ProSpect : une plate-forme logicielle pour l'exploration spectrale des sons et de la musique

Sylvain Marchand (sm@LaBRI.U-Bordeaux.Fr)

SCRIME*
LaBRI†, Université Bordeaux I
351, cours de la Libération
F-33405 Talence cedex – France

Résumé

ProSpect est une architecture logicielle pour la manipulation des sons et plus particulièrement pour l'analyse, la transformation et la synthèse des sons dans les modèles spectraux. Cette architecture résulte de l'extension d'un langage fonctionnel généraliste par un ensemble de types et de primitives spécifiquement sonores. Plusieurs modèles sonores sont disponibles, le nombre et la nature des manipulations possibles sur un son donné dépendant du modèle sonore dans lequel ce son est exprimé. Le fait de pouvoir agir sur les sons directement par les instructions du langage permet de programmer littéralement les sons et facilite grandement le processus de composition musicale. Il est de plus possible d'utiliser un des logiciels écrits au-dessus de *ProSpect* et disposant d'une interface graphique dédiée. Parmi ces logiciels on peut trouver par exemple le programme d'analyse sonore *InSpect* ainsi que *BOXES*, séquenceur audionumérique avec contraintes.

Mots clefs : architecture logicielle, Linux, modèles spectraux, langage fonctionnel

1 Introduction

ProSpect ("Prospect Spectrum") est une architecture logicielle libre, sous licence GPL (*GNU General Public License*) [Fun91], développée pour le système Linux (il existe également une version pour Windows 95/98/NT). Le but de cette architecture est de faciliter la manipulation des sons représentés sous forme spectrale, et plus précisemment leur analyse, transformation et synthèse. Même si la plupart des opérations sur les sons s'y effectuent rapidement étant donnée la puissance des ordinateurs actuels, la manipulation des sons en temps réel n'est pas le but recherché par *ProSpect*. De plus il ne s'agit pas d'un logiciel "final" comme SMS [Ser97b], mais plutôt d'un environnement de développement ouvert dans l'esprit de la bibliothèque ATS [Pam99] et qui peut servir de base à d'autres logiciels.

Après une rapide description de l'architecture logicielle dans la section 2, les différents modèles sonores et les fonctionnalités proposées dans chacun de ces modèles sont parcourus dans la section 3. Enfin la section 4 présente des applications basées sur *ProSpect*, dans des domaines variés allant aussi bien de l'analyse, transformation et synthèse qu'à l'aide à la composition musicale.

*Studio de Création et de Recherche en Informatique et Musique Electroacoustique, soutenu par le Conseil Régional d'Aquitaine, l'Université Bordeaux I, le Ministère de la Culture, la Direction Régionale des Actions Culturelles d'Aquitaine, la Mairie de Bordeaux, l'ENSERB et le Conseil Général de la Gironde.

†Laboratoire Bordelais de Recherche en Informatique, Unité Mixte de Recherche 5800 du CNRS.

2 Architecture logicielle

L'architecture logicielle de *ProSpect* consiste en un ensemble de fonctionnalités de base auxquelles viennent s'ajouter dynamiquement des modules d'extension.

2.1 Base

La base de *ProSpect* est formée d'un interprète Scheme étendu par une bibliothèque de types de base et de fonctions en grande partie écrites en langage C pour des raisons de performance. On retrouve une structure similaire dans ATS [Pam99], mis à part qu'ici le langage fonctionnel considéré n'est plus Common Lisp mais Scheme.

Langage fonctionnel

Dans la version actuelle, *STk* [Gal99] est utilisé comme environnement Scheme de base. Cependant il existe de nombreux autres interprètes Scheme et l'utilisation par exemple de *Guile* dans les versions ultérieures n'est pas exclue. En utilisant un langage fonctionnel, *ProSpect* hérite des bases du λ-calcul et de la programmation symbolique. Il n'y a de distinction *a priori* ni entre programmes et données pour l'aspect programmation informatique, ni entre instruments et partitions pour ce qui a trait à la composition musicale. Il n'y a donc pas cette frontière artificielle que l'on retrouve par exemple dans MPEG-4 [VGS98]. Cependant *ProSpect* n'est pas complètement écrit en Scheme. Contrairement à Common Lisp Music [LLP99], entièrement en Common Lisp, il fait appel à des extensions écrites en C, langage de plus bas niveau mais plus efficace en termes de rapidité.

Traitement du signal

Les possibilités initiales du langage fonctionnel sont étendues par une bibliothèque de types de base et de primitives implantées en C. Ainsi peut on alors manipuler efficacement par exemple des nombres complexes, vecteurs ou autres matrices. Parmi les extensions réalisées, on trouve par exemple de nouvelles fonctions mathématiques, de l'algèbre linéaire ou des primitives d'encodage et de compression de flots de données. *ProSpect* renferme également des fonctions issues de la psychoacoustique, même si le principal objectif des extensions est le traitement du signal. On retrouve alors tout naturellement beaucoup d'outils de base issus de la théorie du signal, comme des fenêtres d'analyse (Hann, Hamming, Blackman, Kaiser), des algorithmes de filtrage, de conversion de taux d'échantillonnage, ou la transformée de Fourier. La transformée de Fourier rapide utilisée dans *ProSpect* est d'ailleurs la FFTW [FJ00] développée au MIT.

Interface graphique

ProSpect est basé sur *STk* (Scheme + *Tk*), et hérite par conséquent de l'interprète Scheme de ce dernier mais aussi du *toolkit* graphique *Tk*, qui fournit un grand nombre d'objets et de primitives graphiques. *ProSpect* propose en plus de nouvelles fonctions pour gérer plus facilement les menus ou les boîtes de dialogue par exemple.

2.2 Extensions

La base logicielle peut être dynamiquement étendue à l'aide de modules (*plug-ins*), écrits en Scheme ou en C par exemple. Les logiciels dérivés de *ProSpect* définissent eux-mêmes de nouvelles extensions, qui peuvent être à leur tour réutilisées dans de nouveaux programmes. Définir une telle extension est aisé :

```
(prospect-link "msc")  ; chargement dynamique de msc.so

(plugin-insert-spectral-format
 '("Spectral Model (Compressed)"
```

```
  (".msc")
  spectral-test-msc
  spectral-load-msc
  spectral-save-msc))

(prospect-provide "msc")
```

Dans l'exemple précédent un nouveau format de fichier de sons spectraux est défini. Il s'agit d'un format nommé "Spectral Model (Compressed)", qui a pour extension par défaut `msc`. La fonction `test` décide si un fichier donné est du bon format, `load` effectue le chargement d'un tel fichier en mémoire et `save` sauvegarde un son spectral dans un fichier avec ce format. Ces trois fonctions sont définies dans `msc.so`, code objet résultant de la compilation d'un programme en C et chargé dynamiquement au début. Les primitives nécessaires à l'interfaçage entre le C et le Scheme sont fournies par `ProSpect` lui-même.

3 Modèles sonores

Les modèles sonores sont eux-mêmes des extensions et *ProSpect* en dispose pour l'instant de trois principaux :

- le modèle temporel (`temporal`), où chaque son est représenté par le niveau de pression acoustique en un point particulier de l'espace, et ce en fonction du temps ;
- le modèle de synthèse additive (`spectral`), dans lequel les sons consistent en un ensemble de partiels, oscillateurs sinusoïdaux dont les amplitudes et les fréquences évoluent lentement dans le temps ;
- le modèle de synthèse additive structurée (`sas`), qui structure les paramètres de la synthèse additive classique en seulement quatre nouveaux paramètres très proches de la perception.

Il est possible de définir un son à partir d'un fichier (fonction `load`) et réciproquement de créer un tel fichier à partir d'un son présent en mémoire (fonction `save`). Avec ces simples fonctions il est aisé d'effectuer un conversion de format au sein d'un même modèle sonore. L'exemple suivant permet de convertir un son temporel du format *WAV* au format *AIFF* :

```
(temporal-save (temporal-load "sound.wav") "sound.aiff")
```

Les principaux formats sonores, comme *AIFF* et *WAV* pour le modèle temporel, sont supportés. En fait la liste des formats supportés par `ProSpect` s'allonge de jour en jour car il est très facile de rajouter un nouveau format, qui viendra s'insérer sous forme de *plug-in* dans l'architecture générale.

Même si il existe des fonctionnalités communes aux différents modèles sonores, l'ensemble des manipulations possibles sur un son donné dépend du modèle dans lequel ce son est exprimé.

3.1 Modèle temporel

Le terme "modèle temporel" désigne la représentation du son sous la forme de niveau de pression acoustique en un point particulier de l'espace. Tout son limité en fréquence peut être représenté par un flot d'échantillons résultant de la discrétisation de cette pression. Il est facile d'enregistrer et de reproduire un tel son, en utilisant des convertisseurs analogique/numérique et numérique/analogique, mais la manipulation du son est difficile et souvent peu intuitive. L'exemple suivant enregistre 10 secondes de son et en normalise l'amplitude avant de le rejouer :

```
(define sound (temporal-record 10))
(temporal-normalize! sound)
(temporal-play sound)
```

On peut connaître le nombre d'échantillons (`size`) et la durée en secondes (`time`) du son, mais il est impossible de les modifier. En revanche on peut tout à fait connaître et changer la fréquence d'échantillonnage (`rate`) des sons temporels. Pour passer par exemple d'un format DAT à 48000 Hz au format CD à 44100 Hz, en perdant inévitablement le contenu fréquentiel du son d'origine au-delà de 22050 Hz, il suffit de faire :

```
(define s (temporal-resample (temporal-load "sound.wav") temporal-rate-cd))
```

Il est également possible de construire de nouveaux sons par des superpositions (`mix`) et des séquences (`seq`) de sons existants, à condition qu'ils aient la même fréquence d'échantillonnage. Le programme suivant construit ainsi le son `s4`, composé de la superposition des sons `s1` et `s2` suivie en séquence du son `s3` :

```
(define s1 (temporal-load "sound1.aiff"))
(define s2 (temporal-load "sound2.aiff"))
(define s3 (temporal-load "sound3.aiff"))
(define s4 (temporal-seq (temporal-mix s1 s2) s3))
```

En fait le nombre de manipulations autorisées sur le modèle temporel est très limité. L'amplitude du son peut être aisément modifiée, en revanche la fréquence, la durée et le timbre sont étroitement liés et il est très difficile de changer l'un sans changer l'autre. Les modèles spectraux quant à eux permettent de séparer ces paramètres, car ils font apparaître le contenu fréquentiel du son à un instant donné. Ils paramètrent le son au niveau du récepteur, en tenant compte si possible de la perception.

3.2 Synthèse additive

L'analyse de McAulay et Quatieri [MQ86], implantée dans Lemur [FH96], extrait des sons les partiels, oscillateurs sinusoïdaux dont les amplitudes et les fréquences évoluent lentement dans le temps. Ces évolutions sont échantillonnées et, comme dans le cas du modèle temporel, il est possible de connaître et de modifier leur taux d'échantillonnage (`rate`). On peut également effectuer des étirements temporels ou des filtrages parfaits. Il est possible d'amplifier et de transposer un son, mais cette transposition ne tient pas compte des formants du son.

Ce modèle permet de reproduire de nombreux sons, à condition toutefois qu'ils soient sans bruit ni transitoires. *ProSpect* supporte de nombreux formats de fichiers spectraux, compressés ou non. Les primitives sur les sons spectraux sont trop nombreuses pour être énumérées ici, mais certaines sont illustrées dans les applications décrites en section 4.

Cependant les modèles sonores basés sur la synthèse additive sont souvent difficiles à utiliser pour l'édition ou la création de sons. La raison de cette difficulté est le trop grand nombre de paramètres éloignés des paramètres musicaux perçus par un auditeur.

3.3 Synthèse Additive Structurée

Le modèle de Synthèse Additive Structurée (SAS) a été introduit dans [DCM99]. Basé sur la synthèse additive, il impose des contraintes sur les paramètres de cette dernière pour en mettre en évidence d'autres, mais cette fois aussi proches de la perception et de la terminologie musicale que possible. Ainsi dans ce modèle tout son monophonique peut être décrit en terme d'amplitude, fréquence, couleur et inharmonicité, grandeurs variant lentement dans le temps.

Ce modèle favorise l'unification de représentation du son et de la musique à un niveau sous-symbolique, car la définition d'un son ou d'une opération musicale peut s'y faire de la même manière.

Un son S dans le modèle SAS est de la forme : (A, F, C, W), où les deux premiers paramètres (l'amplitude A et la fréquence F) sont unidimensionnels, fonction du temps uniquement, tandis que les deux autres (la couleur C et l'inharmonicité W) sont bidimensionnels, fonctions de la fréquence et du temps. Tous ces paramètres sont des fonctions qui varient lentement dans le temps :

- amplitude (*amplitude*) A : temps $\rightarrow$ amplitude
- fréquence (*frequency*) F : temps $\rightarrow$ fréquence
- couleur (*color*) C : fréquence $\times$ temps $\rightarrow$ amplitude
- inharmonicité (*warping*) W : fréquence $\times$ temps $\rightarrow$ fréquence

Dans l'implantation du modèle SAS réalisée, quatre fonctions récupèrent la valeur des paramètres du modèle à un instant donné (`sas-a-get`, `sas-f-get`, `sas-c-get` et `sas-w-get`), et quatre autres permettent de changer ces valeurs (`sas-a-set !`, `sas-f-set !`, `sas-c-set !` et `sas-w-set !`). Il est alors possible d'effectuer toutes les autres manipulations nécessaires sur ces paramètres directement en Scheme, comme indiqué dans la section suivante.

Il existe en fait deux autres modèles sonores dans *ProSpect*. Le modèle harmonique place une contrainte forte de proportionnalité entre les fréquences des partiels. Il s'agit en fait d'une restriction du modèle SAS, la notion d'inharmonicité n'existant pas. Il permet cependant de manipuler l'amplitude, la fréquence et la couleur des sons. Le deuxième modèle est celui du vocodeur de phase [Ser97a], qui utilise la transformée de Fourier à court terme pour produire, à partir d'un son temporel, une séquence de spectres pris à des instants successifs dans le temps. Ce modèle n'est pas implanté directement dans *ProSpect*, mais il l'est dans le logiciel d'analyse *InSpect* décrit dans la section suivante.

4 Applications

ProSpect est une plate-forme de développement qui peut être utilisée seule ou bien servir de base à d'autres logiciels, comme le logiciel d'analyse sonore *InSpect* ou *BOXES*, un séquenceur audionumérique avec contraintes développé par Anthony Beurivé.

4.1 Analyse et synthèse

InSpect [MS99] est un logiciel d'analyse sonore initialement écrit en C + *Tcl/Tk*, puis réécrit en utilisant *ProSpect* comme base. Ce logiciel permet de passer d'un modèle sonore à un autre et notamment d'obtenir l'expression du son dans un modèle spectral à partir de sa version temporelle.

InSpect dispose de nombreuses fonctionnalités, qui sont des extensions de *ProSpect* et sont donc à ce titre réutilisables dans d'autres programmes. Parmi ces fonctionnalités on peut citer de nombreuses méthodes d'analyse spectrale à court terme (et notamment la transformée de Fourier à l'ordre n [Mar98]), des techniques de suivi de partiels ainsi que des algorithmes d'extraction de paramètres sonores ou musicaux (hauteur, volume, brillance).

On peut également y trouver de nombreuses méthodes de synthèse spectrale, ce qui a notamment permis de comparer leurs performances. La plus rapide d'entre elles a été implantée directement dans le noyau du système Linux dans le module de synthèse *ReSpect* [MS99] et *ProSpect* l'utilise pour effectuer la synthèse spectrale en temps réel.

4.2 Programmer les sons

ProSpect a déjà été utilisé directement, sans logiciel supplémentaire, lors de la composition du quatrième fragment de “Sept Couronnes pour Goethe” par Jean-Michel Rivet en 1999. Le modèle sonore utilisé a été le modèle SAS, qui permet de ne pas considérer les sons comme des “boîtes noires” sur lesquelles on applique des fonctions abstraites, mais plutôt comme une matière musicale à sculpter à l'aide de paramètres [Arf98].

4.2.1 Transformations sonores

Il est possible d'agir sur la durée, l'amplitude, la fréquence et le timbre des sons.

Temps

Changer la durée d'un son spectral peut se faire très facilement. L'exemple suivant fait appel à la fonction `stretch-time` qui permet ici d'étirer le son `s0` dans le temps, depuis l'instant initial 0 et jusqu'à la fin du son étiré, en utilisant une vitesse de déplacement dans le son qui est constante dans le temps. Une vitesse de 1 redonnerait le son original.

```
; original
(define s0 (spectral-load "cello.msc"))
; ralenti 2 fois
(define s1
 (spectral-stretch-time s0 0 (* (spectral-time s0) 2) (lambda (t) 0.5)))
; accéléré 2 fois
(define s2
 (spectral-stretch-time s0 0 (* (spectral-time s0) 0.5) (lambda (t) 2)))
```

Cependant la fonction du temps passée en paramètre n'est pas forcément constante. Elle peut être vue comme un signal, avec des variations plus lentes que le signal du son résultant. Ce mécanisme permet de réaliser une composition multiéchelle [Vag98].

Amplitude et fréquence

Amplifier ou transposer un son spectral est tout aussi facile. La fonction d'amplification (resp. de transposition) prend en paramètre le son de référence (ici `s0`) ainsi qu'un facteur d'amplification (resp. de transposition) qui peut varier en fonction du temps.

```
; fade-in
(define (fade-in s) (lambda (t) (/ t (spectral-time s))))
(define s3 (spectral-amplify s0 (fade-in s0)))

; octave supérieure
(define s4 (spectral-transpose s0 (lambda (t) 2)))
```

Couleur et inharmonicité

Effectuer un filtrage parfait d'un son spectral est aisé. On peut également agir sur l'harmonicité d'un son grâce à la distorsion harmonique. L'exemple suivant illustre ces possibilités :

```
(define (filter f a d t) (* a (- (/ f 500) 1)))
(define s5 (warp-amplitude s0 filter))

(define (warper f a d t) (/ (pow f 2) 4000))
(define s6 (warp-frequency s0 warper))
```

Le filtre précédent (`filter`) est en fait une fonction retournant son gain en fonction d'une fréquence `f`, mais aussi d'une amplitude `a`, de la durée totale du son `d` et de la date actuelle `t`. Ce dispositif complexe permet de définir des filtres tout à fait intéressants. On peut aussi

$$\begin{tikzcd}
(A_1, F_1, C_1, W_1) \arrow[dr] & (A_2, F_2, C_2, W_2) \arrow[dl] \\
(A_1, F_1, C_2, W_1) & (A_2, F_2, C_1, W_2)
\end{tikzcd}$$

FIG. 1: synthèse croisée par échange de couleurs.

utiliser le fait que dans le modèle SAS un filtre est simplement défini par sa couleur en fonction du temps :

```
(define (sas-filter! sound filter)
 (do-loop 0 (sas-size sound) 1  ; parcours du son du début à la fin
  (lambda (i)
   (sas-c-set! sound i
    (sas-envelope-multiply (sas-c-get sound) (sas-c-get filter))))))

(define (sas-example)
  (let* ((source (sas-load "source.sas"))
         (filter (sas-load "filter.sas")))
   (sas-filter! source filter)
   (sas-save source "result.sas")))
```

4.2.2 Hybridations sonores

Un des avantages du modèle SAS est son aptitude à produire des sons hybrides. On peut par exemple effectuer de nombreux types de synthèses croisées uniquement en interchangeant des paramètres entre plusieurs sons. La figure 1 illustre une synthèse croisée sur le paramètre couleur entre deux sons S_1 and S_2. La programmation d'une telle transformation avec *ProSpect* est triviale. Bien évidemment il est également possible de mélanger les paramètres de différents sons. Le programme suivant réalise un morphing simple d'un premier son vers un second, sans tenir compte des formants :

```
(define (morpher v1 v2 alpha)
  (* (pow v1 (- 1 alpha)) (pow v2 alpha)))

(define (sas-envelope-blend e1 e2 morpher alpha)
 (if (= (vector-length e1) (vector-length e2))
  (let ((e (make-vector (vector-length e1))))
   (do-loop 0 1 (vector-length e)
    (lambda (i)
     (vector-set! e i (morpher (vector-ref e1 i) (vector-ref e2 i) alpha))))
   e)))

(define (sas-morphing s1 s2 morpher)
 (let* ((n (max (sas-size s1) (sas-size s2)))
        (s (sas-make n)))
  (do-loop 0 n 1
   (lambda (i)
```

```
    (let* ((alpha (/ i (- n 1)))
           (a1 (sas-a-get s1 i))
           (f1 (sas-f-get s1 i))
           (c1 (sas-c-get s1 i))
           (w1 (sas-w-get s1 i))
           (a2 (sas-a-get s2 i))
           (f2 (sas-f-get s2 i))
           (c2 (sas-c-get s2 i))
           (w2 (sas-w-get s2 i))
           (a (morpher a1 a2 alpha))
           (f (morpher f1 f2 alpha))
           (c (sas-envelope-blend c1 c2 morpher alpha))
           (w (sas-envelope-blend w1 w2 morpher alpha)))
     (sas-a-set! s i a)
     (sas-f-set! s i f)
     (sas-c-set! s i c)
     (sas-w-set! s i w))))
  s))

(define (sas-example)
 (let* ((s1 (sas-load "source1.sas"))
        (s2 (sas-load "source2.sas"))
        (s (sas-morphing s1 s2 morpher)))
  (sas-save s "target.sas")))
```

4.3 Aide à la composition

Il existe également des programmes permettant d'effectuer ce genre de transformations musicales sans avoir à manipuler des algorithmes. *BOXES* en est un bon exemple. Il s'agit d'un séquenceur audionumérique avec contraintes, développé par Anthony Beurivé. Il s'agit d'un logiciel d'aide à la composition musicale qui utilise les modèles spectraux de *ProSpect* et dispose donc de possibilités de transformations sonores avancées, ainsi que d'une synthèse en temps réel. Outre le fait de manipuler des sons spectraux, *BOXES* innove en proposant un mécanisme original d'édition avec contraintes qui peut faciliter grandement le processus de composition musicale.

5 Conclusions et perspectives

ProSpect est une plate-forme logicielle libre développée pour le système Linux et qui permet la manipulation (analyse, transformation, synthèse) des sons dans divers modèles sonores, principalement spectraux. Cette plate-forme offre la possibilité de pouvoir programmer les sons en agissant directement sur eux par les instructions d'un langage fonctionnel. *ProSpect* sert de base à des logiciels allant de l'analyse sonore (*InSpect*) à l'aide à la composition musicale (*BOXES*). Son ouverture et son extensibilité permettent de rajouter sans cesse de nouvelles fonctionnalités. Nous espérons que dans l'avenir un plus grand nombre de personnes intéressées interviendront dans son développement et l'utiliseront comme base pour de nombreux autres logiciels dans le domaine de l'informatique musicale.

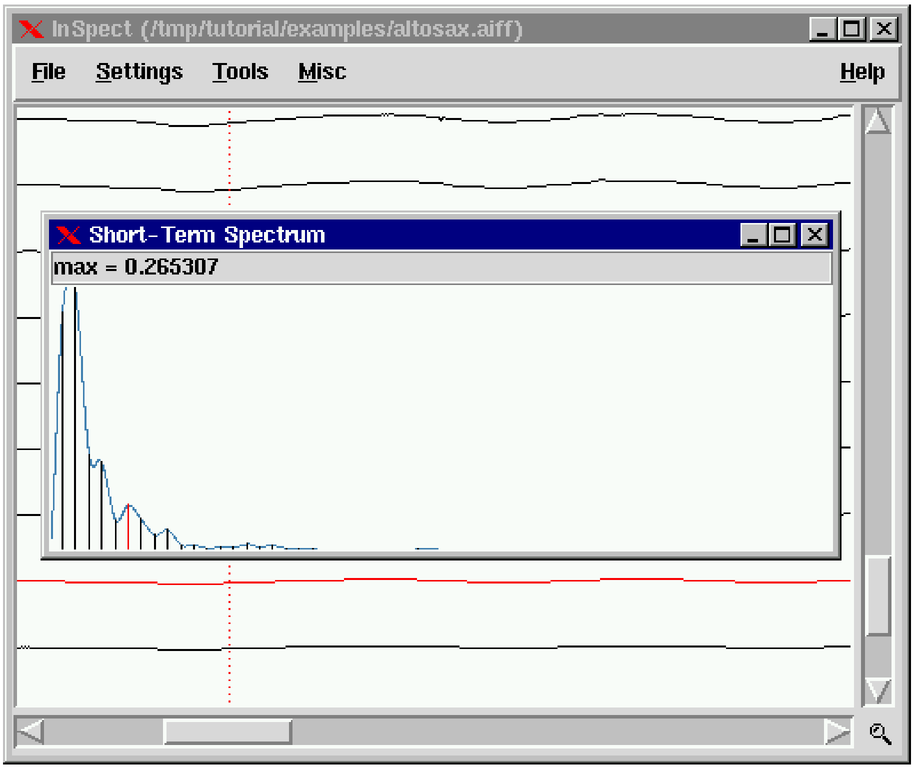

FIG. 2: *InSpect* affichant les évolutions dans le temps des fréquences des partiels d'un son de saxophone alto (sur 0.7 secondes). Au premier plan est affiché un spectre à court terme ainsi que l'enveloppe spectrale correspondante.

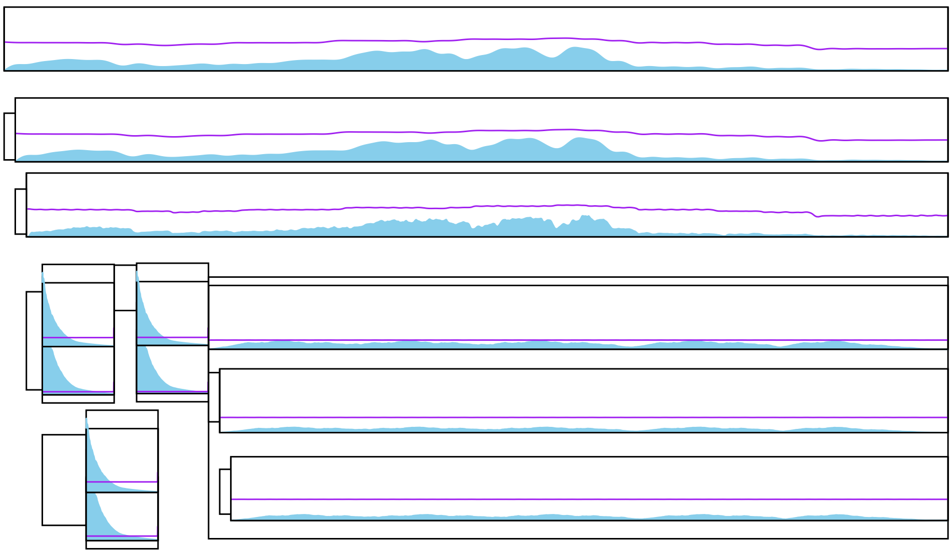

FIG. 3: une pièce musicale composée avec le logiciel *BOXES*.

Références

[Arf98] Daniel Arfib. *Recherches et applications en informatique musicale*, chapter 19 : Des courbes et des sons, pages 277–286. Hermes, Paris, 1998.

[DCM99] Myriam Desainte-Catherine and Sylvain Marchand. Vers un modèle pour unifier musique et son dans une composition multiéchelle. In *Proceedings of the Journées d'Informatique Musicale (JIM'99)*, pages 59–68, Paris, mai 1999. CEMAMu.

[FH96] Kelly Fitz and Lippold Haken. Sinusoidal Modeling and Manipulation Using Lemur. *Computer Music Journal*, 20(4) :44–59, 1996.

[FJ00] Matteo Frigo and Steven G. Johnson. FFTW. URL http ://www.fftw.org, 2000.

[Fun91] Free Software Fundation. GNU General Public License. URL http ://www.fsf.org/copyleft/gpl.html, 1991.

[Gal99] Erick Gallesio. STk. URL http ://kaolin.unice.fr/STk, 1999.

[LLP99] Fernando Lopez-Lezcano and Juan Pampin. Common Lisp Music update report. In *Proceedings of the International Computer Music Conference (ICMC'99)*, pages 399–402, Pékin, octobre 1999. International Computer Music Association (ICMA).

[Mar98] Sylvain Marchand. Improving Spectral Analysis Precision with an Enhanced Phase Vocoder using Signal Derivatives. In *Proceedings of the Digital Audio Effects (DAFx'98) Workshop*, pages 114–118, Barcelone, novembre 1998.

[MQ86] Robert J. McAulay and Thomas F. Quatieri. Speech Analysis/Synthesis Based on a Sinusoidal Representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4) :744–754, 1986.

[MS99] Sylvain Marchand and Robert Strandh. InSpect and ReSpect : spectral modeling, analysis and real-time synthesis software tools for researchers and composers. In *Proceedings of the International Computer Music Conference (ICMC'99)*, pages 341–344, Pékin, octobre 1999. International Computer Music Association (ICMA).

[Pam99] Juan Pampin. ATS : a Lisp Environment for Spectral Modeling. In *Proceedings of the International Computer Music Conference (ICMC'99)*, pages 44–47, Pékin, octobre 1999. International Computer Music Association (ICMA).

[Ser97a] Marie-Hélène Serra. *Musical Signal Processing*, chapter Introducing the Phase Vocoder, pages 31–90. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[Ser97b] Xavier Serra. *Musical Signal Processing*, chapter Musical Sound Modeling with Sinusoids plus Noise, pages 91–122. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[Vag98] Horacio Vaggione. Transformations morphologiques : quelques exemples. In *Actes des Journées d'Informatique Musicale (JIM)*, page G1, 1998.

[VGS98] B. L. Vercoe, W. G. Gardner, and E. D. Scheirer. Structured Audio : The creation, transmission, and rendering of parametric sound representations. *Proceedings IEEE*, 86(5) :922–940, 1998.