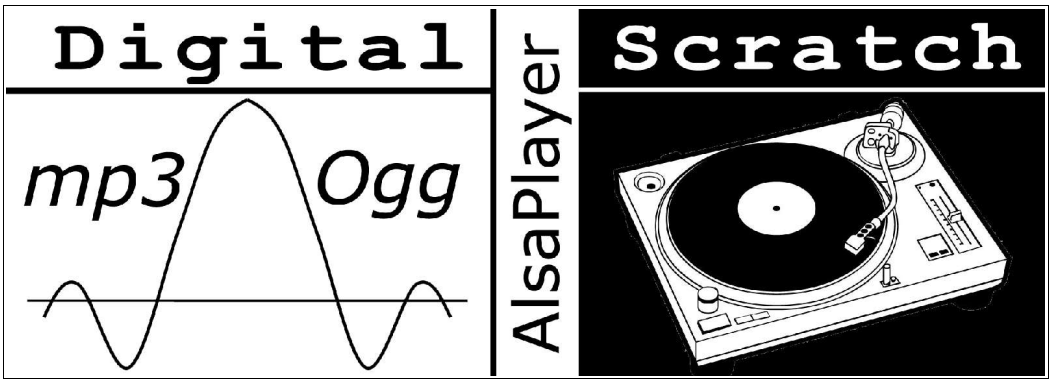
utbm
Université de Technologie de Belfort-Montbéliard

TX5x *Travaux de laboratoire*

Semestre *Printemps 2004*

DIGITAL SCRATCH

Une application de pilotage de lecture de fichier audio par platine vinyle



http://www.digital-scratch.org

Etudiant *Rosener Julien (julien.rosener@utbm.fr)*

Suiveur *Lacaille Nicolas (nicolas.lacaille@utbm.fr)*

Sommaire

1 Introduction

L'unité de valeur TX5x à pour trait de permettre aux étudiants de réaliser un projet d'assez grande envergure. De plus, elle permet de proposer ses propres sujets. C'est dans ce cadre que j'ai développé une seconde version de l'application *Digital-Scratch*, un projet qui me tient à cœur depuis un an. En effet, j'ai déjà réalisé une TX avec Cyril Coquilleau sur ce sujet. Cependant la première fois était plutôt une sorte de test, on devait pouvoir répondre à la question : « est ce que cela est possible de réaliser une application qui remplisse ces fonctionnalités ? ». Après beaucoup de travail et d'essais infructueux, il s'est avéré que l'idée était concrétisable. Ce rapport présente donc en détails l'élaboration de *Digital-Scratch* depuis la longue phase de conception jusqu'à la première version implémenté. Effectivement, cela n'est qu'une première version; mais le sujet m'est particulièrement cher et je continuerais à le développer d'une part pour terminer toutes les fonctionnalités de bases et aussi pour améliorer l'efficacité du système. Plus qu'un simple sujet d'informatique; ce projet est un exercice de développement complet d'un système mettant en oeuvre toutes les entités que l'on retrouve dans les projets conséquents.

Digital-Scratch est un logiciel appliqué à l'une des principales activités tournant autour des musiques électroniques : le mixage par platine vinyle.

Pour bien comprendre l'intérêt de l'idée, revenons sur ce qu'est le mixage. Dans sa version la plus primitive, c'est l'enchaînement de deux musiques (chacune délivrée par une platine vinyle). Ce qui est vraiment particulier à bon nombre de musique électronique, c'est le rythme binaire autour desquelles elles sont architecturées. Ainsi, pour que l'enchaînement des musiques soit le moins perceptible, on tente de modifier leur vitesse, pour que leurs battements soient semblables. L'enchaînement se fait ensuite par l'intermédiaire d'une table de mixage qui se charge de mixer les 2 musiques pendant un certains temps.

Le principe de *Digital-Scratch* est le suivant : pouvoir mixer de la musique que l'on ne possède pas en disque vinyle, tout en utilisant ses propres platines. Pourquoi garder ses platines alors que les technologies numériques permettent de réaliser cela avec des CDs ? Parce qu'on peut toucher les disques lors de leur lecture. Ainsi, on obtient une très grande précision de décalage et de vitesse dans la musique qui est jouée. Par extension, on peut également réaliser des « scratchs » qui sont des mouvements de va et vient rapide du disque.

La solution ici proposées est d'utiliser un ordinateur pour analyser les mouvements pratiqués par l'utilisateur sur la platine. Ce dernier pourra lire, de manière synchronisée avec l'utilisateur, un fichier numérique de musique (par abus de langage, on parlera souvent de « mp3 », l'un des formats les plus utilisés).

2 Analyse du domaine

Le domaine d'utilisation est celui du « djing » (le nom de l'activité qui consiste à mixer de la musique). Pour bien comprendre quel est la place de Digital-Scratch dans ce domaine, étudions le d'une part l'état existant de cette discipline et ensuite ce que peut apporter *Digital-Scratch*.

2.1 Etat existant

Voici un diagramme de classe très simplifié (les notions de méthodes et d'attributs ne nous intéressent pas pour le moment) servant à décrire l'état actuel du domaine.

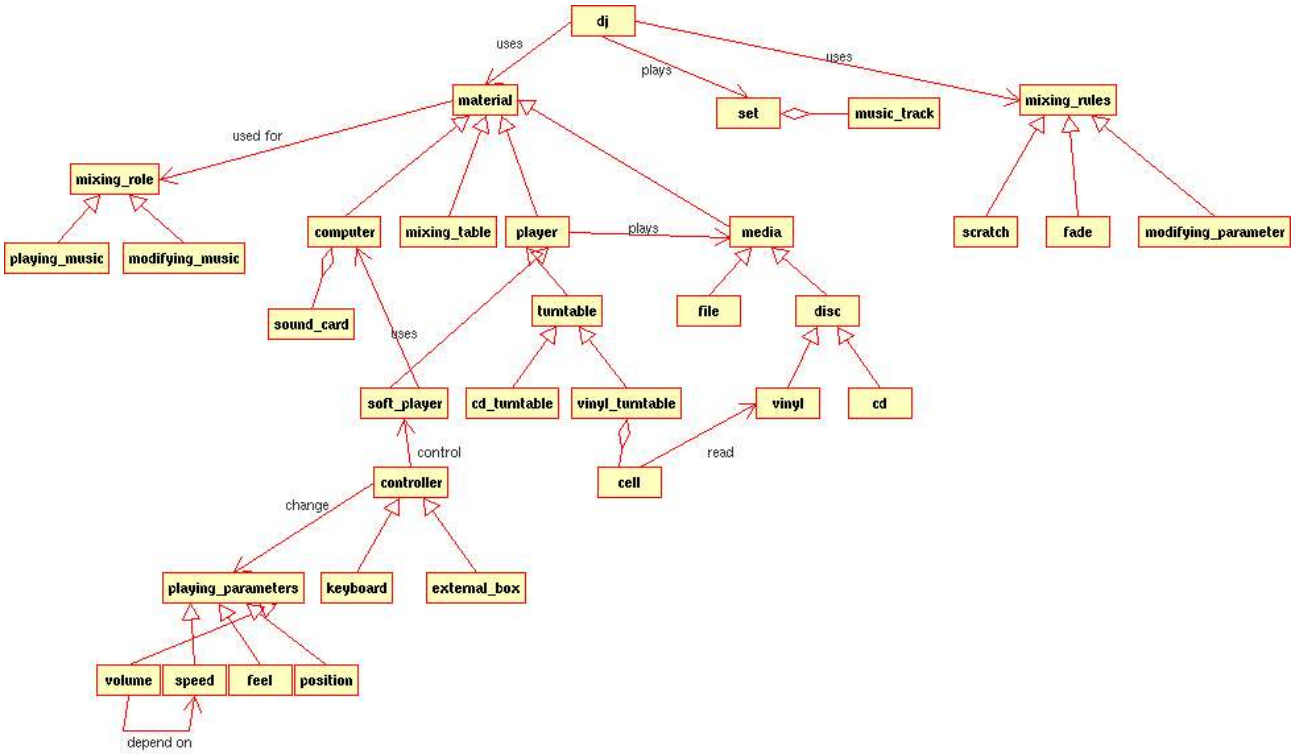


Illustration 1 - Diagramme de l'état existant du domaine du mixage

Tout part du DJ (Disc Jockey, littéralement « qui passe des disques » dont le but est de faire ce qu'on appel un « set », c'est une compilation de musique mixées. Cet enchaînement se fait grâce à des techniques de mixage; le schéma n'est pas exhaustif mais fourni comme exemple le « fondu-enchainé », la modification de paramètres (on entend par la de basse, médium, aigu par exemple) ou encore la technique du scratch (un va et vient très rapide de la musique sur un court instant).

La partie qui nous intéresse le plus est la partie traitant du materiel sont le DJ se sert justement pour mixer, c'est-à-dire jouer des disques et modifier leur sonorité en les enchaînant. Le matériel en lui même est le suivant : un lecteur de musique, et surtout plusieurs, dont on mélange les sons via une table de mixage. Justement, ces lecteurs sont de plusieurs types : il y a les platines vinyle pour les disques classiques, les platines CD pour le format numérique et enfin les lecteurs logiciels qui permettent de lire tout types de fichiers audio numériques. A part pour les appareils spécialisés (qui

ne sont pas spécialement dédié au mixage), seul l'ordinateur peut remplir le rôle de lecteur logiciel si on le dote du logiciel adéquat. Pour l'activité de mixage, il s'agit de modifier certain paramètres de lectures, comme la vitesse, le sens ou encore la position dans la musique. Pour cela on doit pouvoir contrôler le lecteur logiciel : le contrôleur le plus classique est tout simplement le clavier, d'ailleurs bon nombres de logiciels de mixage simule deux platines à l'écran que l'on pilote au clavier. Récemment sont apparus des contrôleurs externes, ce sont des boîtiers qui permettent de piloter un lecteur, on y trouve une touche, play, pause, modification de vitesse, etc...

Ce paragraphe a donc présenté l'activité du mixage, en allant de la techniques en elle-même jusqu'au matériel utilisé.

2.2 La solution *Digital-Scratch*

Le diagramme précédent à présenté le domaine d'étude, cependant, on peut soulever un problème. Il y a une distinction du type de lecteur suivant le format physique du support contenant la musique. Cela oblige a changer de type de lecteur à l'intérieur d'un set si on possède certaines musiques sur CD, vinyle ou encore mp3. Ce qu'on peut faire, c'est tout transformer en mp3 pour ne plus utiliser qu'un seul lecteur, le lecteur logiciel. Pourtant, cela n'intéresse personne, pourquoi ? En fait tout les Djs savent modifier les paramètres de lectures avec des platines vinyles (c'est presque toujours avec ce support que l'on apprend), faire de même avec un clavier d'ordinateur ou un contrôleur externe, est bien moins aisé et précis. Le problème est posé : concilier le contrôle des paramètres de lectures avec une platine vinyle, avec les fichiers audio numériques que l'on ne peut lire que grâce à un lecteur logiciel.

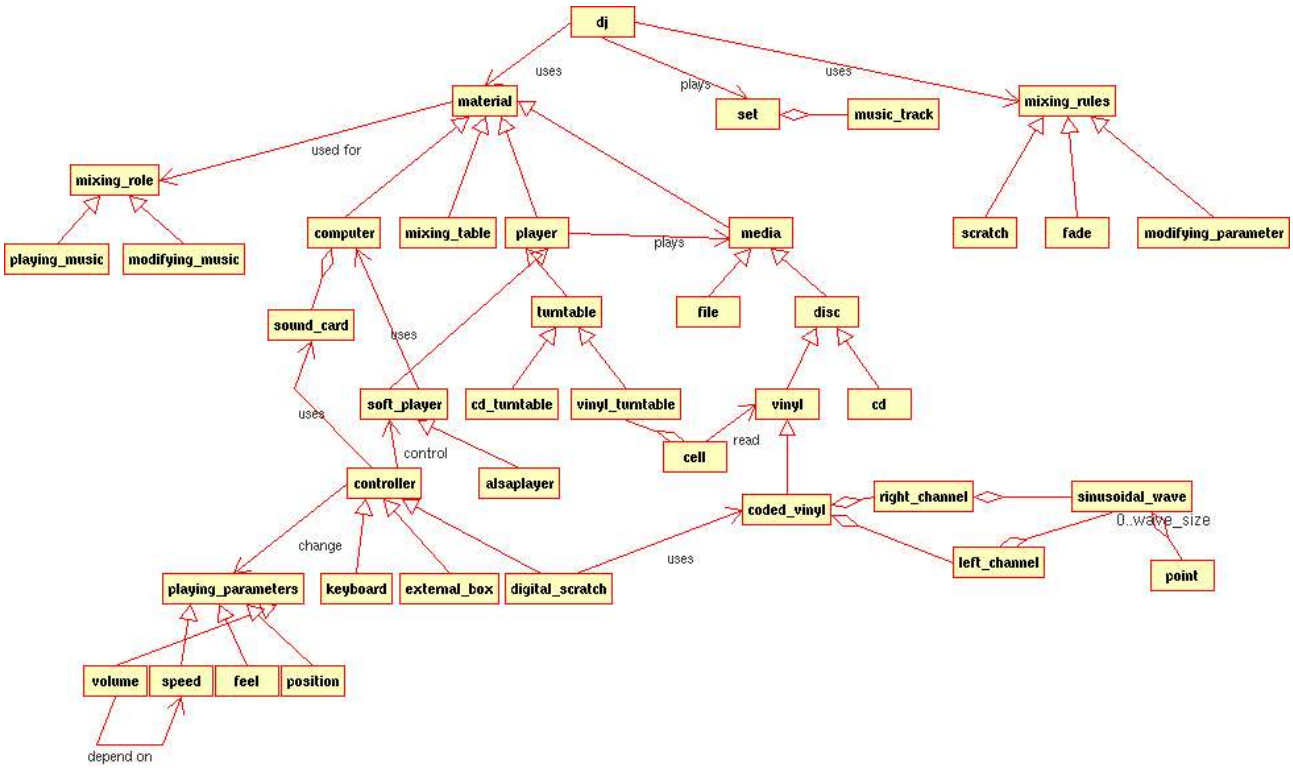


Illustration 2 - Diagramme modélisant la solution qu'apporte Digital-Scratch

Le diagramme ci dessus est basé sur celui décrivant l'état existant, cependant on y a placé *Digital-Scratch*. En fait *Digital-Scratch* est tout simplement un contrôleur de lecteur logiciel qui utilise un disque vinyle et donc une platine vinyle pour le lire.

L'idée est de laisser toujours le DJ devant sa platine qu'il pilote comme il sait le faire. La différence est qu'au lieu d'y placer ses propres disques, on y laisse un disque spécialement créé pour le système *Digital-Scratch*. Ce disque est codé : on le lit avec une platine vinyle dont la sortie n'est pas relié à la table de mixage mais à l'ordinateur qui peut décoder le son que la platine lui envoie. Ainsi, ce dernier peut déterminer les paramètres de lectures : vitesse, sens et position. Il n'y a plus qu'a piloter le lecteur logiciel avec ses paramètres. On rentrera plus tard dans les détails de conception du disque codé. Conclusion : on peut mixer des mp3 en utilisant ses platines vinyles.

3 Le mix digital : produit existants

3.1 Historique de *Digital-Scratch*

L'idée d'une application telle que Digital-Scratch m'est venu au début des années 2000 en pleine explosion du « marché » du MP3 et de ses équivalents. A cette époque, aucun système ne proposait la fonctionnalité qui consiste à mixer ces fichiers grâce au pilotage d'une platine. Seul des produits tout-en-un proposait 2 lecteurs et une table de mixage virtuel sur PC. Finalement, la concrétisation de Digital-Scratch n'a commencé à naître que fin 2003.

Entre temps, des sociétés ont développé des systèmes similaires mais comportant tous leur petites particularités Etudions leurs spécificités, elles pourront influencer celles de Digital-Scratch.

3.2 Les autres produits

3.2.1 *Final-Scratch* de Stanton Magnetics

L'application la plus semblable à la mienne est développée par Stanton Magnetics (http://www.stantonmagnetics.com). Cette dernière est commercialisé sous le nom de *Final Scratch* (http://www.finalscratch.com).

Le principe est le même que pour *Digital-Scratch* : un disque vinyle timecodé est lu par une platine dont le son est analysé par un PC qui se charge de lire un fichier audio à la vitesse, au sens et à la position correspondant à la platine. Cependant, il y a quand même quelque différences.

La partie hardware est différente : en effet, la philosophie de Digital-Scratch est de ne rajouter aucune autre partie physique à ce que peut posséder un DJ classique, 2 platines, une table de mixage et en l'occurrence un PC. Pour contourner le problème de l'amplitude du son généré par une cellule de platine, *Final Scratch* propose un ampli phono externe sous forme d'un pack contenant 2 cartes sons USB. Au final, on a plus qu'a brancher cette partie externe sur un port USB de son PC et d'y relier aussi ses 2 platines.

Le timecode aussi est différent. C'est une sinusoïde déphasée entre les 2 voies pour la restitution de la fréquence et du sens. La différence se trouve au niveau du système de positionnement qui code un mots de 16 bits sur une voie (via des différences d'amplitude). Je ne possède pas plus d'informations à propos du timecode dans la mesure ou celui-ci est breveté et que ses caractéristiques ne sont pas divulguées.

La partie software est elle aussi différente. *Final Scratch* utilise un logiciel permettant le mixage virtuel (*Traktor*). Cette application existait déjà avant Final Scratch d'où ses fonctionnalités déjà relativement avancées (comme la détection de BPM et le bouclage sur 4 temps, 8 temps, 16 temps, etc... ou encore une boite à effets permettant d'appliquer des effets sonores à la lecture en temps réel).

3.2.2 *Dvinyl 2020* de Sound Graph

Voici le système *Dvinyl* (http://www.d-vinyl.dj) commercialisé par Sound Graph (http://www.soundgraph.com).

Dvinyl fonctionne également sur le principe du lecteur de propriété du signal provenant d'un vinyle timecodé pour piloter un logiciel de lecture de fichier audio. Je ne dispose d'aucune informations sur le timecode utilisé, je sais seulement qu'il n'intègre pas de système de positionnement, ce qui le simplifie considérablement. Le logiciel propose aussi énormément de fonctionnalités avancées.

3.2.3 *DVS* de Mixvibes

Encore un système semblable : *DVS* de la société Mixvibes (http://www.mixvibes.com). Il est pratiquement identique à *Digital-Scratch* mais il me semble que son système de positionnement est presque identique à celui de *Final Scratch*. Autre points commun avec *Final Scratch* : le logiciel. *DVS* utilise Mixvibes, un logiciel de mixage virtuel déjà existant. Ici encore, ce dernier possède une multitude de fonctionnalités de traitement du son.

3.2.4 *Digital-Scratch* : les idées des produits existants

Digital-Scratch reprend quant à lui les caractéristiques d'un peu tout les systèmes vu précédemment. Le timecode est basé sur les même concept que celui de *Final Scratch*, mais est quand même différent.

La partie hardware est inexistante comme pour *DVS* (pas de boîtier externe, pas d'amplificateur, etc...) permettant de laisser une grande flexibilité : 1 platine correspond à 1 carte son. *Final Scratch* est lui limité à 2 platines alors que *Digital-Scratch* peut simuler autant de platines que nécessaire.

Comme les autres systèmes, Digital-Scratch utilise un logiciel de lecture de fichier audio déjà existant. Il s'agit d'*Alsaplayer* (http://www.alsaplayer.org). Par contre, ce dernier ne possède actuellement aucune fonctionnalité avancée de traitement du son. Cependant, il est totalement évolutif donc ces fonctionnalités existeront à termes très certainement. Ce logiciel a été choisi pour ces qualités de lectures inverses.

4 Cahier des charges

Le but de l'application est de pouvoir mixer de la musique avec un ordinateur, tout en conservant le « touché » des platines vinyles. Ce cahier des charges décrit les différentes fonctionnalités que devra proposer l'application pour répondre à ce but.

De prime abord ces dernières sont relativement simples à recenser : il faut qu'au final, on utilise sa platine exactement de la même manière que si elle fonctionnait normalement avec un disque normal. Cependant, mis à part les caractéristiques classiques de lecture de musique, il y a d'autres points à définir et à mettre en œuvre dans l'application.

4.1 Matériel pour utiliser l'application

Digital-Scratch devra tourner sur un PC possédant une ou plusieurs cartes sons équipées d'entrées ligne stéréo et de sorties stéréo classique. Bien entendu, il faudra également au moins une platine permettant de lire des disques vinyles. L'intérêt du mixage est d'utiliser plusieurs sources sonores : en effet, on doit pouvoir utiliser l'application avec 2, 3 voire plus de platines. En dernier lieu viendra se greffer une table de mixage permettant de « réunir » les différentes sources sonores pour ne faire qu'un unique son général. Voici le schéma le plus classique, le mixage avec 2 platines :

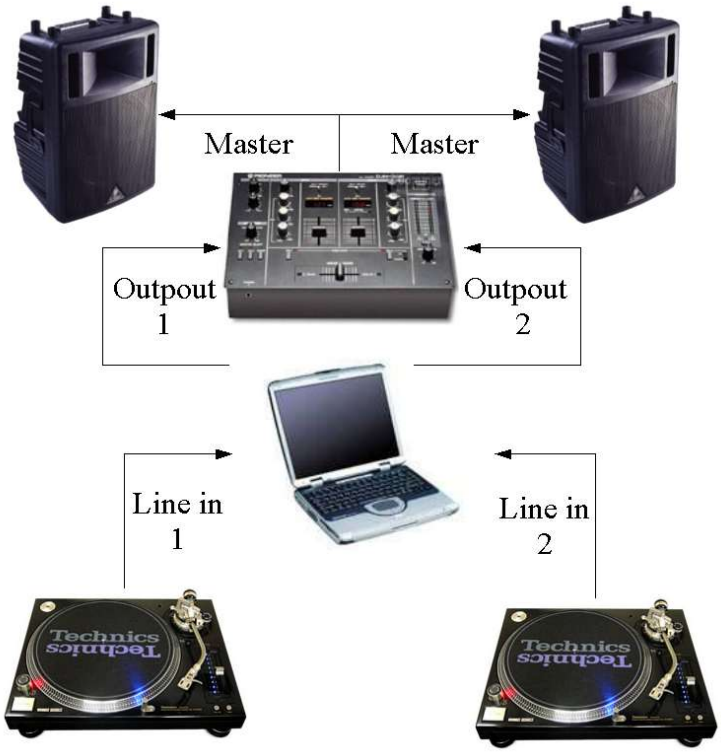


Illustration 3 - Matériel composant le système Digital-Scratch

4.2 Lecture

L'aspect principal est celui de la lecture de fichier audio. Il faut que l'application soit capable de lire ce fichier selon les caractéristiques suivantes :

- la vitesse de lecture (ou *pitch*) qui doit être proportionnelle à celle de rotation de la platine,
- le sens de lecture qui doit être dépendant du sens de rotation de la platine,
- le volume de lecture qui dépend de la vitesse de rotation de la platine (une rotation élevée rend un son plus fort),
- la position de lecture, en effet, par exemple lorsque l'on place la cellule de la platine au milieu du disque, on se trouve au milieu de la musique.

Revenons sur le système de positionnement. Un évolution intéressante par rapport aux disques classiques est d'intégrer un système de positionnement relatif. Dans le cas d'un disque normal, une musique peut atteindre environ 12 minutes au maximum. Si une musique est plus longue on est coincé. Voici une solution : considérons 12 minutes sur le vinyle codé, rien ne nous empêche d'assimiler ces 12 minutes comme valant 20 minutes par exemple (si l'on désire jouer une musique de 20 minutes). Ainsi, en fonctionnement normal, si l'on place la cellule au milieu du disque, on écoute la 6è minute de la musique. En positionnement relatif, on se trouverais à la 10è minute de notre musique de 20. On peut alors jouer des musiques plus longue que la longueur maximale enregistrable. On perd bien entendu une certaine précision puisque les échelons de temps seront plus grand, mais on peut aisément doubler le temps maximum du disque.

Un dernier mode intéressant de positionnement est celui où on ne gère justement pas le système de positionnement. Un gros souci lors du « scratch » est le problème des sauts de cellule. En bougeant trop rapidement le disque, la platine, voire la table qui la supporte, vibre et la cellule se décolle du disque. Elle retombe rarement dans le même sillon, donc la musique « saute ». Pour pallier ce problème les disques de « scratch » contiennent la même musique sur 1 cm de large (plusieurs dizaines de sillons). Pour ne pas en arriver là, il suffit de désactiver le système de positionnement, si la cellule saute, on ne sautera pas dans la musique.

4.3 Le disque vinyle

Sans vouloir déjà interpréter les fonctionnalités décrites ci-dessus, on sait déjà que tout le système sera basé autour d'un disque vinyle « codé » ou « timecodé ». Celui-ci devra répondre à certaines caractéristiques, et l'application devra proposer des services permettant de contrôler le bon fonctionnement de l'ensemble platine-disque.

- le disque (et surtout son contenu) doit être lisible par une platine classique et fournir en sortie de platine le « son » le plus proche possible de celui souhaité,
- il doit permettre de détecter de manière simple, constante et sans ambiguïté la vitesse et le sens de rotation du disque,
- un système de positionnement sera également mis en place pour détecter l'emplacement courant de la cellule.

Du coté applicatif il s'agit également de proposer des fonctionnalités liées directement au disque timecodé :

- permettre un étalonnage du signal sortant de la platine, c'est-à-dire permettre à l'utilisateur de définir le sens de rotation courant et la vitesse de rotation qu'il considère comme étant à 0% de transformation (quand on parle d'un « pitch à 0% », cela signifie que le disque tourne à la vitesse d'origine, sans modification de la platine par l'utilisateur),
- proposer un système permettant de savoir à tout instant si le disque est correctement lu

par l'application (problème de saturation, de perte de signal, etc…).

4.4 Le lecteur de fichiers audio

Une fois le signal du vinyle timecodé analysé, il faudra lire les fichiers audio via un player, celui-ci devra proposer :

- la lecture du plus grand nombre de formats audio possible (mp3 bitrate fixe, mp3 bitrate variable, ogg Vorbis, wav, mp3 pro, etc…) dans tout les sens et à des vitesses non constantes,
- un équalizer pour filtrer les fréquences,
- une boite à effets autorisant l'echo, le flange, le delay ou encore le wah wah, etc… .

4.5 L'interface utilisateur

L'interface devra pourvoir proposer au moins toutes les fonctionnalités que propose un lecteur classique comme on peut en trouver actuellement :

- paramétrage des différents éléments : choix de carte son, utilisation du système de positionnement, volume à 0¨de pitch, étalonnage du disque, etc…,
- un lecteur de fichier audio avec les fonctionnalités de bases comme la lecture manuelle, la pause manuelle, etc…,
- une playlist complètement manageable : ajout, suppression, décalage d'ordre des musiques ainsi que lecture de fichiers *.pls* et *.m3u*,
- un système de monitoring de la qualité du signal provenant du vinyle timecodé,
- un affichage des états courant de lecture et des caractéristiques (vitesse, sens, position).

5 Conception du système

Dans cette partie je vais expliquer comment j'ai conçu l'application dans son état actuelle. Cette phase de conception pure est terminé, la phase d'implémentation ne l'est pas encore tout à fait.

5.1 Diagramme de conception

Voici un diagramme de conception de l'application basé sur les idées défini dans la première partie d'analyse du domaine.

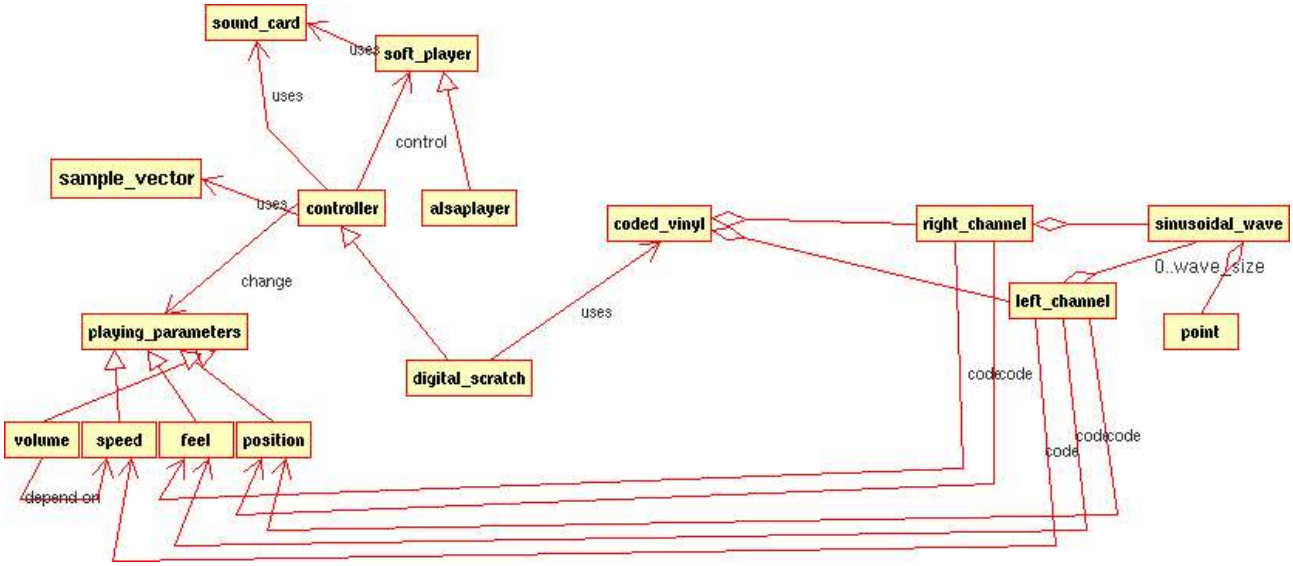


Illustration 4 - Diagramme de conception de Digital-Scratch

Au centre, on retrouve *Digital-Scratch* qui est un contrôleur dont ce dernier a pour but de piloter un lecteur logiciel. Le choix d'*Alsaplayer* comme lecteur logiciel, fera l'objet d'une petite discussion plus loin.

Notre contrôleur, ici *Digital-Scratch*, n'est autre qu'un composant logiciel qui utilise la ou les cartes sons de l'ordinateur pour récupérer le signal émis par la ou les platines. Sans vouloir déjà détaillé le contenu du vinyle codé, notons que se dernier joue sur des paramètres distincts entre les 2 voies (un disque vinyle possède un signal stéréo) qui le compose. Les signaux présent sur les 2 voies code de manière simple ou combinée les paramètres de lectures. Le paragraphe suivant est une discussion sur le vinyle codé, sur les différentes idées pour concevoir le code et bien-sûr, sur le système finalement retenu.

5.2 Le disque vinyle codé

5.2.1 Introduction

Pour pouvoir renseigner à tout instant la vitesse et le sens de lecture du mp3, il faut récupérer des informations précises provenant du vinyle en mouvement.

Ce fameux vinyle doit donc posséder une structure qui, par l'intermédiaire de la cellule et de son diamant, génère un son reconnaissable et analysable par le PC (via la carte son). Nous appellerons cette structure « timecode », car elle code le temps (par l'intermédiaire de la fréquence de rotation du vinyle et de l'emplacement de la cellule). On doit ainsi pouvoir mettre en oeuvre la détection des 3 caractéristiques d'une platine en mouvement : le sens et la vitesse de rotation du disque et la position de la cellule.

Avant de détailler le timecode utilisé, il convient de définir avec précisions les notions de platine vinyle, disque vinyle et cellule de lecture.

5.2.2 La platine vinyle

Pratiquement toutes les informations qui suivront, sont extraits du site http://www.son-video.com à la page http://www.son-video.com/Rayons/Hifi/PlatineTD/CatPlatineTD.html.

Une platine vinyle est composée des éléments suivants

- Le plateau tournant
- Le moteur et son alimentation
- Le bras
- La cellule
- Le socle
- Les accessoires

Le plateau tournant supporte et entraîne le disque. Il est étroitement associé au système d'entraînement (moteur). L'objectif commun à toutes les platines est d'offrir un entraînement très régulier et silencieux. La cellule se comporte comme un microphone qui « écoute » les gravures du sillon. La moindre vibration est donc directement amplifiée et audible.

Illustration 5 - Platine vinyle en éclatée

5.2.3 Le disque vinyle

Avant de détailler les technologies mises en œuvre pour lire les disques, il est important de comprendre le fonctionnement de l'enregistrement phonographique.

Pour produire un disque, l'enregistrement est tout d'abord masterisé. Cette finalisation consiste à ajuster tous les paramètres de l'enregistrement pour que celui-ci soit compatible avec le support de diffusion. Cela existe aussi pour le CD mais avec d'autres contraintes. Pour le vinyle, il faut limiter

la dynamique, notamment dans les graves pour conserver une taille de sillon compatible. Le responsable de cette opération décide de la manière de traiter la bande-son pour conserver les aspects artistiques (en relevant délicatement certains passages par exemple).

Tous les disques ne sont pas identiques. Entre les modèles standards des années 70 à 80 et les modèles audiophiles (disponibles mais rares depuis les années 70) en vinyle 180 ou 190g... il y a un monde: les modèles haut de gamme acceptent une dynamique bien plus importante. Les disques standard sont pressés en 125g. Il existe des pressages spéciaux de haute qualité en 160, 180 et même 190g.

La gravure nécessite une égalisation pour accentuer les aigus et réduire les graves suivant une courbe normalisée par la RIAA. Une courbe inverse est appliquée dans le préampli en sortie de cellule. Ce traitement permet d'améliorer considérablement le rapport signal / bruit des disques.

Le support (appelé flan) est un disque en aluminium recouvert d'une laque de nitrocellulose. La gravure se fait par un burin placé sur un bras tangentiel. La vitesse d'avance du bras est réglable pour permettre une durée plus ou moins longue. Plus la durée est longue et plus les sillons sont serrés et donc, moins il y a de capacité dynamique. On peut aussi augmenter la dynamique en utilisant une vitesse plus grande, 45tr/min, sur un 30cm ou en employant un système de compression de dynamique (dBX). Cette dernière solution nécessite un décodeur (expanseur) ce qui en limite la diffusion. On tirera de ces flans un moule métallique, la matrice de pressage, réalisée par galvanoplastie. Le disque sera ensuite pressé à chaud (160°) avec une pâte de PVC/acétate de vinyle. La couleur noire est donnée par un colorant. Certains disques sont pressés en couleur pour des éditions collector, cela ne changeant rien au son.

Les disques microsillons ont été fabriqués avec 2 vitesses normalisées: 33 et 45 tr/min. Toutes les platines proposent ces 2 vitesses. Les disques plus anciens, c'est-à-dire avant les microsillons sont le plus souvent en 78tr/min. Cependant, les débuts de l'industrie phonographique ont connu des tâtonnements traduits par des vitesses très variées (40, 60, 78 tr/min).

Caractéristiques des disques	33 tr/min	78 tr/min
Largeur des sillons	25...100 µm	100...125 µm
Pas entre sillons	85...140 µm	250...300 µm
Bande passante	30 Hz...15 kHz	100 Hz...10 kHz
Rapport signal/bruit	env. 60 dB	env. 35 dB

5.2.4 La cellule et le problème de transformation du son de sortie

La cellule phono délivre un signal de très bas niveau: 3mV pour les modèles à aimant mobile, 0,3mV pour celle à bobine mobile. Afin de permettre un bon rapport signal / bruit, le disque est gravé avec une pré-accentuation qui remonte les aigus et réduit les graves. La courbe de préaccentuation a été définie par l'association des éditeurs phonographiques américains, la fameuse RIAA qui a donc donné son nom à cette courbe. Le préamplificateur phono utilise une courbe inverse définie suivant des normes rigoureuses.

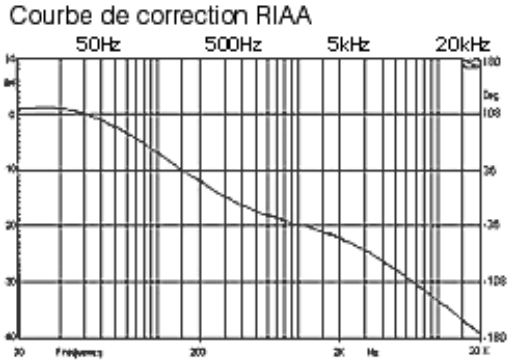


Illustration 6 - Courbe de correction RIAA

La courbe de lecture doit être rigoureusement inverse à celle de la gravure des vinyles. Voici la formule permettant d'appliquer les corrections que l'on doit appliquer à la lecture, si l'on veut récupérer les timbres originaux et les différents équilibres :

$$dB = 10\log_{10}\left(1+\frac{1}{4\pi^2 f^2 {t_2}^2}\right) - 10\log_{10}\left(1+\frac{1}{4\pi^2 f^2 {t_1}^2}\right) - 10\log_{10}\left(1+4\pi^2 f^2 {t_3}^2\right)$$

A l'aide d'un tableur, on peut facilement trouver la valeur de fréquence pour laquelle il n'y à aucune correction . On trouve alors une valeur de 1026 Hz, disons pour simplifier, 1kHz.

5.2.5 Le timecode, le code du vinyle de *Digital-Scratch*

Cette dernière valeur est très intéressante : en effet, vu la faible intensité du signal généré par une cellule, on ne peut utiliser l'entrée ligne d'une carte son. Reste alors l'entrée micro, qui elle, est capable de « remonter » le signal d'entrée en appliquant un filtre de +20dB. Il reste un problème : par défaut les cartes sons n'intègre pas de manière matérielle la compensation calqué sur la courbe RIAA. Le signal récupéré n'est donc pas le bon. Or, il existe une valeur qui ne subit aucune transformation si un filtre RIAA était présent, c'est justement cette fréquence de 1026Hz. On peut donc affirmer maintenant que si l'on presse un vinyle avec un « son » à 1026Hz, celui-ci apparaîtra de la même manière à 1026Hz dans la carte son. Autre intérêt : si on possède une platine intégrant d'emblée le décodage RIAA, celui-ci n'aura presque aucune influence (au moins pour la vitesse de rotation à 0% d'augmentation). C'est grâce à la fréquence du signal recueilli que l'on pourra jouer le mp3 à la vitesse correspondante à la vitesse de rotation physique du disque.

Deuxième caractéristique du « son » à utiliser : la forme. Pour pouvoir facilement détecter la fréquence et l'amplitude du time code, il convient d'utiliser un signal sinusoïdale, plus apte a être pressé sur un disque vinyle que le « parfait » signal carré.

Troisième caractéristique. Il faut en dernier lieu pouvoir détecter le sens de rotation du vinyle. Pour cela, nous allons jouer sur les 2 pistes (stéréo) du disque. Chaque voie possèdera le même timecode à un décalage d'un quart de période prêt. Comme cela, si on considère que c'est la voie de gauche qui est en avance sur celle de droite dans le sens normal de lecture (front montant de G avant celui de D), il sera aisé de détecter un sens inverse si l'ordre d'arrivée des fronts change. Si pour un front montant de gauche on suit par un front montant de droite, c'est que l'on tourne dans le sens normal, sinon, si c'est un front descendant qui suit sur la voie de droite, c'est que l'on tourne dans le sens inverse.

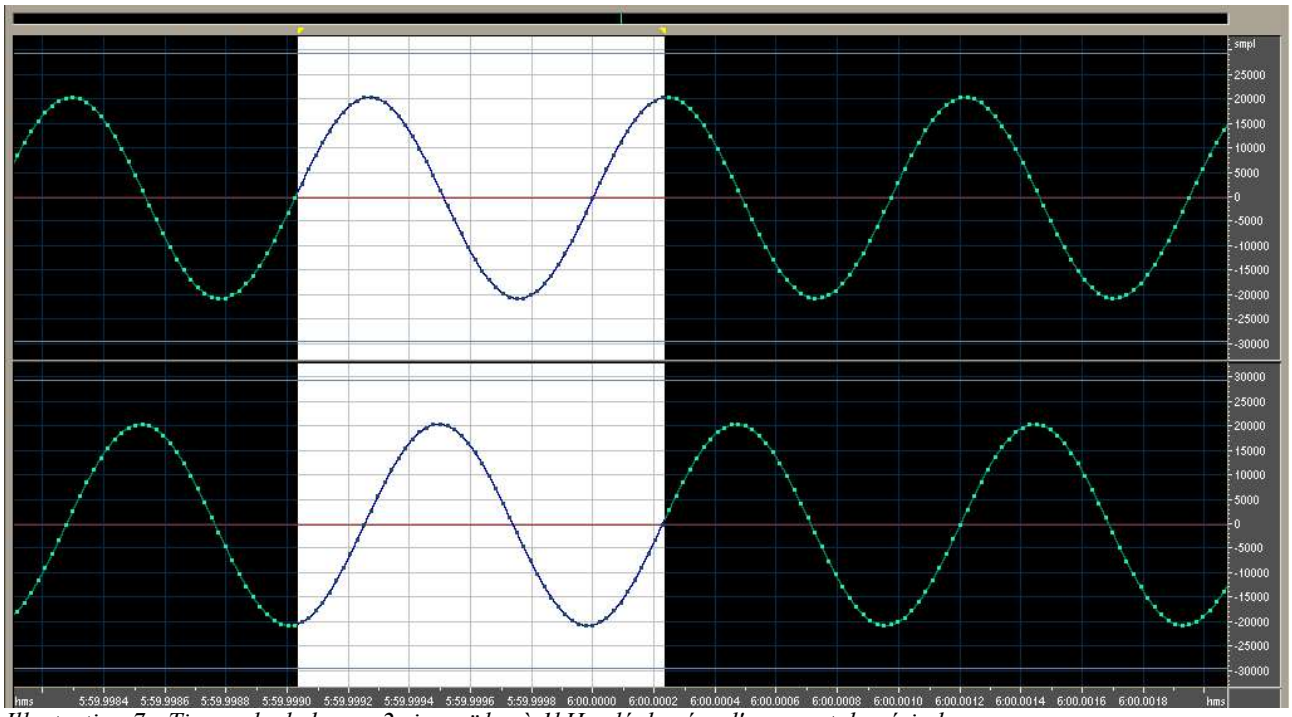


Illustration 7 - Timecode de base : 2 sinusoïdes à 1kHz, déphasées d'un quart de période

5.2.6 Le problème du positionnement de la cellule

Après avoir simulé la vitesse de rotation du disque vinyle et son sens, il reste à repérer l'emplacement de la cellule. On entend par « emplacement » la position diamétrale de la cellule. Dans le cas de l'usage d'un disque classique, si l'on place la cellule au début du disque, on entendra le début du morceau, de même pour la fin. Il s'agit donc de simuler cet emplacement pour lire le fichier mp3 à partir de l'endroit exact correspondant à l'emplacement de la cellule.

Pour cela, plusieurs solutions sont possibles. Tout d'abord, on peut penser à faire diminuer progressivement le volume sonore du timecode sur l'ensemble du disque. Ainsi à tout instant connaissant l'amplitude du timecode, on peut déterminer la position de la cellule. Après avoir effectués des tests, on s'aperçoit que l'amplitude n'est pas une donnée viable. Elle a tendance a osciller ou, lors de problème à la surface du vinyle, d'être complètement fausse pendant un temps relativement long. La solution, de part son manque de précision, ne peut pas être retenue.

La deuxième solution par d'une constatation simple : deux notions nous sont directement accessibles dans un timecode, l'amplitude et la fréquence. Si l'amplitude n'est pas une donnée à tout instant satisfaisante, tentons de « coder » notre position de manière fréquentielle. Problème : on ne peut pas toucher à notre fréquence de base qui nous permet de déterminer la fréquence de rotation du disque. Voici les 2 premières solutions étudiées pour la première version de l'application

5.2.6.1 Timecode 1 : jouer sur le forme

L'idée ici est d'utiliser un décalage temporel des pics de la sinusoïde. En temps normal, la valeur maximum d'une sinusoïde se trouve centrée par rapport au front montant et descendant qui l'encadre. L'intérêt ici est de décaler soit vers l'avant soit vers l'arrière (en termes temporel) ce maximum (et de manière équivalente le minimum) de manière a détecter son emplacement et par là de proposer un codage de la position de la cellule. Voici un exemple sur cette courbe :

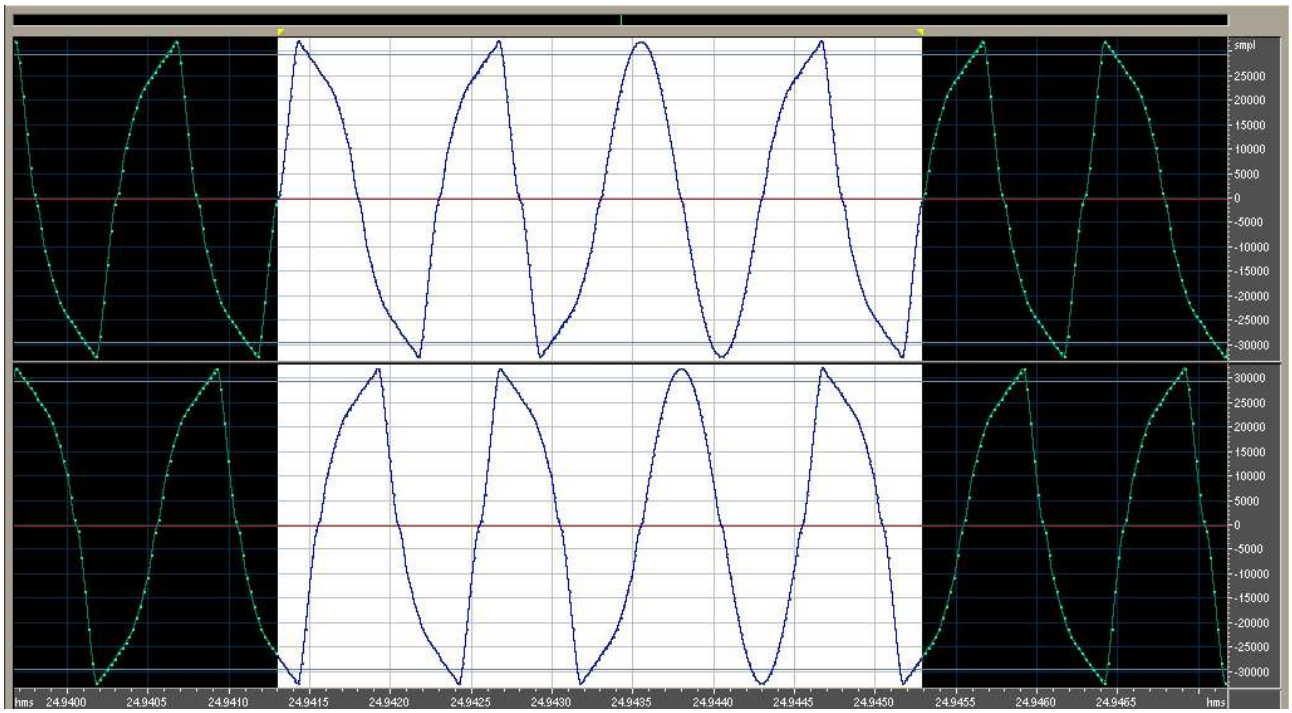


Illustration 8 - Timecode utilisant la modification de forme

Ici, le codage s'effectue sur 4 périodes de signal (2 informations, la partie haute et la partie basse) que l'on peut doubler en utilisant la période correspondante sur l'autre voie. Cela nous ramène à 16 informations toutes les 4 périodes recueillies. Si l'ensemble de ces 16 informations représente un chiffre hexadécimal d'un nombre de type 0xXXXX, on peut coder 65536 valeurs différentes.

5.2.6.2 Timecode2 : rajouter des informations fréquentielles

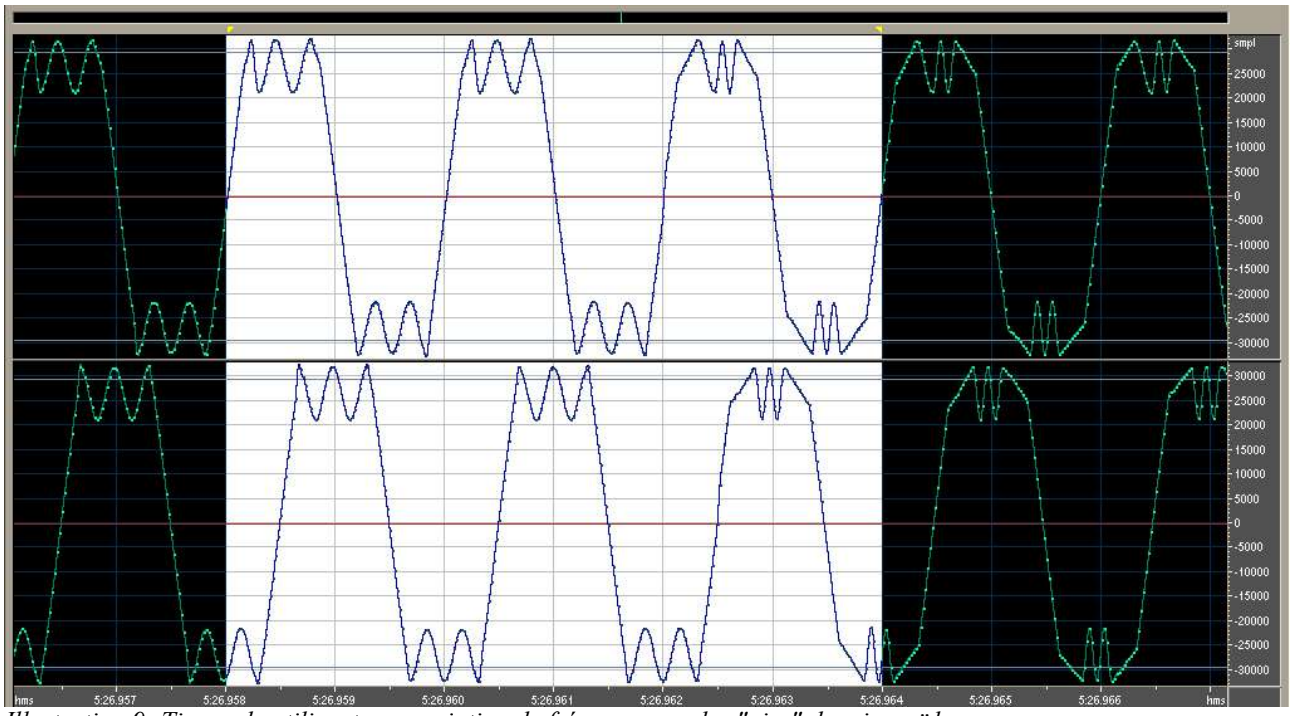


Illustration 9- Timecode utilisant une variation de fréquence sur les "pics" des sinusoïdes

Autre solution, rajouter à chaque pics de sinusoïde une informations de 4 nombres binaires 0 ou 1. Pour cela, il suffit de placer de manière centré au niveau de chaque pic une autre sinusoïde de 4 demi périodes dont chacune peut prendre 2 fréquences, l'une haute représentant un 0, l'autre faible représentant un 1. Ensuite, on utilise les 2 pics haut et bas de chaque voie. Ceci nous ramène à 4 informations par pics, donc 16 informations en tout, c'est-à-dire un nombre de 16 bits pouvant prendre également 65536 valeurs. On travaille donc cette fois-ci sur seulement une période, la réactivité est donc très grande. La courbe précédente illustre cela.

5.2.7 Les tests

Avant de passer à une phase finale de pressage, nous avons décidé de réaliser une dubplate (pour simplifier, une « gravure » d'un disque vinyle possédant une bonne qualité sonore mais rapidement altérable). Celui-ci contient un timecode simple, c'est-à-dire une sinusoïde déphasée entre les 2 voies. La première chose à faire, est simplement d'enregistrer le disque dans différentes conditions et d'observer les différentes altérations que peut prendre le signal (à l'opposé d'un format numérique qui ne doit théoriquement pas en fournir).

5.2.7.1 Lecture normale

Etudions tout d'abord, la lecture normale. Dans un premier temps, on peut constater la qualité générale de sortie du signal : en effet on constate très peu de bruit autour du signal. On peut également noter que le déphasage n'est pas tout à fait à la hauteur de celui réalisé numériquement, en effet, il n'y a pas tout à fait ¼ de période de différence. Cependant cela ne devrait pas être vraiment très gênant dans la mesure ou le déphasage est bien présent. Deuxième remarque, l'oscillation générale de l'ensemble du signal qui rend la détection de l'amplitude peut aisée.

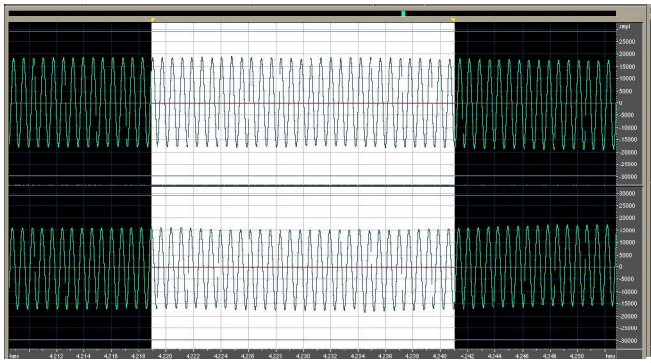


Illustration 10 - Enregistrement d'une sinusoïde (1)

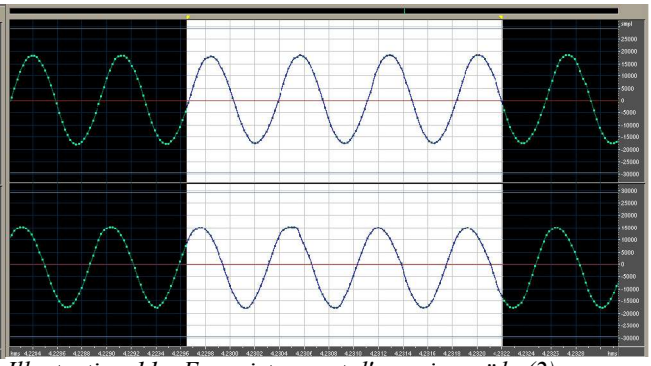


Illustration 11 - Enregistrement d'une sinusoïde (2)

Je n'ai pas d'image des enregistrements des 2 timecodes précédemment évoqués, mais le résultat est très mauvais, il n'y a aucune chance de pouvoir utiliser le système de positionnement. Peut-être est ce du au fait que ce ne sont que des dubplates et non des vrais vinyles pressés ?

5.2.7.2 Arrêt lent

Ensuite, on procède à un arrêt (relativement lent) de la rotation du disque. Ici, on constate d'une part que l'allure générale possède quelques problèmes d'amplitude, en effet à deux reprises, le signal forme une « bosse » plutôt inexpliqué mais dont il faut tenir compte. Le signal baisse également progressivement en intensité, concrètement on l'entend, lorsque qu'on arrête le disque, le son est de moins en moins fort.

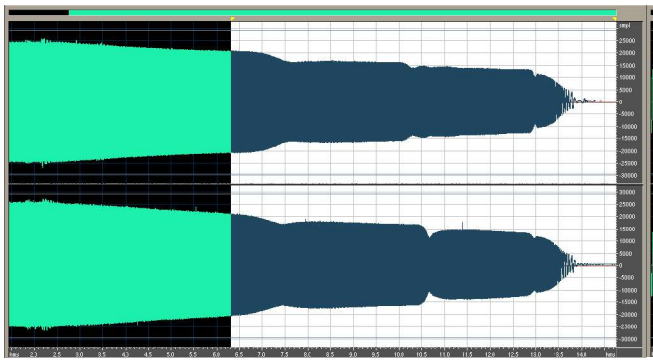


Illustration 12 - Arrêt lent de la platine (1)

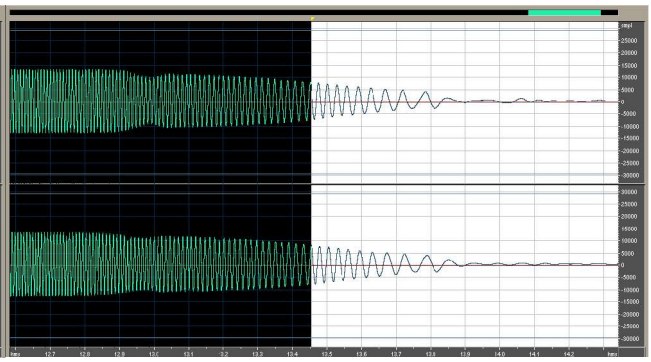


Illustration 13 - Arrêt lent de la platine (2)

5.2.7.3 Pose de cellule

Dans le cas suivant, on procède à la pose de la cellule alors que le disque est entrain de tourner, ce que l'on fait régulièrement si l'on cherche un passage spécifique dans un morceau de musique. Dès l'impact de la cellule avec le disque, le signal se forme a peu de chose prêt correctement, en tout cas, l'allure générale est présente. Il y a cependant de gros écarts d'amplitude donnant des saturations fortes et donc une perte importante de signal. Cette première phase dure environ 150 ms. Ensuite on peut noter l'oscillation très prononcée de l'ensemble de la courbe. Ce phénomène est particulièrement remarquable durant un peu moins d'une seconde.

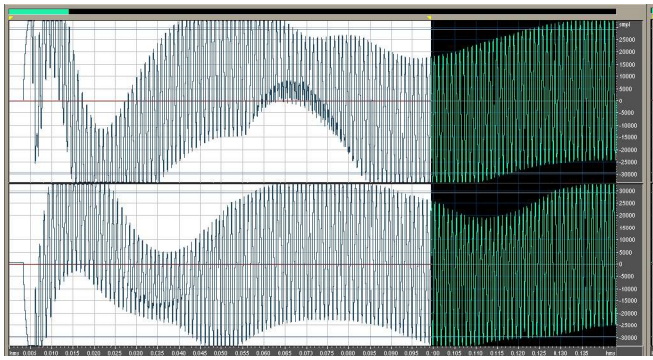


Illustration 14 - Pose de cellule (1)

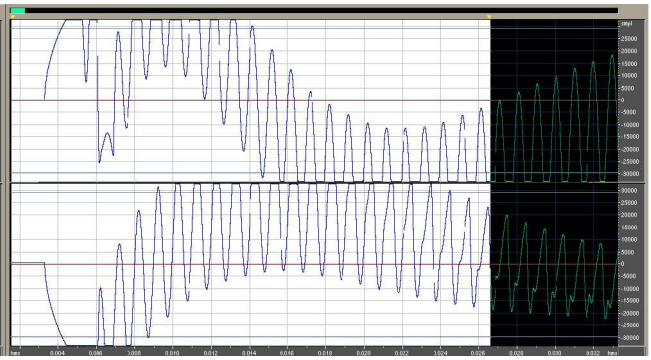


Illustration 15 - Pose de cellule (2)

5.2.7.4 Scratch

Dernier test : le « scratch ». On entend par cela, un changement rapide de sens de rotation avec des accélérations et des décélérations plus que prononcées. Dans ce test, on réunit pratiquement tout les problèmes rencontrés dans les précédents : forte oscillations de l'allure générale, risque de saturation lors des changements de vitesse rapide, déformations de la pointes des sinusoïdes. Cependant, on notera peu de problème lors du changement de sens, la courbe est a peine déformée, et cela est normal puisque le déphasage s'inverse.

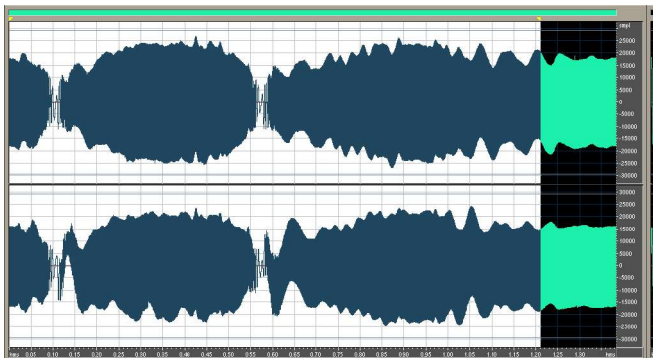


Illustration 16 - Scratch (1)

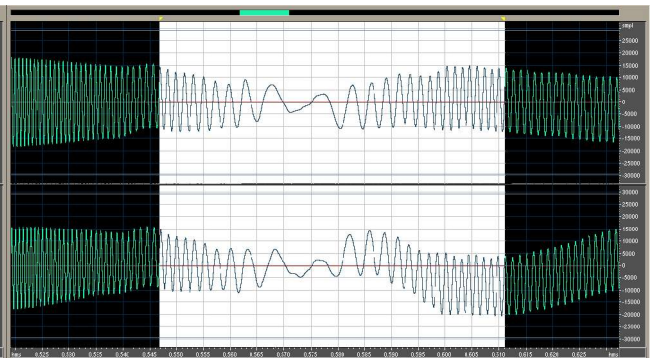


Illustration 17 - Scratch (2)

5.2.8 Timecode final : un travail sur la forme et la fréquence

5.2.8.1 Vue générale

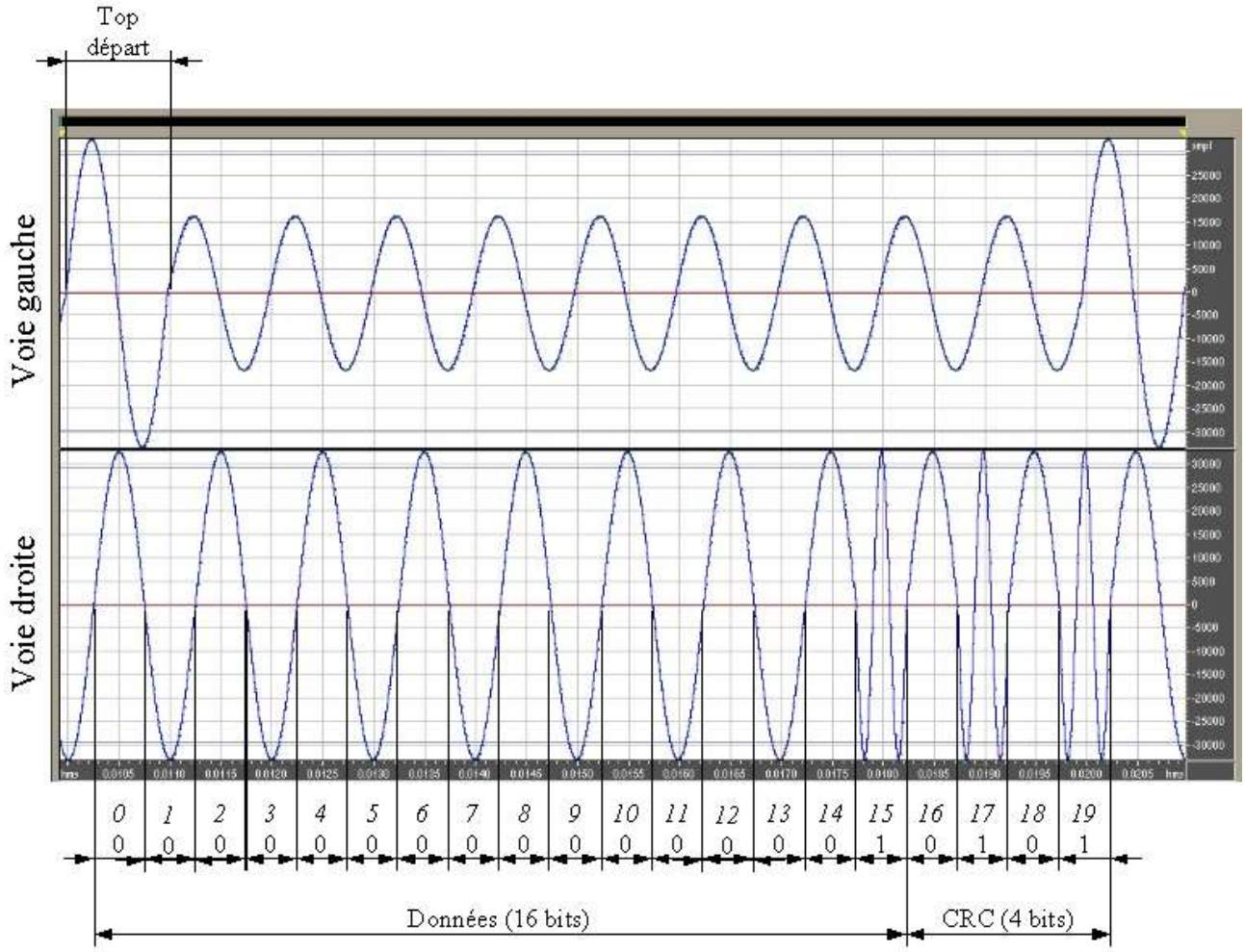


Illustration 18- Timecode utilisant la différence d'amplitude et de fréquence pour coder la position

Cette partie va vous présenter le timecode final, celui qui est pressé sur un disque vinyle. Il tente d'utiliser plusieurs notions vu dans les timecode précédent. Ci-dessus, le timecode final.

On utilise ici 2 formes distinctes mais quand même liées entre la voie gauche et la voie droite.

5.2.8.2 Vitesse de rotation

Le premier problème est celui de la vitesse de rotation du disque : on la détecte via la voie de gauche qui est une sinusoïde à 1kHz.

5.2.8.3 Sens de rotation

Ensuite, il faut détecter le sens : la voie de droite est décalée d'un quart de période par rapport à celle de gauche. Attention cependant, parfois, la sinusoïde de la voie de droite est cadencé à 3kHz, des fronts « inexacts » apparaissent. Il faut toujours se référer à la voie de gauche qui est la référence en matière de fréquence de rotation. L'algorithme de recherche de sens sera celui-ci (de manière très simlifié) :

- On prend 2 front consécutifs de la voie de gauche
- On cherche le « milieu » de ces 2 fronts
- On cherche le front de la voie de droite le plus proche de ce milieu
- Si le premier front de gauche est le même que celui de droite, on tourne dans le sens normale sinon, c'est que le disque tourne dans le sens contraire.

5.2.8.4 Système de positionnement

Le gestion de la position s'effectue de la manière suivante. On va utiliser 10 périodes de la voie de droite pour « coder » une position. Avec 10 périodes, on a 20 demi-périodes qui représenteront chacune un bit. Ces 20 bits sont découpés en 16 bits de données de positionnement et 4 bits de données CRC permettant de vérifier l'intégrité des données précédentes. Pour détecter le début d'un ensemble de 20 bits, on utilise l'autre voie, celle de gauche. Cette dernière est constituée d'une sinusoïde de valeur maximum 16384 suivant un codage d'amplitude sur 16bits. On a placé tout les 10 périodes une période dont la valeur maximum est de 32768 (2 fois plus que la précédente). Cette différence d'amplitude est suffisamment importante pour être détecter sans trop de problèmes. C'est cette période qui est appelée « top départ » sur le schéma précédent.

Un bit 0 sera en fait une demi-période à 1KHz, un bit 1 sera une demi-période à 3KHz. Avec 16 bits de données, on peut donc coder 65536 positions sur le disque. Sachant que 10 périodes représentent à 1Khz 10 ms, on peut coder toutes nos valeurs sur 655 secondes, à savoir 10 min 55 seconde (valeur tout à fait acceptable pour le pressage sur disque).

La vérification d'intégrité de la trame de données est effectué via un algorithme de CRC-4. Tout les détails pour la recherche de ces 4 bits en fonctions de la trame et aussi la vérification de la validité des données à partir du CRC se trouve dans le document *Contrôle CRC* en annexe1.

Bien entendu sur une durée de 10 périodes, la détection de la position ne sera pas très rapide. Elle sera cependant très sûre via le système de vérification de la trame codant la position. On estime qu'il faut au maximum 3 codes de position complet pour être certain de pouvoir déterminer la position. En effet, la première peut être incomplète, la 2è erronée mais la dernière sera certainement correcte. Cela représente 3x10 périodes, donc 3x10ms=30ms.

5.2.8.5 Vue schématique

Voici une vue schématique du timecode dans le contexte du système *Digital-Scratch*. Le vinyle est composé de 2 voies, gauche et droite, qui sont toutes les 2 composées de signaux sinusoidaux.

Cependant, la notion de sinusoïdes fait apparaître celle de période et dans notre cas de demi période ou symboliquement de « vague sinusoïdale ». Ainsi chaque voie est une ensemble de vague sinusoïdale de propriété différentes. Chacune d'elles sera composées d'une liste de points (les échantillons du signal).

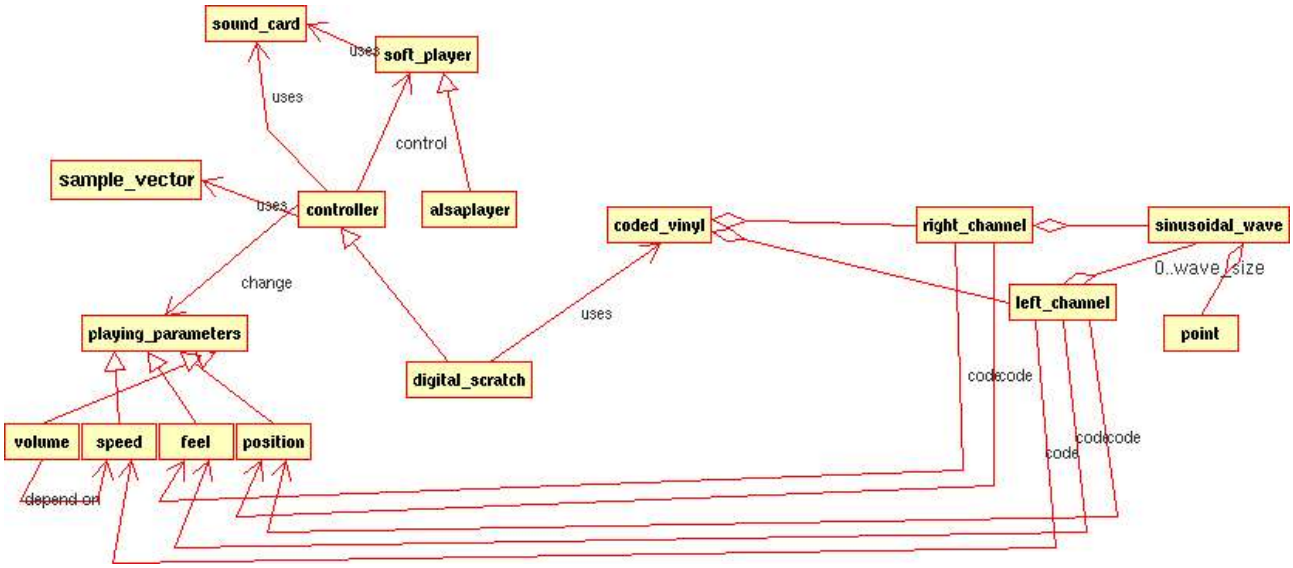


Illustration 19 - La structure du vinyle codé

5.3 L'analyse du signal

5.3.1 Présentation

Lorsqu'on joue le disque via la platine vinyle, on récupère un signal analogique plus ou moins proche de la réalité (cf. partie *Time Code*). On récupère ce signal grâce à l'entrée ligne de la carte son par l'intermédiaire de « buffer », c'est-à-dire des échantillons de données sonores. Le principe du programme est simple : pour chaque buffer de données récupérées, on recherche toutes les informations dont nous avons besoin (vitesse de lecture, sens de rotation, emplacement de la cellule). La partie qui suit traitera de l'analyse des données sonores mais également de la manière de stocker toutes ces informations.

5.3.2 L'analyse des données récupérées par la carte son

Après avoir récupéré un buffer de données sonores, on l'analyse et de cette analyse naîtrons les paramètres de lectures.

L'idée est d'analyser les données et de les représenter sous la forme définie dans le paragraphe précédent, c'est-a-dire, en les structurant comme une liste de vagues sinusoïdales de type différent.

Pour la voie de gauche, les types sont : « amplitude normale » et « amplitude top départ ». Pour la voie de droite, les types sont : « bit 0 » et « bit 1 ».

5.3.2.1 Recherche des vagues sinusoïdales

Pour trouver les vagues sinusoïdales, il faut repérer les fronts (montants et descendants). Pour répondre à ce problème, on peut penser dans un premier temps rechercher tout les « passages » de la courbe de données par la valeur 0. En effet, c'est la théorie. Seulement après quelques tests pratiques avec une platine, on se rend compte par exemple que le signal possède un offset (décalage en amplitude) non négligeable, la valeur 0 n'est donc plus celle où se produisent les fronts. Autre problème : tout l'ensemble du signal peut osciller de manière non négligeable (cf partie *Test* du *Time code*). La courbe peut alors ne pas passer par 0 pendant un temps indéfini. Tout ceci peut être source de multiples erreurs.

Si la valeur de définition des fronts n'est pas 0, il s'agit de trouver « à tout instant » laquelle elle est. Voici une solution plus ou moins exacte : définir une courbe moyenne à la courbe des maximums et des minimums. On le distingue concrètement sur ce schéma :

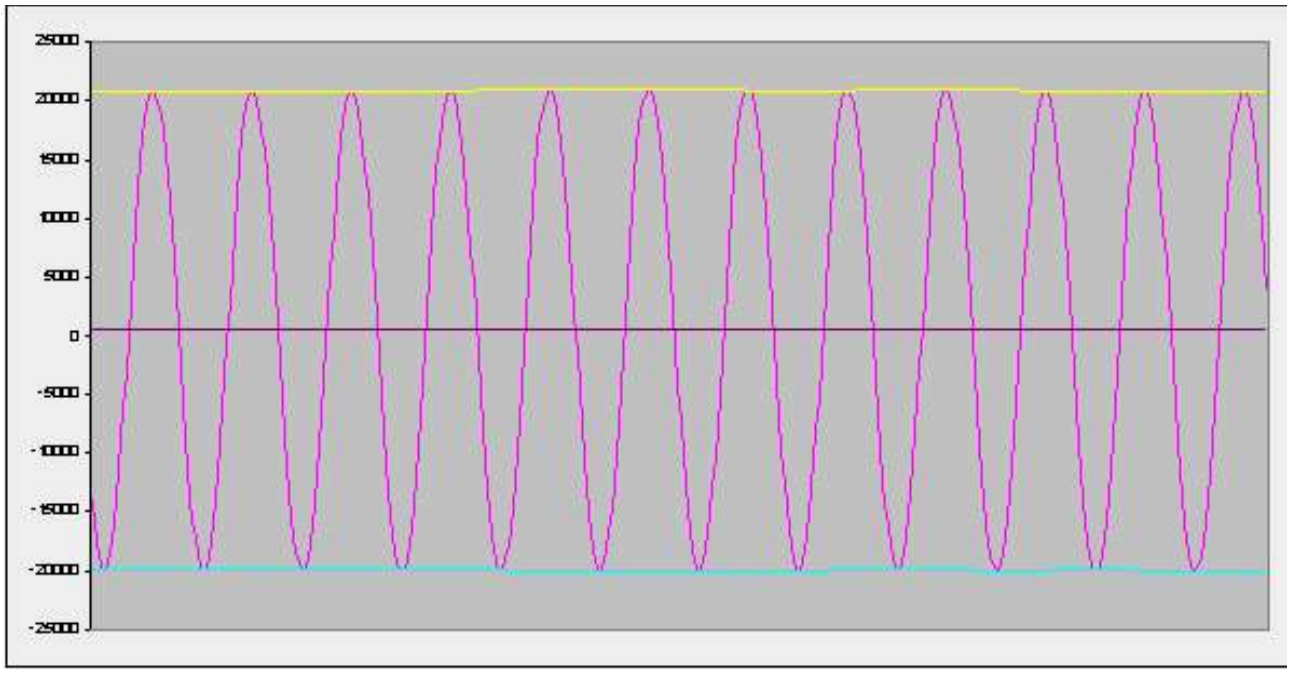


Illustration 20 - Courbe moyenne sur des données correctes

La sinusoïde représente le timecode issu du vinyle. La courbe jaune est celle qui relie tous les maximums de la sinusoïde, celle du bas pour les minimums. La courbe du milieu est la moyenne de ces 2 dernières. On s'aperçoit maintenant facilement du décalage de cette dernière qui ne se trouve pas en 0 mais plutôt aux alentours de 400-500.

Cependant, un gros problème entrave cette démarche : si un maximum n'est pas vraiment bon, c'est-à-dire si le timecode est mal restitué, la courbe des maximums sera fausse et par voie de conséquence la courbe moyenne. Ceci est également valable pour la courbe des minimums. En voici une illustration :

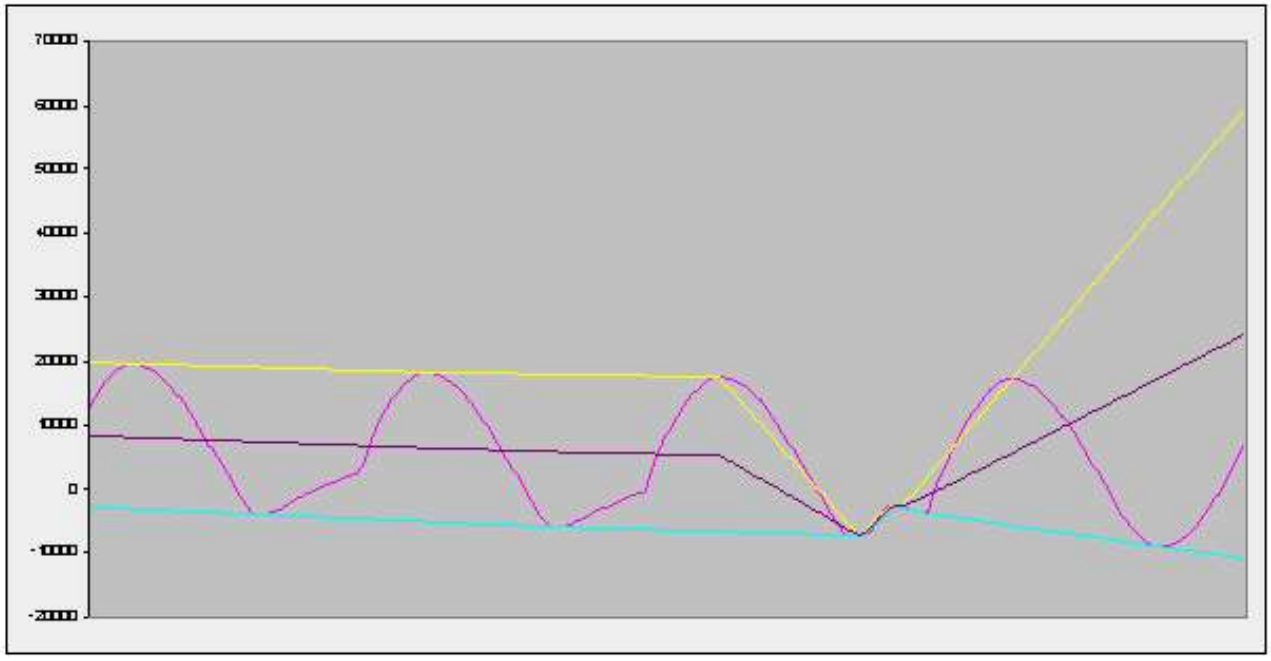


Illustration 21 - Courbe moyenne sans correction sur des données altérées

Ici le timecode est clairement altéré, on note tout particulièrement deux « faux » maximum sur la fin, qui faussent complètement la courbe des maximums et dans une moindre mesure celle des minimums. Plusieurs solutions sont envisageables, cependant elles posent toutes d'autres problèmes. En voici une très simple qui permet d'améliorer déjà beaucoup la courbe moyenne. Tout maximum se situant en dessous de la barre des 0 ne sera pas prix en compte, et tout minimum supérieur à 0 sera occulté. Voici ce que cela peut donner dans le cas d'un timecode fortement altéré :

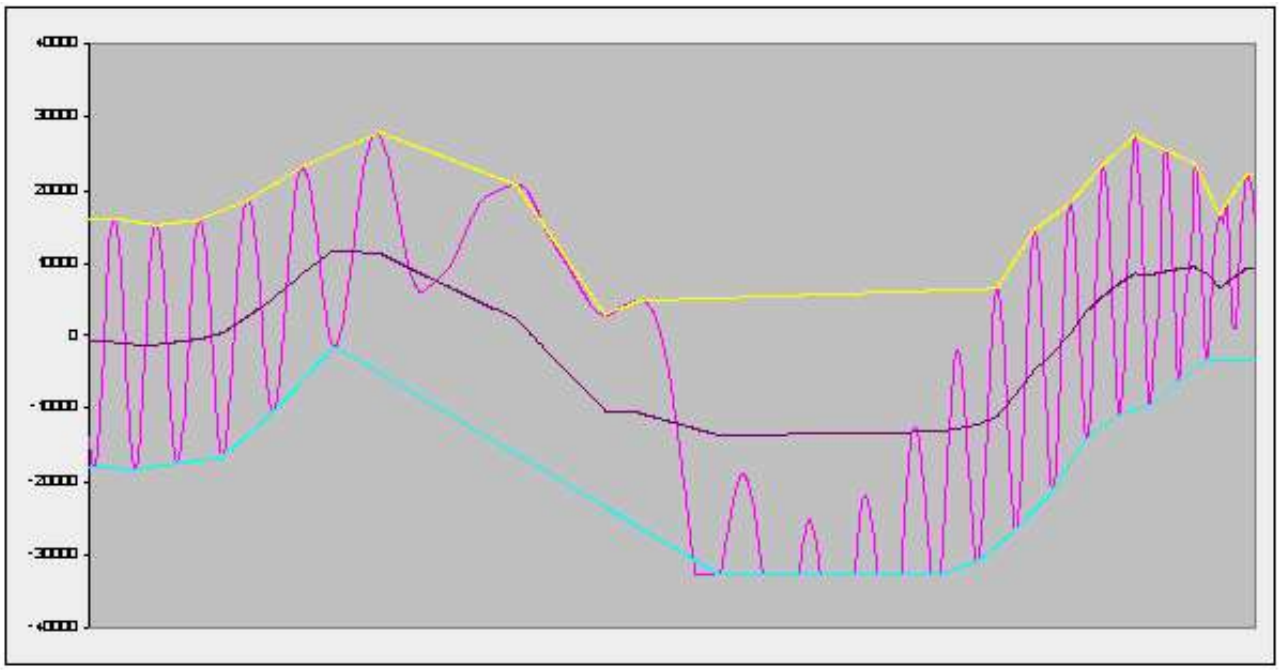


Illustration 22 - Courbe moyenne avec correction sur des données altérées

Ici, on voit nettement la courbe moyenne suivre le cours général du timecode. Evidemment dans les cas extrêmes, le fait de ne pas tenir compte des max en dessous de 0 n'est pas l'idéal, mais fonctionne la plupart du temps. Evolution possible : un lissage de la courbe moyenne permettant d'occulter encore les erreurs de détection de maximum et de minimum. En tout les cas, une détection de front par rapport à 0 aurait été catastrophique dans un cas comme celui la.

5.3.2.2 Evolution de la structure des vagues sinusoïdales

Si on récupère régulièrement des buffers de données puis qu'on les transforme en structure de vagues sinusoïdales, on risque rapidement de saturer l'ordinateur avec des milliers d'éléments. De plus, on n'a rapidement plus besoin de certains éléments, seuls les derniers reçus sont intéressant.

On procède donc régulièrement à la concaténation des données courantes avec les précédentes (en liant correctement le dernier « bout » de vague précédent avec le premier « bout » de vague courant). Une fois qu'on a détecté les paramètres de lectures, on élimine les vieilles vagues sinusoïdales. On considère qu'il faut toujours en garder 40 pour pourvoir analyser correctement (pour coder une position, on utilise 20 vagues, donc, dans le pire des cas il en faut 40 pour la détecter).

5.3.2.3 Détection de la fréquence

Une fois la recherche de vagues sinusoïdales terminées, on peut passer à la recherche de la fréquence du signal. C'est très simple, il suffit de déterminer les périodes et de les rapporter au taux d'échantillonnage courant. Plus précisément on additionne toutes les demis périodes que l'on divise par leur nombre total pour faire une moyenne, puis on détermine la fréquence, voici la formule utilisée :

$$fréquence = \frac{taux\,d'échantillonnage}{\frac{dernier\ front - premier\ front}{nombre\,demi\,période} \times 2}$$ ¶

5.3.2.4 Détection du sens

Pour déterminer le sens de rotation du disque, on utilise un système de déphasage. En effet, le timecode est pratiquement identique sur la voie gauche et sur la voie droite, au déphasage prêt : la voie de droite possède ¼ de période de retard par rapport à celle de gauche. Pour repérer ce déphasage, il suffit d'analyser les fronts. Si l'on prend un front de la voie gauche et le front immédiatement suivant (en terme temporel) de la voie de gauche, on constate ceci :

- front montant suivit d'un front montant : sens normal
- front descendant suivit d'un descendant : sens normal
- front montant suivit d'un front descendant : sens inverse
- front descendant suivit d'un front montant : sens inverse

De manière condensée on peut dire : lorsque qu'un front est différent de celui qui suit, c'est que le disque tourne a l'envers, si par contre les fronts sont les mêmes, le disque tourne dans le sens normal.

5.3.2.5 Détecter la position

Pour détecter le position, il faut d'abord attendre 2 vagues sinusoïdales de type « top départ » sur la voie de gauche puis analyser les 20 vagues suivantes sur la voie de droite. Pour vérifier l'intégrité des données de la voies de droite, on applique l'algorithme inverse de CRC qui valide ou non la position.

La position ne doit pas être renseignée à tout instant, en effet elle n'est pas suffisamment souvent

détecté pour piloter de manière régulière le lecteur. On privilégie d'abord la fréquence que l'on peut améliorer grâce à un système d'asservissement avec la position. Concretement, si la position détecté sur le vinyle, n'est pas la même que celle sur le mp3, on rattrape le retard ou l'avance en « trichant » un peu sur la fréquence.

5.3.2.6 Gestion du volume

Le volume n'est pas détecté directement avec les données provenant du timecode. En fait, le volume est dépendant de la vitesse de rotation, voici une courbe volume = f(vitesse) :

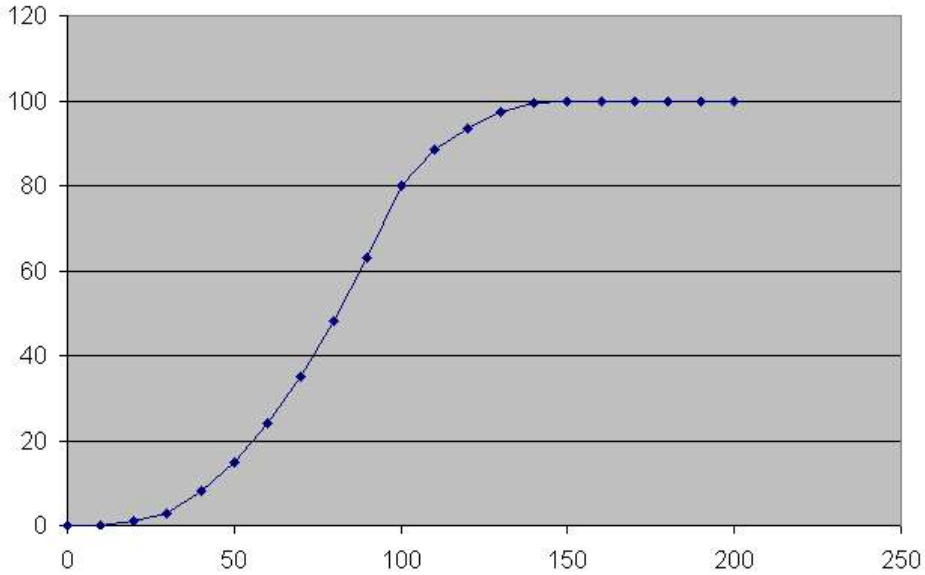


Illustration 23 - Courbe volume = f(vitesse)

5.4 Organisation du travail

Ce projet se voulant être un modèle de développement d'un système informatique, tout le travail s'est architecturé autour d'une solution de développement libre sur Internet. Cette solution est proposé par *gna!* (www.gna.org). Ces derniers proposent un accès CVS de gestion des versions et l'hébergement d'un site internet traitant de l'application. Voici les liens pour accéder à tout ces services :

- Site internet www.digital-scratch.org

- Gestion du site et du code source via CVS :

  En anonyme :
  Sourcecode repository :
  cvs -d:pserver:anonymous@cvs.gna.org:/cvs/dscratch co dscratch
  Webpages repository :
  pserver:anonymous@cvs.gna.org:/cvs/dscratch.homepage co dscratch

Avec un compte sur gna! membre du projet digital-scratch :
Sourcecode repository :
export CVS_RSH=ssh
cvs -d username@cvs.gna.org:/cvs/dscratch co dscratch
Webpages repository :
export CVS_RSH=ssh
cvs -d username@cvs.gna.org:/cvs/dscratch.homepage co dscratch

L'utilisation de la technologie proposée par CVS est très intéressante pour le développement à plusieurs. Ainsi, le code est proposé à quiconque, et si ce dernier le modifie, il place sa version sur le serveur CVS. La dernière version est toujours celle présente sur ce serveur, de plus, le système gère un historique de toute les versions.

6 Implémentation

Voici la phase d'implémentation qui s'attache à justifier le choix de la plateforme, du langage de programmation et du player.

6.1 Choix de la plateforme

Un des premières questions concernant le projet a été la recherche de la plate-forme pour réaliser le projet. Le choix s'est porté entre *GNU/Linux* et *Microsoft Windows,* au moment du premier projet.

Nous avons choisi *GNU/Linux* car ce système d'exploitation véhicule les images que nous voulons apporter au projet : gratuité, libre disponibilité du code source et des binaires de l'application. De plus, la plupart des outils de développement sont disponibles gratuitement et sont souvent bien documentés.

L'autre intérêt de *GNU/Linux* est qu'il est multi-plateforme. Notre projet pourra aussi être utilisé par quelqu'un possédant un PC normal ou bien par un *Macintosh* ou encore d'autres systèmes supportant *GNU/Linux*.

Pour respecter la philosophie de ce système d'exploitation, nous placerons notre projet en licence GPL. Ainsi les personnes intéressées pourront télécharger le code source du programme, et le modifier si elles le souhaitent. Le projet pourra constamment être amélioré.

En ce qui concerne la couche de gestion des cartes son, indispensables à notre projet, nous avons choisi *ALSA* (informations sur le site www.alsa-project.org) : cette couche est intégrée par défaut dans les nouvelles versions de *GNU/Linux* à l'heure où ces lignes sont écrites (noyau 2.6.x).

6.2 Choix du player

Pour le module de lecture des musiques de notre projet, nous avons eu le choix entre plusieurs solutions :

- écrire entièrement un module qui pourrait décoder, lire et modifier la vitesse des morceaux musicaux. Cette hypothèse a rapidement été écartée en raison de la difficulté de réalisation.
- utiliser une application existante et l'interfacer avec notre projet. Ce choix permettrait de nous concentrer sur l'objectif premier qui est de réaliser un mécanisme permettant de détecter la vitesse et ainsi de ne pas « réinventer la roue », surtout lorsque celle-ci a été développée pendant de nombreuses années par des équipes mettant en jeu plusieurs personnes.

Ainsi nous avons choisi d'adopter la seconde solution.

Nous avons ensuite mené des recherches pour déterminer quelle application était la plus intéressante pour notre projet.

Les critères auxquels devait répondre le module de lecture étaient :

- une licence GPL assurant que l'application soit dans le domaine du libre. Ainsi nous pourrons examiner son fonctionnement en étudiant le code source si besoin est pour pouvoir l'intégrer dans notre projet.
- la gratuité, offerte par la licence GPL.
- un projet maintenu régulièrement pour éviter la mauvaise surprise de voir émerger des bugs non traités ou bien encore l'abandon du logiciel qui nous obligerait de changer une partie du projet.
- un âge d'au moins un ou deux an qui donne au projet une base solide.

AlsaPlayer a retenu notre attention : il répond à tous ces critères et bien au-delà.

Il offre en outre la possibilité de lire les cinq ou six formats sonores les plus répandus (en plus du format .mp3 que nous avions choisi initialement). Il permet aussi de lire le morceau musical à l'envers (il est un des seuls au monde à la proposer même parmi les solutions commerciales payantes).

Il dispose aussi d'une interface de contrôle externe permettant à un programme quelconque de le piloter, ce qui sera très utile pour notre programme.

Les informations concernant ce logiciel se trouvent sur le site http://www.alsaplayer.org.

6.3 Langage de programmation

Toute l'application est développée en C++. Ce langage est l'idéal pour gérer le compromis suivant : utiliser les API Alsa et Alsaplayer qui sont en C, mais aussi bénéficier d'un langage orienté-objet utilisant facilement la modélisation du système effectuée. La compilation est effectuée avec *g++*.

7 Manuel d'installation et d'utilisation

7.1 Contenu du package d'installation

Un fichier compressé portant ce nom contient tout le projet, c'est-à-dire :

- les sources du projet
- le fichier d'automatisation de la compilation
- le script de lancement
- un fichier d'aide sommaire

Pour décompresser tous ces fichiers il suffit de taper la commande suivante :

```
tar xvzf digitalScratch-v1.0.tar.gz
```

7.2 La compilation

La compilation permet de créer les fichiers binaires du programme. Elle a été automatisée et la commande `make` permet de lancer cette opération.

Une fois celle-ci terminée, les binaires sont générés et installés dans *bin/*.

Attention : Pour être compilé, le projet requiert plusieurs librairies.

- *libasound* : les fonctions bas niveau *ALSA*
- *alsaplayer-dev* : la librairie d'interfaçage avec *AlsaPlayer*
- *lm* : la libraire mathématique de GNU/Linux

Nota : cette opération de compilation à l'aide de la commande `make` peut être réutilisée en cas de modification du programme pour régénérer les binaires.

Pour l'instant la configuration des cartes sons est écrite directement dans le fichier `main.cpp`, cependant un système de gestion des options de lancement sera mis au point.

8 Evolutions

Le projet n'est pas terminé en terme d'implémentation, il reste des fonctions essentielles à développer comme la gestion du système de positionnement. Cependant, de nouvelles évolutions sont déjà interessantes à évoquer.

8.1 L'interface graphique

Pour l'instant il n'y a pas d'interface graphique, car il est d'abord important de réaliser un noyau solide composé de l'interface avec les cartes sons et *AlsaPlayer* et le système d'analyse du signal. Cependant, voici la maquette d'une petite étude sur les fonctionnalités que devra proposer l'interace.

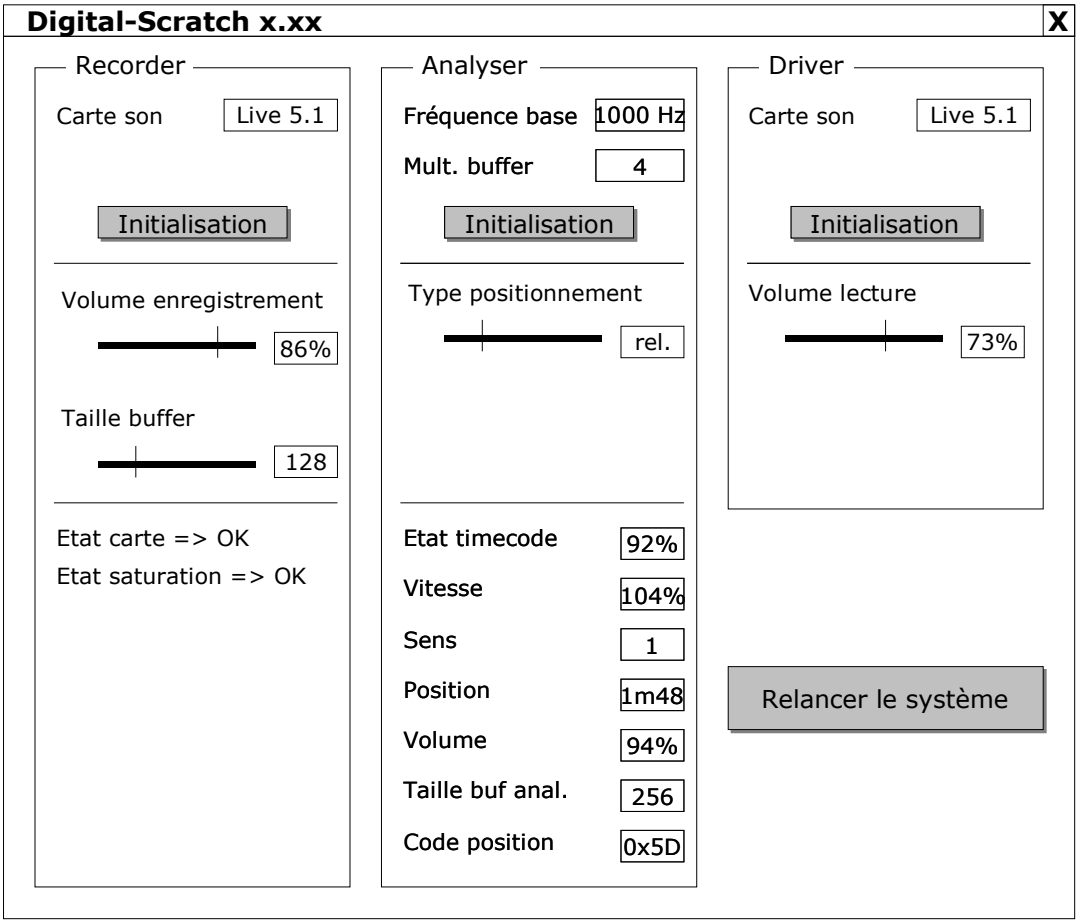


Illustration 24 - Maquette d'interface graphique

8.2 Plugins

Une évolution interressante serait de permettre à Digital-Scratch d'utiliser des plugins. Par exmple un plugin d'effets sur le son du mp3. On pourrait imaginer les fonctions suivantes :

- un equalizer graphique,
- un système de détection des BPM (Beats Per Minute),
- des effets sonores en tous genres,...

9 Conclusion

Le développement de ce projet a été très enrichissant et intéressant car il a touché à plusieurs domaines : le monde de la musique et informatiquement parlant le traitement du signal ainsi que *GNU/Linux*.

Je n'avait jamais fait de projets si conséquents sous GNU/Linux et la multitude d'outils libres qui est offerte est un avantage. De plus réaliser une application de traitement du signal a été une chose nouvelle, il faut concevoir des algorithmes robustes et optimisés en raison du nombre élevé de traitements à effectuer par intervalle de temps. En effet la nature « temps réel » du programme a impliqué une grande rigueur.

La nature mécanique du son inhérente au vinyle a été plus complexe que prévue et a nécessité plusieurs algorithmes de correction : en effet le son qui sort de la platine n'est pas parfait, et la moindre rayure ou la moindre anomalie d'utilisation du vinyle peut modifier le son qui devient quasiment inexploitable. Beaucoup de problèmes sont réglés, toutefois il y a des points à améliorer comme le temps de réactivité en cas de lever et poser de cellule sur le vinyle.

Extrait de la conclusion du rapport précédent :

« Voir notre projet fonctionner et évoluer grâce aux différentes procédures codées a été très motivant. En effet, nous n'avions rarement réalisé de projet aussi concret au sein de l'UTBM et cela restera une excellente expérience que nous conseillons à tous les étudiants. »

10 Bibliographie - Référence

- Alsaplayer : A pcm audio player for Linux and other similar systems
  http://www.alsaplayer.org/
- MixVibes reliable and performance solutions for Djs
  http://www.mixvibes.com/
- SoundGraph D-Vinyl
  http://www.d-vinyl.dj/
- Le site de Final Scratch
  http://www.finalscratch.com/
- Son-vidéo.com, vente en ligne de matériel sono
  http://www.son-video.com
- Le contrôle CRC, un article de Sylvain Nahas
  http://docs.sylvain-nahas.com/crc.html
- Le pseudo code de la recherche d'un CRC (en anglais)
  http://www.ibrtses.com/delphi/dcrc.html
- AMS Standard Cyclic Redundancy Check - CRC16 -Status: Baseline
  http://ams.cern.ch/AMS/Dataformats/node26.html
- Audio Linux Sound Architecture, le site de développement des drivers Alsa
  http://www.alsa-project.org

11 Annexe

11.1 Le contrôle CRC

11.1.1 Introduction

Ce document consiste à expliquer l'intérêt et le fonctionnement du contrôle CRC.
Le CRC (Cyclic Redundancy Code) est un code permettant de vérifier l'intégrité d'un mot formé de plusieurs bits.
Pourquoi vérifier cette intégrité ? Beaucoup de système, notamment informatique, utilisent le transfert de données. Par exemple, un réseau informatique : il s'agit d'envoyer des données d'un point A à un point B. Celles ci sont transférées par « paquet » de plusieurs bits qui renferment l'adresse de l'émetteur, l'adresse du destinataire et bien sûr, les données elles mêmes. Mais comment s'assurer, une fois réceptionnées, que ces données sont conformes à celles qui ont été envoyées ? Plusieurs solutions existent.
La première et la plus simple serait de ré envoyer la trame de données, cependant, cela est bien trop coûteux en taille.
D'autres méthodes consistent à rajouter quelques bits de vérification à la trame. Si ces bits sont le résultat d'une fonction mathématique appliqué aux données, il suffit d'appliquer une fonction inverse à la réception pour déterminer si oui ou non la trame est bien la même que celle envoyé. C'est cette méthode qui nous intéresse ici. Une technique simple est d'utiliser un bit de parité (1 si la trame est paire, 0 si elle est impaire); voilà une méthode peu coûteuse en place mais pas forcément très sûre. De manière un peu plus efficace on peut se rajouter un « cheksum » qui n'est autre que la somme des bits constituant de la trame de données. C'est une solution plus coûteuse mais qui permet de détecter déjà beaucoup d'erreurs.
Nous arrivons maintenant à notre méthode qui consiste à calculer un code de redondance cyclique à partir de la trame des données. Ce code se détermine par des calculs relativement simple basés sur l'arithmétique modulaire. Nous verrons plus tard que la qualité de ses résultats est fonction d'un polynôme dit « générateur ». Selon ce dernier, on peut obtenir des probabilités d'intégrité de la trame proche des 100%.

11.1.2 Rappel sur l'arithmétique sur les nombres binaire

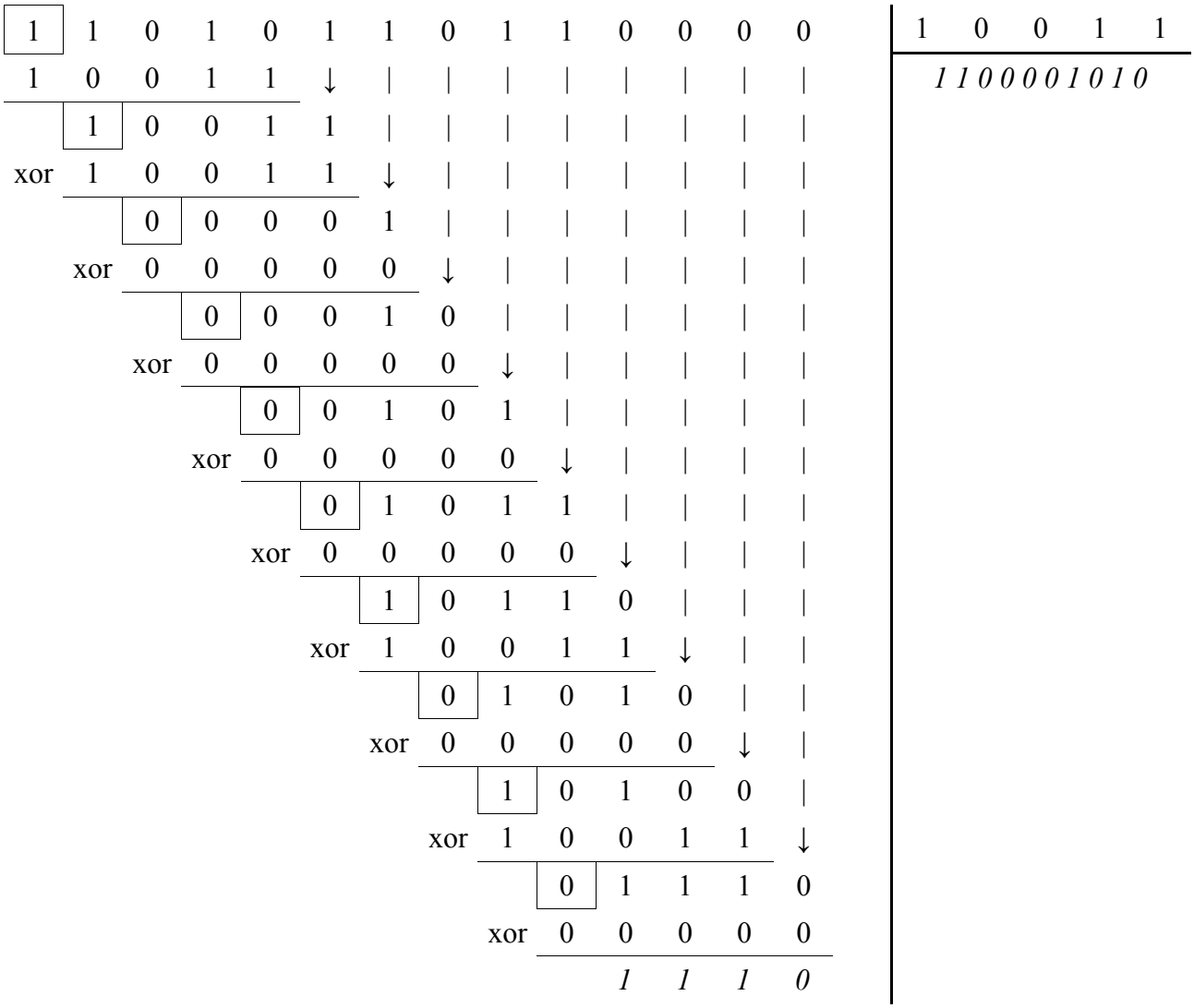
Je ne m'attarderai ni sur les définitions mathématiques, ni sur la théorie de l'arithmétique modulo 2. Cependant quelques petites connaissances dans ce domaine s'impose.

11.1.2.1 L'opérateur OU exclusif (XOR)

Cet opérateur est très utilisé dans l'algorithme de création du CRC, en voici ces caractéristiques :

A	*B*	*A* ***xor*** *B*
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

11.1.2.2 La division entre 2 nombre binaires



Pour ce faire, on sélectionne le bit de poids fort du dividende, ici il vaut 1, il sera le bit de poids fort du quotient. Maintenant, on peut faire (1 x diviseur) = 1 0011. On réalise ensuite un XOR entre le dividende et ce résultat qui est décalé de suffisamment de bit pour commencer au début du dividende. Ensuite, on ajoute au résultat de l'opération le bit suivant correspondant au dividende (on le fait « descendre »).
L'opération s'enchaîne ainsi de suite. On sélectionne le bit de poids fort du résultat précédemment trouvé, ici, c'est un 1. On lui applique donc un XOR avec 1 0011. Le résultat sera 0000 auquel on fera descendre 1, ce qui donne 00001. Ici, le bit de poids fort est un 0, donc (0 x diviseur) = 00000. On refait l'opération avec le XOR.
Ces opérations seront réalisées jusqu'à ce qu'on ne puisse plus « descendre » de bit depuis le dividende, autrement dit, on réalise (taille_du_dividende – taille_du_diviseur + 1) opérations (la taille correspondant au nombre de bits).

11.1.3 Fonctionnement

Le calcul va utiliser le principe de la division polynomiale. Ainsi, on va diviser la trame d'entrée A par un polynôme et donc récupérer le CRC. A l'arrivée on redivisera la trame B complète (A+CRC)

par ce même polynôme. Si le reste est nul, c'est que la trame de départ est la même que la trame d'arrivée.
Le choix du polynôme générateur est fonction de la qualité du résultat du CRC. Sans rentrer dans les détails mathématiques, le plus simple est d'utiliser ceux qui sont définie comme étant de bonne qualité. Les CRC-16 et 32 donne une fiabilité de près de 100% au niveau de l'intégrité de la trame.
Exemples :

Nom	Polynôme générateur
CRC-4	$x^4 + x^2 + x^1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
CRC-16 SDLC (CCITT)	$x^{16} + x^{12} + x^5 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-16 Reverse	$x^{16} + x^{14} + x^1 + 1$
SDLC Reverse	$x^{16} + x^{11} + x^4 + 1$
CRC-32 (ethernet)	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

L'algorithme de base qui renvoi la trame d'origine suivit de son CRC est le suivant :

```
Début
      Décaler, vers la gauche, du degré du polynôme générateur le mot d'entrée
      Faire la division (mot d'entrée / polynôme générateur)
      Faire le calcul (mot d'entrée XOR polynôme générateur)
Fin
```

L'algorithme qu'il faut appliquer à une trame reçu (contenant un CRC) est le suivant :

```
Début
      Faire la division (trame reçu / polynôme générateur)
      Si le reste est nul Alors la trame est correcte
      Sinon retransférer la trame
Fin
```

Voici un exemple de recherche d'un CRC (il réutilise l'exemple de la division vu plus haut). On veut transférer le mot : 11 0101 1011. On décide d'utiliser le polynôme générateur de degré 4 (le degré étant le nombre de bit – 1); ici 10011.
D'abord on décale 11 0101 1011 de 4 rang vers la gauche, résultat : 11 0101 1011 0000.
Ensuite, on cherche le reste de la division de 11 0101 1011 0000 par 10011, résultat : 1110 (c'est le FCS pour **F**rame **C**ontrol **S**equence, nommé par abus de langage CRC).
Enfin on applique un XOR entre le mot d'entrée (décalé de 4 rang) et le reste de la division, 11 0101 1011 0000 XOR 1110 = 11 0101 1011 1110, ce qui correspond bien au mot de départ « suivi » du CRC.

On transfert cette trame. Si on la récupère telle quelle, on obtient 11 0101 1011 1110. On divise ceci par 10011, et on trouve un reste nul, le transfert s'est donc correctement déroulé, aucune perte de données n'est à déplorer.

11.1.4 Implémentation en C

Voici une implémentation en langage C du problème. On trouve 3 fonctions :

- `int get_code_crc(int mot, int polynome_generateur)` qui renvoie le mot passé en paramètre suivit de son CRC a partir du polynôme générateur (en hexadécimal).
- `int reste_division_ou_exclusif(int dividende, int diviseur)` qui renvoie le reste de la division du dividende par le diviseur.
- `int degre_int(int mot)` qui renvoie le degré d'un mot binaire (max 32 bits).

```
//-----------------------------------------------------------------------------
// Recupere le nombre de bit (degre) d'un mot (short int)
int degre_int(int mot)
{
	int degre = 0;
	int i     = 0;

	for (i = 0; i < (sizeof(int)*8); i++)
	{
		if ((mot & 0x0001) == 1) degre = i+1;
		mot = mot >> 1;
	}

	return degre;
}

//-----------------------------------------------------------------------------
// Donne le reste de la division entre 2 nombre binaire (division utilisant
// les OU exclusifs)
int reste_division_ou_exclusif(int dividende, int diviseur)
{
	int i          = 0;
	int tdividende = 0;
	int tdiviseur  = 0;
	int div_interm = 0; // diviseur intermedaire
	int reste      = 0;

	// recherche des degres des operandes
	tdividende = degre_int(dividende);
	tdiviseur = degre_int(diviseur);

	// recherche du reste de la division
	div_interm = diviseur;
	reste = dividende >> (tdividende - tdiviseur);

	for (i = (tdividende - tdiviseur+1); i > 0; i--)
	{
		reste = reste ^ div_interm;

		if (i > 1)
		{
			reste = reste << 1;
			reste = reste | ((dividende >> (i-2)) & 0x0001);
			if (((reste >> (tdiviseur-1)) & 0x0001) == 0) div_interm = 0;
			else div_interm = diviseur;
		}
	}

	return reste;
}


//-----------------------------------------------------------------------------
// A partir d'un mot binaire (short int), renvoie ce mot suivit de son CRC
// defini par un polynome generateur (ex : 0x15 = X4 + X2 + 1)
```

```
int get_code_crc(int mot, int polynome_generateur)
{
      int crc = 0;

      // decaler le mot vers la gauche du degre-1 du polynome generateur
      mot = mot << (degre_int(polynome_generateur)-1);

      // chercher le reste de la division entre le mot et le polynome generateur
      crc = reste_division_ou_exclusif(mot, polynome_generateur);
      // concatenation du mot et du polynome generateur
      mot = mot ^ crc;

      return mot;
}
```

En reprenant l'exemple précédent, voici comment utiliser ce programme :

```
int main (int argc, char *argv[])
{
      int code = get_code_crc(0x35B, 0x15);
      printf("code + crc : %X  \n", code);


      return 0;
}
```

TX52 – Pilotage de lecture de fichier audio numérique par platine vinyle

---

Etudiant Julien Rosener, GI05, ILC (julien.rosener@utbm.fr)

Suiveur Nicolas Lacaille (nicolas.lacaille@utbm.fr)

Semestre Printemps 2004

Résumé

Digital-Scratch est une application de pilotage de mp3 par platine vinyle. Le principe est d'utiliser un disque vinyle contenant un timecode (son crypté) analysé par un PC qui se charge de lire un mp3 à la vitesse, au sens de rotation et à l'emplacement appliqué par la platine.

L'application tourne sous GNU/Linux équipé de l'API ALSA (Advanced Linux Sound Architecture) et utilise le lecteur *AlsaPlayer* pour lire tous les formats audio actuels. Ce dernier est entièrement pilotable via *Digital-Scratch*.

Ce projet très technique (traitement du signal, contrainte de temps réel, algorithmes optimisés pour une exécution rapide) a été très enrichissant et intéressant.

Le logiciel est développé sous la licence GPL qui le place dans le domaine du logiciel libre ce qui permet d'étendre la communauté de développeurs. Ainsi, *Digital-Scratch* aura la possibilité d'évoluer sans cesse. Le projet est développé en C++.

Mots clés

- GNU/Linux
- API ALSA
- Timecode
- Musique
- Vinyle
- Mp3
- API Alsaplayer
- C++