

Initiation a la recherche

**Gédoux Architecte multimédia   
 Gautier  
 N2**

La synthèse additive avec Clojure

Table des matières

Introduction4

Etat de l’art5

Contexte et environnement de travail1

Clojure2

Overtone2

Harmonikit2

Présentation du projet1

Analyse du programme2

L’OSC et ses outils2

Problèmes rencontré2

Amélioration2

Ouverture2

Conclusion1

Introduction

Dans le cadre de mon module d’initiation à la recherche il m’a été demandé de travailler sur un projet bien particulier : Harmonikit, le synthétiseur additif que Rich Hickey à présenté lors de sa dernière conférence à la ClojureCon.

Ce projet a été un défi de taille pour moi, en effet ayant à la base une formation électronique, je me suis pris de passion pour l’informatique après mon stage orienté dans ce domaine. Plus encore j’ai du apprendre un langage très original qui a une structure totalement différente des autres langages que j’ai pu utiliser auparavant.

Je me suis vraiment attaché à ce projet car il concerne un domaine qui me passionne depuis mon enfance : la musique électronique.

Dans ce rapport je vais vous présenter les environnements sur lesquels j’ai travaillé et parler un peu de leurs spécificités puis je vous parlerais de la réalisation de mon projet et de l’évolution de mes objectifs en fonction de mes avancés, de mes découvertes et des difficultés rencontrées tout au long de ces six mois.

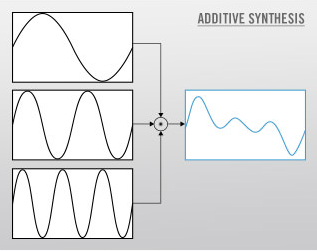
Etat de l’art

Lorsque l’on veut aborder un projet avec une démarche nouvelle, notamment dans un projet de recherche, il est important de réaliser un état de l’art de ce qui a été fait avant celui-ci. Un état de l’art rassemble nos recherches sur tout ce qui a été fait avant, dans le thème abordé par notre projet.

## La synthèse additive

Pour réaliser mon état de l’art je me suis tout d’abord intéressé à la synthèse additive, le synthétiseur sur lequel je devais travailler étant basé sur ce principe.

La synthèse additive consiste à créer un signal périodique en superposant plusieurs sinusoïdes harmoniques. Elle est basée sur le principe de Fourier qui nous dit que tout signal périodique peut être théoriquement décomposé en une somme de sinus et cosinus de fréquences multiples de la fréquence fondamentale du signal.



*Schéma représentant une synthèse additive*

A l’époque la synthèse additive était peu utilisée car très énergivore en terme d’utilisation des processeurs, on lui préférait donc la synthèse soustractive. Cette dernière part du principe inverse : filtrer un signal périodique très riche à l’aide de filtres fréquentiels pour obtenir le son voulu.

Mais désormais, grâce aux progrès des processeurs en matière de puissance de calcul,

La Synthèse additive commence à être de plus en plus utilisée sur les synthétiseurs numériques.

L’inconvénient principal de la synthèse additive est l’importance d’avoir un nombre d’oscillateur conséquent de manière à créer des sons intéressants qui se détachent des sons purs donnés par les sinusoïdes.

Ce type de synthèse requiert donc une innovation dans la manière de contrôler ces sinusoïdes, la première tentative à été réalisé par Iannis Xenakis, compositeur de musique concrète réputé (qui est pour la petite anecdote le grand père de ma copine) avec l’UPIC, une table graphique permettant d’actionner les oscillateurs en dessinant des lignes et des points.

Différentes méthodes ont ensuite vu le jour et sont implémentées dans certains synthétiseurs que je vais vous présenter ci dessous.

## Quelques Synthétiseurs additifs

### Le Parsec



Le Parsec comporte un nombre impressionnant de 1024 Partiels (Les sinusoïdes harmoniques), qui sont reparties sur deux oscillateurs principaux.

Si le parsec est innovant avec son nombre de sinusoïde utilisé, il ne l’est pas dans son fonctionnement : en effet il utilise exactement les même contrôle qu’un synthétiseur soustractif lambda.

### Le Phosphor



Le Phosphor est un synthétiseur un peu différent : il utilise 26 partiels, réparti sur deux oscillateur, on peut a l’instar du synthétiseur sur lequel j’ai travaillé modifier l’amplitude de chaque partiels séparément. Pour le reste, l’enveloppe et les effets s’effectuent sur les seuls oscillateurs.

### Le Razor



Le Razor est, en terme de contrôle le synthétiseur additif le plus innovant qu’il m’est été donné de voir.

Il utilise jusqu’à 320 partiels, mais ce qui fait sa force c’est que l’intégralité des effets et modulations qu’il propose on aussi été créés sur le principe de la synthèse additive. Ce qui donne des effets complètements nouveaux comme le Formant qui permet de faire rentrer en résonnance tel ou tel partiels, et bien d’autres effets mais il serait un peu fastidieux de tous les décrire ici.

Contexte et environnement de travail

## Clojure et Overtone

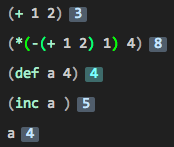
Clojure est un langage Lisp (pour list processing) : c’est un langage informatique de traitement de liste. Il a été inventé par Rich Hickey, fondateur de Conitect, car ce dernier voulait avoir un langage Lisp qui soit totalement compatible avec la plateforme Java.

Le développement via Clojure se fait à l’aide d’un REPL (pour Read-Eval-Print-Loop)

Qui se comporte exactement comme un terminal : une fois l’instruction exécutée, elle est évaluée puis compilée, et la boucle revient au point de départ et redemande une nouvelle instruction.

Ce Type de compilation est pour ma part très efficace. Contrairement a d’autres langages comme le langage C où il faut compiler l’ensemble du programme pour le faire fonctionner et si il y a une erreur, il faut tout recompiler après correction.

Ici si par exemple vous avez compilé l’ensemble du programme, et que vous voulez modifier une fonction ou corriger une erreur, vous n’avez besoin de recompiler que la partie du programme qui pose problème. Le gain de temps est conséquent.



Nous avons ici un exemple de différentes commandes simple.

Comme on peu le voir avec la variable « a », elle n’est modifiée qu’au moment ou la modification est exécutée, elle revient à son état initial ensuite.

C’est une des particularités fondamentales de Clojure et c’est une des caractéristiques qui en font un langage très efficace.

Personnellement j’ai eu au départ, beaucoup de mal à comprendre ce type de fonctionnement contrairement au java par exemple ou chaque variable est stockée et est modifiée jusqu'à une nouvelle intervention.



Overtone est une bibliothèque de Clojure qui utilise le moteur audio de SuperCollider pour créer une multitude d’objets sonores, d’effets et d’outils adapté à la création audio.

C’est cette bibliothèque qui a permit en partie de créer Harmonikit.

Présentation du projet

## Harmonikit

#### Structure

Harmonikit est un synthétiseur additif sur Ordinateur qui a été Créé par Rich Hickey, sur Clojure grâce entre autre a la bibliothèque Overtone.

Il a été présenté par ce dernier à la dernière ClojureCon de 2013 (meeting où se réunissent tous les acteurs de Clojure). D’après ses dires M. Hickey aurait codé Harmonikit en deux heure pendant un voyage ne train.

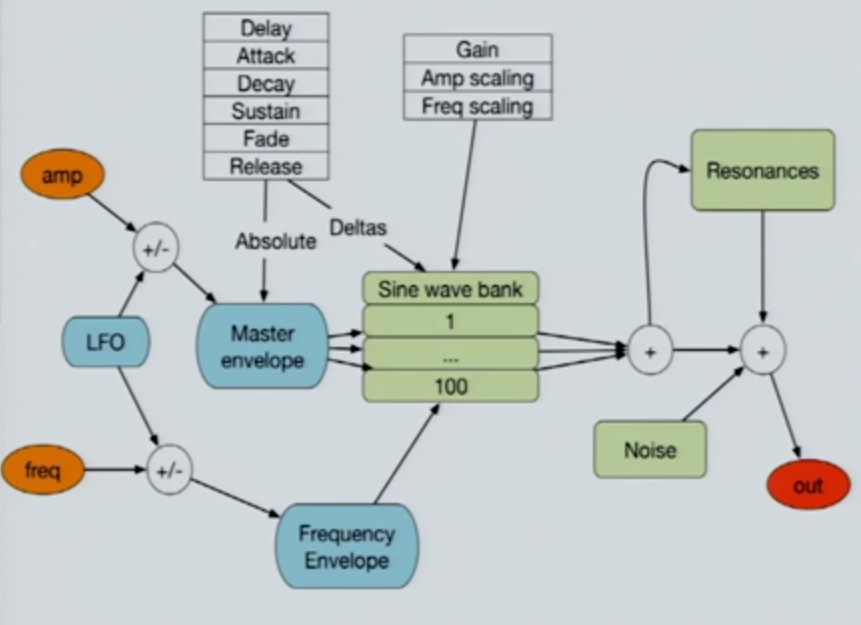
Il l’a uploadé sur Github de manière à ce que tout le monde puisse l’utiliser, le modifier, l’améliorer.

Au niveau de son fonctionnement en tant que synthétiseur, Harmonikit utilise des effets relativement classiques, mais la manière de les utiliser est quant à elle plus originale.

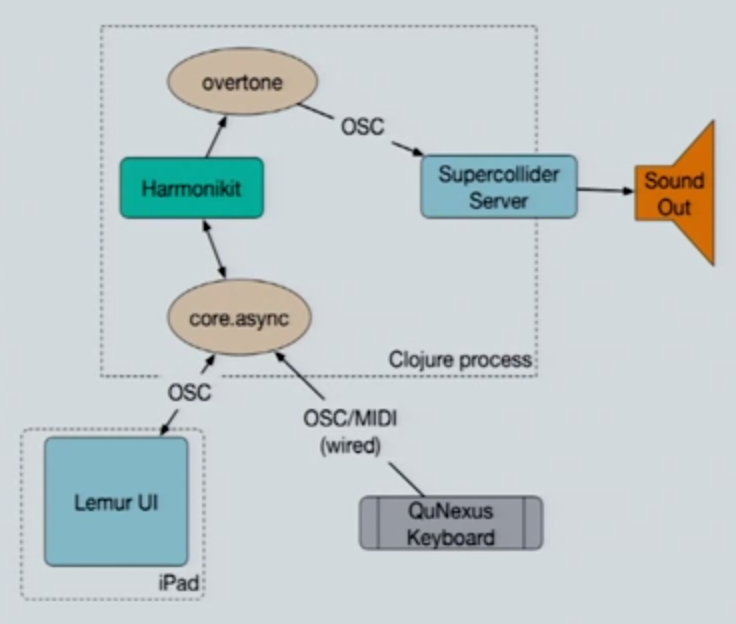
Par exemple l’enveloppe sonore de notre synthétiseur (qui correspond à l’amplitude d’un son au cours du temps), n’est pas seulement globale comme dans sa forme standard d’utilisation : elle s’applique aussi de manière globale mais peut aussi être utilisé de manière individuelle pour chaque Harmonique.

En d’autre terme on pourra définir une attaque globale (l’attaque définie l’amplitude à laquelle le son sera joué à la pression de la note. Plus l’attaque est grande plus le son mettra un temps à atteindre son volume maximal) mais aussi définir l’attaque de chaque Harmonique individuellement. Par exemple on pourra définir une attaque plus faible pour les harmoniques aigus, de manière à ce quelle arrive avant les harmoniques graves dans notre son.

Le Programme ne fonctionne pas seulement avec Overtone, il utilise d’autres bibliothèques que je vais vous présenter.



*Schéma récapitulatif de la structure d’Harmonikit*



*Schéma des interactions entre Harmonikit et les différentes*

*bibliothèques et interfaces utilisées*

Harmonikit utilise la bibliothèque d’Overtone, qui elle même utilise le moteur audio de SuperCollider de manière a faire du son.

Harmonikit est controlé par un clavier d’une part, ici le QuNexus qui communique en OSC, et par une interface graphique d’autre part, permettant de modifier les harmoniques, les effets, etc. Ici c’est l’application Lemur sur Ipad qui sert d’interface graphique, elle communique aussi en OSC avec Harmonikit.

C’est la bibliothèque core.async qui permet de stabiliser le tout en permettant d’envoyer des commandes de manière asynchrone. Ainsi si l’on joue une note avec deux appareils différents, elles seront jouées l’une après l’autre.

#### Fonctionnement

Harmonikit fonctionne de la manière suivante :

* Tout d’abord on instancie la variable patch, qui est une collection de tout les symboles dont on a besoin, classé sous forme de vecteur.
* On va ensuite « patcher » Overtone, c’est a dire réécrire certaines fonctions de la bibliothèque de base. En effet, Rich Hickey ne pouvait pas charger Overtone live pour gérer les générateurs audio de ses harmoniques. Il lui a donc fallut réécrire certaines fonctions d’Overtone pour pouvoir utiliser de manière différente certains objets de SuperCollider, dont les Ugens qui sont des générateurs de son. La réécriture va aussi permettre d’adapter certaines fonctions à notre contexte, comme par exemple la fonction synthdef qui permet de définir un instrument.
* Une fois Overtone Patchée on va s’attaquer à tout la structure de notre synthétiseur en réalisant une fonction pour chaque paramettre présent dans notre patch de départ et les interactions qu’ils peuvent avoir avec le son de notre instrument. En commencant par une fonction patch->buf qui va mettre tout notre patch dans un buffer pour pouvoir gérer la multitimbralité de notre instrument. Ce qui est impossible à faire avec les instruments de base d’Overtone qui n’ont qu’un timbre spécifique. On peut encore citer la fonction harm qui va définir les caractéristiques de chaque harmonique en les opposant aux valeurs de notre master-enveloppe, qui s’occupe de la forme globale de notre son.
* Une fois que toutes les fonctions qui permettent l’interaction avec les sons ont été créé, on va gérer l’interaction avec notre interface OSC, c’est sur ces fonctions que j’ai eu à faire le plus de modification.
  + La fonction edit-patch va nous permettre de modifier les valeurs présentent dans notre collection patch, ce qui va amener à modifier le son. Elle fonctionne de la manière suivante : lorsque l’on utilise un objet OSC cette fonction va regarder le chemin de notre objet et en fonction de son nom va faire des modification. Par exemple si je veux baisser le gain, le bouton que je vais utiliser aura comme chemin /gain. edit-patch va donc modifier la variable gain en fonction de l’argument qu’à mon bouton.
  + La fonction transmit-patch va envoyer les informations du patch à l’interface OSC de manière à ce que les changements soient affichés. Par exemple pour le gain la fonction va envoyer la valeur du gain en argument et lui donner le chemin /gain comme cela mon bouton sera positionné au bon endroit sur mon interface OSC.
* On va ensuite instancier plusieurs variables et initialiser HarmoniKit :
  + On va définir l’atom apatch qui est une variable spécifique qui met en mémoire les valeurs de notre collection patch.
  + Le client osc.
  + Le serveur osc.
  + Le channel utilisé par async.
  + On va aussi initialiser apatch avec un patch prédéfini.
* une fois l’initialisation terminée on lance la boucle principale qui utilise async pour transférer les données. Dans la boucle :
  + on va remplacer les valeurs de apatch par celle de la fonction edit-patch.
  + on va jouer les sons par rapport aux messages OSC reçus par le QuNexus.
  + on va supprimer les sons après pression de la touche.

## L’OSC et ses outils

L'Open Sound Control est un format de transmission de données entre ordinateurs, synthétiseurs, robots ou tout autre matériel ou logiciel compatible, conçu pour le contrôle en temps réel. Il utilise le réseau au travers des protocoles [UDP](https://fr.wikipedia.org/wiki/UDP) ou [TCP](https://fr.wikipedia.org/wiki/TCP) et apporte des améliorations en termes de rapidité et flexibilité par rapport à l'ancienne norme [MIDI](https://fr.wikipedia.org/wiki/MIDI).

Il fonctionne de manière très simple, lorsque l’on utilise un objet utilisant l’OSC , ce dernier va envoyer cinq informations :

* L’adresse IP de l’hôte de la connexion
* Le port utilisé
* Le chemin, c’est a dire le non de l’objet, qui est unique.
* Le type de l’argument que l’objet envoi (float, int, double, etc.)
* L’argument

Voici un exemple :

{:src-port 51857, :src-host 192.168.0.37, :path /harmonics/ascale/2, :type-tag f, :args (-1.0)}

L’interface graphique utilisant l’OSC que Rich Hickey avait choisi, Lemur, coutait relativement cher. Le premier défi que je me suis donné a été de transposer cette interface graphique sur une application gratuite, à défaut d’être open-source : TouchOSC.

L’autre avantage que TouchOSC a par rapport a Lemur est ca portativité, en effet Lemur de fonction que sur les appareils de la marque à la pomme, tandis que TouchOSC  
fonctionne aussi sur Android, je peux donc à présent utiliser Harmonikit aussi avec mon téléphone.

Cependant les contrôles que Propose TouchOSC en comparaison avec Clojure sont différents dans leur fonctionnement, j’ai donc du faire beaucoup de modification pour pouvoir adapter Harmonikit à TouchOSC.

Créer une interface depuis TouchOSC est relativement simple : depuis un éditeur nous avons accès à différents objets, des boutons poussoir, des interrupteurs, des faders, etc.

Une fois ceux ci configuré on leurs ajoute un identifiant OSC pour qu’ils puissent êtres reconnus par notre programme. Le reste se fait ensuite avec Clojure, en fonction des actions que l’ont veut effectuer.



Pour recréer Harmonikit sur TouchOSC j’ai tout d’abord du faire l’interface sur l’éditeur en donnant les mêmes noms de chemin aux nouveaux objets que ceux utilisé sur Lemur, et en leurs attribuant les mêmes caractéristiques.

Une fois cette étape terminée, comme je l’ai dis plus haut certains objets n’avaient pas du tout le même comportement entre Lemur et TouchOSC.

Par exemple les interrupteurs qui permettent d’utiliser ou non une harmonique :

* Les interrupteurs sur Lemur, lorsqu’ils sont regroupés comme c’est le cas ici par groupe de 12, utilisent un seul chemin et envoient 12 arguments sur celui-ci , rassemblés en 1 vecteur.
* A l’inverse pour TouchOSC les interrupteurs rassemblés en groupe de 12 de la même manière, utilisent 12 chemins contenant 1 argument chacun

J’ai donc du modifier fondamentalement les fonctions edit-patch et trasmit-patch pour qu’elle puissent fonctionner de nouveau.

## Amélioration

J’ai effectué quelques améliorations au programme de base, que ce soit au niveau des effets sonores ou de l’ergonomie

#### Les effets audio

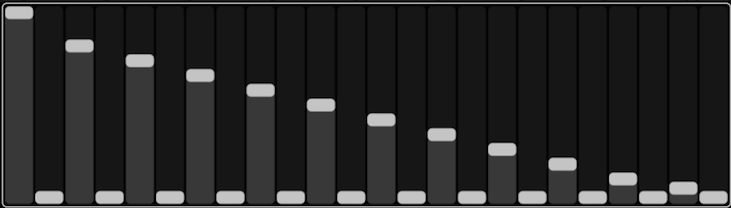
J’ai ajouté deux effets audio en plus de ceux déjà présent :



* Le premier effet qui s‘appelle « Formant » va définir un maxima d’énergie (ici 1) de l’amplitude de trois de nos harmoniques et fixée toute les amplitudes de nos autres harmoniques à 0, maxima d’énergie que l’on va pourvoir déplacer en fonction de la position de notre bouton. ainsi nous ajoutons un effet dynamique à notre amplitude d’harmonique.



* Le second effet que j’ai appelé « Shape effect » à pour but de donner une forme spécifique à chaque gain de nos harmoniques, de manière à donner un effet d’étouffement à notre son, comme la pédale d’atténuation d’un piano. Pour ce faire on utilise une look-up table qui contient la forme scéniques que l’ont veut donner au gain de chaque Harmonique.
  + lorsque l’on parcourt la plage de valeur du premier bouton de cet effet, les valeurs du gain vont êtres stockées dans un tampon et remplacées par celles de la look-up table.
  + à l’inverse le second bouton permet de remettre les valeurs du gain à leurs états précèdent en remplaçant ces dernières par celles présentes dans le tampon.



#### Sauvegarde et chargement

Une autre amélioration intéressante a été l’utilisation de fonctions de sauvegarde et de chargement de patch.

La fonction de sauvegarde va recopier l’atom apatch dans un fichier .edn qui sert à stocker les informations de manière efficace pour être réutilisé dans un programme.

La fonction de chargement va remplacer les valeurs de apatch par celle contenues dans le fichier .edn concerné.

Je me suis limité au stockage et au chargement de 5 patches différents.



## Ouverture

Il reste bien sur beaucoup d’améliorations à faire pour rendre Harmonikit vraiment viable, c’est ce que disait Rich Hickey dans sa conférence. Je pense notamment au fait de pouvoir ajouter un bruit blanc, que je n’ai pas eu le temps de réellement appréhender. Mais aussi faire une vrai librairie sonore et non pas se limiter à l’utilisation de fichier .edn pour charger ses patchs.

Je pense aussi à faire de Harmonikit un instrument VST(pour Virtual Studio Technology) qui lui permettrait d’être utilisé sur beaucoup de softwares de création musicale , comme Ableton, Reason ou encore Cubase.

Et enfin essayer d’ajouter de nombreux effets audio comme ceux utilisés par Razor qui se bases aussi sur la synthèse additive.

Le fait de rajouter des Harmoniques pourra aussi être très intéressant en terme de qualité et de définition audio, car avec seulement 24 harmoniques la synthèse additive n’est pas vraiment crédible comparée à la synthèse soustractive. Mais cela va nous amener encore une fois a repenser toute l’architecture car il est impossible de gérer tout les paramettre de chaque harmoniques un à un.

On pourrait donc réaliser des patterns globaux, à l’instar de mon effet « Shape » que l’ont appliquerai à ces harmoniques de manière à toute les changer en même temps,

Conclusion

J’ai trouvé ce projet passionnant, il m’à permis d’avoir un point de vue totalement différent de celui que je me faisait arbitrairement de la recherche, et de la recherche en informatique plus précisément.

D’un autre coté j’ai toujours été baigné dans le monde de la musique et cette expérience me motive encore plus dans l’atteinte de mon projet professionnel c’est a dire m’immerger dans le monde musical tout en restant dans le domaine informatique, comme par exemple en travaillant dans des entreprise comme Ableton ou encore Native Instrument qui sont orienté sur ce genre de sujet.

Etant moi même compositeur de musique électronique, j’ai pour projet de transformer Harmonikit en plugin VST pour pouvoir l’utiliser sur mes interfaces audio préférées.

Bibliographie

* <http://clojure.org/>
* <http://clojure-doc.org/>
* <http://overtone.github.io/>
* <https://fr.wikipedia.org/wiki/Synth%C3%A8se_sonore_additive>
* <http://www.native-instruments.com/fr/products/komplete/synths/razor/>
* <http://www.kvraudio.com/news/audio_damage_releases_phosphor_additive_synth_15851>
* <https://shop.propellerheads.se/product/parsec/>
* <http://hexler.net/software/touchosc>
* <https://liine.net/en/products/lemur/>