Gauthier Lebrun

RAPPORT DE FIN D'ÉTUDES

de l'IUP Génie Informatique de LA ROCHELLE

Travail effectué à l'I.R.C.A.M.

Développement d'un logiciel de visualisation et d'analyse de signaux sonores

Xspect

Février - Juin 1997

Responsable à l'IRCAM : Xavier Rodet
Responsable à l'IUP : Michel Ménard

Institut de Recherche et de Coordination Acoustique/Musique

Remerciements

Je tiens à remercier Xavier Rodet, responsable de l'équipe Analyse / Synthèse, pour sa patience, son aide attentive, et ses précieux conseils pendant toute la durée de mon stage.

Je remercie l'IUP Génie Informatique de La Rochelle de m'avoir permis de réaliser ce stage à l'IRCAM, et particulièrement Michel Ménard et Bernard Besserer.

Je remercie Dominique François, et Guillaume Levy pour le code clair et bien écrit dont j'ai hérité.

Enfin je remercie Philippe Depalle, co-responsable de l'équipe Analyse / Synthèse, Christophe Vergez, Stéphane Rossignol, Stéphan Tassart, Geoffroy Peeters ainsi que les autres membres de l'équipe qui par leur accueil chaleureux, ont largement contribué à la réussite de mon stage.

Abstract

IRCAM (Institut de Recherche et Coordination Acoustique / Musique) is one of the most important music research center in the world. The aim of IRCAM is to promote work in common between music composers and scientists. Research at IRCAM is based on four axis : acoustic phenomenoms, sound synthesis, computer assisted creation, and computer interaction.
The Analysis and Synthesis Team builds tools in order to help musicians in understanding, studying and creating new sounds. In computer music, as in speech field, sound signals are examined, listened to, measured, analysed, processed and synthesized by musicians and scientist. For this type of work, a signal visualisation and analysis program (SVAP) is needed.
Also, scientists wanted a SVAP with the following properties: portability on different UNIX platforms, use of standard graphical libraries, client-server architecture on ethernet, suitability to compare signals, support for arbitrary long signals and differents file formats, user adaptibility and extensibility, and a reasonable price for a multi-users and multi-machine licence. Having not found a program which could fulfill their prerequisites, the Analysis and Synthesis Team decided to write a new SVAP, named Xspect.
Created two years ago, Xspect was providing many of these capabilities but a lot of others were still lacking. After the correction of a few features (like resource and memory management), I was able to add some new functionalities. Indeed, Xspect is now able to handle its own colormap in order to work together with other color intensive applications. Moreover, Xspect has now an improved sound player (on SGI and DEC ALPHA architectures) and extended analysis capabilites (like zero-padding for Fast Fourier Transform). Finally, Xspect now offers a new breakpoint function editor, for example to create filter templates.
Xspect is now a reliable software for scientists who want to study sounds with precision. It is always under development in order to follow the scientists needs. New features such as sonagram display are planed.

Résumé

L'IRCAM (Institut de Recherche et Coordination Acoustique / Musique) est l'un des plus importants centres de recherche musicale au monde. Le but de cet institut est de promouvoir le travail en commun entre les compositeurs et les scientifiques. La recherche à l'IRCAM est basée sur quatre axes : les phénomènes acoustiques, la synthèse sonore, la composition assistée par ordinateur, et l'interaction avec l'ordinateur.

L'équipe analyse/synthèse conçoit des outils afin d'aider les musiciens dans la compréhension, l'étude et la création de nouveaux sons. En musique électronique, les signaux sonores sont examinés, écoutés, mesurés, analysés, synthétisés par les musiciens et les scientifiques. Pour ce type de travail, un programme de visualisation et d'analyse de signaux est nécessaire.

Les scientifiques désiraient un programme répondant aux caractéristiques suivantes : portabilité sur différentes plateformes UNIX, utilisation de bibliothèques graphiques standards, une architecture client-serveur sur éthernet, aptitude à recevoir de longs signaux et différents formats de fichiers, possibilité de personalisation et d'extension, et un prix raisonnable pour une licence multi-utilisateurs et multi-machine. N'ayant pas trouvé de programmes commerciaux répondant à ces prérequis, l'équipe analyse/synthèse a décidé de créer son propre programme, appelé Xspect.

Créé il y a deux ans, Xspect répondait à la plupart de ces caractéristiques, mais il restait un bon nombre de spécificités à développer. Après une phase de corrections (telles que la gestion des ressources et de la mémoire), j'ai pu ajouter de nouvelles fonctionalités. En effet, Xspect gère maintenant sa propre palette de couleurs, lui permettant de fonctionner avec d'autres applications gourmandes de couleurs. De plus Xspect peut maintenant jouer les sons quelque soit leur fréquence d'échantillonnage. Enfin, Xspect a été complété d'un éditeur de fonctions breakpoints nous permettant par exemple d'éditer des gabarits de filtres.

Xspect est donc un outil quotidien pour les scientifiques de l'IRCAM qui désirent étudier les sons de manière précise. Xspect est continuellement en cours de développement afin de suivre les désirs des scientifiques, et de nouvelles fonctionnalités telles que la visualisation de sonagrammes sont prévues.

Table des matières

Table des figures

Introduction

De toutes les nouvelles technologies, l'informatique est sûrement celle qui a réussi à s'intégrer aux mieux dans tous les domaines de la vie courante. Ainsi, le milieu artistique fait lui aussi appel à ses services pour assouvir les besoins des compositeurs dans la recherche de sons nouveaux.

C'est dans l'esprit d'une continuelle interaction entre les musiciens, les compositeurs, et les scientifiques qu'a été créé l'IRCAM (Institut de Recherche et Coordination Acoustique / Musique), cherchant ainsi à mettre au service des musiciens les dernières nouveautés de la recherche scientifique.

Parmi les multiples pôles de recherche, l'équipe "Analyse / Synthèse" étudie les différentes stratégies de modélisation et de transformation des sons. C'est dans cette équipe que s'est déeroulée mon stage dans le but de poursuivre le développement d'un logiciel de visualisation et d'analyse des signaux sonores intitulé Xspect.

Après une bref description de l'IRCAM, je présenterai Xspect et les outils de développements mis à ma disposition. J'exposerai ensuite les améliorations que j'ai apporté à Xspect. Enfin, j'apporterai quelques points concernant l'adéquation entre la formation apportée par l'IUP et la réalité d'un tel stage.

Première partie

Généralités

Chapitre 1

Présentation de l'IRCAM

En 1970, le président Georges Pompidou demande à Pierre Boulez de créer et de diriger un institut de recherche musicale associé au futur Centre national d'Art Contemporain. L'IRCAM (Institut de Recherche et Coordination Acoustique / Musique) est alors fondé dans le but de lier la science à la musique, et d'intégrer la connaissance technologique à l'imagination du compositeur, "*montrant ainsi que recherche et création,(...), peuvent être perçues à la fois comme un processus de mutation et de prolongation de la connaissance accumulée*[1]".

1.1 Les objectifs

1. Une recherche multiforme mutuellement enrichissante pour les chercheurs et les compositeurs.
2. Un espace de création en temps réel en offrant aux jeunes talents un cadre de réalisation technique performant.
3. Un grand pôle pédagogique pour les compositeurs qui doivent se familiariser avec le maniement des technologies contemporaines, ainsi que pour les étudiants qui doivent avoir accès aux démarches qui guident la recherche et la création musicale aujourd'hui.
4. Une médiation active en développant une activité éditoriale, en proposant une médiathèque, ou encore par les nombreuses journées portes ouvertes.

1.2 La recherche

Au service de la musique, la recherche scientifique explore de nouveaux territoires acoustiques. Elle élargit les règles du jeu informatique, elle étudie la vie et les métamorphoses des timbres, elle projette des outils inédits.

L'IRCAM a privilégié quatre axes de recherche :

1. **Les phénomènes acoustiques :** Etude des sons, des lieux d'écoute, perception auditive.

1. Laurent BAYLE, directeur de l'IRCAM (1997)

2. **La synthèse sonore :** en explorant de nombreux modèles afin d'accroître le potentiel sonore et permettant l'exploration de sonorités inédites.
3. **La composition assistée par ordinateur :** recherche sur la représentation symbolique des formes et matériaux musicaux.
4. **L'interaction avec l'ordinateur :** allant de l'interface homme / machine jusqu'au temps réel lors des concerts.

De ces quatre axes de recherche ont été fondés les différents pôles de recherche et de développement suivants :

L'acoustique instrumentale : Cette équipe étudie le fonctionnement des instruments de l'orchestre en cherchant à :

- Améliorer les méthodes de fabrication,
- Etendre les possibilités techniques et sonores,
- Etablir des modèles de simulation synthétiques d'instruments existants.

L'acoustique des salles : Cette équipe étudie et évalue les espaces acoustiques, permettant ainsi de développer :

- des outils d'aide à la conception acoustique,
- des simulateurs d'effet de salle ou de mouvement de sources sonores.

La psychoacoustique : Cette équipe étudie les mécanismes de l'audition et de la perception du discours musical et fournit :

- des repères pour l'élaboration de structures sonores complexes,
- des méthodes pour le contrôle des sons de synthèse.

Les représentations musicales : C'est avec une collaboration étroite avec les musiciens que cette équipe tente :

- de formaliser les structures du discours musical,
- de développer des outils de conception assistée par ordinateur,
- des méthodes d'analyse et de description musicales automatisées.

Les systèmes temps-réels : Cette équipe est chargée de perfectionner les plates-formes existantes telles que la Station d'Informatique Musicale, ou d'étendre les possibilités d'interaction aux dispositifs multimédias.

Les services en ligne : Face à une croissance de plus en plus importantes des réseaux, cette équipe cherche à :

- créer des accès à distance pour les outils et contenus de la recherche musicale.
- développer des technologies permettant de faciliter et de maîtriser leur circulation.

L'analyse/synthèse : c'est au sein de cette équipe, dirigée par Xavier RODET et Philippe DEPALLE, que j'ai poursuivi le développement du logiciel Xspect.

1.3 L'équipe Analyse-Synthèse

1.3.1 Présentation

Cette équipe étudie les différentes stratégies de modélisation des sons. En étroite relation avec les musiciens et les chercheurs, elle cherche à créer des outils leur permettant d'étudier, de transformer, ou même de créer les sons.

L'analyse du son permet de structurer de manière intelligible l'information véhiculée par le son. Cette analyse est primordiale, car elle permet d'expliquer les phénomènes sonores et offre des moyens d'action pour le modifier d'une manière précise (comme augmenter la vitesse sans changer la hauteur du son, par exemple).

Par la synthèse des sons, on propose aux musiciens plusieurs outils ou modèles, permettant la création des sons à partir des éléments d'analyse. Plusieurs modèles sont proposés à l'IRCAM, allant du synthétiseur Chant au modèle physique de trompette (en cours de finalisation).

1.3.2 Les réalisations

Les réalisations de l'équipe visent donc au développement d'outils de traitement du signal, ainsi qu'à l'élaboration de méthode d'analyse et de synthèse de sons (que ce soit de la parole, ou un instrument).
L'équipe a produit de nombreux outils pour l'analyse et la synthèse sonore, parmi lesquels nous pouvons citer :

Chant : bibliothèque d'outils proposant à la fois une analyse et une synthèse fondées sur le modèle de la voix. Cette bibliothèque va bientôt être intégrée à *Diphone*.

SVP - Audiosculpt (Super Vocodeur de Phase) : Système modulaire d'analyse, de traitement, et de synthèse de sons qui intègre le concept de traitement du signal par bloc. C'est un outil puissant de modification de timbre. Une interface graphique, Audiosculpt, a été développée afin de rendre son utilisation plus intuitive.

Diphone : Programme fondé sur le concept du contrôle par diphone généralisé. Déjà connue en synthèse de la parole, cette méthode est généralisée à la synthèse des sons musicaux. Elle fournit un moyen puissant de construire une phrase musicale à partir d'un dictionnaire d'éléments sonores en les interpolant (on peut ainsi, dans une même phrase, passer continuement d'un son de violon à une voie chantée).

Outre ces réalisations logicielles, l'équipe analyse/synthèse réalise des projets en collaboration avec le monde industriel et culturel. Parmi les nombreux projets, nous pouvons citer sans doute le plus médiatique de l'IRCAM : la synthèse de la voix artificielle du castrat du film *Farinelli* de Gérard Corbiaud.

Chapitre 2

Présentation générale de Xspect

2.1 Introduction

L'analyse des sons demande des outils permettant leur visualisation, écoute, mesure, ou comparaison. L'IRCAM a donc besoins pour ces travaux d'un logiciel capable d'effectuer ces manipulations. Il doit en outre répondre aux caractéristiques suivantes :

- Portabilité sur différentes plates-formes UNIX, notamment DEC ALPHA et SGI.
- Utilisation de librairies graphiques standards.
- Supportant l'étude de long signaux, et des formats de fichiers différents.

N'ayant pas trouvé sur le marché un logiciel répondant à leurs critères (techniques et économique), l'équipe Analyse/Synthèse s'est appliquée au développement d'un tel outil, depuis près de 3 ans.

Xspect est un logiciel graphique permettant l'affichage, la manipulation la comparaison des sons d'une manière simple et intuitive. Il est également capable d'effectuer différentes analyses telles que FFT ou calcul de la Fréquence fondamentale sur des parties de signaux.

Écrit en langage C, et faisant appel à la librairie graphique X/MOTIF, Xspect a été développé de manière simultanée sur les systèmes DEC Alpha OSF1 V3.2, et SGI IRIX 5.3 et IRIX 6.2. De par son écriture, Xspect est facilement portable sur d'autres plates-formes UNIX.

Xspect tente dans la mesure du possible de manipuler les fichiers et les résultats à la manière d'Emacs. Ainsi, chaque son est placé dans un *buffer*. Utilisant le "*memory mapping*[1]", Xspect accepte par conséquent des signaux de toutes tailles, seulement limités par la taille du disque.

Xspect ouvre des fenêtres (*Windows*) sur l'écran. Ces fenêtres peuvent être découpées verticalement ou horizontalement en *Frame* et *SubFrame* (Cf. FIG 2.1). Cela permet donc facilement de comparer, de regarder les différents canaux, d'avoir un spectre sous un

1. Memory mapping: opération consistant à organiser un fichier dans la mémore virtuelle

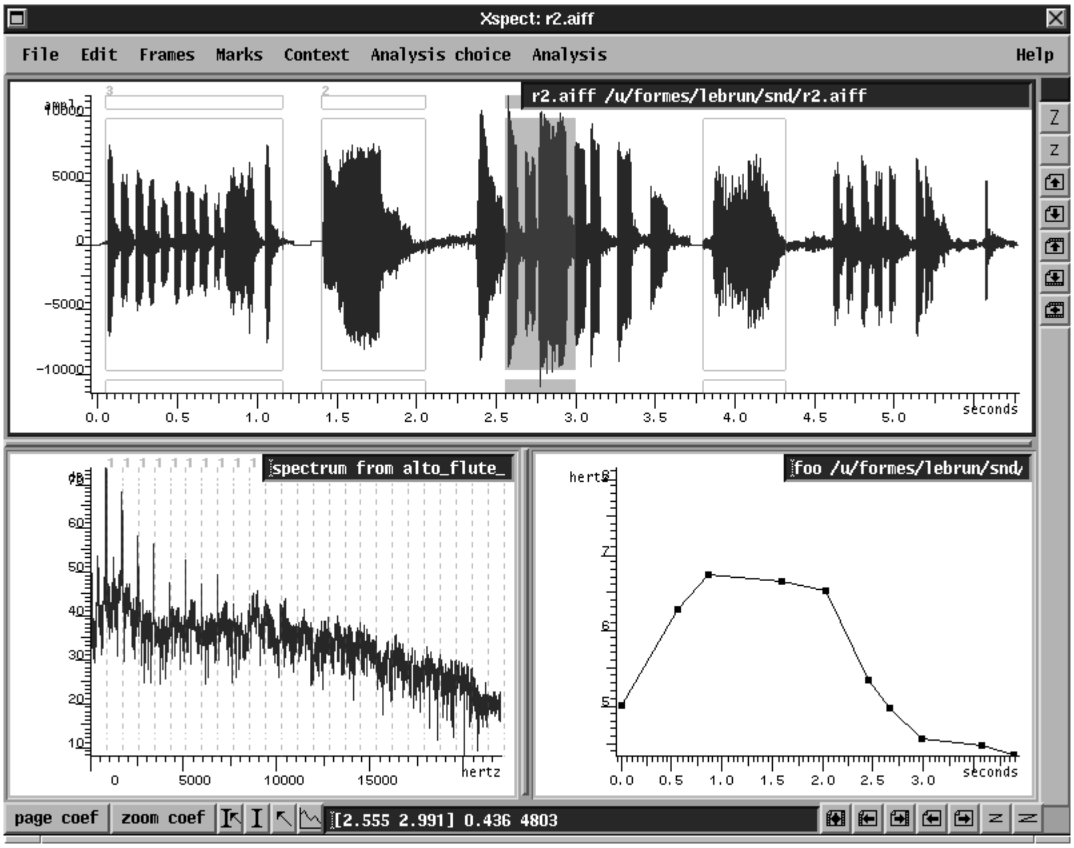

FIG. 2.1 – *Exemple d'affichage*

signal, etc.

Dans chaque *Frame*, une ou plusieurs *Views* peuvent être superposées. Ces *views* proposent de multiples sélections.

Xspect permet également la synchronisation automatique d'actions entre plusieurs *views*. Par exemple, le déplacement d'une sélection peut forcer le rafraîchissement d'une analyse située sur une autre *view*. Xspect offre ainsi beaucoup de facilités pour ajuster précisément les affichages, les analyses, les zooms, etc.

Dans la figure 2.1, la fenêtre est décomposée en trois *frames*. Dans la *frame* du haut, on peut apercevoir des sélections (3 désactivées, et une activée). Dans la *frame* en bas à gauche, on peut voir un spectre ainsi que les harmoniques associées (en pointillées). Dans la *frame* en bas à droite, nous pouvons voir une fonction breakpoint.

2.2 Les principaux objets graphiques

Les *windows*, les *frames*, les *views* et les *buffers* sont les principaux objets graphiques du logiciel.

2.2.1 Les *Windows*

Une *window* est la fenêtre de travail de Xspect. Elle possède une barre de menus permettant l'accès à toutes les fonctions de visualisation et d'analyse. Quelques boutons apparaissent également permettant le zoom, le déplacement au début ou à la fin du signal, ainsi que les changements de mode (sélection, édition,).

2.2.2 Les *Frames*

Lorsqu'on désire afficher plusieurs signaux différents dans une même fenêtre, on peut diviser celle-ci en *Frames*. Ce découpage peut se faire verticalement ou horizontalement grâce aux fonctions Split Frame Horizontally et Split Frame Vertically.
Dans une fenêtre donnée, il n'y a qu'une seule *Frame* active à la fois. Cette *Frame* active est marquée d'un bord bleu (les *frames* inactives ayant un bord rouge).

2.2.3 Les *Views*

Par souci de convivialité et de précision, le programme permet de superposer plusieurs signaux dans une *frame*. Ainsi, à chaque *view* correspond un et un seul signal (que ce soit un son ou une analyse). L'utilisateur peut donc superposer plusieurs signaux identifiés par un *identifier box* qui contient le nom du *buffer*. On peut donc par exemple superposer les deux canaux d'un fichier son stéréo.
Une *view* active aura son *identifier box* de couleur bleu. À l'inverse, une *view* inactive aura son *identifier box* de couleur rouge.

2.2.4 Les *Buffers*

Mettre un signal dans une *view* consiste à créer un *buffer*. Cette nouvelle structure contient le signal à afficher dans la *view*, qu'il provienne d'un fichier son ou d'une analyse. Le *buffer* contiendra toutes les données caractéristiques du signal. Il est identifié par un identifier (le nom du fichier son ou le type de l'analyse) qui sera spécifié dans l'*identifier box* de la vue dans laquelle il est affiché.
Ainsi, un même *buffer* peut être associé à plusieurs *views*.
Ceci est particulièrement utile pour l'affichage de plusieurs canaux d'un même signal (ex. son stéréo). En effet, les différents canaux appartiennent tous à un unique *buffer*. Aussi, pour afficher les différents canaux d'un signal, il suffit de créer plusieurs *views*, et d'affecter chaque canal à la *view* désirée. Le numéro du canal d'un son multi-canaux est précisé dans l'identifier.

2.3 Outils de marquage et de sélection (*marks*)

Pour comparer, mesurer, mettre des repères sur des signaux, il est indispensable de pouvoir sélectionner des parties spécifiques et marquer des points en particulier.
On désignera par *mark* une valeur du signal pointée par un curseur. Tous les éléments décrits ci-dessous sont constitués d'une ou plusieurs *marks*.

Il est possible de créer plusieurs *marks* sur un signal, mais de la même façon que pour les *frames* et les *views*, il ne peut y avoir qu'une seule *mark* active.

2.3.1 Les cursors

Les cursors ne sont constitués que d'une seule *mark*. En cliquant avec le bouton gauche de la souris sur le signal, on dessine un trait vertical (ou SHIFT+clic gauche pour un trait horizontal) : c'est un cursor.
La valeur pointée par le cursor (temps, amplitude, fréquence en fonction du signal) apparaît lors de sa création dans l'*identifier box* de la vue.

2.3.2 Les segments

Un segment est composé de deux *marks* : début et fin. En cliquant sur le bouton gauche de la souris, puis en se déplaçant sur le signal on crée une bande verticale (shift+bouton gauche : bande horizontale). Cette bande délimite une partie du signal étudié. Dans l'identifier box apparaissent les valeurs de début et de fin de segment. Par abus de langage, on appelera "Sélection" la bande affichée.

2.3.3 Les diphones

Un diphone est composé d'un segment et d'un cursor (centre du diphone). Cette fonctionnalité est utile pour les travaux de repérages de sons particuliers sur un signal ; l'idée étant qu'une phrase peut-être décomposée en une succession de sons d'articulation (transition entre phonèmes / notes) et sons d'appui (voyelle stable / tenue de note) que l'on appelle diphone.

2.3.4 Les *grids*

Une *grid* apparait lors d'une analyse spectrale avec calcul de F0 (fréquence fondamentale). Cette *grid* est composée de plusieurs cursors positionnés à intervalles réguliers d'une longueur de F0. Les différentes harmoniques du signal sont donc ainsi facilement identifiables.

2.4 Les Synchronisations (menu *Frames*)

Cette fonction permet de synchroniser plusieurs vues ensembles. Ainsi, une action engendrée sur une vue provoquera une réaction sur une autre vue qui lui est liée. Nous appellerons vueSource la vue de départ sur laquelle on effectue manuellement une action ; et vueTarget la vue d'arrivée, celle qui subit l'effet de la synchronisation.

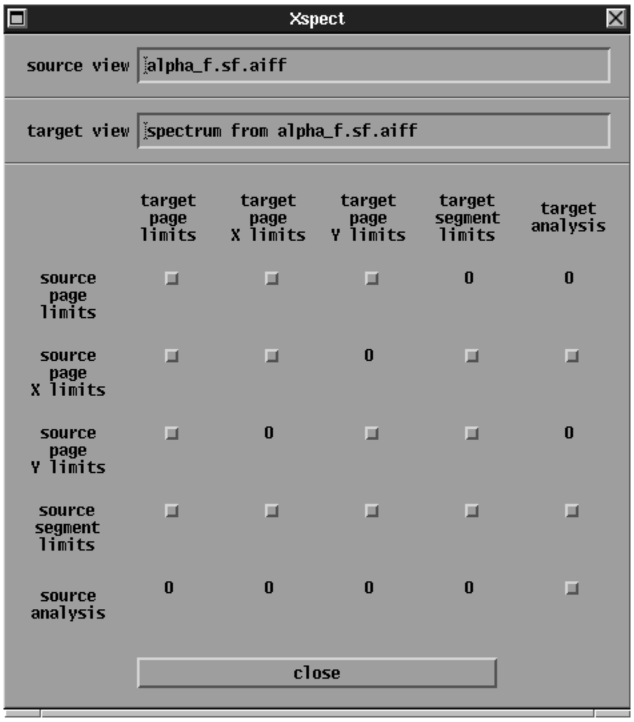

FIG. 2.2 – *Fenêtre de synchronisation*

2.4.1 Énoncé des synchronisations disponibles

Description des actions de la source qui lancent la synchronisation :

Label	Description
Source page limits	Tout changement des limites de la page du signal.
Source page X limits	Tout changement des limites de la page suivant l'axe des X.
Source page Y limits	Tout changement des limites de la page suivant l'axe des Y.
Source segment limits	Tout changement du segment actif.
Source analysis	Toute analyse recalculée (la viewSource doit être une analyse)

Description des actions sur la cible :

Label	Description
Target page limits	Mise-à-jour des limites de la page (en X et Y).
Target page X limits	Mise-à-jour des limites de la page en X.
Target page Y limits	Mise-à-jour des limites de la page en Y.
Target segment limits	Mise-à-jour du segment actif
Target analysis	L'analyse est refaite sur la sélection de la vue qu'elle concerne (la viewTarget doit être une analyse).

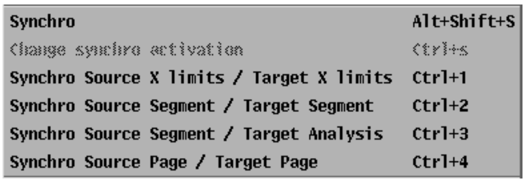

FIG. 2.3 – *Nouveaux items de synchronisation*

FIG. 2.4 – *Indicateur de synchronisation active*

2.4.2 Ajout de raccourcis clavier

La mise en œuvre d'une synchronisation est une opération relativement longue à réaliser. En effet, le programme demande à l'utilisateur de sélectionner la vue de destination, puis ouvre la fenêtre des synchronisation. L'utilisateur sélectionne la synchronisation voulue, puis referme la fenêtre.

Or, il s'avère que dans la pratique seules quelques synchronisations sont utilisées de manière courante. L'idée a donc été d'affecter des raccourcis à ces opérations.

J'ai donc rajouté quatres items dans le menu *Frames*(Cf. FIG. 2.3).

Il suffit donc d'un simple appel à ces items (ou mieux, avec leurs raccourics clavier) pour créer une synchronisation. La possibilité de création de synchronisation par l'ancienne méthode reste évidemment possible car elle nous offre un choix de synchronisations plus étendu.

2.4.3 Activation /Désactivation de Synchronisation

Lorsqu'on a créé une synchronisation, et qu'on se trouve sur la *view* Source, un symbole 'A' apparaît à côté de l'Identifier Box (Cf. FIG. 2.4). Cela indique donc qu'une synchronisation a été activée pour cette *view*.

De même, il est possible de désactiver une synchronisation. Cela est très utile lorsqu'on désire écouter un large extrait et que le signal est synchronisé avec une analyse... Le programme doit recalculer l'analyse alors qu'on en n'a pas besoin.
L'utilisateur peut donc désactiver une synchronisation d'une *view* source par l'item `Change Synchro activation` du menu *Frame*. La synchronisation n'est pas détruite mais simplement désactivée. La lettre 'D' apparaît alors à la place de la lettre 'A'.

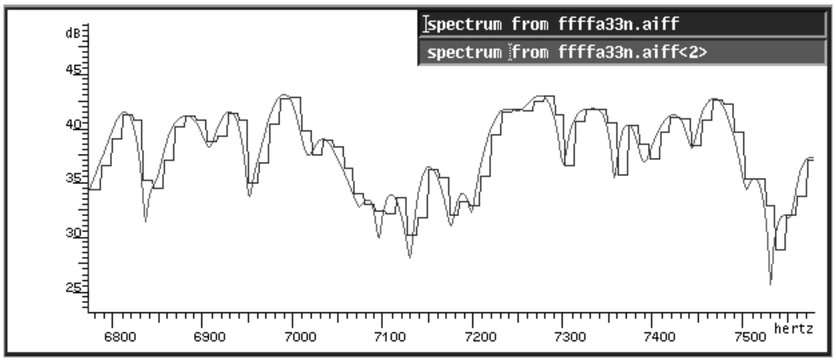

FIG. 2.5 – *Exemple de zéro-padding : En escalier : spectre normal ; courbe lisse : le même spectre mais sur un nombre d'échantillons multipliés par* 2^5

2.5 Les analyses

Xspect permet d'effectuer de nombreuses analyses sur les sons :

Set analysis type to spectrum : Analyse spectrale

Set analysis type to F0 from signal : pour avoir en plus le calcul de la fréquence fondamentale et la grille

Set analysis type to F0 from spectrum : pour faire le calcul de la fréquence fondamentale à partir d'un spectre

Set analysis type to square mag. fft : pour représenter le carré de l'amplitude du spectre

Set analysis type to log. fft : pour représenter le logarithme du spectre (représentation par défaut)

Toutes ces analyses peuvent se faire sur les fenêtres suivantes : Hanning, Hamming, exact Blackmann, Blackmann, Blackmann-Harris 3 (-67 dB), Blackmann-Harris 3 (-61 dB), Blackmann-Harris 4 (-92 dB), Blackmann-Harris 4 (-72 dB), Triangular, Rectangular.

Afin d'améliorer, la lisibilité des analyses, j'ai permis à Xspect d'affiner le calcul de ses analyses FFT par la méthode du *Zéro-padding.* Cette méthode augmente le nombre de points sur lesquels vont être calculés la FFT et effectue ainsi une interpolation. La figure 2.5 nous montre un spectre calculé de manière classique ; et le même spectre mais calculé sur un nombre de points multipliés par 2^5.

Chapitre 3

Outils de Développement

3.1 Le langage

Xspect est un programme écrit en C. Il fait appel à de nombreuses fonctions graphiques de X, Xt (X Toolkit), et MOTIF ainsi qu'à des librairies de calcul développées en interne à l'IRCAM. Des conventions d'écriture ont été adoptées afin de faciliter la relecture des sources et permet une certaine cohérence dans le code. Ainsi, des préfixes sont associés aux fonctions ayant des mécanismes d'appel spéciaux :

Type de fonction	Préfixe
les callbacks	XsCb
les timeouts callbacks	XsTcb
les actions	XsAc
les utilitaires	XsUt

Dans l'optique d'améliorer la qualité du développement à l'IRCAM, un groupe d'étude a été créé : c'est le groupe MQDL (Méthode et Qualité du Développement Logiciel).
Le but de ce groupe est d'inciter les programmeurs à structurer leurs méthodes de travail, par exemple, en fournissant une documentation précise du travail effectué, des améliorations apportées, etc. Ainsi, une charte du développement en C a été proposée afin d'améliorer la lisibilité des programmes.

3.2 L'éditeur de texte : emacs

J'ai utilisé lors de mon travail à l'IRCAM l'éditeur de texte EMACS qui reste le meilleur environnement de travail sous UNIX. En effet, il propose à l'utilisateur différents outils de base facilitant l'écriture du code C : mots clés en couleurs, indentation automatique, ou encore interface avec CVS (Contrôle de version) sont des fonctionnalités qui améliorent considérablement le travail de développement.

3.3 Le fichier Makefile

Comme le programme comporte plus d'une vingtaine de modules, il est indispensable pour la bonne gestion de la compilation de faire appel à un fichier makefile.

Ce fichier tient compte des différentes plates-formes et fait donc appel à des fichiers différents selon l'architecture du système.
Il fait le lien avec les librairies utilisées et place les objets résultants dans le répertoire `$ARCH/$SYSTEM/obj` et les exécutables dans le répertoire `$ARCH/$SYSTEM/bin`; où `$ARCH` est la variables d'environnement indiquant la plate-forme utilisée (SGI, ou ALPHA) et `$SYSTEM` la version du système utilisée (IRIX 5.3 ou 6.2, OSF1, etc.)
Lorsqu'on lance la compilation par la commande make, sans arguments, la compilation se fait de manière optimum.

Des arguments peuvent êre passés en paramètres à `make` :

Clean : supprime tous les .o et exécutables. Ceci est utile lorsqu'on désire recompiler entièrement le programme.

Proto : compile le programme avec le flag DEBUG. Ainsi, toutes les lignes de debug seront prises en considération et lors de l'exécution les messages apparaîtront sur la sortie standard. Make proto génère l'exécutable xspect.proto. Cette option est utile lorsqu'on veut "tracer" le déroulement du programme. De plus, l'option `-g` est également passé en paramètre, autorisant ainsi le passage du programme dans un débogueur.

3.4 CVS : Contrôle de version

Afin de différencier les versions de Xspect, nous avons utilisé le gestionnaire CVS. CVS est un outil permettant de gérer l'évolution et l'élaboration d'un logiciel et autorisant plusieurs développeurs à travailler en même temps puis à rassembler les différentes modifications.
L'utilisation de ce programme s'est limitée, en ce qui me concerne, à la gestion des différentes versions d'Xspect. Ainsi, nous pouvons suivre étape par étape les différentes modifications qui ont été apportées au programme. Nous pouvons également revenir à une version plus ancienne, etc. La totalité des sources de Xspect sont gérées et placées dans un répertoire "repository" appelé CVSROOT.

3.4.1 Récupération initiale des sources

J'ai pu ainsi récupérer tous les sources de Xspect en tapant simplement la commande suivante :
cvs update xspect
Cette action copie tous les fichiers de Xspect contenus dans le repository et les place dans mon répertoire courant. Des répertoires CVS sont créés et contiennent les informations propres au programme CVS. Il est donc fortement déconseillé de les effacer.

3.4.2 Création d'une nouvelle branche

Afin de ne pas toucher aux versions précédentes, j'ai créé une branche personnelle de travail. Cette branche s'appelle Gauthier. Ainsi, lorsqu'on voudra récupérer les sources du programme, on indiquera quelle est la branche voulue.
La branche a été créée par la séquence d'appels suivante :

```
cvs tag -b Gauthier (crée une nouvelle branche)
```

```
cvs update -r Gauthier (récupère les sources)
cvs commit
```

Le fait d'avoir récupéré les sources avec l'option -r rend la branche Gauthier sticky. Cela veut dire que la mise à jour des fichiers par l'action commit se fera par défaut dans cette branche.

3.4.3 Mise à jour des fichiers dans le repository

Lorsqu'on a fait une modification que l'on juge assez importante et fiable pour qu'on puisse la conserver, il est bon de placer les sources dans le repository et ainsi créer une nouvelle version des sources.
Cette action se fait par l'appel suivant :

```
cvs commit
```

Comme la branche a été rendue sticky, la mise à jour se fera dans la branche Gauthier. Lors de l'ajout des fichiers dans la base (repository), un éditeur de texte (celui de son choix) apparaît pour permettre d'écrire des commentaires sur les modifications ajoutées.

3.4.4 Ajout et Retrait de fichiers du Repository

Il est possible d'ajouter des fichiers à la base. Cette opération se fait simplement en se plaçant dans le répertoire qui contient le fichier et en tapant la commande suivante :

```
cvs add nom_du_fichier
```

De même, il est possible de supprimer un fichier de la base par la commande suivante :

```
cvs remove nom_du_fichier
```

3.5 Internet

L'internet est une véritable mine d'informations pour les développeurs. Ainsi, de nombreux sites nous proposent des pages d'aides et d'explications très détaillées sur les fonctions C, X/MOTIF.
D'autre part, il est souvent utile de pouvoir demander conseil aux autres lorsque la solution semble difficile à résoudre. Ainsi, les news groups consacrés à X/MOTIF et à la programmation sous UNIX permettent un contact rapide avec une population chaque jour croissante de développeurs du monde entier.
Je citerai notamment les news group suivant :

```
fr.comp.windows.x11
comp.windows.x.motif
```

Comme de nombreuses modifications ont été apportées à Xspect, j'ai créé une page consacrée aux nouveautés d'Xspect. Ainsi, chacun peut être au courant des dernières nouveautés en consultant la page : `http://www.ircam.fr/equipes/analyse-synthese/lebrun/work/work_index.html`

Deuxième partie

Mon travail à l'IRCAM

Introduction

Xspect a été développé en C et utilise X/MOTIF comme couche graphique. Le source du programme comporte pas moins de 35000 lignes de code utilisant la plupart des fonctionnalités X. Mon premier travail a donc été de comprendre les mécanismes spécifiques de X/MOTIF tels que la gestion des resources et les actions/translations. Les deux premiers chapitres exposent donc ces spécificités et la manière dont elles sont utilisées dans Xspect.

Xspect est très gourmand en couleurs car il permet de nombreuses fonctionnalités de sélection. Aussi, lorsqu'une autre application, tournant en parallèle, nécessite beaucoup de couleurs, Xspect ne pouvaient afficher les signaux. Mon travail a donc été de créer une palette de couleurs spécifique à Xspect. C'est l'objet du troisième chapitre.

Une fonctionnalité primordiale de Xspect est de pouvoir jouer les signaux visualisés. Le quatrième chapitre explique les diverses améliorations apportées à Xspect à ce sujet.

Le cinquième chapitre traite de la fusion d'un éditeur de fonctions breakpoints (écrit en C avec librairie X) dans Xspect. Un exemple d'application étant de pouvoir éditer des filtres et de les appliquer directement sur un spectre.

Le dernier chapitre traite des diverses autres améliorations apportées au programme : protection des analyses, taille des fenêtres d'analyse, amélioration de la gestion de la mémoire et des resources, etc.

Chapitre 1

Utilisations des ressources dans X/MOTIF Leurs applications dans Xspect

Une particularité des programmes X/MOTIF est qu'ils sont entièrement paramétrables par l'utilisateur grâce au mécanisme des ressources. Ainsi, l'utilisateur peut créer un fichier de ressources dans lequel il indiquera la couleur de ses barres de menus par exemple.
Pour le programmeur, c'est un moyen très simple pour adapter un programme dans une langue différente, en définissant par exemple les libellés des menus dans un fichier de ressources. Il n'aura ensuite qu'à modifier ce fichier pour changer la langue du programme, sans devoir le recompiler.
Ce chapitre nous décrit succintement le mécanisme de la gestion des ressources sous X/MOTIF, et comment il a été utilisé dans Xspect.

1.1 Généralités

Les widgets et les applications peuvent déclarer toutes leurs variables comme ressources. Dans la pratique, seules celles ayant besoin de valeurs entrées par l'utilisateur seront définies en ressources (comme les configurations de fenêtres par exemple).
La bibliothèque Xt permet de définir les ressources en utilisant une structure `XtResource` déclarée comme suit:

```
typedef struct {
  String resource_name; /* specifiee par le symbole XmN */
  String resource_class; /* specifiee par le symbole XmC */
  String resource_type; /* type de la variable actuelle */
  Cardinal resource_size; /* taille determinee par sizeof() */
  Cardinal resource_offset; /* determinee par XtOffsetOf() */
  String default_type; /* valeur immediate ou fonction */
  XtPointer default_address; /* adr. de la valeur par defaut */
} XtResource, *XtResourceList;
```

Pour utiliser des ressources dans des widgets existants, il suffit d'écrire un fichier de ressources.

Normalement, ce fichier se trouve dans le répertoire */usr/lib/X11/app-defaults* et a pour nom le même que l'argument `classname` de `XtVaAppInitialize()`.
La gestion des ressources s'effectuent en plusieurs étapes :

1. Au démarrage de l'application, par un appel `XtAppInitialize`, la bibliothèque Xt lit les fichiers de ressources, que ce soit .Xdefaults ou un fichier utilisateur tel que `xspect.ad` (en utilisant le programme *xrdb*). Puis, Xt les fusionne dans une seule base interne au programme qui sera utilisée à chaque création de widget.
2. Si aucun fichier de ressources n'est utilisé (c.-à-d. qu'on ne s'est pas servi de *xrdb* avant de lancer le programme), alors la base de ressources est remplie par les ressources codées en "dur" : c'est le champ *fallback_resources* de la fonction XtApInitialize (pointeur sur un tableau XtResourceList).
3. Quand on crée un widget, l'appel à `XtVaCreateManagedWidget()` lit la base de ressources et affecte automatiquement les ressources des widgets aux valeurs contenues dans la base.

Note:

- Si la valeur d'une resource est codée en "dur" par `XtVaManagedWidget` ou `XtVaSetValues`, cette valeur écrase celle lue dans la base.
- Pour récupérer les valeurs des ressources de l'application, on doit faire un appel à `XtGetApplicationResources`.
- Si la ligne de commande comporte des ressources, elles écrasent celles spécifiées par la base. Par exemple :

  ```
  xspect +askBeforeExit
  ```

1.2 Syntaxe

- Chaque entrée de la base est un couple spécification / valeur.
  `spécification = nom de l'application + '.' + Nom de la resource`
  `valeur = ':' + chaîne de caractères`
  Exemple : `xterm.scrollBar:on`
- Une * peut être utilisée comme "joker" à la place du nom de l'application (ex.:`*scrollBar`). On affecte alors la valeur à toutes les ressources ayant la propriété (ici `scrollBar`) dans leur base.
- Une ligne commentaire comence par !.

1.3 Utilisation des ressources dans Xspect

1.3.1 Le fichier de resource xspect.ad

L'utilisateur peut (re-)définir l'environnement de **Xspect** en modifiant le fichier de ressources `xspect.ad`. Le fichier original `xspect.ad` se trouve dans le répertoire

/u/formes/src/xspect/Xspect/src/app-default. Il comporte les ressources de **Xspect** que l'on peut personaliser telles que les couleurs, mais aussi les racourcis claviers.

IMPORTANT: il est indispensable de copier ce fichier dans son compte avant de le modifier.

Pour charger le fichier de ressources, on utilise le programme *xrdb*:

```
xrdb -merge xspect.ad
```

1.3.2 Le fichier include resource.h

Le fichier *include* **resource.h** comporte les ressources par défaut de Xspect. Ces ressources peuvent être modifiées en les initialisant dans un fichier de ressources (tels que `xspect.ad` ou `.Xdefaults`).
Par exemple :

```
xspect.margintop:10 xspect.askBeforeExit:false
```

Voici la liste des ressources définies dans resource.h :

```
askBeforeExit
audioDisplay (Alpha seulement)
autoPlaceWindow
buttonBarSize
createViewWarning
firstViewCreated
fileFormat
frameActiveBkgColor
frameActiveShadowType
frameBkgColor
frameShadowType
idActiveBkgColor
idBkgColor
idLeftPosition
marginBottom
marginLeft
marginRight
marginTop
maxSecScreen
maxSecVerify
mode
nColors
openFilesIn
rawWarning
selectionQuestion
sfDir
signalFullPage
splitPercent
watchbufferLimits
zoomMode
privateColormap
menuBarColor
buttonColor
nbSamplesBeforeWarning
```

1.3.3 Initialisation des ressources

L'initialisation des ressources dans le programme se fait par l'appel de la fonction `AppInitialize` (qui se trouve dans le module **resources.c**). Cette fonction comporte l'appel suivant :

```
widget=XtVaAppInitialize(appContext,''XXspect'', optionDesc,
       XtNumber(optionDesc), pargc, pargv, resourcesString,NULL);
```

La variables `resourcesString` passée en argument représente donc la liste des ressources par défaut de l'application. Cette variable a été initialisée dans **resources.h**.

1.3.4 Ajout d'une option sous Xspect

Cette partie s'adresse essentiellement au programmeur qui désire ajouter une option au programme.
Les options sont les ressources propres à Xspect et trouvent leur valeur par défaut dans le fichier `resource.h`.
Une option doit donc pouvoir être spécifiée dès le lancement du programme mais peut aussi être modifiée en cours d'exécution par l'intermédiaire de la boîte de contexte d'options.
L'ajout d'une option dans Xspect se fait en plusieurs étapes. Nous prendrons exemple sur l'option privateColormap qui permet de passer en palette de couleurs privées :

1. Modification de la structure Options (fichier typedefs.h). Cette structure regroupe toutes les options propres à Xspect. Ceci permet de les centraliser et de pouvoir les manipuler rapidement si l'utilisateur les modifie par l'intermédiaire du contexte d'Options :

   ```
   typedef struct _Options { ....  Boolean privateColormap; } Options;
   ```

2. Cette option aura une valeur par défaut qu'il faut définir comme constante dans le fichier defines.h :

   ```
   #define PRIVATE_COLORMAP False
   ```

3. Il faut ensuite modifier le fichier resources.h. Nous avons besoin de deux constantes XtNprivateColormap et XtCprivateColormap qui seront nécessaires dans le tableau resources. Ces constantes permettent de donner des noms aux ressources :

   ```
   #define XtNprivateColormap "privateColormap"
   #define XtCprivateColormap "PrivateColormap"
   ```

4. Dans le même fichier, il faut modifier le tableau resources en ajoutant les caractéristiques de la nouvelle resource : noms, type, taille, et valeur par défaut.

   ```
   static XtResource resources[] = {
     ...
     { XtNprivateColormap, XtCprivateColormap,
       XtRBoolean, sizeof(Boolean),
       XtOffset(OptionsPtr, privateColormap), XtRImmediate,
       (XtPointer) PRIVATE_COLORMAP
     },
   };
   ```

5. Toujours dans resources.h, il faut modifier le tableau des options optionDesc qui permet de spécifier ce qui devra être tapé pour lancer une option au démarrage de Xspect.

```
static XrmOptionDescRec optionDesc[] = {
  ....,
  {"-privateColormap", XtNprivateColormap, XrmoptionNoArg, "false"},
  {"+privateColormap", XtNprivateColormap, XrmoptionNoArg, "true"},
};
```

6. En lançant Xspect avec l'options -h, on obtient la liste de toutes les options avec leur descriptifs. Il faut donc rajouter cette nouvelle option dans cette liste, ce qui se fait dans le fichier resource.c dans la fonction usage.

```
fprintf(stderr, " -/+privateColormap Launch with the private
colormap.\n");
```

7. Pour terminer, il faut rajouter cette option dans la boîte de contexte d'options, en rajoutant un champ context.

Chapitre 2

Actions et Translations sous Xspect

Pour traiter des actions liées aux widgets, la librairie Xt utilise le mécanisme des *callbacks*. Ainsi, lorsque l'utilisateur clique sur un bouton, il suffit de lier l'événement à la fonction correspondante voulue.
Or, il se peut que l'utilisateur veuille implémenter une action qui n'est pas prévue par le widget. A ce moment là, on utilise le mécanisme des actions/translations.

2.1 Généralités

Lorsqu'aucune classe de widget n'a les *callbacks* nécessaires ou les caractéristiques voulues, on utilise la technique des Actions / Translations. Cette méthode est en effet plus simple que de réécrire un widget et effectue la même chose.
Ainsi, on utilise les actions pour ajouter des comportements à un *Primitive* ou *Core* widget. (On peut également les utiliser dans le code d'un widget pour implémenter une caractéristique du widget).

2.1.1 Actions

Les actions et les *callbacks* diffèrent par la méthode d'implémentation des fonctions:

- Pour les **callbacks**, le déclencheur est défini dans le widget (qui peut ou non être lié à un événement). Quand le déclencheur est activé, les routines sont appelées par `XtCallCallbacks()` ou `XtCallCallbackList()`.
- En revanche, les **Actions** sont appelées directement par le mécanisme de translation de Xt, comme le résultat d'une combinaison d'événements.

Pour déclarer une action, il faut d'abord déclarer une **Table des Actions** qui organise les chaînes de noms d'actions avec des pointeurs sur fonctions. La table est déclarée par `XtAppAddActions()`. On crée ensuite une table de translations stockée généralement dans le fichier de resources de l'application (on peut également la coder en "dur").
Le schéma suivant montre une table de translation et une table d'action, et indique comment chacune est utilisée lorsqu'un événement est reçu pour activer la fonction correspondante.

```
Key Event (Return key pressed)
```

Translation Table

```
static char default Translations[]=
        ''<Key>Return:Activate()'';
```

Action Table

```
static XtActionsRec actions[]={
{''Activate'',Activate}
};
```

Action Function

```
Activate(w,event,params,num_params)
widget w;
XEvent *event;
String *params;
Cardinal *num_params
{
/*action code */
}
```

Translation Table : Traduit un événement en un nom d'action. Les translations se trouvent donc dans le **fichier de resources.**

Action Table : Traduit le nom d'action en un pointeur sur fonction. Elle est donc spécifiée dans le **code**.

Schéma type de la création de la table des actions :

```
main(...)
{
        /* autres declarations */
        ...
        static XtActionsRec actions[]={
               {``RedrawSmallPicture'',RedrawSmallPicture},
        }
        ...
        /* XtAppInitialize */
        ...
        XtAppAddActions(app_context,actions,XtNumber(actions));
        ...
        /* XtMainLoop */
}
```

Les actions ajoutées par `XtAppAddActions` sont globales au contexte de l'application, et donc utilisables par n'importe quelle instance de widget de l'application.
Typiquement, une action est juste un pointeur sur une fonction à 4 arguments :

1. un widget.

2. un événement.
3. une chaîne contenant tous les arguments spécifiés pour l'action dans la table de translation.
4. le nombre d'arguments de la chaîne.

2.1.2 Translations

La table des translations permet de déterminer à quels widgets les actions sont destinées et lesquelles vont les déclencher.
Bien sur, la librairie Xt comporte les fonctions pour charger cette table : ce sont `XtParseTranslationTable`, `XtAugmentTranslation`, et `XtOverrideTranslation`.
Fragment de code montrant la déclaration des tranlations :

```
static char defaultTranslations[]=
       ``<Btn1Down> : set()\n\
         <Btn2Down>, <Btn2Up> : quit()'';

XtTranslations myTranslations;
...

myTranslation = XtParseTranslationTable(defaultTranslation);
XtOverrideTranslations(farewell,myTranslations);
```

On remarquera que la table des translations (`defaultTranslations`) est définie en tant que chaîne de caractères et peut donc se trouver dans un fichier de resources. L'utilisation du fichier de resources est avantageuse au départ car elle permet de modifier cette table sans devoir recompiler tout le programme.
En pratique, certaines parties de la table des translations seront codées en "dur" dans le programme pour définir certains comportements qui ne seront pas modifiables par l'utilisateur. En revanche, d'autres parties de la table des translations seront insérées dans un fichier de resources pour permettre à l'utilisateur de personaliser les comportements du logiciel à sa manière.

Voici un exemple montrant une partie du fichier de resources d'une application où l'on définit une table de translations :

```
*XmPrimitive.baseTranslations:\
    <Expose>:  RedrawSmallPicture()
```

On notera que, puisque les fichiers de resources sont des chaînes de caractères, nous n'avons pas besoin de guillemets. Néanmoins, pour les tables de translations comportants plusieurs lignes, nous devons terminer chaque ligne par `\n\`.
La resource affectée ici est `baseTranslations` (nouveau dans R5). Cette resource a le même nom pour tous les widgets. Comme le fichier de resources est constitué exclusivement de chaînes de caractères, Xt affecte donc la table des translations avec les données du fichier de resources en passant par un convertisseur de resources. Ce convertisseur appelle la fonction `XtParseTranslationTable()` pour nous quand nous spécifions les translations dans un fichier de resources.

2.2 Utilisation dans Xspect

Xspect utilise fortement les actions / translations.

Tables de translations définies dans `xspect.h`:

translationsWindow	translationsDrawNoView	translationsDrawCallUdi
translationsPopupPage	translationsMixMode	translationsSynchro
translationsPopupZoom	translationsDrawSelMode	translationsShortCutSynchro
translationsIdw	translationsDrawSigMode	translationsTranslationOutput
translationsIdwChgActView	translationsDrawEditMode	
translationsMoveButton	translationsDrawVisSel	

On trouve la table des actions dans `resources.h`:

```
static XtActionsRec actions[] =
{
  {"Motion",            XsAcMotion},
  {"Up",                XsAcUp},
  {"ActivateFrame",     XsAcActivateFrame},
  {"ActivateWindow",    XsAcActivateWindow},
  {"DeActivateWindow",  XsAcDeActivateWindow},
  {"ActivateSelection", XsAcActivateSelection},
  {"CreateSelectionX",  XsAcCreateSelectionX},
  {"CreateSelectionXY", XsAcCreateSelectionXY},
  {"CreateSelectionY",  XsAcCreateSelectionY},
  {"IdwChgActView",     XsAcIdwChgActView},
  {"PagePopupMenu",     XsAcPagePopupMenu},
  {"ZoomPopupMenu",     XsAcZoomPopupMenu},
  {"IdwPopupMenu",      XsAcIdwPopupMenu},
  {"ModifySelection",   XsAcModifySelection},
  {"ModifySelectionXY", XsAcModifySelectionXY},
  {"MoveButtonWidget",  XsAcMoveButtonWidget},
  {"OkCallUdi",         XsAcOkCallUdi},
  {"OkVisitSelection",  XsAcOkVisitSelection},
  {"OkSynchro",         XsAcOkSynchro},
  {"OkOutput",          XsAcOkOutput},
  {"SelectAll",         XsAcSelectAll},
  {"ShiftSelection",    XsAcShiftSelection},
  {"ShiftSelectionXY",  XsAcShiftSelectionXY},
  {"ShiftSignalX",      XsAcShiftSignalX},
  {"ShiftSignalXY",     XsAcShiftSignalXY},
  {"ShiftSignalY",      XsAcShiftSignalY},
  {"OkShortCutSynchro", XsAcOkShortCutSynchro},
  {"OkEditAltBtn1",     XsAcOkEditAltBtn1},
  {"OkEditAltBtn2",     XsAcOkEditAltBtn2},
  {"OkEditAltBtn3",     XsAcOkEditAltBtn3},
  {"OkEditMotion",      XsAcOkEditMotion},
  {"OkEditKey",         XsAcOkEditKey},
  {"OkEditAltCtrlBtn1", XsAcOkEditAltCtrlBtn1},
  {"OkEditAltShiftBtn3", XsAcOkEditAltShiftBtn3},
  {"OkEditAltCtrlBtn3", XsAcOkEditAltCtrlBtn3},
};
```

Les fonctions indiquées ci-dessus se trouvent se trouvent dans `Xa.c`.

Chapitre 3

Manipulation de Palettes de Couleurs sous X/MOTIF

Lorsqu'une application a besoin de couleurs pour pouvoir fonctionner, elle crée ses propres couleurs et les alloue dans la palette de couleurs par défaut.
Or, il se peut que cette palette soit pleine. Par exemple, lorsqu'une autre application tourne en même temps et qui a déjà allouée la palette avec ses couleurs, il ne reste plus de place à notre application pour pouvoir allouer ses couleurs.
La solution est donc de pouvoir créer une palette de couleurs propre à l'application et invisible aux autres. Aprés un bref descriptif des capacités de Xlib à gérer les palettes de couleurs, un exemple sera présenté.

3.1 Les palettes de couleurs sous Xlib

Une palette de couleurs physique est un registre dans lequel l'écran lit les intensités RVB[1] qui vont générer les couleurs à lécran. La plupart des stations n'ont qu'une palette de couleurs physique. Dans ce cas, toutes les fenêtres de l'écran sont interprétées en utilisant la même palette de couleurs. Certaines stations ont plusieurs palettes physiques, dans ce cas plusieurs fenêtres peuvent avoir leur palette indépendente.

Si la palette de couleurs physique ne peut être modifiée, elle est dite *immuable*. Les systèmes monochromes ont une palette de couleurs immuable. Dans ce cas, aucune application cliente ne peut allouer de couleurs privées et toutes les valeurs RVB sont prédéfinies. Dans ces systèmes, `XCreateColormap` fonctionnera, mais il donnera seulement une autre copie de la palette par défaut.

Dans la plupart des stations de travail, on peut écrire de nouvelles valeurs dans la palette de couleurs physique. Ces palettes sont dites *modifiables*.

X gère de multiples palettes de couleurs en gardant des *palettes de couleurs virtuelles* en mémoire et en les installant sur instruction du gestionnaire de fenêtres (*window manager*). L'*installation* d'une palette de couleurs est l'opération effectuant le passage d'une palette virtuelle dans la palette physique. Seules les palettes installées sont utilisées

1. RVB = Rouge, Vert, Bleu

pour déterminer les couleurs apparaissant à l'écran. Quand il n'y a qu'une seule palette physique et qu'une palette virtuelle est installée, la palette virtuelle qui avait été installée précédemment devient *désinstallée*.

Sur la plupart des stations de travail couleurs, il est assez fréquent que le client alloue toutes les couleurs de la palette. Les palettes virtuelles sont une solution à ce problême. Quand un client ne peut pas allouer les couleurs dont il a besoin dans la palette installée, il peut créer une nouvelle palette virtuelle. Le *window manager* installera alors cette palette virtuelle lorsque l'application est utilisée.

Quand une palette de couleurs virtuelle est installée et qu'il n'y a qu'une seule palette de couleurs physique, tous les clients qui utilisaient l'ancienne palette seront affichée en fausses couleurs puisque les valeurs des pixels de leurs fenêtres seront interprétées d'après la nouvelle palette.

Quand une application crée une palette virtuelle, elle doit modifier l'attribut `colormap window` de sa fenêtre principale (`topLevel`) pour que le *window manager* puisse trouver quelle palette il doit installer. Par défaut cet attribut indique la palette par défaut. Si ses sous-fenêtres utilisent différentes palettes de couleurs, une propriété peut être modifiée pour l'indiquer au *window manager*.

3.2 Fonctions pour la manipulation de palettes de couleurs

Les fonctions suivantes peuvent être utilisées par les applications ayant besoin de gérer leurs propres palettes de couleurs : `XCreateColormap()`, `XFreeColormap()`, `XListInstalledColormaps()`, `XCopyColormapAndFree()`, `XSetWindowColormap()`.

Les fonctions `XInstallColormap()` et `XUninstallColormap()` sont utilisées par le gestionnaire de fenêtres pour installer et désinstaller les palettes de couleurs.

3.2.1 XCreateColormap()

Description

Cette fonction crée une palette de couleurs virtuelle, soit avec aucune entrée alouée, soit avec toutes ses entrées allouées en lecture/écriture, qui correspondent au *visual* passée en paramêtre. Si aucune entrée n'est allouée, elle peuvent être soit allouées en lecture/écriture, soit en lecture seule. Si toutes les entrées sont allouées en lecture/écriture, la palette est totalement privée et a juste besoin que ses couleurs soient stockées avec `XStorecolors()`.

Arguments

```
Colormap XCreateColormap(display, w, visual, alloc)
        Display *display;
        Window w;
        Visual *visual;
        int alloc;
```

display : Spécifie la connection au Serveur X.

w : Spécifie la fenêtre de l'écran sur laquelle on veut créer la palette.

visual : Spécifie un type de support visuel supporté par l'écran. Si le type de visuel n'est pas l'un de ceux supporté par l'écran, alors la fonction renvoie une erreur *BadMatch*.

alloc : Spécifie les entrées de la palette à allouer. Ce paramètre peut être *AllocNone* ou *AllocAll*.

3.2.2 XFreeColormap()

Description

Cette fonction désinstalle la palette virtuelle et libère les resources associées à la palette. Elle envoie un événement `ColormapNotify` à toutes les fenêtres qui utilisaient la palette de couleurs.

Arguments

```
XFreeColormap(display, colormap)
        Display *display;
        Colormap colormap
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à libérer.

3.2.3 XListInstalledColormaps()

Description

Cette fonction liste les palettes actuellement installées.

Arguments

```
Colormap *XListInstalledColormaps(display, w, num_return)
        Display *display;
        Window w;
        int *num_return;
```

display : Spécifie la connection au Serveur X.

w : Spécifie la fenêtre qui détermine l'écran.

num_return : Spécifie le nombre de palettes actuellement installées.

3.2.4 XCopyColormapAndFree()

Description

Cette fonction copie toutes les entrées de la palette existante dans une nouvelle palette et libère ces entrées de l'ancienne palette. Elle est utilisée quand l'allocation de couleurs échoue et que certaines couleurs ont déjà été allouées. Pour les applications ayant un besoin important de couleurs qu'elles ne peuvent assouvir, elles peuvent appeler `XCopyColorMapAndFree`, modifier l'attribut *colormap window*, et continuer d'allouer les couleurs dans la nouvelle palette.

La fonction `XCopyColorMapAndFree` crée une palette du même type de *visual* et pour le même écran que celui de la palette passée en paramètre. Elle retourne l'identificateur de la nouvelle palette.

Arguments

```
Colormap XCopyColormapAndFree(display, colormap)
        Display *display;
        Colormap colormap
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à copier.

3.2.5 XSetWindowColormap()

Description

Cette fonction modifie l'attribut *colormap window* d'une fenêtre. La palette doit avoir le même de type de *visual* que la fenêtre sinon une erreur *BadMatch* est renvoyée.

Arguments

```
XSetWindowColormap(display, w, colormap)
        Display *display;
        Window w;
        Colormap colormap;
```

display : Spécifie la connection au Serveur X.

w : Spécifie la fenêtre.

colormap : Spécifie la palette à copier.

3.2.6 XInstallColormap()

Description

Cette fonction installe une palette de couleurs. Chaque fenêtre utiliant cet identificateur de palette dans son attribut *colormap* reçoit un événement *ColormapNotify*.

Arguments

```
XInstallColormap(display, colormap)
        Display *display;
        Colormap colormap;
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à installer.

3.2.7 XUninstallColormap()

Description

Cette fonction supprime la palette virutelle de l'ensemble des palettes installées. Sur les systêmes à une seule palette physique, la palette par défaut est réinstallée. Elle envoie un événement *ColormapNotify* aux fenêtres qui utilisaient la palette spécifiée.

Arguments

```
XUninstallColormap(display, colormap)
        Display *display;
        Colormap colormap;
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à désinstaller.

3.3 Application dans Xspect

La gestion des couleurs dans Xspect se trouve essentiellement dans le module `color.c`. On y définit les couleurs à utiliser dans le programme, et si nécessaire la palette privée de Xspect.
Une des premières tâches du programme principal (fonction `main` de `xspect.c`) est d'appeler la fonction `initColors(...)`. Cette fonction essaie d'allouer les couleurs dans la palette de couleurs par défaut. Si cette allocation échoue alors on crée une nouvelle palette virtuelle propre à Xspect grâce à la fonction `XCopyColormapAndFree()`.

Si une palette virtuelle est créée, l'utilisateur a alors la possibilité de l'activer ou de la désactiver dans le menu Context|Option, par le champ privateColormap.

L'allocation des couleurs s'effectue par l'appel à la fonction `getcolors()`. Celle-ci crée une table des couleurs et essaie de l'allouer par un appel à `XAllocColorCells()`.

On trouvera en annexe le code source d'un programme d'exemple reprenant en grande partie la méthode utilisé dans Xspect. Cette-à-dire qu'il essaie d'allouer les couleurs dans la palette courante, et en cas d'échec il crée une palette virtuelle. Pour tester, les résultats, nous affichons une mire des couleurs.

3.3.1 Remarques sur XAllocColorCells

Outre l'allocation de couleurs dans une palette, cette fonction nous permet de superposer des couches graphiques sans les détruire. Cette méthode permet d'améliorer les performances du programme en réduisant le nombre d'éléments graphiques complexes à réafficher. Elle est donc particulièrement utile lors des sélections.

L'astuce permettant de dessiner sans détruire ce qui a déjà été dessiné repose sur le fait que l'on peut dessiner sur un plan de la zone graphique. Cela est possible en utilisant le composant **plane_mask** du contexte graphique. Ce sont ces autres plans qui contiennent les informations de ce qui a déjà été dessiné. L'inconvénient de cette méthode est qu'elle nous oblige à allouer plus de couleurs que nous aurions normalement besoin.

```
XAllocColorCells(display, colormap, contig, plane_masks_return,
                 nplanes, pixels_return,npixels)
        Display *display;
        Colormap colormap;
        Bool contig;
        unsigned long plane_masks_return[];
        unsigned int nplanes;
        unsigned long pixels_return[];
        unsigned int npixels;
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à désinstaller.

contig : Valeur booléenne indiquant que les plans sont contiguës ou non..

plane_masks_return : Renvoie un tableau des masques des plans.

nplanes : Indique le nombre de plans dans le tableau précédent.

pixels_return : Renvoie un tableau de pixels.

npixels : Indique le nombre de pixels.

Chapitre 4

Correction des problèmes de son sous Xspect

4.1 Introduction

Xspect se doit de fonctionner sous SGI et DEC ALPHA. Comme ce sont deux architectures différentes, la gestion du son n'est pas la même. Ainsi, sous ALPHA la gestion du son est laissée à la charge d'un serveur audio dont on peut fixer la fréquence de sortie (actuellement à 44.1 kHz).
Cette différence de gestion du son implique donc deux écritures de la fonction `playFromSelection`. Pour indiquer au compilateur quelle fonction il faut compiler, le fichier play.c comporte deux parties :

```
#if defined(__sgi)
        :
        :
        Code pour SGI
        :
        :
#elif defined(__alpha)
        :
        :
        Code pour ALPHA
        :
        :
#endif
```

4.2 Machines DEC ALPHA : fusion des sources de playsf

De nombreux problèmes existaient lorsqu'on jouait des sons à partir de Xspect. Ainsi, sur les machines DEC ALPHA, les sons étaient joués à la vitesse d'échantillonnage du serveur audio : un son échantillonné à 11kHz joué sur un serveur à 44kHz était entendu 4 fois trop vite...

De plus, on désire pouvoir régler le volume directement à partir de Xspect. Cette fonctionalité avait été implantée pour les machines SGI, mais elle ne l'était pas pour les machines DEC ALPHA.

Rolf Woehrman a développé le programme playsf qui permet de jouer tous les sons quelque soient leur fréquence d'échantillonnage, leur type (float ou short), et indépendamment de l'architecture sur laquelle ils ont été conçus (en Big ou Little Endian).
Ce programme fonctionnant bien, notamment pour les machines DEC ALPHA, je m'en suis donc largement inspiré pour réécrire la fonction `playFromSelection` de Xspect.

4.2.1 Fonction playFromSelection (play.c)

Cette fonction permet de jouer un segment de son. Elle est appelée par la fonction *callback* `XsCbPlayFromSelection`, en réponse à un appel au menu Marks/Play Segment.

Prototype : `void playFromSelection(View *view,int iselFrom)`

view : spécifie la vue contenant le buffer et la sélection. Il nous permet de récupérer l'adresse du buffer et le numéro du canal à jouer. En effet, la fonction `playFromSelection` ne joue qu'un canal.

iselFrom : spécifie le numéro de la sélection.

Une fois la sélection récupérée, j'initialise le serveur audio par la fonction `InitAudio`.
Je calcule ensuite le début et la fin du segment (en position en secondes), puis j'initialise une mémoire tampon de lecture (bufferToPlay) positionnée au début du segment. Enfin, j'effectue une boucle de lecture des échantillons. La sortie audio se fait dans cette boucle par la fonction `writeAudio`.

C'est dans cette boucle que ce font également différentes conversions :

- Conversion en Short par la fonction `STconvertToShortBuffer`.
- Si nécessaire, une inversion de l'ordre des octets (conversion Big Endian en Little Endian), par la fonction `STswap16`.

Lorsque le buffer a été bien converti, on peut modifier le volume de sortie en multipliant chaque échantillon par la variable level. Cette variable récupère la valeur output Level de la fenêtre Context/Play.

4.2.2 Fonctions InitAudio et writeAudio

Ces fonctions proviennent de playsf.

`void Initaudio (Buffer *buffer)` : Cette fonction crée une connection vers le serveur indiqué par la variable AUDIOFILE en fonction du taux d'échantillonnage du buffer. C'est dans cette fonction qu'on va initialiser (s'il y a lieu) la variable audioConversion qui va permettre de convertir le buffer à jouer à la bonne fréquence d'échantillonnage.

`int writeAudio (char *buf, int samples, int sampleWidth)` : Cette fonction envoie un nombre de *samples* échantillons (contenus dans la variable buf). Si la fréquence de l'échantillon est différente de celle du serveur, il faut convertir cet échantillon de manière à ce qu'il soit audible. C'est la fonction `AConvert`. Puis, la variable buf est envoyée au serveur audio : fonction `AFPlaySamples`.

4.2.3 Nouvelles bibliothèques

Pour pouvoir intégrer les sources de playsf (fonctions InitAudio, writeAudio), de nouvelles bibliothèques ont dû être inclues au Makefile. Ainsi, ces fonctions utilisent la bibliothèque ST (pour *Sound Tools*), que Rolf Woehrman a développée. Cette bibliothèque se trouve actuellement dans `/u/sol/woehrman/src/STtools/libST`.

Pour utiliser les fonctions AConvertAllocInit et Aconvert, il faut inclure la bibliothèque aconvert. On peut la trouver dans `/u/formes/src/xspect/aconvert` avec ses sources.

Le fichier Makefile de Xspect a donc été modifié de manière à tenir compte de ces nouvelles bibliothèques :

```
LIB_STOOLS_DIR   =       /u/sol/woehrman/src/STtools/libST/\$(ARCH)
LIB_PRIV_DIR     =       /u/formes/src/xspect/aconvert

INCL_PRIV_DIR    =       /u/formes/src/xspect/aconvert
INCL_STOOLS_DIR  =       /u/sol/woehrman/src/STtools/include
```

> *ATTENTION : L'ordre des bibliothèques sur la ligne de compilation est important. Ainsi, il faut d'abord mettre les bibliothèques les plus dépendantes en premier lieu, suivies des plus indépendantes. Par exemple, la bibliothèques* aconvert *ne doit pas se trouver APRÈS les bibliothèques* AF *et* AFUtil *car elle les utilise. On indiquera donc sur la ligne de commande la liste des bibliothèques suivantes :* cc ... -laconvert -lAF -lAFUtil

4.3 Correction des bugs des machines SGI

La lecture des sons sur les machines SGI fonctionnait bien pour quasiment tous les sons sauf pour ceux échantillonné en flottants, et nécessitant une inversion des octets (Big $\rightleftarrows$ Little Endian).

Après étude du code existant, je me suis apperçu que la fonction `copyShortToPlay` effectue la conversion en short et l'inversion éventuelle de l'ordre des octets.
Il s'est avéré qu'elle était éronnée dans le cas des échantillons en float. En effet, 2 erreurs se trouvaient dans la partie DATA_TYPE_FLOAT :

1. La multiplication par 32768 était effectuée sur le pointeur et non sur la valeur pointée par celui-ci ! D'où (gros) problème...

2. La multiplication se faisait avant le swapFloat. Cette fonction est utilisée pour inverser l'ordre des octets, lorsque cela est nécessaire. Il faut donc effectuer la multiplication APRES l'inversion des octets.

J'ai donc aboutit au code suivant (où `pvalues` pointe sur le buffer à jouer) :

```
case DATA_TYPE_FLOAT:
   if (buffer->swap)
      for (i = 0; i < length && pvalues < pvaluesMax; i++)
      {
         float tmp = *((float *) pvalues);
         swapFloat(&tmp);
         tmp = tmp * 32768.0 * level;

         pvalues += step;
         *(ptr++) = (short) tmp;
      }
   else
      for (i = 0; i < length && pvalues < pvaluesMax; i++)
      {
         *(ptr++) = (short) (*((float *) pvalues) * 32768.0 * level);
         pvalues += step;
      }
break;
```

Chapitre 5

Fusion de XSedit à Xspect

5.1 Introduction

XSedit est un petit programme d'édition de graphique développé par Xavier Rodet au sein de l'IRCAM. Ce programme permet de créer, et d'éditer des courbes linéaires par morceaux (appelées fonctions *breakpoints*), ce qui est particulièrement utile lorsqu'on désire appliquer un gabarit de filtre à un spectre.

Ce programme, écrit en C, possède une interface graphique utilisant les fonctions X. Développé en parallèle à Xspect, ce programme comporte des structures semblables ou comparables à celles de Xspect, sa fusion dans Xspect étant déjà prévu lors de sa création.

5.2 Chargement d'un fichier de fonction breakpoint

Pour charger une fonction breakpoint, afin de l'éditer, il a été nécessaire d'ajouter un nouvel élément dans le menu *File*.
Comme les fichiers de fonction breakpoint ne possèdent pas d'entête, l'utilisateur doît indiquer la structure du fichier. Pour cela, une boîte de dialogue a été créée afin de déterminer (Cf. Fig 5.1):

- Le nombre de colonnes : par défaut 2 colonnes : temps / fréquence.
- Ordre des colonnes : X Y ou Y X.

Les fichiers de fonction breakpoints étant des fichiers ASCII, le problème de swap BIG $\leftrightarrow$ LITTLE Endian ne se pose pas. Les points sont stockés sous formes de liste chaînée. Il suffit d'appler la fonction `intBrkpfStruc` afin d'initialiser la liste.
Toutes les fonctions de gestion de la liste existent déjà dans les sources de XSedit et sont réutilisables telles quelles :

initBrkpfStruc : Initialise la structure de BrkpStuc. Celle-ci comporte tous les renseignements concernants la gestion de la liste et de son affichage.

lookForNearCel : Pointe sur la cellule la plus proche correspondant aux coordonnées passées en paramètre.

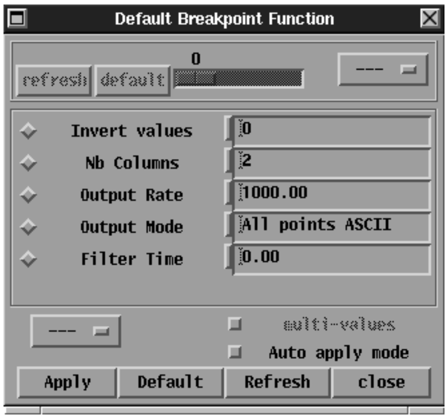

FIG. 5.1 – *Paramètres par défaut pour une fonction breakpoint*

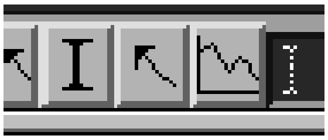

FIG. 5.2 – *Bouton de passage en mode édition*

removeCellInList : Supprime une cellule de la liste.

addNewPoint : Ajoute une cellule à la liste.

5.3 Création d'un mode Édition

Trois modes existent dans Xspect, permettant ou inhibant diverses opérations de sélection. J'ai utilisé le même principe afin de passer, lorsque la nature du vbuffer est une fonction breakpoint, dans le mode d'édition de la fonction (Cf. FIG 5.2). Cette fonction appelle la table des actions / translation `translationsEditMode`.

Cette appel à la table de tanslations inhibe les comportements de la souris existants et affectent les comportements suivants :

Action	Description
`Bouton 1`	Sélection en X
`Bouton 2`	Déplacement du signal en X
`Bouton 3`	Complétion de la sélection en X
`ALT+Bouton 1`	Créer un point
`ALT+Bouton 2`	Déplacer un point
`ALT+Bouton 3`	Marquer/Démarquer un point
`CTRL+Bouton 1`	Réactive la sélection
`CTRL+Bouton 2`	Déplaceement de la sélection en X
`SHIFT+Bouton 1`	Sélection en Y
`SHIFT+Bouton 2`	Déplacement de la sélection en Y
`SHIFT+Bouton 3`	Complétion en Y
`ALT+SHIFT+Bouton 2`	Démarquer les points dans un rectangle
`ALT+SHIFT+Bouton 3`	Marquer les points dans un rectangle
`ALT+CTRL+Bouton 1`	Supprimer un point
`ALT+CTRL+Bouton 3`	Tuer tous les points marqués

Comme pour toute table de translations, celle-ci a été déclarée dans les fichiers `xspect.h`, et `resources.h`. De même, les fonctions correspondants à la réponse à ces événements se trouvent dans le fichier `Xa.c`.

En réalité, cette table des translations reprend les actions de sélection et de déplacement de signaux déjà existant dans le mode Orienté signal (celui par défaut dans Xspect).
Il a donc suffit de recopier cette table en rajoutant les nouvelles actions d'édition : ajout et suppression de points, sélection.

5.4 Sauvegarde d'une fonction Breakpoint

Une fois la fonction éditée, il est nécessaire de pouvoir la sauvegarder. Plusieurs modes de sortie sont proposés dans le menu Context | Breakpoint function default format (Cf. Fig 5.1) :

SVP Filter Gabarit : Permet de sauvegarder les données de la fonction sous le format de SVP (Super Vocoder de Phase). Lors de la sauvegarde, le programme demande le Time Filter nécessaire à l'entête.

Sample ASCII : Le programme rééchantillonne le signal et sauvegarde le fichier dans un format ASCII. La fréquence de l'échantillonage de sortie est fixée dans cette fenêtre (champ Output Rate).

Sample Binary : Le programme rééchantillonne le signal et sauvegarde le fichier dans un format binaire.

All Points ASCII : Le programme sauvegarde tous les points de la fonction dans un fichier ASCII.

Selected Points ASCII : Le programme sauvegarde la liste des points marqués dans un fichier ASCII.

Lorsqu'on a déterminé le type de sortie désiré, la sauvegarde se lance par l'appel à la fonction `Save Breakpoint function` dans le menu File. Une boîte de dialogue s'ouvre pour demander le nom du fichier.

5.5 Création d'une nouvelle fonction breakpoint

Contrairement aux autres signaux qui doivent exister pour pouvoir être ouverts, Xspect doit maintenant avoir la possibilité de créer une nouvelle fonction Breakpoint.
Cela implique donc un nouveau mode d'ouverture : c'est l'item New Breakpoint function du menu File.
Ce mode permet de créer un nouveau *buffer* de type *Breakpoint function* et passe directement dans le mode d'édition.
Par défaut, la taille du buffer est [0 .. 10] en X et [0 .. 0,1] en Y. On peut facilement changer la taille en modifiant les valeurs du contexte de *display* (menu Context | Display).

5.6 Exemple d'utilisation : édition d'un gabarit

La figure 5.3 nous présente un exemple caractéristique d'utilisation de l'éditeur de fonctions *breakpoints* : le filtrage d'un signal sonore.
Dans la *frame* de gauche (en haut), nous pouvons voir le signal sonore ('Fa 3' de flûte).
Puis, nous avons calculé son spectre (*frame* de droite, en haut). Grâce à l'éditeur de fonctions breakpoints nous avons sélectionné un partiel (harmonique), en l'occurence il s'agit de l'harmonique fondamentale (F0), que nous avons sauvé sous forme de gabarit.
Nous avons ensuite appliqué une analyse SVP sur le signal en appliquant le gabarit. Nous obtenons alors une resynthèse du signal (*frame* de gauche en bas). En effectuant un spectre de ce signal (*frame* de droite, en bas), on observe bien que seul le premier partiel a conservé son intégrité.

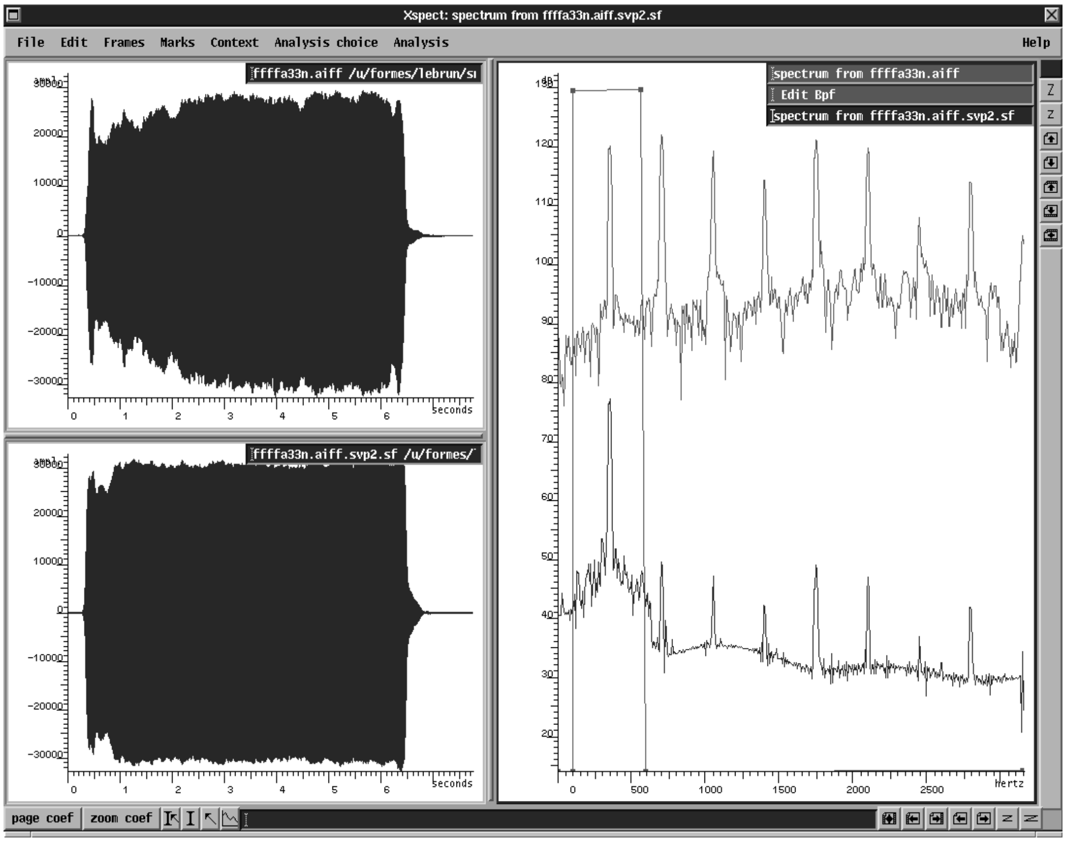

Fig. 5.3 – *Exemple d'utilisation de l'éditeur de fonctions breakpoints*

Chapitre 6

Adéquation IUP / Le stage

Ce stage de fin d'études de cinq mois est vraiment un élément important dans notre formation. En effet, la durée nous permet d'aborder des projets beaucoup plus important que lors des précédents stages.

Ainsi, ce stage m'a permis d'aborder un programme assez conséquent (plus de 35.000 lignes de code !) comportant la plupart des fonctionlités de la programmation sous UNIX, ainsi que celles du monde X / MOTIF. La formation que l'IUP m'apporté me semble bien adapté à la réalité d'un tel stage.

Ainsi, l'IUP a adopté le système UNIX, système de plus en plus utilisé et promu à un bel avenir. Utilisé également à l'IRCAM, je n'ai donc eu aucun problème d'adaptation au système. De plus, le cours de programmation C sous UNIX m'a été d'un grand secours et m'a permis une compréhension rapide du programme.

Je regrette néanmoins que l'IUP ne m'ait pas préparé à la programmation sous X/MOTIF, et notamment aux mécanismes particuliers de la gestion des resources et des tables d'actions / translations.
De plus, l'IUP ne nous prépare pas à la gestion de "gros" programmes qui demandent, par exemple, l'utilisation d'un fichier Makefile. Pourtant l'utilisation de ce fichier est indispensable pour faciliter la gestion des différents fichiers qui composent le programme. Il serait bon que l'IUP nous enseigne la création et l'utilisation de ce fichier, ce cours pourrait s'insérer dans celui d'UNIX de première année.
A l'IRCAM, le maniement d'une FFT est une opération aussi courante qu'une addition ! Aussi, le cours de traitement de signal m'a aidé dans la compréhension des opérations et traitements effectués sur les sons.

Enfin, l'IRCAM acceuille de nombreux compositeurs et scientifiques étrangers. La maîtrise de l'anglais a été un atout supplémentaire lors des multiples séminaires présentés. De plus, toutes les documentations existantes sur X/MOTIF sont en anglais. La spécialisation des cours d'anglais informatique est donc un avantage incontestable.

Conclusion

La première partie de ce rapport nous a permis de situer le contexte dans lequel s'est déroulé mon stage. La deuxième partie a été écrite afin de faciliter la prise en main du développement du logiciel par de futurs programmeurs.

Xspect est un outil qui est largement utilisé pour diverses applications de traitement du signal à l'IRCAM. En outre, il a été proposé dans différents endroits (laboratoire ou grandes entreprises) en version de démonstration. Ainsi, Citroën a récemment acquis une version SGI. Cet outil semble répondre aux exigences des utilisateurs avec efficacité. C'est un logiciel évolutif qui permet d'intégrer de multiples fonctionnalités. La façon dont il a été programmé permet, en effet, de le faire progresser dans divers domaines d'analyse, de représentations et de manipulations des sons. En continuel développememt, de nouvelles fonctionnalités sont d'ors et déjà prévues telles que l'affichage de sonagrammes.

Ce stage m'a donc permis tout d'abord de prendre en charge la responsabilité de la programmation d'un logiciel de taille relativement importante et utilisant des fonctionnalités du monde Xwindow qui m'étaient inconnues. Il a donc nécessité une phase d'apprentissage de ces mécanismes, ainsi que des capacités d'adaptation, d'écoute vis-à-vis des utilisateurs, et de rigueur pour pouvoir poursuivre le développement du logiciel en conservant l'optique prévue initialement.

Enfin ce stage m'a permis de pouvoir appliquer mes connaissances techniques dans institut dont la vocation est de promouvoir les liens entre la science et un domaine qui me passionne : la musique.

Annexes

Exemple de création de palettes de couleurs

```
/* INCLUDES */
/* -------- */

/* standard */
#include <stdio.h>
#include <stdlib.h>

/* XLib */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>

/* MOTIF */
#include <Xm/FileSB.h>
#include <Xm/DrawingA.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>

/* DEFINES */
/* ------- */
#define charset XmSTRING_DEFAULT_CHARSET
#define MAX_PLANE          2
#define MAX_CELL           ((MAX_COLOR) * (1 << (MAX_PLANE)))
#define MAX_COLOR          16
#define CANNOT_OVERLAY     0
#define CAN_OVERLAY        1
#define MAX_CHAR           256
#define UNDEFINED          -1
#define WIDTH     400
#define HEIGHT     300

/* TYPES */
/* ----- */
typedef char STRING[MAX_CHAR];

/* Constantes de couleurs */
/* ---------------------- */
static short valStd8[][3] =
{
  0xFF, 0xFF, 0xFF,
  0x00, 0xFF, 0xFF,
  0xFF, 0x00, 0xFF,
  0x00, 0x00, 0xFF,
  0xFF, 0xFF, 0x00,
  0x00, 0xFF, 0x00,
  0xFF, 0x00, 0x00,
  0x00, 0x00, 0x00
};
```

```
/* Variables globales */
/* ------------------ */
Display *display;
int screen_num;
Window window;
unsigned long planeMasks[MAX_PLANE];
unsigned long pixels[MAX_COLOR];
XColor exactDefs[MAX_CELL];

Colormap colormap, mycolormap;
int iwhite = UNDEFINED;
int iblack = UNDEFINED;
int ifg = UNDEFINED;
int ibg = UNDEFINED;
Pixmap pixmap;



void blkToZero(str)
  char *str;
{
  int i;
  char *ptr = str;

  for (i = 0; i < strlen(str); i++, ptr++)
    {
      if (*ptr == ' ')
*ptr = '0';
    }
}

void fin()
{
  /* Restore the old colormap */
  if (mycolormap!=colormap)
    {
      printf("Restore the old colormap\n");
      XFreeColormap(display,mycolormap);
    }

  exit();
}


/* Programme Principal */
/* ------------------- */
main(argc,argv)
int argc;
char *argv[];
{
  Widget topLevel,text_w, dialog,drawing_a, pb;
  XtAppContext app;
  XGCValues gcv;
  GC gc;

  extern void exit(), echo_file();
  void DessinePalette();
  void initColors();
  void drawing_area_callback();

  topLevel=XtVaAppInitialize(&app, "Demos", NULL, 0,
     &argc, argv, NULL,
     XmNwidth, WIDTH,
     XmNheight, HEIGHT,
     NULL);

  display = XtDisplay(topLevel);
```

```
  screen_num = DefaultScreen(display);
  window=XtWindow(topLevel);

  initColors(8);
  XtVaSetValues(topLevel,XmNcolormap,mycolormap,NULL);

  /* Create a drawing area widget */
  drawing_a=XtVaCreateWidget("drawing_a",
    xmDrawingAreaWidgetClass, topLevel,
    NULL);
  XtAddCallback(drawing_a,XmNexposeCallback,drawing_area_callback,NULL);

  gc = XCreateGC(XtDisplay(drawing_a),
  RootWindowOfScreen(XtScreen(drawing_a)), 0, NULL);
  XtVaSetValues(drawing_a,XmNuserData,gc,NULL);

  /* Create a pixmap the same size as the drawing area */
  pixmap=XCreatePixmap(XtDisplay(drawing_a),
       RootWindowOfScreen(XtScreen(drawing_a)),WIDTH,HEIGHT,
       DefaultDepthOfScreen(XtScreen(drawing_a)));


  /* Clear pixmap with white */
  XSetForeground(XtDisplay(drawing_a), gc, WhitePixelOfScreen(XtScreen(drawing_a)));
  XFillRectangle(XtDisplay(drawing_a), pixmap,gc,0,0,WIDTH,HEIGHT);


  DessinePalette(drawing_a);


  /* Add a pushbutton the user can use to close */
  pb = XtVaCreateManagedWidget("Close",
       xmPushButtonGadgetClass, drawing_a,
       NULL);

  /* if activated call the callback */
  XtAddCallback(pb,XmNactivateCallback, fin,NULL);

  /* create a file selection dialog */
  dialog=XmCreateFileSelectionDialog(topLevel,"filesb",NULL,0);
  XtAddCallback(dialog,XmNcancelCallback,exit,NULL);
  XtAddCallback(dialog,XmNokCallback,echo_file,NULL);

  XtManageChild(dialog);

  /* main loop */
  XtManageChild(drawing_a);
  XtRealizeWidget(topLevel);
  XtAppMainLoop(app);

}


/* Fonction Callback appelee lors de l'activation de la fenetre */
/* ou de sa re-exposition */
/* ---------------------------------------------------------- */
void drawing_area_callback(widget,data,cbs)
Widget widget;
XtPointer data;
XmDrawingAreaCallbackStruct *cbs;
{
  XEvent *event=cbs->event;
  Display *dpy=event->xany.display;

  if (cbs->reason == XmCR_EXPOSE || cbs->reason == XmCR_ACTIVATE)
    {
      GC gc;
```

```
      XtVaGetValues(widget,XmNuserData,&gc,NULL);
      XCopyArea(dpy,pixmap, event->xany.window,gc,0,0,WIDTH,HEIGHT,0,0);
    }
}


/* Write the name of the selected file on the screen */
/* ------------------------------------------------ */
void echo_file(fs,client_data,cbs)
Widget fs; /* file selection box */
XtPointer client_data;
XmFileSelectionBoxCallbackStruct *cbs;
{
  char *filename;

  if (!XmStringGetLtoR(cbs->value,charset,&filename))
    return;

  if (!*filename)
    {
      printf("No file selected.");
      XtFree(filename);
      return;
    }

  printf("Filename given : %s \n",filename);
  XtFree(filename);
}


/* Draw the colormap in a window */
/* ----------------------------- */
void DessinePalette(widget)
Widget widget;
{
  int i;
  int step = (4 * WIDTH / 5) / MAX_COLOR;
  GC gc;

  XtVaGetValues(widget,XmNuserData,&gc,NULL);

  for (i=0;i<MAX_COLOR;i++)
    {
      XSetForeground(XtDisplay(widget), gc, exactDefs[i].pixel);
      XFillRectangle(XtDisplay(widget),pixmap,gc,i*step,0,step,HEIGHT);
    }
}


/* colormap initialisation */
/* ----------------------- */
void initColors(ncolor)
int ncolor;
{
  XVisualInfo visualInfo;
  int defaultDepth=DefaultDepth(display,screen_num);

  int nplane, ncell;

  colormap = DefaultColormap(display,screen_num);
  mycolormap=colormap;

  nplane = 2;
  ncell = ncolor * (1 << nplane);

  if (defaultDepth == 1)
    {
```

```
      /* must be StaticGray, use black and white */
      printf("The screen can only have 2 colors.");

      ncolor = 0;
      nplane = 2;
      ncell = 2;
    }

  if (!XMatchVisualInfo(display, screen_num, defaultDepth, PseudoColor, &visualInfo))
    {
      if (!XMatchVisualInfo(display, screen_num, defaultDepth, DirectColor, &visualInfo))
{
  /* No PseudoColor or TrueColor visual available at defaultDepth.
   * Some applications might try for a GrayScale visual
   * here if they can use gray to advantage, before
   * giving up and using black and white.
   */
  printf("initColors: TODO HERE: try for a GrayScale... \n");
}
    }

  if ( getColors(ncolor, nplane, ncell, &planeMasks[0], &pixels[0], &exactDefs[0]) == CANNOT_OVERLAY )
      printf("We can't allocate enough colors for overlay.\n");
}

int getColors(maxColors, maxPlanes, maxCells, planeMasks, pixels, exactDefs)
     int maxColors;
     int maxPlanes;
     int maxCells;
     unsigned long planeMasks[MAX_PLANE];
     unsigned long pixels[MAX_COLOR];
     XColor exactDefs[MAX_CELL];
{
  STRING name[MAX_CELL];
  XColor *pExDef;
  Colormap defaultCmap = colormap;
  int nbCells = maxCells;
  int power2maxPlanes;
  int i, j, k;
  int class;

  void createPrivateMap();

  /* compute our private color map */
  createPrivateMap(&valStd8[0][0], name, maxColors, maxPlanes, maxCells, planeMasks, pixels, exactDefs);
  iwhite = 0;
  ibg = 1; /* cyan */
  iblack = 7;
  ifg = 3; /* blue */

  if (XAllocColorCells(display, defaultCmap, False, planeMasks, maxPlanes, pixels, maxColors) == 0)
    {
      fprintf(stderr,"Can't alloc enough colors in the current color map. Creation of a new one. \n");

      if ((defaultCmap=XCopyColormapAndFree(display,colormap)) == BadAlloc)
fprintf(stderr, "\nCan't Create new colormap\n");

      if (XAllocColorCells(display, defaultCmap, False, planeMasks, maxPlanes, pixels, maxColors) == 0)
{
  fprintf(stderr,"\nCan't allocate %d colors after created the new color map\n\
                        This is a big problem !\n",maxColors);
  exit(0);
}
    }

  /* allocated colorcells succesfully,
   * now set their colors
```

```
   */

  for (i = 0; i < maxCells; i++)
    {
      exactDefs[i].flags = DoRed | DoGreen | DoBlue;
      if (! XParseColor(display, defaultCmap, name[i],  &exactDefs[i]))
  fprintf(stderr,"basic: color name %s not in database", name[i]);

      printf("%s\n",name[i]);
    }

  /* set pixel value in struct to the allocated ones */
  for (i = 0; i < (1 << maxPlanes); i++)
    for (j = 0; j < maxColors; j++)
      {
unsigned long msk = 0;

for (k = 0; k < maxPlanes; k++)
  {
    int bool = ((1 << k) & i) >> k;
    msk |= bool * planeMasks[k];
  }

exactDefs[i * maxColors + j].pixel = pixels[j] | msk;
      }

  /* this sets the color of the read/write cells */
  XStoreColors(display, defaultCmap, exactDefs, nbCells);

  if (defaultCmap!=colormap)
    {
       mycolormap=defaultCmap;
       XInstallColormap(display,mycolormap);
    }

  if (nbCells >= 4)
       return (CAN_OVERLAY);
   else
       return (CANNOT_OVERLAY);
}


void createPrivateMap(val, name, maxColors, maxPlanes, maxCells, planeMasks, pixels, exactDefs)
     short val[][3];
     STRING name[MAX_CELL];
     int maxColors;
     int maxPlanes;
     int maxCells;
     unsigned long planeMasks[MAX_PLANE];
     unsigned long pixels[MAX_COLOR];
     XColor exactDefs[MAX_CELL];
{
  int i, j;
  int fr1, fg1, fb1;
  int fr2, fg2, fb2;
  int fr3, fg3, fb3;
  double coeff = 1.5;

  printf("createPrivateMap..\n");

  /* plan 1 */
  fr1 = 0;
  fg1 = 255;
  fb1 = 0;

  /* plan 2 */
  fr2 = 255;
```

```
  fg2 = 0;
  fb2 = 0;

  /* plan 1 & 2 */
  fr3 = 255;
  fg3 = 255;
  fb3 = 0;

  i = 0;
  for (j = 0; j < maxColors; j++)
      {
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", val[j][0],  val[j][1],  val[j][2]);
blkToZero(name[i * maxColors + j]);
      }

  i = 1;
  for (j = 0; j < maxColors; j++)
      {
double cr, cg, cb;

cr = (coeff * val[j][0] + fr1) / (coeff + 1.);
cg = (coeff * val[j][1] + fg1) / (coeff + 1.);
cb = (coeff * val[j][2] + fb1) / (coeff + 1.);

if (cr > 255.) cr = 255.;
if (cg > 255.) cg = 255.;
if (cb > 255.) cb = 255.;
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", (int) cr, (int) cg, (int) cb);
blkToZero(name[i * maxColors + j]);
      }

  i = 2;
  for (j = 0; j < maxColors; j++)
      {
double cr, cg, cb;

cr = (coeff * val[j][0] + fr2) / (coeff + 1.);
cg = (coeff * val[j][1] + fg2) / (coeff + 1.);
cb = (coeff * val[j][2] + fb2) / (coeff + 1.);

if (cr > 255.) cr = 255.;
if (cg > 255.) cg = 255.;
if (cb > 255.) cb = 255.;
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", (int) cr, (int) cg, (int) cb);
blkToZero(name[i * maxColors + j]);
      }

  i = 3;
  for (j = 0; j < maxColors; j++)
      {
double cr, cg, cb;

cr = (coeff * val[j][0] + fr3) / (coeff + 1.);
cg = (coeff * val[j][1] + fg3) / (coeff + 1.);
cb = (coeff * val[j][2] + fb3) / (coeff + 1.);

if (cr > 255.) cr = 255.;
if (cg > 255.) cg = 255.;
if (cb > 255.) cb = 255.;
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", (int) cr, (int) cg, (int) cb);
blkToZero(name[i * maxColors + j]);

      }
}
```