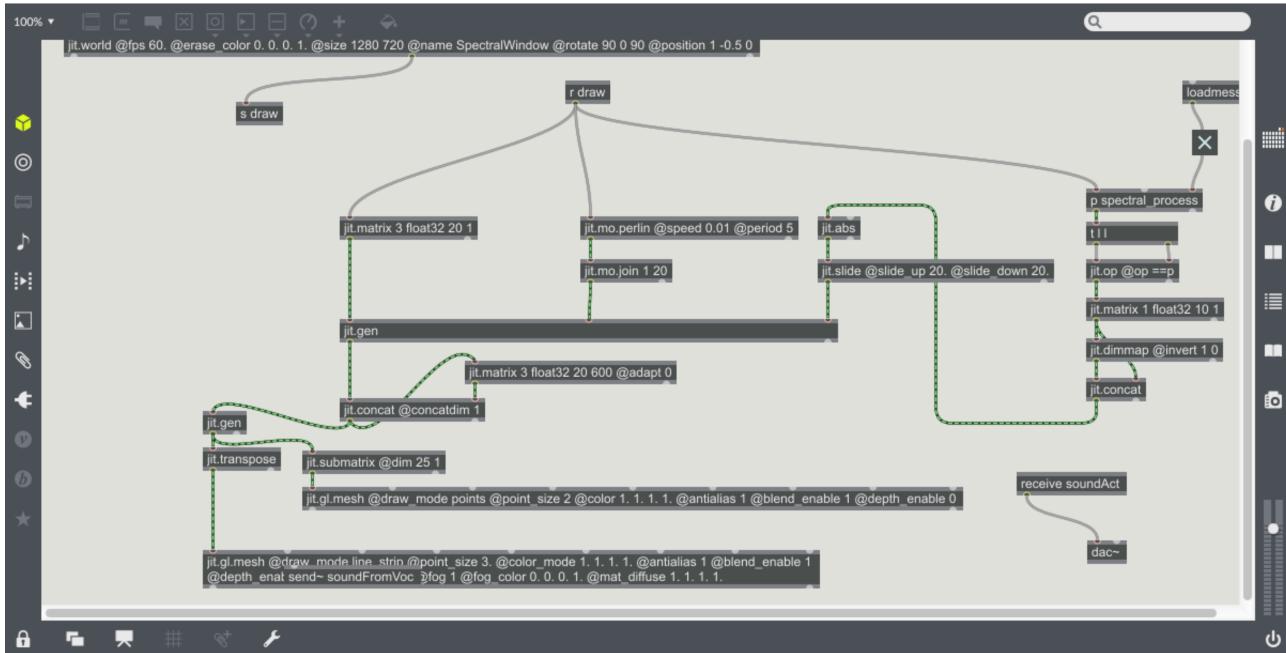


FIGURE 4.10 – Le patch qui control les objets jitter

4.4 La mise en œuvre compositionnelle

Dans le cadre artistique, on a créé une composition de courte durée afin de mettre en valeur l’importance artistique du vocodeur de phase via une source d’exemples fructueux. Le but de cette petite pièce de composition est d’utiliser le minimum d’information sonore (3-4 enregistrements de . < 10 secondes), afin de créer un univers sonore engendré par paramétrisation sur le vocodeur de phase présenté tout au long de ce mémoire.

Chaque son est produit dans MaxMSP en utilisant un des codes développés et présentés dans ce mémoire. Chaque son est un produit direct d’un ou deux sources sonores. Les sources sont des enregistrements de piano ou de la guitare effectués et joués par l’auteur. L’orchestration numérique de la pièce était réalisée sur le logiciel du mixage *Reaper*.

Les enregistrements des sons d’origine étaient faits avec un échantillonnage de $48kHz$ mais les enregistrements dans Max sont reconfigurés à $44,1kHz$. L’enregistrement sur MaxMSP est effectué par un *buffer~* et un simple objet *record~*.

La pièce composée agit d’une pièce acousmatique. L’idée de composition origine de la simulation de l’addition de deux sinusoïdes avec une fréquence différente, mais proche. Ce battement sonore

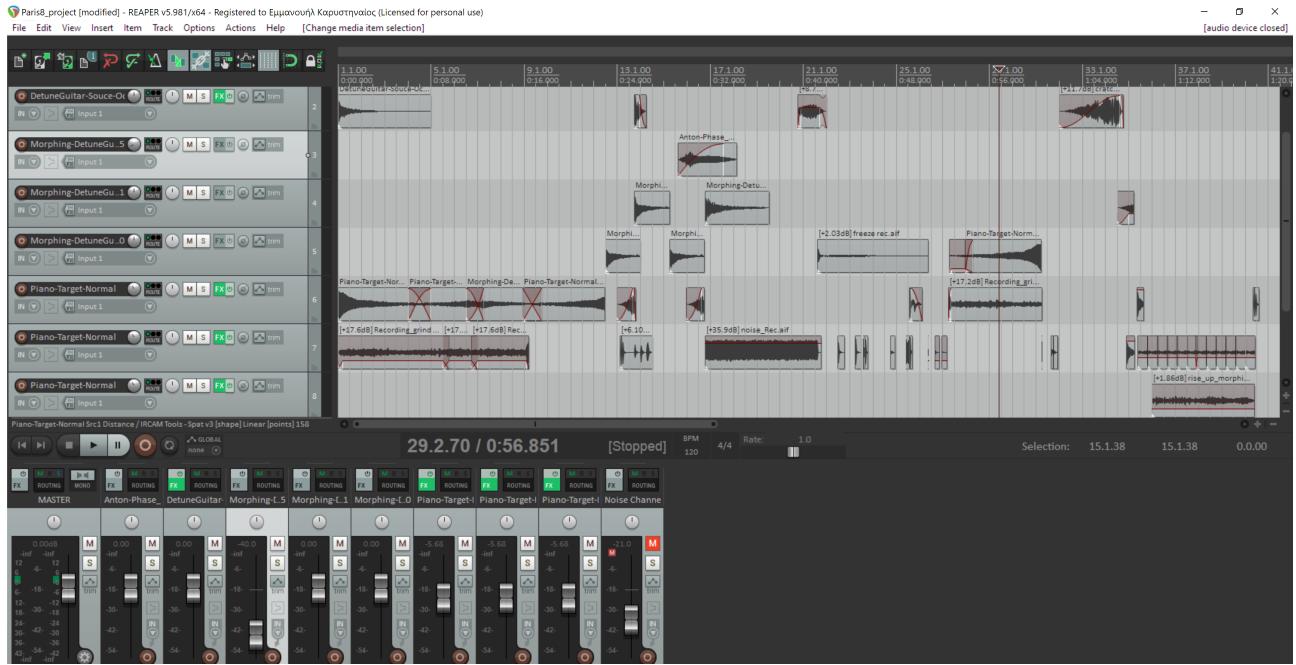


FIGURE 4.11 – Partie du mixage de Diakrotima sur Reaper

est reflété par des crescendos qui vont et viennent pendant la durée totale de la pièce. Par conséquent, le nom de la pièce est *Diakrotima*, qui signifie battement sonore en grec.

D'une manière quasi régulière les sons se précèdent presque consécutivement. Chaque son est différent et similaire, cela souligne l'existence d'un vocodeur de phase. Cette méthode compositionnelle est inspiré par la musique de Morton Feldman¹. "Chaque son comme si presque efface de sa mémoire ce qui est arrivé auparavant", disait Morton Feldman. C'est sur ce dire de Feldman que l'inspiration de cette composition émerge.

Pendant le processus de mixage les seuls effets complémentaires utilisés sont un de-noiser² pour enlever le bruit électomagnétique des enregistrements et un outil de spatialisation³. Aucun effet de modification sonore a été ajouté afin d'avoir des sons strictement produits par un vocodeur de phase.

Le format finale de la pièce est un fichier *wav* normalisé en stéréo. La pièce, les enregistrements d'origine aussi bien que les enregistrements de MaxMSP sont disponibles sur le répertoire github de ce projet de recherche.

1. Hirata Catherine Costello, *The sounds of the sounds themselves : Analyzing the early music of Morton Feldman*, 1996.

2. de-noiser de Izotope série RX7

3. SPAT de Ircam Tools

Analyse structurelle

La pièce, *Diakrotima*, a une forme asymétriquement périodique conforme à une structure varié et complexe. Le début est fort et dynamique avec un son de source qui a pour origine un enregistrement d'une E2⁴ de piano joué en forte. En même temps, un son d'origine de frottement avec un plectre sur la sixième corde de la guitare ajoute de la tension bruyant et continue sur la pièce.

La deuxième vague du battement arrive avec une F2⁵ de la guitare jouée en fortissimo d'une mode pizzicato Bartók. La suite consiste à une modification du même son. Quelques sons percussifs de courte durées sont réalisés par un double vocodeur de phase pour morphing. Les deux sons utilisés sont le pizzicato Bartók à la guitare ainsi qu'un son de frottement avec le plectre.

La paramétrisation d'un son percutant agit sur une courbe gaussienne du fenêtrage Gabor avec σ suffisamment petit, ainsi qu'une transposition de l'un de deux sons vers le registre aigüe. Le patch utilisé est le *Super Phase vocodeur* contenant du morphing, de la transposition, du freeze et du changement de la temporalité. Pour construire plus des sons à base du bruit, on construit une courbe gaussienne très serré avec $\sigma \simeq 0$, l'amplitude du bruit au maximum avec $\alpha = 1$ et un freeze temporel.

Tout autre son présenté dans la pièce est une variation du paramétrage. Parfois, on choisit de mélanger des modifications percutantes de sons avec une transposition de la hauteur. Les autres produits sonores sont de la variation des valeurs de phase et d'amplitude parmis le *Super Phase Vocoder*.

4.4.1 Évaluation Artistique

Dans le cadre de recherche, il s'agit d'une suite logique d'application de la critique de l'œuvre. Ici, donc, on se focalise sur un point de vue épistémologique, de l'évaluation de la mise en

4. la fondamentale presque à 83Hz.

5. la fondamentale presque à 87Hz.

œuvre, des applications du vocodeur de phase dans ce contexte artistique.

Dans la pièce *Diakrotima* se trouve une palette des sons produits exclusivement par un vocodeur de phase avec très peu de matériel sonore à l'origine. On a pu ainsi démontré que les possibilités d'un vocodeur sont presque infinies dans un contexte artistique, mais on remarque encore quelques défauts.

Quand on applique un changement de la hauteur plus d'une octave vers le bas, on retrouve une perte de qualité considérable. Également, si un son n'est pas très harmonique, au sens qu'il contient des caractéristiques très percutantes et qu'il n'a pas de fréquence persistante, on se rend compte que toute opération de changement de la hauteur ou du rythme de lecture n'est pas optimale. Au-delà des problèmes techniques liés à la construction de l'algorithme, on peut constater que les sons produits avec le vocodeur de phase ont tous un caractère rond. C'est-à-dire que l'effet de reproduire de sons à partir des sinusoïdes donne l'impression que tout son est similaire et par extension pas naturel.

L'effet d'avoir des sons arrondis ou artificiels est parfois bien désirable. Pour autant, le résultat reste toujours robuste et de haute qualité. Or, le vocodeur de phase ne semblerait pas être le premier choix des outils informatiques pour des compositeurs qui souhaiteraient créer des pièces acoustiques avec des sons d'une haute qualité réaliste.

Chapitre 5

Conclusion

5.1 Résumé de la recherche

Dans ce mémoire, nous avons parcouru des notions mathématiques de l'analyse spectrale en se focalisant sur le vocodeur de phase. Nous avons cité toute information nécessaire pour qu'un amateur se plonge dans le domaine du processus spectral du signal. De plus, nous avons détaillé le processus pour créer un vocodeur de phase sur MaxMSP permettant une connaissance plus profonde parmi l'application. En outre, nous avons proposé plusieurs modifications du code dans un contexte artistique et nous avons composé une pièce acousmatique en utilisant ces modifications.

En d'autres termes, cette recherche sert à des fins documentaires ainsi qu'épistémologique. Notre but est d'informer les musiciens et les compositeurs des notions complexes. Ce travail me sera aussi enrichissant, car il m'a permis 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 découvre de nouvelles manières pour manipuler cet effet et on applique notre critique. Cette problématique, constitue le but final de la recherche.

5.2 Applications

Dans le second chapitre, nous avons parcouru plusieurs applications du vocodeur de phase mais les capacités de cet outil sont quasiment infinies. Pour donner une idée de ces capacités, 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 son¹.

Le vocodeur de phase qui est appliqué exclusivement sur la voix humaine peut être également modifié pour transformer un certain type de voix. Par exemple une voix d'homme peut être transformée en une voix féminine ou bien tout autre type de voix humaine. Cette opération est construite sur une base des profils spectraux de différents types des voix qui permettent d'interpoler entre ces profils spectrales comme un morphing sous paramètres²

Une amélioration du vocodeur de phase consiste à appliquer des ondelettes. En lieu d'appliquer la transformation simple de Fourier, on peut remplacer l'exponentiel complexe de Fourier par une fonction d'ondelette. Cette fonction est équivalente de l'application de plusieurs transformées de Fourier à différentes tailles de fenêtres permettant de capter simultanément plus de détail fréquentiel et ponctuel (attaque)³.

Les outils spectraux et spécialement ceux d'analyse-synthèse sont fréquemment utilisés par les compositeurs parmi les logiciels DAW tels que Ableton Live, Cubase, Reaper, etc. Une des bibliothèques fréquemment utilisée est TRAX fait par *IRCAM Tools*⁴.

5.3 Discussion

Le vocodeur de phase est aujourd'hui plus qu'un simple outil d'analyse spectrale. Des méthodes ont été développées pour la prévision de manière stochastique sur une resynthèse optimale des

1. Axel Roebel, *Shape-invariant speech transformation with the phase vocoder*, 2010.

2. Axel Roebel, *A new approach to transient processing in the phase vocoder*, 2003.

3. Beltrán, José R and Beltrán, Fernando, *Additive synthesis based on the continuous wavelet transform*, 2003.

4. <https://www.flux.audio/project/ircam-trax-v3/>

ondes sonores.

Ces méthodes considèrent des formules probabilistes pour reconstruire des signaux afin d'obtenir une meilleure technique de prédiction de la fréquence. Pour plus d'informations à ce sujet, le livre de Roads and al.⁵ était plutôt éclairant.

5.4 Recherche pour l'avenir

On peut étendre cette recherche sur les processus spectraux en utilisant les principes tels que Deep Learning. Spécifiquement, j'envisage une approche sur le Morphing spectrale en temps réel avec de l'apprentissage pour automatiser la paramétrisation du vocodeur de phase correspondant⁶.

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 afin de développer une méthode intelligente pour préciser les différentes notes contenue dans un son polyphonique⁷. L'extraction des fréquences dominantes de chaque voix 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 prochaine recherche, je souhaiterais aborder le domaine de l'analyse Topologique de la musique. C'est une interprétation de la musique symbolique comme des structures topologiques bidimensionnelles, c'est-à-dire des graphes orientés. À partir de ces graphes on pourra extraire des informations du style musical ou de la structure harmonique d'une pièce musicale. La théorie de graphes contient plusieurs méthodes spectrales telles que la transformation Fourier des graphes et toute une infrastructure mathématique basée sur les concepts investigués dans ce mémoire.

Une des pistes alternatives, également intéressante à étudier pour ma part,, serait l'utilisation des VAE (Variational Autoencoders) pour identifier la distribution de probabilité d'un ensemble

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

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

7. Polyphonie : Assemblage de voix ou d'instruments, sans préjuger de leur nature

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 A

Appendix

A.1 FFT Cooley - Tukey algorithm

1

```
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
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
```

1. Le code est recuperé du lien www.algorithm-archive.org distribué sous une licence *MIT*

```

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   end
27
28   function bitreverse(a::Array)
29     # First, we need to find the necessary number of bits
30     digits = convert(Int, ceil(log2(length(a))))
31
32
33     indices = [ i for i = 0:length(a)-1]
34
35
36     bit_indices = []
37     for i = 1:length(indices)
38       push!(bit_indices, bitstring(indices[i]))
39     end
40
41     # Now stripping the unnecessary numbers
42     for i = 1:length(bit_indices)
43       bit_indices[i] = bit_indices[i][end-digits:end]
44     end
45
46     # Flipping the bits
47     for i = 1:length(bit_indices)
48       bit_indices[i] = reverse(bit_indices[i])
49     end
50
51     # Replacing indices
52     for i = 1:length(indices)
53       indices[i] = 0
54       for j = 1:digits
55         indices[i] += 2^(j-1) * parse(Int, string(bit_indices[i][end-j]))
56       end
57       indices[i] += 1
58     end
59   end
60
61   indices
62 end

```

```

52     end
53
54     b = [ float(i) for i = 1:length(a) ]
55     for i = 1:length(indices)
56         b[i] = a[indices[i]]
57     end
58
59     return b
60
61 end
62
63 function iterative_cooley_tukey(x)
64
65     N = length(x)
66
67     logN = convert(Int, ceil(log2(length(x))))
68
69     bnum = div(N,2)
70
71     stride = 0;
72
73     x = bitreverse(x)
74
75     z = [Complex(x[i]) for i = 1:length(x)]
76
77     for i = 1:logN
78
79         stride = div(N, bnum)
80
81         for j = 0:bnum-1
82
83             start_index = j*stride + 1
84
85             y = butterfly(z[start_index:start_index + stride - 1])
86
87             for k = 1:length(y)
88
89                 z[start_index+k-1] = y[k]
90
91             end
92
93         end
94
95         bnum = div(bnum,2)
96
97     end
98
99     return z
100
101 end
102
103 function butterfly(x)
104
105     N = length(x)
106
107     half = div(N,2)
108
109     n = [ i for i = 0:N-1]
110
111     half = div(N,2)
112
113     factor = exp.(-2im*pi*n/N)
114
115
116     y = [0 + 0,0im for i = 1:length(x)]
117
118 end

```

```
87
88     for i = 1:half
89         y[i] = x[i] + x[half+i]*factor[i]
90         y[half+i] = x[i] - x[half+i]*factor[i]
91     end
92     return y
93 end
94 function approx(x, y)
95     val = true
96     for i = 1:length(x)
97         if (abs(x[i]) - abs(y[i]) > 1e-5)
98             val = false
99         end
100    end
101    println(val)
102 end
103 function main()
104     x = rand(128)
105     y = cooley_tukey(x)
106     z = iterative_cooley_tukey(x)
107     w = fft(x)
108     approx(y, w)
109     approx(z, w)
110 end
111 main()
112
```

Listing A.1 – Algorithme de Cooley-Tukey avec des diagrammes papillon

Listings

2.1 DFT	17
Exponential.js	48
A.1 Algorithme de Cooley-Tukey avec des diagrammes papillon	68

Bibliographie

Fernando Beltrán, José R et Beltrán. Additive synthesis based on the continuous wavelet transform : A sinusoidal plus transient model. *DAFX03*, 3 :1–6, 2003.

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

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

Cycling74.com. Max8 release date, 2018. URL <https://cycling74.com>.

Coralie Diatkine. Audiosculpt 3.0 user manual, 2011. URL [http://support.ircam.fr/docs/](http://support.ircam.fr/docs/AudioSculpt/3.0/co/FFT%20Size.html)
[AudioSculpt/3.0/co/FFT%20Size.html](http://support.ircam.fr/docs/AudioSculpt/3.0/co/FFT%20Size.html).

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

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

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

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

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.

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

James Cooley et John W. Tukey. An algorithm for the machine calculation of complex fourier series. *JSTOR*, 1965.

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

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

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.

Gérald Grisey. Écrits ou l’invention de la musique spectrale. Paris, 2008. MF Éditions.

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

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

Catherine Costello Hirata. The sounds of the sounds themselves : Analyzing the early music of morton feldman. *Perspectives of New Music*, pages 6–27, 1996.

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

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

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

Babu V. Suresh Wahid Khabou P. G. Reshma, P. Gopi Varun. Analog implementation of fft using cascade current mirror. *Microelectronics Journal*, 60, fevrier .

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

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

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

Axel Roebel. Shape-invariant speech transformation with the phase vocoder. 2010.

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