Introduction aux effets audio

Option audionumérique - ensea
S. Reynal

25 janvier 2017

Ce TP est une introduction à quelques aspects du traitement du signal audio :

- mise en forme du signal destinée à en optimiser la dynamique (noise-gate / compresseur) et l'occupation spectrale (EQ)
- spatialisation du signal (panning, ajout d'ambiance réverbérante)

Le développement sera réalisé, selon votre goût :

- soit avec l'API Java Sound pour un traitement temps réel (acquisition via le microphone - traitement - restitution immédiate sur la sortie audio),
- soit sous Matlab pour un traitement en temps différé (quelques fonctions matlab utiles en traitement audio : `wavread, wavwrite, spectrogram, colorera, mp3read`). On pourra alors appliquer le traitement, soit à un fichier MP3, soit à un enregistrement réalisé indépendamment avec un micro directement pendant la séance.

1 Echantillonage et quantification

Il s'agit ici de tester quelques conséquences audibles du sous-échantillonage et du bruit de quantification... L'idée clé est de lire un signal audio (enregistré par vos soins via un micro ou bien téléchargé depuis internet ou votre clé USB... etc etc), puis de :

- le sous-échantillonner : pour cela, vous allez décimer le signal, c'est-à-dire, construire un nouveau signal à partir d'un échantillon pris tous les N échantillons du signal original ;
- ou bien de réduire le nombre de bits de quantification : pour cela, la méthode la plus simple consiste à arrondir la valeur de chaque échantillon au nombre sur n bits le plus proche (par exemple en utilisant le "et logique" de matlab ou de java pour mettre à 0 les bits de poids faible).

le sous) Dans les deux cas, il faut écouter attentivement !

MATLAB : écrire une fonction `xs=underSample(xe, N)` puis `xs=bitCruch(xe,N)`.

JAVA : dans la boucle infinie de traitement, réaliser un arrondi par mise à zéro des bits de poids faible. Quant au sous-échantillonnage, le plus simple est de mettre à la même valeur N échantillons successifs.

2 Réalisation d'une noise-gate

Il s'agit habituellement de la première étape de traitement d'un signal audio enregistré en condition réelles, c'est-à-dire avec du bruit ambiant. Le noise-gate est un effet qui coupe (=réduit au silence) les parties du signal se trouvant en-dessous d'un certain niveau, dans le but de supprimer le bruit. L'idée clé est qu'en présence de signal utile (parole ou musique), le bruit est généralement masqué (cf. cours psychoacoustique sur le masquage fréquentiel), et l'auditeur ne le perçoit pas ; en revanche il est clairement audible lors des plages de silence musical ou de pause entre deux phrases. Lorsque le signal sera mélangé à d'autres signaux lors de l'étape de mixage, il est souvent primordial (sauf raisons artistiques particulières) que le moins de bruit ambiant subsiste. On dit que la porte est ouverte lorsque le signal passe tel quel au travers du traitement ; qu'elle est fermée si le signal est réduit (ou atténué).

MATLAB : Ecrire une fonction `xs=noiseGate1(xe, seuil, attenuation)` qui prend le signal audio sous forme d'un vecteur en entrée xe, et restitue le signal après application d'une noise gate très simplifiée, consistant à atténuer d'une quantité fixée (en dB) les échantillons se trouvant sous le seuil (paramètre *seuil*). Il est intéressant de normaliser le signal avant de le traiter afin de s'affranchir des problèmes de rapport niveau/seuil.

JAVA SOUND : dans la boucle infinie de traitement, atténuer d'une quantité fixée (en dB) les échantillons se trouvant sous le seuil (paramètre *seuil*).

Tester l'effet de différents seuils et différentes atténuations (ne pas se contenter de spectrogrammes, mais écouter !!!). Commentaires ?

Améliorations possibles Une noise gate améliorée dispose de paramètres dynamiques supplémentaires (cf. figure 1) :

- attack time : c'est le temps que met la porte à s'ouvrir lorsque le signal dépasse le seuil
- holding time : c'est le temps minimal pendant lequel la porte reste ouverte (même si le signal passe sous le seuil) ; ceci permet notamment d'éviter les effets de hachage, la porte ne pouvant pas se fermer tant que ce temps n'est pas écoulé.
- release time : c'est le temps (supérieur au précédent, donc) que met la porte à se fermer lorsque le signal passe sous le seuil (à condition que le holding time ait été atteint).

3 Réalisation d'une spatialisation par délai et réverbération

Dans cette partie vous devez implémenter un algorithme de spatialisation à partir de délais et de réverbération. L'objectif est de donner une impression subjective de spatialisation, comme si la voix avait été enregistrée dans une pièce à l'acoustique réverbérante.

Implémentation d'un délai : le signal d'entrée est retardé, puis réinjecté à l'entrée après atténuation. Les paramètres sont :

— le retard, en mille-secondes
— le feedback, ou taux de réinjection (atténuation de la boucle de réaction).

Testez plusieurs valeurs de délai, de quelques ms à quelques centaines de ms, et commentez sur la perception spatiale de l'enregistrement. Testez également la superposition de deux ou trois délais avec des paramètres très différents entre eux.

L'implémentation de la réverbération qui vous est demandée repose sur l'algorithme de convolution par transformée de Fourier. Des réponses impulsionnelles se salles typiques sont fournies sur mon site (fichiers SDIR). La sortie de la réverbération est obtenues en convoluant le signal d'entrée $x(n)$ avec la réponse impulsionnelle de la salle $h(n)$: $y = x * h$. Pour accélérer le calcul dans la version temps réel sous JAVA SOUND (temps qui est proportionnel au carré de la longueur N de la réponse impulsionnelle), on passe dans le domaine fréquentiel : $Y = X \cdot H$. Il est utile de calculer la transformée de Fourier de h une fois pour toute, et celle de x, par FFT (algorithme en temps $N \ln N$).

JAVA SOUND : On pourra utiliser une des nombreuses bibliothèques de transformée de Fourier rapide (FFT) disponible sur internet.

Remarque : attention au choix du fenêtrage (hamming, blackman, hanning,...) lors de l'application de la FFT. Attention également au fait que le signal réverbéré est plus long que le signal original (de N exactement).

4 Réalisation d'un égaliseur

Cette ultime partie consiste à réaliser un égaliseur sous forme de filtres à réponse impulsionnelle infinie. Ce système est composé de :

— un filtre passe-haut non-résonant à 4 pôles, et l'équivalent en filtre passe-bas ;
— un filtre "shelf", équivalent au filtre analogique de fonction de transfert

$$T(p) = \frac{1 + \tau_1 p}{1 + \tau_2 p}$$

— un filtre paramétrique passe-bande, de fréquence centrale f_0 et de coefficient de qualité Q, tous deux variables.

Exemple de filtre passe-bas d'ordre 4 coupant à $f_e/10$:

$$y[n] = (1 * x[n-4]) + (4 * x[n-3]) + (6 * x[n-2]) + (4 * x[n-1]) + (1 * x[n-])$$

$$+(-0.85 * y[n-4]) + (3.53 * y[n-3]) + (-5.52 * y[n-2]) + (3.84 * y[n-1])$$