Time stretching en temps réel dans le cadre du live coding

Abdeselam El-Haman Abdeselam[1]
Superviseur : Bernard Fortz

[1]Université Libre de Bruxelles
aelhaman@ulb.ac.be

Abstract

Overtone est une libraire en Clojure qui est utilisée pour faire du Live Coding (l'art de programmer en « vif »). Une des techniques les plus utilisées dans le domaine de la musique synthétisée est le time-stretching, qui consiste à rallonger ou rétrécir une pièce musicale sans changer sa tonalité. Le time-stretching est intéressant dans le live-coding lorsqu'on peut modifier les paramétres de celui-ci en temps réel. Dans cet article une méthode de time-stretching avec des approches différentes sera analysée et utilisée en temps réel avec Overtone.

Introduction

Depuis le débuts de la musique éléctronique, le "resampling" de pièces musicales (c'est à dire, la manipulation de celles-ci) a un rôle important pour pouvoir manipuler des pièces musicales (rajouter des filtres, des envelopes, etc . . .).

Un son est un signal qui est défini par sa durée, sa fréquence et son amplitude. La fréquence définit son "pitch" ou tonalité. La tonalité d'un son est plus grave si la fréquence est plus petite (donc une période plus grande).

La manipulation d'un son dans le temps est utilisée très souvent lors du mixage et DJing. Ce genre de manipulations aident à rétrécir ou prolonger cette pièce pour, par exemple, mettre un son à la même vitesse que le "tempo" d'une chanson.

Cette manipulation aura comme résultat aussi un effet secondaire. La prolongation du signal provoquera aussi que la fréquence de ce signal soit modifié, et avec ça, un changement de tonalité non désiré se produira [Oppenheim and Schafer (2009)].

Le time-stretching est une technique pour éradiquer ce problème : changer le tempo d'un son sans changer sa tonalité.

Plusieurs algorithmes existent pour implementer cette technique. Dans cet article l'état de l'art du time-stretching sera étudié et plusieurs de ces algorithmes seront implémentés pour l'application en temps réel de cette technique sur Overtone, une librairie pour qui a été conçue pour le traitement et synthèse de son.

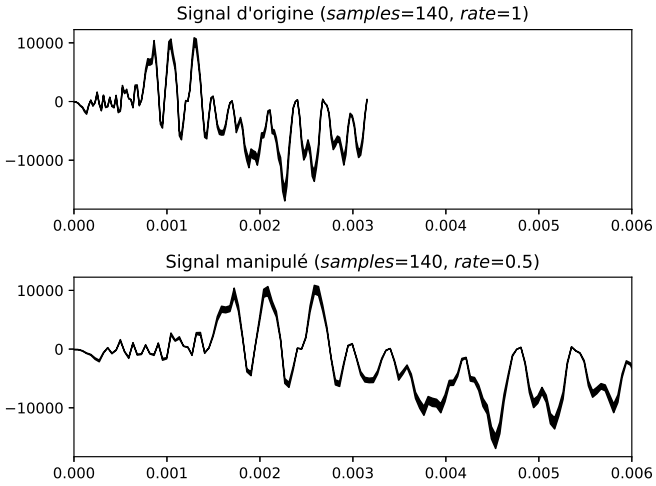

FIGURE 1 – Échantillon d'une guitare lu à des vitesses différents

État de l'art

Méthode OLA

En général la méthode utilisée dans les algorithmes TSM se résume en plusieures étapes distinctes.

Décomposer le signal d'entrée en grains Le signal d'entrée est décomposé en *grains* [citer article] d'une taille rélativement petite et fixe. Chacun de ces ces grains sa un décalage *hopsize* H_a (hopsize d'analyse). On peut représenter ça tel que :

$$x_m(r) = x(r + mH_a) : r \in [-N/2 : N/2 - 1]$$

où x est notre signal d'entrée en format discret de taille $L \in \mathbb{Z}_{\geq 0}$. N est la taille du grain x_m.

La taille de ces grains doît est généralement d'une taille entre 50ms et 100ms. Ceci est important pour trouver le *pitch local* dans l'intervale $[mH_a - N/2 : mH_a + N/2 + 1]$. Si l'intervale est grand, plusieures tonalités apparaissent dans cet intervale, donc ce grain n'est pas représentatif.

Modifier le hopsize Une fois le signal décomposé, avec l'information de pitch dans chaque grain, on va définir un nouveau hopsize H_s qui va servir à récomposer ces grains. Ce H_s est défini tel que :

$$H_s = \alpha H_a$$

Dans lequel α représente le *facteur d'échelle*. L'effet du TSM est fixé par le *facteur d'échelle*. Pour un effet de rétrécissement :

$$\alpha < 1$$

Au contraire, pour un effet d'élargissement :

$$\alpha > 1$$

Superposition de grains Avec H_s défini, on peut réconstruire le signal résultat de cette modification, en récomposant le nouveau signal avec les *grains* avec un décalage de H_s au lieu de H_a. Alors ces grains sont superposés avec une différence de H_s et additionés. C'est à dire :

$$y(t) = \sum_{m \in \mathbb{Z}} x_m(t - mH_s)$$

où $y(t)$ est le signal de sortie par rapport au temps t. u

Fenêtrage Hormis le fait que cette méthode est utilisée comme base des algorithmes qui seront illustrés plus tard dans ce document, elle crée beaucoup d'*artéfacts* à cause d'un déphasage entre les grains qui provoque des discontinuités qui se traduit par des sons lourds et courts qui n'étaient pas dans le signal initial. Ceci est dû à cause d'un déphasage important entre les grains puisque $H_s \neq H_a$.

L'algorithme *OLA* se base sur cette méthode basique et rajoute des fénêtres pour avoir une fluidité dans la transition d'un grain à un autre. Les grains x_m sont donc appliqués à une fonction de fênetrage w qui est définie :

$$w(r) = \frac{1}{2}\left(1 - cos\left(\frac{2\pi(r + N/2)}{N - 1}\right)\right)$$

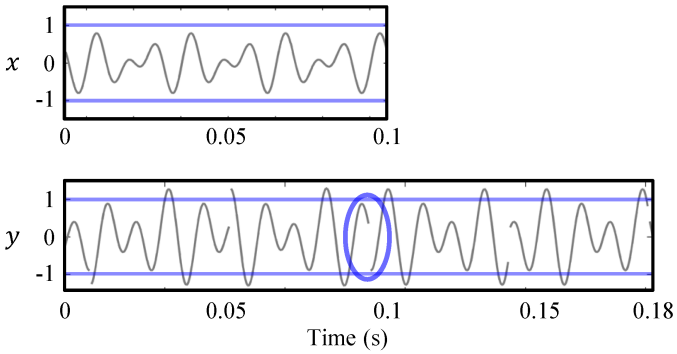

FIGURE 2 – Exemple d'artefact qui se produit quand le même grain d'analyse est utilisé pour reconstruire le signal avec le tempo modifié [Driedger and Müller (2016)].

Où w est la *fenêtre de Hanning*.

Cette fenêtre a la propriété que

$$\sum_{n \in \mathbb{Z}} w\left(r - n\frac{N}{2}\right) = 1$$

qui rassure la continuité d'amplitude dans le signal résultant.

Méthode WSOLA

Le problème de la méthode OLA est que, étant le hopsize H_a fixe, chaque séparation entre grains est le même, peu importe la différence de phase quand les grains sont décalés et réassemblés par rapport au hopsize H_s.

L'algorithme *WSOLA* [Verhelst and Roelands (1993)] régle ce problème en rajoutant un marge de décalage Δ_m au hopsize H_a afin qu'entre les grains x_m et x_{m+1} il y ait un minimum de discontinuités.

Ce décalage Δ_m est défini étant la position dans laquelle les parties superposées des grains x_m et x_{m+1} se ressemblent au plus. Pour ceci nous devons définit Δ_{max} étant le maximum que la valeur de Δ_m doit avoir. Alors $\Delta_m \in [-\Delta_{max} : \Delta_{max}]$. Nous avons alors le nouveau grain de synthèse

$$y_m(r) = x(r + mH_a + \Delta_m) \ : \ r \in [-N/2 : N/2 - 1]$$

qui va être superposé en suivant le procédé *OLA* pour réassembler les grains décrit avant.

Choix du décalage Δ_m est choisi en trouvant la meilleure resultat si on applique une fonction de corrélation entre les parties des grains qui se superposent. Plusieures formules

peuvent être choisies pour trouver une corrélation entre 2 variables (une regréssion linéaire, etc ...). On peut utiliser le coefficient corrélation croisée pour cette fin :

$$c_c(m,\delta) = \sum_{n=0}^{N-1} (n + \tau^{-1}((m-1)N) + \Delta_{m-1} + N) \\ x(n + \tau^{-1}(mx_m) + \delta) \quad (1)$$

avec $max(m,\delta) \rightarrow \delta = \Delta_m$ et τ notre fonction de TSM.

Méthode vocodeur de phases

WSOLA et OLA manipulent le signal dans le temps, et en particulier la méthode WSOLA arrive à trouver des résultats assez positifs pour garder la périodicité du signal. Ceci va éviter les artefacts d'un mauvais "overlap" car les ondes sinusoïdales aurant toujours une périodicité continue.

Le *vocodeur de phases* [Flanagan and Golden (1966)] est une méthode qui va prendre des grains comme les méthodes précédentes mais va décomposer ce grain en plusieures sinusoïdales, qui pourront être rallongées. Tout signal peut être décomposé en signaux sinusoïdaux grâce à la *transformée de Fourier* décrite pour un signal discret telle que :

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} \qquad k = 0, \ldots, N-1$$

Une autre approche qui serait plus approprié ici est la *Short Time Fourier Transform* (STFT) qui sert à trouver des informations locales de courte durée pour avoir une meilleure précision :

$$X(m,k) = \sum_{r=N/2}^{N/2-1} x_m r w(r) exp(-2\pi i k r/N)$$

Le calcul de cette transformée n'est pas très efficace, c'est pour ça qu'il existe l'algorithme *Fast Fourier Transform* (FFT) [Portnoff (1976)] , qui s'occupe de calculer cette STFT et son inverse (ISTFT). La FFT impose que que le grain analysé soit d'une taille d'une puissance de 2 (512, 1024, 2048, 4096).

Dans ce cas, on définit X^{Mod} un frame d'un STFT modifié. Faudra reconstruire le signal de sortie y qui correspond au X^{Mod}. Pour ceci, on doit trouver l'inverse comme ceci :

$$x_m^{Mod}(r) = \frac{1}{N} \sum_{k=0}^{N-1} X^{Mod}(m,k) exp(2\pi i k r/N)$$

Et puis on a :

$$y_m = \frac{w(r) x_m^{Mod}(r)}{\sum_{n\in\mathbb{Z}} w(r - nH_s)^2}$$

qu'on pourra utiliser pour reconstruir y de la même façon que pour les méthodes OLA.

Déphasage Les grains X^{Mod} consécutifs auront des décalages de phase quand ils seront superposés (cf Figure 3), ce qui provoque des artefacts. Pour résoudre ce problème, le *vocodeur de phases* décalera encore la phase des grains modifiés pour avoir une continuité de phase entre les grains superposés.

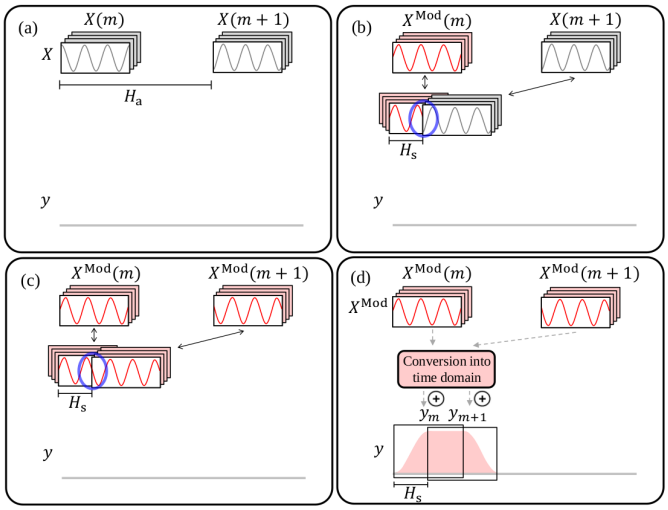

FIGURE 3 – Fonctionnement du vocodeur de phases [Driedger and Müller (2016)]. (a) Les frames sont décomposés par le STFT. (b) Les phases des 2 frames sont clairement pas les mêmes si on les overlap sans les manipuler. (c) La phase des frames sont propagées pour avoir une correspondence. (d) ISTFT pour reconstruire le signal après avoir fait l'overlap des frames modifiés.

Méthode développée

Le live-coding (aussi nommée *programmation à la volée*) est une technique de programmation de façon improvisée afin d'avoir des résultats artistiques. Il existe plusieurs domaines dans lesquels le live coding s'applique, comme la génération de graphismes ou de la musique. Évidement, dans cet article l'approche musicale du live-coding a été exploitée pour pouvoir implémenter un algorithme de time stretching en temps réel.

J'ai implementé un granulateur qui utilise la méthode OLA basique. Pour faire cela j'ai utilisé la librairie *Overtone* en Clojure qui est utilisée pour faire de la synthèse de

son en temps réel et du live-coding en base de *SuperCollider* [Aaron and Blackwell (2013)].

Overtone Clojure est un langage fonctionnel qui a plusieures caractéristiques intéressantes dans notre sujet. Premièrement,en étant un langage fonctionnel hérité de Lisp, donc toutes les données sont manipuléss de façon mathématique avec des fonctions [Hickey (2008)].
Deuxièmement, Clojure utilise le interpreteur de ligne de comande *Leiningen*, qui permet d'exécuter des commandes en temps réel tapées par l'utilisateur.

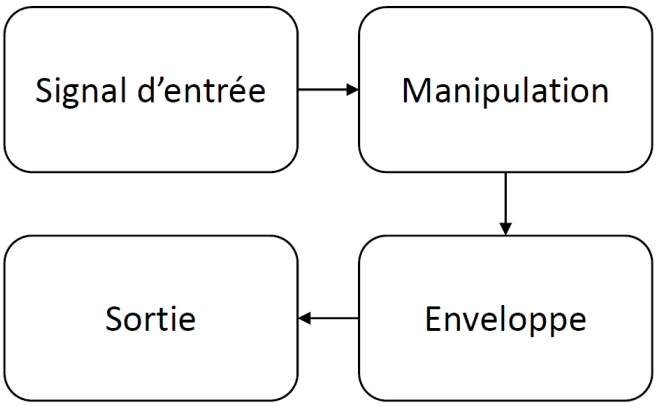

FIGURE 4 – Fonctionnement d'un synthétiseur [Driedger and Müller (2016)]. Le fonctionnement d'un synthétiseur se résume en une "boîte noire" qui prend un signal, le manipule (filtres, enveloppe, etc...) et ensuite fait un output. Plusieures manipulations peuvent être imbriquées.

D'un autre coté, SuperCollider est un environnement et un langage de programmation pour faire de la synthèse en temps réel. SuperCollider fonctione basiquement avec des "synth" (synthétiseur) qui prennent un signal (que ce soit une onde ou un buffer avec des segments temporels de son), les manipule et puis les renvoit.

Ces manipulations sont faites avec des UGens (Unit Generator) qui sont des blocs pour analyser/synthétiser des sons (par exemple, un filtre passe bas, un oscilateur sinusoïdale, un ugen pour lire un buffer, etc...). Les résultats d'un UGen peuvent être passés à un autre UGen qui peut faire d'autres manipulations (voir Figure 4). Aussi, les paramètres des "synth" sont modifiables en temps réel. C'est à dire, lorsque cet instrument est en train de jouer, nous pouvons changer, par exemple, la modulation d'un filtre et faire qu'il sonne différement lorsqu'il est lu par le serveur.

Overtone profite du langage fonctionnel et les capacités de synthèse de SuperCollider pour faire une liaison entre les 2, le fonctionnement des langages fonctionnels se ressemblant fortement au fonctionnement des synthétiseurs.

Granulateur Le granulateur, comme son nom indique, est un synthétiseur qui manipule des grains comme on les a vu précédement. Ce granulateur peut servir à implémenter beaucoup d'algorithmes avec des grains, mais dans ce cas là nous allons l'utiliser pour implémenter un algorithme OLA asynchrone.

Dans cette approche, le granulateur se compose d'un *scheduler* et d'un générateur de grains [Truax (1994)].
Le *scheduler* s'occupe de lancer le bon grain au bon moment à la sortie grâce à des paramètres pour définir le type de grain et à quelle vitesse les lancer. Dès qu'il veut lancer un grain, il va génerer le grain dont on a besoin grâce au générateur de grains.

Pour ceci on va gérer 4 paramètres.
— **Le rate**
— **La durée d'un grain**
— **Le délai entre 2 grains**
Le fonctionnement de ce granulateur peut se résumer dans le pseudocode suivant :

Algorithm 1 Scheduler du granulator

```
scale ⇐ le ratio
duration ⇐ durée d'un grain
delay ⇐ délai entre grains grain
phasor ⇐ pointeur dynamique qui avance au rythme du scale
last ⇐ le moment où le dernier grain a été lancé
time ⇐ le temps courant
N ⇐ la taille du morceau analysé
last = time
while phasor < N do
    if time ≤ last + delay∗duration then
        generateGrain(phasor, duration)
        last = time
    end if
    phasor = update(phasor)
    time = update(time)
end while
```

Le phaseur est un pointeur dynamique qui va parcourir le fichier à lire à la vitesse du ratio passé en paramètre. Les fonctions update du pseudocode vont mettre a jour ces variables (par rapport au temps). La méthode generateGrain prend en paramètre un indice et la durée, donc cette méthode va renvoyer un grain qui commence à $phasor$ et qui finit à $phasor + duration$.

Implementation Ce granulateur a été implémenté en Overtone (voir Annexe pour code source) en définissant un "synth" (méthode defsynth), et pour créer ce synth on a utilisé les UGen suivants. L'UGen `trig1` est un fonction qui va activer l'ugen auquel on va lui associer, il prend une onde en paramètre (dans notre cas, un `sin-osc`), il va server de *scheduler*. Puis nous avons un `phasor` qui va parcourir le fichier à la vitesse définie par notre paramètre de ratio. On a donc associé notre trigger à un UGen `env-gen` qui va créer un grain grâce à une enveloppe basée sur la fenêtre de Hanning. Finalement un UGen `play-buf` va jouer le grain qui sera trig à la position du `phasor`.

Grâce au fait que ça a été défini dans un synth, les paramètres peuvent être modifiables lors de son exécution, nous donnant aussi la possibilité de le modifier en temps réel tel un VST ou "plugin" [Tanev and Božinovski (2014)].

FIGURE 5 – Instrument MIDI (Akai MPD218). Cet instrument a été utilisé pour changer les paramètres du granulateur en temps réel.

MIDI À continuer demain

Résultats

On va analyser les morceaux musicaux de différents styles et les paramètres qu'on a du utiliser pour

Remerciements

Je remercie mon prometeur, Bernard Fortz, de m'avoir fait découvrir les yeux au monde du live-coding et la programmation fonctionnelle.

Je tiens aussi à remercier l'UrLab pour m'avoir accueilli au SmartMonday pour partager avec eux tout ce que j'ai appris lors de mes recherches pour ce mémoire.

Je voudrais aussi remercier mon collègue Antoine Passemiers pour m'avoir donné autant de conseils lors de la réalisation de ce mémoire.

Et pour finir je voudrais remercier à Sam Aaaron pour tout son travail dans le domaine du live-coding et de la pédagogie avec Overtone et Sonic Pi.

References

Aaron, S. and Blackwell, A. F. (2013). From sonic pi to overtone : Creative musical experiences with domain-specific and functional languages. In *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design*, FARM '13, pages 35–46, New York, NY, USA. ACM.

Driedger, J. and Müller, M. (2016). A review of time-scale modification of music signals. *Applied Sciences*, 6(2).

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

Hickey, R. (2008). The clojure programming language. In *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, pages 1 :1–1 :1, New York, NY, USA. ACM.

Oppenheim, A. V. and Schafer, R. W. (2009). *Discrete-Time Signal Processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Portnoff, M. (1976). Implementation of the digital phase vocoder using the fast fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(3) :243–248.

Tanev, G. and Božinovski, A. (2014). *Virtual Studio Technology inside Music Production*, pages 231–241. Springer International Publishing, Heidelberg.

Truax, B. (1994). Discovering inner complexity : Time shifting and transposition with a real-time granulation technique. *Computer Music Journal*, 18(2) :38–48.

Verhelst, W. and Roelands, M. (1993). An overlap-add technique based on waveform similarity (wsola) for high quality time-scale modification of speech. In *Proceedings of the 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing : Speech Processing - Volume II*, ICASSP'93, pages 554–557, Washington, DC, USA. IEEE Computer Society.

Appendices

Démonstrations

Lien vers une vidéo démonstration de l'usage du granulateur avec le MPD218 : `https://twitter.com/Abdulitokun/status/866394006917451777`

Le code source, ainsi que le rapport se trouvent dans le lien suivant : `https://github.com/Abde-/info-f-308mem`