## Equalizer

### Qu’est-ce qu’un Equalizer ?

Un égaliseur (equalizer, equaliser ou "EQ" en anglais) ou correcteur de timbre est un appareil ou logiciel de traitement du son. Il permet d'atténuer ou d'accentuer une ou plusieurs bandes de fréquences composant un signal audio.

Une source sonore est composée d'une multitude d'ondes sonores réparties sur un large spectre de fréquences audio. L'être humain adulte perçoit ainsi, de 20 Hz à 20 kHz. Il est possible de distinguer et isoler certaines bandes de fréquences afin de leur appliquer un traitement spécifique. En agissant sur une plus ou moins large gamme de fréquences : son graves, médium, aigus, la correction permet d'atténuer ou au contraire de renforcer le timbre du son. À la différence des correcteurs les plus souvent rencontrés sur les amplificateurs Hifi, autoradios, téléviseurs, etc..., les égaliseur interviennent sur des bandes précises et sont généralement plus performants, en particulier pour renforcer certaine fréquences sans trop générer de nuisances (souffle, bruit de fond, saturation, distorsion...).

Plusieurs types d'égaliseurs sont exploités. Le correcteur le plus simple et le plus répandu exploite 2 ou 3 filtres - grave, médium, aigus- pour traiter sélectivement la bandes. Ils sont présents sur les amplificateurs hifi, les autoradios, les guitares électriques ou encore sur les tables de mixage. Les dispositifs électroniques de correction plus perfectionnés peuvent être assimilés à la gamme des égaliseurs dits "graphiques", adjectif symbolisant l'effet produit lorsqu'on analyse le son : l'impact graphique en bosse ou en creux est révélateur du traitement : chaque bande de fréquence impacte directement la courbe sonore produite.

### Choix de la base de travail

Une fois le concept de fonctionnement d’un égaliseur assimilé, nous nous sommes mis à la recherche d’exemples concrets dans le langage de programmation que nous utilisons pour notre projet. Nous sommes finalement tombés sur un site de partage de code Japonais. Bien évidemment, Java est un langage universel et nous n’avons donc pas rencontrés de problème pour comprendre le code et tester celui-ci afin de vérifier qu’il correspondait bien à nos attentes (<http://www.189works.com/article-9543-1.html>). Après des essais en interne, nous avons finalement décidé que le programme était satisfaisant et nous l’avons donc sélectionné comme base de travail.

### Fonctionnement

Les principales fonctionnalités de l’égaliseur sont assurées par deux classes de l’API Android.

* Le package AudioEffect permet la manipulation d’effets sonores offerts par le framework audio d’Android et propose ainsi plusieurs classes pour y parvenir, dans le cadre de notre application on utilisera **android.media.audiofx.Equalizer**. Comme son nom le laisse penser, la classe Equalizer fournit des méthodes pour agir sur la fréquence d’un signal audio.

Une fois initialisé avec le signal audio à modifier (ici la musique qui est en cours de lecture par notre application), on récupère la quantité de bandes de fréquence (plage de fréquence) supportée par le morceau. Pour chacune de ces bandes on récupère alors la fréquence centrale et on lui associe une seekbar (barre de modification) variant entre sa valeur minimale et sa valeur maximale en décibels, permettant ainsi de jouer avec le morceau et d’amplifier les aigus ou les graves selon l’envie.

* Le deuxième composant fait ici figure de gadget : il s’agit du **Visualizer**. Cette classe permet de récupérer les informations de fréquence d’une bande sonore afin d’y associer un effet visuel simple.

### Implémentation

A l’origine, cette application fonctionne de façon indépendante mais dans la cadre de notre projet il a bien évidemment fallu l’inclure et l’associer aux différentes classes existantes. C’est pourquoi, l’activité Equalizer n’est lancée que lorsque l’utilisateur le demande par le biais de l’interface. La piste en cours de lecture lui est alors transmise et les modifications apportées par l’utilisateur ont un impact direct sur le morceau en cours d’écoute. Les valeurs de l’Equalizer sont ensuite réinitialisées à chaque changement de musique.

## TimeStretching

### Concept

Le time stretching est un effet audio numérique qui a pour but de raccourcir ou allonger la durée et/ou le tempo d’un échantillon sonore sans modifier sa hauteur tonale. C’est le procédé réciproque du « Pitch Shifting » qui consiste à changer la hauteur d’un signal sonore sans en modifier sa longueur.

### Fonctionnement

Il existe diverses méthodes pour effectuer du time-stretching sur un signal sonore, toutes ces méthodes utilisent des concepts de physique et de manipulation de signal audio avancés mais nous allons tout de même tenter d’expliquer le principe de fonctionnement de la méthode à base de « Phase Vocoder ».

**Phase vocoder**

L’utilisation d’un phase vocoder (vocodeur de phase en français) est une technique complexe faisant appel à des éléments de CPL (codage prédictif linéaire – une méthode d’analyse/synthèse développée dans le cadre des technique d’encodage, de transmission et de réception de la parole). Elle utilise les transformées de Fourrier en continu.

### Implémentation

Comme expliqué précédemment, les notions abordées par cette fonctionnalité sont très compliquées et il nous a été impossible d’inclure celle-ci dans notre projet. Le principal problème a été le manque d’exemples concrets disponibles en Java, en effet il ne semble exister aucun code open-source dans ce langage de programmation, les logiciels comme Audacity (logiciel libre pour la manipulation de données) ne sont disponibles qu’en C++.

Il existe néanmoins sur la plate-forme Android une application (Audio Speed Changer) qui inclut cette fonctionnalité. A des fins pédagogiques nous avons donc réalisés de la retro-ingénierie (reverse engineering) sur cette application afin d’en apprendre d’avantage.

### Reverse Engineering

Le reverse engineering est l’activité qui consiste à étudier un objet pour en déterminer le fonctionnement interne ou la méthode de fabrication (<http://fr.wikipedia.org/wiki/R%C3%A9tro-ing%C3%A9nierie>).   
Notre but n’était évidemment pas de nuire au développeur et à son application mais bien de comprendre son implémentation !

Notre méthode fut la suivante :

* Récupération de la version gratuite de l’application (.apk, format des applications installables Android) sur internet.
* Transformation du fichier .apk en .zip et ouverture de ce dernier. On y trouve tous les fichiers qui composent notre application et en particulier le fichier classes.dex qui représente la version compilée de l’application.

[screen1] légende : Contenu d’un fichier .apk

* Transformation du fichier .dex en .jar grâce au programme dex2jar (<http://code.google.com/p/dex2jar/>).
* Le .jar généré contient alors toutes les fichiers .class de l’application et il est désormais possible de les lire grâce à un utilitaire tel que JD-GUI (<http://java.decompiler.free.fr/>) qui reconstruit les fichiers .java associés.

[screen2] Légende : Interface une fois le projet ouvert avec JD-GUI

Malheureusement en fin de compte nous nous retrouvons devant un code **obfuscated**, c’est-à-dire qu’il a été rendu illisible à l’œil humain par le développeur lui-même, empêchant ainsi toute analyse.

Après contact avec le développeur, celui-ci nous a conseillé d’abandonner l’idée d’implémenter cette fonction si nos connaissances en manipulation du son n’étaient pas assez importantes, tout en ajoutant que si nous décidions de persister il serait plus aisé de travailler en langage C ou C++, langages que personne dans notre groupe ne maitrise assez. C’est donc avec regret que nous laissons de côté le time-stretching.

Les références pour le time stretching:

http://www.daimi.au.dk/~pmn/sound/timestretch/paper-071002A.doc

http://www.engr.uvic.ca/~kzhang/theproject1.html

http://en.wikipedia.org/wiki/Audio\_timescale-pitch\_modification

http://my.fit.edu/~vkepuska/ece3551/The%20DSP%20Dimension/Tutorials/TimeStretchingOverview.pdf

http://www.dspdimension.com/admin/pitch-shifting-using-the-ft/