Traitement du Signal Temps Réel

Sujet BE: Autotune et Réverbation

Olivier Perrotin & Thomas Hueber

DURÉE : $4 \times 4\,\text{h}$
DATE DE RENDU : 23 Janvier 2024

Introduction

1 But du BE

Le cours magistral ($2 \times 2\,\text{h}$) a abordé différents aspects liés à la conception d'un « système temps-réel » :

- Définition(s) d'un système temps-réel ($\neq$ système à exécution rapide)
- Modèles théoriques de conception (synchronous/scheduled, etc.)
- Choix du hardware (DSP, GPU, etc.)
- Systèmes d'exploitation (OS standards, rôle de l'ordonnanceur, etc.)
- Techniques d'implémentation logicielle (bonnes pratiques, risque d'inversion de priorité, etc.)
- Audio sur PC (« mille-feuille » logiciel, API audio, etc.) et traitement audio temps-réel (modèle producteur-consommateur, buffer underrun/overrun, overlap-add, buffer circulaire).

Un focus particulier a été mis sur les systèmes dits « soft », destinés à une implémentation sur un OS standard. Dans le cadre du BE TSTR ($4 \times 4\,\text{h}$), nous vous proposons de mettre en pratique certains des points abordés dans le cours, au travers d'un des projets suivants, **au choix** :

1. implémentation d'un effet audio de type *autotune* par synthèse additive
2. implémentation d'un effet audio de type *réverbération* à convolution

2 Ressources

Les ressources du BE sont dans le dossier `BE_tstr_2023_v1` sur Chamilo. Vous y trouverez :

- L'API `RtAudio` `rtaudio-4.1.1.tar`[1]
- Un répertoire `c/` contenant des fonctions en C qui vous seront utiles, et qui sont décrites en Annexe 2.
- Un répertoire `python/` contenant un script Python permettant de tracer des signaux enregistrés en format binaire par votre programme.
- Un répertoire `impulse_response/` qui contient un exemple de réponse impulsionnelle en format binaire (pour le sujet 2).

1. source et doc : `https://www.music.mcgill.ca/~gary/rtaudio/`

3 Evaluation

Vous effectuerez une démonstration de votre programme en 5 min lors de la seconde partie de la 4e séance. Vous fournirez un fichier ZIP (nommé `nom1_nom2_tstr_projet_2023.zip` (remplacer « `projet` » par `reverb` ou `autotune`), à envoyer à olivier.perrotin@grenoble-inp.fr) contenant :

- Un `README` précisant la marche à suivre pour compiler et exécuter votre programme
- Un répertoire `bin/` avec les différents exécutables
- Un répertoire `src/` contenant les sources (nettoyées et commentées) nécessaires à le compilation des différents exécutables (et le Makefile associé)
- Un répertoire `files/` contenant les enregistrements effectués, et dont le nom de chaque fichier est explicite (par exemple `Q11_in` pour enregistrement de l'entrée pour la question 11) et est proprement référencé dans le rapport.
- Un rapport *court* (max 4 pages, nommé `nom1_nom2_tstr_projet_2023.pdf` – remplacer « `projet` » par `reverb` ou `autotune`) détaillant vos choix d'implémentation et les résultats obtenus. Les instructions pour lesquelles une réponse est attendue dans le rapport (réponse à une question, tracé d'une figure, etc.) sont indiquées en bleu dans les consignes suivantes. Vous êtes encouragés à y indiquer aussi les divers problèmes rencontrés.

Ces projets seront implémentés en standard C (ou C++) sous Linux, **sur les machines de l'école**, en s'appuyant sur l'API `RtAudio` développée par G. Paul Scavone. Une implémentation sous Windows ou MacOS, sur vos machines personnelles, est possible mais nous n'aiderons pas sur les problèmes inhérents à l'utilisation de ces OS.

> Dans votre rapport, accordez un soin particulier à *l'orthographe*, ainsi qu'aux *figures* (titres des axes explicites, texte lisible, usage efficace des couleurs, etc.).

Prise en main de l'API RtAudio

1 Fonctionnement de base de l'API

1. Compilez RtAudio sur Linux en suivant les étapes suivantes
   a) Téléchargez l'API `RtAudio` `rtaudio-4.1.1.tar` et décompressez-là dans le répertoire de votre choix.
   b) Compilez la librairie principale (`autoconf` puis `./configure` puis `make all`).
   c) Compilez les programmes d'exemples (`cd tests` puis `make all`).
2. Compilez et lancez `duplex`, aidez-vous de `audioprobe` si nécessaire. Que font ces deux programmes ?
3. Étudiez le code source `duplex.cpp` (vu en cours) qui permet la gestion d'un flux audio E/S :
   a) Identifiez les paramètres de configuration du flux audio (fréquence d'échantillonnage, nombre de buffers internes, taille des buffers, etc.).
   b) Identifiez la fonction de callback audio `inout()` et son prototype. Que fait cette fonction par défaut ? elle copie le in dans le out. On entend simplement notre propre voix.
   c) Pour faire passer plusieurs paramètres du programme principal (`main`) à cette fonction de callback audio, il est possible d'utiliser des structures de données. Implémentez cette solution pour faire passer à la fois la taille du buffer et la fréquence d'échantillonnage à `inout()`.
   d) Par la suite, nous travaillerons avec des doubles. Configurez l'API `RtAudio` pour fonctionner avec ce format (cf. flag `RTAUDIO_FLOAT64`).

2 Écriture d'un fichier – observer l'évolution des variables

Il existe peu d'outils simples en C qui permettent d'observer et tracer l'évolution des variables à un instant donné. Une solution est d'écrire les variables dans un fichier pendant l'exécution, puis de les tracer avec un programme annexe. La fonction `write_buff_dump()` qui vous est fournie (cf. Annexe 2.2) permet de remplir progressivement un buffer appelé `buffer_dump` avec la variable ou le tableau de votre choix au cours de l'exécution du programme.

6. Initialisez le tableau `buffer_dump` dans votre programme, avec la taille de votre choix. Fonctions utiles : `malloc()`, `calloc()`, `free()`. Où faites-vous cette initialisation et pourquoi ? avant l'appel de inout()
7. Copiez le buffer d'entrée de la fonction `inout()` dans le buffer `buffer_dump` au fur et à mesure de l'exécution en utilisant `write_buff_dump()`.
8. Écrivez le buffer `buffer_dump` sur le disque dans un fichier binaire. Fonctions utiles : `fopen()`, `fwrite()`, `fclose()`. Où placez-vous l'écriture de `buffer_dump` dans le programme, et pourquoi ?
9. Le script « `python/plot_dump.py` » permet de lire le fichier binaire écrit et de tracer les données. Adaptez le nom du fichier à lire dans le script, et personnalisez l'affichage de la figure selon vos besoins. Ce même script écrit le signal sur le disque en format `.wav` afin de vous permettre de l'écouter, dans le cas où le signal enregistré est audio.
10. Répétez la procédure avec un deuxième buffer pour aussi écrire le signal de sortie de la fonction `inout()`.

> Pour chaque lancement du programme, l'entrée et la sortie du programme sont maintenant écrites dans des fichiers.
>
> - Dans la suite du BE, il vous sera régulièrement demandé de joindre ces fichiers pour vérifier votre implémentation. Il vous sera aussi parfois demandé d'écrire des variables intermédiaires de votre programme pour les observer.
> - Au-delà des consignes, il est aussi vivement encouragé d'écrire et observer les variables de votre programme qui vous semblent nécessaire pour vous aider dans le débugage de votre code.

SUJET 1 : Effet *Autotune* par synthèse additive

Dans ce BE, vous implémenterez une version temps-réel en C/C++ de l'autotune en partant du code source `duplex.cpp` (si vous êtes à l'aise avec les MakeFile, vous pouvez si vous le souhaitez créer votre propre exécutable « `autotune` » en adaptant légèrement le fichier `tests/Makefile`. Sinon, travaillez directement dans `duplex.cpp`). Dans le fichier « `c/somefunc.cpp` » de l'archive `BE_tstr_2023_v1`, vous trouverez un ensemble de fonctions destinées à vous faciliter l'implémentation, listées en Annexe 2.

1 Analyse du signal

Fréquence fondamentale : Nous avons vu en cours qu'une méthode d'estimation de f_0 utilise le deuxième maximum de l'auto-corrélation du signal à analyser.

11. Implémentez la fonction d'auto-corrélation sur le buffer d'entrée de `inout()` puis extrayez-en le deuxième maximum pour obtenir f_0. Vous pouvez vous aider de l'algorithme 2 fourni en Annexe 1. Tracez f_0 mesuré en fonction du temps avec `plot_dump.py` après l'avoir écrit dans un fichier. Joignez au tracé de f_0 le fichier contenant le signal d'entrée correspondant.
    a) Quelle est la limite de fréquence basse que vous pouvez mesurer ? Que faire pour mesurer des fréquences plus basses ?
    b) Quelle est la limite de fréquence haute que vous pouvez mesurer ? Rajoutez une limite dans votre code.
12. Implémentez un buffer circulaire sur lequel vous effectuerez l'estimation de f_0. Tracez la courbe de f_0 obtenue avec différentes tailles de buffer et commentez. Joignez à chaque fois le fichier contenant le signal d'entrée correspondant.

Analyse harmonique : Pour effectuer une synthèse additive, il est nécessaire de connaître les fréquences et amplitudes des harmoniques à générer.

13. Soit `n_fft` le nombre de points sur lequel vous calculez une Transformée de Fourier Discrète (TFD). Quelle est la valeur du bin de la TFD le plus proche de f_0 mesuré ?
14. Calculer la TFD du signal (cf. Annexe 2.3) et extrayez les valeurs de fréquence et d'amplitude pour chaque harmonique. La TFD d'un cosinus calculée par FFT vous donnera deux pics d'amplitude `n_fft/2`. Quelles amplitudes attendez-vous en théorie ? Déduisez-en la correction à apporter à la valeur fournie par la FFT pour obtenir l'amplitude réelle de la TFD.

2 Synthèse du signal

Première implémentation : La synthèse additive génère un signal par l'addition de cosinus d'amplitudes et fréquences correspondant à chaque harmonique du signal à reconstruire (cf. équation dans le cours).

15. Implémentez la synthèse additive à partir des fréquences et amplitudes des harmoniques uniquement et envoyez le résultat sur le buffer de sortie. Écrivez le signal de sortie dans un fichier et joignez-le au rapport. Tracez ensuite ce signal de sortie et mesurez à la main sur le signal le f_0 perçu. D'où vient ce phénomène ?

> Par défaut, les valeurs -1 et 1 du buffer de sortie correspondent au niveau maximal d'amplitude en valeur absolue de votre carte son. Pour protéger vos oreilles :
> - Vérifiez à ne pas envoyer des valeurs allant au-delà de cet intervalle dans le buffer de sortie.
> - L'amplitude [-1,1] peut être très forte en fonction de votre matériel sonore. Au premier lancement de votre programme, attendez d'entendre le niveau sonore avant de mettre le casque sur vos oreilles.

Ajout de la phase : L'information de phase permet de rendre cohérentes les trames de signal contiguës.

16. Rappelez l'équation de la phase de l'harmonique de la trame courante en fonction de la fréquence et la phase de l'harmonique de la trame précédente.
17. Inclure la phase dans la synthèse additive. Écrivez le signal de sortie dans un fichier et joignez-le au rapport. Tracez ensuite ce signal de sortie et commentez-le. Les transitions entre chaque trame sont-elles parfaites ?

3 Autotune

Entre l'analyse et la synthèse, il vous est maintenant possible de modifier la valeur de f_0 (et des harmoniques) pour réaliser l'effet de votre choix. L'autotune consiste à arrondir f_0 à des valeurs discrètes sur une gamme en demi-tons (gamme musicale occidentale). Un demi-ton correspond à $1/12$ d'une octave sur une échelle logarithmique. Le passage de f_0 en `Hz` en à f_0 en demi-tons (`ST`) et inversement se fait par :

$$f_{0ST} = 12 \log_2 (f_{0Hz}) \tag{3.1}$$

$$f_{0Hz} = 2^{\frac{f_{0Hz}}{12}} \tag{3.2}$$

18. Implémentez un autotune en passant f_0 en `ST`, en arrondissant la valeur, et en repassant en `Hz`. Pour des effets plus forts, arrondissez à 3, 4 ou 6 demi-tons.

SUJET 2 : Effet *Réverbération* à convolution

Dans ce BE, vous implémenterez une version temps-réel en C/C++ d'un effet de réverbération en partant du code source `duplex.cpp` (si vous êtes à l'aise avec les MakeFile, vous pouvez si vous le souhaitez créer votre propre exécutable « `reverb` » en adaptant légèrement le fichier `tests/Makefile`. Sinon, travaillez directement dans `duplex.cpp`). Dans le fichier « `c/somefunc.cpp` » de l'archive `BE_tstr_2023_v1`, vous trouverez un ensemble de fonctions destinées à vous faciliter l'implémentation, listées en Annexe 2.

1 Lecture d'un fichier – chargement de la réponse impulsionnelle

L'effet de réverbération exploite un enregistrement de la réponse impulsionnelle d'un environnement acoustique (une salle par exemple). Dans le répertoire « `impulse_response/` » de l'archive `BE_tstr_2023_v1`, vous trouverez un fichier intitulé « `impres` » contenant les échantillons d'une réponse impulsionnelle, en format binaire (sans en-tête) au format double (64 bits), et donc facilement lisible avec `fread()`.

11. Lisez la réponse impulsionnelle en vous aidant de l'algorithme 1 en Annexe 1. Fonctions utiles : `fopen()`, `fread()`, `fseek()`, `ftell()`, `fclose()`.
12. Où placez-vous la lecture de la réponse impulsionnelle dans le programme, et pourquoi ?

2 Implémentation dans le domaine temporel

13. Implémentez la convolution d'une trame dans le domaine temporel. Vous pouvez vous aider de l'algorithme 3 de l'Annexe 1.
14. Implémentez l'overlap-add pour effectuer la convolution du signal en temps-réel. Écrivez le signal de sortie dans un fichier et joignez-le au rapport. Commentez le résultat obtenu.
15. Mesurez le temps d'exécution du callback audio à l'aide de la fonction `get_process_time()` (cf. Annexe 2.1). Quels paramètres du flux audio ont une influence sur le temps d'exécution du callback ?
16. Ajustez les paramètres du flux audio et tracez l'évolution du temps d'exécution du callback en fonction de la variation de chaque paramètre. Pour chaque mesure, vous enregistrerez les signaux d'entrée et de sortie que vous joindrez au rapport.
17. Quelles valeurs de paramètres permettent les meilleures performances ? Cela est-il acceptable ?

> Par défaut, les valeurs -1 et 1 du buffer de sortie correspondent au niveau maximal d'amplitude en valeur absolue de votre carte son. Pour protéger vos oreilles :
>
> - Vérifiez à ne pas envoyer des valeurs allant au-delà de cet intervalle dans le buffer de sortie.
> - L'amplitude [-1,1] peut être très forte en fonction de votre matériel sonore. Au premier lancement de votre programme, attendez d'entendre le niveau sonore avant de mettre le casque sur vos oreilles.

3 Implémentation dans le domaine fréquentiel

18. Implémentez la convolution dans le domaine fréquentiel en vous aidant des fonctions présentées en Annexe 2.3. Quelle doit être la taille de la Transformée de Fourier Discrète ? Pourquoi ?
19. Répliquez les mesures de temps d'exécution. De nouveau, ajustez les paramètres du flux audio et tracez l'évolution du temps d'exécution du callback en fonction de la variation de chaque paramètre. Pour chaque mesure, vous enregistrerez les signaux d'entrée et de sortie que vous joindrez au rapport.
20. Quelles valeurs de paramètres permettent les meilleures performances ? Concluez sur les performances du système.

Annexes

1 Quelques algorithmes

Algorithm 1: Lire un fichier binaire

```
// Ouvrir un fichier
Ouverture avec fopen()
// Calculer sa taille
Mettre le pointeur de fichier à la fin du fichier avec fseek()
Lire la position du pointeur de fichier avec ftell()
Remettre le pointeur de fichier au début du fichier avec fseek()
// Lire un fichier
Lecture avec fread()
```

Algorithm 2: Demie auto-corrélation (indices positifs)

```
// Pour chaque élément de la corrélation (indices positifs)
for n = 0 ; n < N_y ; n++ do
    y[n] = 0 ;
    // Calcule la corrélation
    for k = n ; k < N_x ; k++ do
        y[k] = y[k] + x[k] × x[k − n]
    end
end
```

Algorithm 3: Convolution

```
// Pour chaque élément de la convolution
for n = 0 ; n < N_y ; n++ do
    y[n] = 0 ;
    // Cherche les indices min et max à chaque décalage
    if n < N_h − 1 then k_min = 0 ;
    else k_min = n − (N_h − 1) ;
    if n < N_x − 1 then k_max = n ;
    else k_max = N_x − 1 ;
    // Calcule la convolution
    for k = k_min ; k < k_max ; k++ do
        y[k] = y[k] + x[k] × h[n − k]
    end
end
```

2 Fonctions fournies

Dans le fichier « `c/somefunc.cpp` » de l'archive `BE_tstr_2023_v1`.

2.1 Mesurer un temps d'exécution

```
double get_process_time();
```

Retourne l'horloge du système.

2.2 Écrire dans un buffer

```
int write_buff_dump(double* buff, const int n_buff, double* buff_dump, const int
n_buff_dump, int* ind_dump);
```

Copie le tableau `buff` de taille `n_buff` dans `buff_dump` de taille `n_buff_dump`. La copie commence à l'indice `ind_dump` du tableau `buff_dump` qui est incrémenté au fur et à mesure. Quand `ind_dump` atteint `n_buff_dump`, alors on ne peut plus écrire dans `buff_dump`.

`double* buff` :	Tableau à copier.
const int `n_buff` :	Nombre d'éléments du tableau à copier (si `n_buff` = 1, cela revient à un passage par adresse de la variable `buff`).
`double* buff_dump` :	Tableau qui reçoit la copie.
`const int n_buff_dump` :	Taille du tableau qui reçoit la copie.
`int* ind_dump` :	Tête d'écriture du tableau qui reçoit la copie. Celle-ci est *passée par adresse* pour être mise à jour à l'appel de la fonction.

2.3 Calculer une transformée de Fourier avec l'algorithme FFT

```
int get_nextpow2(int n);
```

Retourne la puissance de 2 juste supérieure à `n`.

```
int fftr(double *x, double *y, const int m);
int fft(double *x, double *y, const int m);
```

Calcule la transformée de Fourier d'un signal réel (`fftr`) ou d'un signal complexe (`fft`) avec une FFT.

`double *x` :	La partie réelle signal. Ces valeurs seront remplacées par la partie réelle de la transformée de Fourier après exécution.
`double *y` :	Le partie imaginaire du signal. Ces valeurs seront remplacées par la partie imaginaire de la transformée de Fourier après exécution.
const int m :	Taille de la TFD.

```
int ifft(double *x, double *y, const int m);
```

Calcule la transformée de Fourier inverse d'un signal avec l'algorithme FFT.

`double *x` :	La partie réelle de la transformée de Fourier. Ces valeurs seront remplacées par la partie réelle du signal après exécution.
`double *y` :	Le partie imaginaire de la transformée de Fourier. Ces valeurs seront remplacées par la partie imaginaire du signal après exécution.
`const int m` :	Taille de la TFD.

3 Compilation sur Windows

1. Installer le compilateur `tdm-gcc` : `https://github.com/jmeubank/tdm-gcc/releases/download/v10.3.0-tdm64-2/tdm64-gcc-10.3.0-2.exe`
2. Téléchargez l'API `RtAudio` `rtaudio-6.0.1.tar` et décompressez-là dans le répertoire de votre choix.
3. Déplacez les fichiers `RtAudio.cpp` et `RtAudio.h` du répertoire `rtaudio-6.0.1` dans `rtaudio-6.0.1\test`,
4. Compiler votre code avec la commande [2] :

```
C:\TDM-GCC-64\bin\g++ -Wall -D__WINDOWS_WASAPI__ -Iinclude -o duplex
duplex.cpp RtAudio.cpp -lole32 -lwinmm -lksuser -lmfplat -lmfuuid
-lwmcodecdspuuid
```

2. source : `https://www.music.mcgill.ca/~gary/rtaudio/compiling.html`