

Sujets BE Traitement du Signal en Temps-Réel - PHELMA - 2014

Thomas Hueber, CNRS/GIPSA-lab
v1.1

Introduction

Le cours magistral (2x2 heures) 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*, *time-triggered/event-based model*, etc.)
- Choix du hardware (DSP, GPU, FPGA, SoC, etc.)
- Systèmes d'exploitation (OS temps-réel vs. 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/overflow*, *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 (4x4 heures), nous vous proposons de mettre en pratique certains des points abordés dans le cours, au travers des projets suivants :

- a) implémentation d'un effet audio de type *reverb* à convolution (**obligatoire**)
- b) implémentation d'autres effets audio de type « *écho* » (*delay simple*, *flanger*, *chorus*) (facultatif)

Evaluation :

- Vous effectuerez une démonstration de votre programme en 5mn lors de la seconde partie de la 4^{ème} séance.
- Vous fournirez un fichier ZIP (nommé *nom1_nom2_tstr_2015.zip*, et à envoyer à thomas.hueber@gipsa-lab.fr) contenant :
 - o Un README précisant la marche à suivre pour compiler et exécuter votre programme
 - o un répertoire bin/ avec les différents exécutables
 - o un répertoire src/ contenant les sources (nettoyées et commentées) nécessaires à la compilation des différents exécutables (et le Makefile associé)
 - o un rapport **court** (max 4 pages, nommé *nom1_nom2_tstr_2015.pdf*) détaillant vos choix d'implémentation et les résultats obtenus (fonctionnement général, temps de latence estimés pour chaque callback audio, problèmes rencontrés, etc.).

Ces projets seront implémentés en standard C (ou C++) sous Linux en s'appuyant sur l'API RTAudio développé par G. Paul Scavone (une implémentation sous Windows ou MacOSX est tout à fait possible si vous le souhaitez). Dans un premier temps, nous vous invitons à prendre en main cet outil.

Prise en main de l'API RTAudio

Ce que vous devez faire :

- 1) Téléchargez l'API RTAudio à l'adresse suivante <https://www.music.mcgill.ca/~gary/rtaudio/> et décompressez la dans le répertoire de votre choix
- 2) Compilez la *library* principale (`./configure` puis `make all`)
- 3) Compilez les programmes d'exemples (`cd tests` puis `make all`)
- 4) Étudiez le 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 (nombre de *buffers* internes, taille des *buffers*, stratégies en cas d'*overrun/underrun*, etc.). Identifier les mécanismes de gestion de la priorité du *thread* audio (cf documentation de *RtAudioStreamFlags*)
 - b. Identifiez la fonction de *callback* audio et son prototype. Comment faire passer plusieurs paramètres du programme principal (*main*) à cette fonction de *callback* audio ?

Effet *reverb* à convolution (en temps-réel)

Le principe d'un effet audio de type *reverb* à convolution vous a été expliqué en cours. Ce type d'effet exploite un enregistrement de la réponse impulsionnelle d'un environnement acoustique (une salle par exemple). Cet enregistrement vous sera fourni. Ce que vous devez faire :

- 5) Télécharger l'archive http://www.gipsa-lab.fr/~thomas.hueber/cours/ressources_tstr_v1_1.zip. Elle contient les réponses impulsionnelles de différentes pièces, une simulation *offline* (i.e. non-temps-réel) d'une *reverb* à convolution en Matlab (script *testReverbOffline.m* dans le répertoire *matlab*). Étudiez cette implémentation (notamment la taille des différents signaux). Étudiez le comportement dans les domaines temporel et fréquentiel.
- 6) Implémentez une version temps-réel, basée sur la technique *d'overlap-add* en partant du source *duplex.cpp* (vous pouvez si vous le souhaitez créer votre propre exécutable « *reverb* » en adaptant légèrement le fichier *tests/Makefile*). Dans le fichier « *c/somefunc.cpp* » de l'archive *ressources_tstr_v1_1*, vous trouverez un ensemble de fonctions destinées à vous faciliter l'implémentation temps-réel. Ces fonctions sont relatives au calcul des FFT et FFT-inverse, et à la mesure du temps d'exécution.
 - a. Réponse impulsionnelle : Dans le répertoire « *c/* » de l'archive *ressources_tstr_v1_1*, vous trouverez un fichier intitulé « *impres* » contenant les échantillons d'une réponse impulsionnel, en format binaire (sans entête) au format double (64 bits), et donc facilement lisible avec *fread()*. Vous configurerez donc l'API RTAudio pour fonctionner avec ce format (cf. flag `RTAUDIO_FLOAT64`).
 - b. Implémentez la convolution dans le domaine temporel. Mesurez le temps d'exécution du *callback* audio à l'aide de la fonction *get_process_time()*. Ajustez les paramètres du flux audio pour essayer de limiter les *overruns* (et donc le phénomène de *glitch* audio). Concluez.
 - c. Implémentez la convolution dans le domaine fréquentiel et répliquez les mesures. Concluez sur les performances du système.

Effet de type écho (delay/chorus/flanger)

Toujours en partant de l'exemple *duplex* fourni dans l'API RTAudio, nous vous proposons ici d'implémenter 3 effets de type « écho », s'appuyant sur l'ajout à un signal d'une version retardée de lui même : $s(t) = e(t) + e(t - \text{delay}(t))$

- 1) Implémentez un *buffer* circulaire (vous pouvez vous appuyer sur ce que nous avons vu en cours théorique).
 - 2) Implémentez les effets audio suivants :
 - a. *Delay* : $\text{delay}(t) = K$ (de l'ordre de la dizaine de ms).
 - b. *Flanger*: $\text{delay}(t)$ est une fonction sinusoïdale
 - c. *Chorus*: $\text{delay}(t)$ est un nombre aléatoire
-