# Développement d'un effet de réverbération par convolution
# Cahier des charges

Paul ADENOT Yannick MEINE Gael MOTTE
Armand ROSSIUS Nicolas SILVA

18 décembre 2009

## Table des matières



# 1 Le projet

Dans le cadre du projet «Fil rouge» proposé par les enseignants du département informatique de l'INSA de Lyon, nous nous proposons de développer un effet de réverbération par convolution.

## 1.1 Principe

La réverbération par convolution utilise le procédé mathématique du même nom pour produire, à partir d'une réponse impulsionnelle et d'un autre signal, un signal de sortie correspondant au second signal «mélangé» au premier. Une réponse impulsionnelle est la sortie d'un système (dans notre cas un lieu ou un équipement audio) en réaction à un signal impulsionnel. Dans le cadre de ce projet, tous les signaux sont des ondes sonores. Signal impulsionnel : mathématiquement, c'est une onde sans harmonique et très brève. En pratique, on utilise l'éclatement d'un ballon de baudruche ou un pistolet starter.

Dans le cadre du projet, la réponse impulsionnelle sera l'enregistrement de la réverbération produite par un lieu réel ou un équipement audionumérique. Le second signal d'entrée sera la piste audio à traiter.

Ce traitement permettra d'appliquer au second signal d'entrée l'effet de réverbération du lieu ou de l'équipement du premier signal d'entrée.

## 1.2 Architecture logicielle

Le logiciel développé aura la forme d'un plugin (logiciel dépendant d'un logiciel hôte, qui, le plus souvent, lui ajoute des fonctionnalités), écrit en `C++`, et sera compatible avec la majorité des logiciels de musique assistée par ordinateur (MAO), parce qu'utilisant le Software Development Kit (SDK)[1] Virtual Studio Technology (VST), créé par Steinberg, célèbre entreprise dans le domaine de l'audionumérique. Ce SDK est *de facto* un standard dans le monde de la MAO, et son utilisation est gratuite.

La convolution étant une opération coûteuse au point de vue temps CPU, nous nous proposons d'effectuer le calcul sur le processeur graphique de la machine (le GPU), qui est particulièrement optimisé pour le calcul en virgule flottante massivement parallélisé, ce qui correspond tout à fait à la convolution. Cela permettra d'alléger le CPU lors de l'utilisation de multiple instances du plugin dans le programme hôte, et de proposer une latence imperceptible lors de l'enregistrement et du jeu en direct dans l'interface audionumérique de la machine.

## 1.3 Gestion de projet

Cinq personnes feront partie du projet, se répartissant les tâches en fonction de leurs affinités avec les technologies utilisées. Les répartitions sont d'hors et déjà faites. Le projet sera libre, et placé sous licence GPLv2.

[1] Kit de développement logiciel, ensemble d'outils fourni par un éditeur à destination des développeurs pour rendre possible la création d'applications sur un support donné

# 2 Technologies

## 2.1 Langage C++

Bien qu'il soit possible de programmer avec le SDK VST dans bon nombre de langages, nous avons choisi le C++ car il est d'une part très rapide (la rapidité étant la caractéristique principale de choix), et d'autre part compatible nativement avec des solutions de calcul sur GPU. De plus, tous les membres du projet connaissent ce langage. Enfin, C++ est le seul langage supporté officiellement par Steinberg pour son SDK, bien que des projets aient porté le SDK dans d'autres langages, tels que Delphi et Java.

Nous utiliserons le compilateur libre gcc dans sa version 4.4.2, qui suit la spécification C++03, norme actuelle du C++.

## 2.2 VST SDK

Le VST SDK est un ensemble de classes C++ reposant sur une interface de programmation (API) sous-jacente écrite en C. Cela permettra un développement aisé, puisqu'orienté objet. De plus, ce SDK fournit une abstraction matérielle complète, ainsi qu'une abstraction logicielle, puisque son fonctionnement est garanti avec bon nombre de logiciels hôtes. Nous pourrons alors nous concentrer sur l'écriture des algorithmes, tout en étant sûr du fonctionnement du logiciel en dehors de notre environnement de test.

Le plugin aura besoin d'une interface graphique conviviale, qui devra être développée, le SDK ne proposant que des primitives pour des *widgets* basiques tels que des boutons ou des potentiomètres. Nous utiliserons pour cela la bibliothèque libre (sous les termes de la BSD License) VSTGUI (http://sourceforge.net/projects/vstgui/), qui propose une interface de développement d'interfaces graphiques de plus haut niveau.

## 2.3 Calcul sur GPU

Le logiciel utilisera une solution logicielle permettant de réaliser le calcul sur le GPU, et donc de diminuer la charge du processeur principal. À l'écriture de ces lignes, nous devons encore choisir entre deux solutions, OpenCL de Khronos Group ou CUDA de nVidia.

**CUDA** (Compute Unified Device Architecture) est disponible sur les cartes graphiques nVidia Geforce 8 series et postérieures (à partir du modèle 8400GS[2]). Cela veut dire qu'il faut obligatoirement une carte graphique satisfaisant cette spécification pour que le calcul sur GPU soit possible. Cette technologie est mature, puisque la version 2 est considérée comme stable et la version 3 disponible en version beta.

CUDA est, du côté utilisateur, un ajout au langage C, proposant quelques fonctions pour travailler sur la carte graphique, et quelques ajouts syntaxiques afin de contrôler au mieux la gestion du calcul en parallèle. Le chaine de compilation pour un programme CUDA demande d'utiliser le compilateur nvcc, sur les fichiers programmés en C-CUDA (d'extension .cu, puis d'effectuer l'édition des liens avec les

[2]Liste complète : http://www.nvidia.com/object/cuda_learn_products.html

autres fichiers compilés.

Les principales primitives fournies par CUDA permettent l'appel de fonctions de calcul (appelé *kernel*), et le transfert de données depuis la mémoire principale (désignée comme *mémoire hôte*) vers la mémoire de la carte graphique (désignée comme *mémoire du périphérique*). De plus, il est possible de changer les options de parallélisation (tels que le nombre d'unité de calcul à utiliser, le nombre de fil d'exécution par unité de calcul et le nombre de blocs mémoire à utiliser par unité de calcul). CUDA propose de plus une bibliothèque capable de calculer des transformées rapide de Fourier, de manière optimisée, par le biais de la carte graphique.

C'est une technologie propriétaire, les sources du compilateur `nvcc` n'étant pas disponibles.

**OpenCL** (Open Computing Language), est une alternative multiplateforme à CUDA. Elle se base sur les mêmes principes au niveau de la programmation (un ajout au langage C) et possède l'avantage d'être à la fois multiplateforme (nul besoin de posséder un GPU de la marque nVidia) et plus générique dans le sens où OpenCL est aussi conçu pour permettre l'optimisation du calcul parallèle sur le *CPU* en plus du *GPU*. Les désavantages d'OpenCL par rapport à la technologie de nVidia résident principalement dans sa nouveauté. CUDA possède une plus ample documentation et probablement une grande communauté à ce jour, ce qui représente un avantage non négligeable.

La programmation avec OpenCL est proche de CUDA. On retrouve exactement les mêmes notions de *kernel*, *mémoire hôte*, *mémoire du périphérique*. Bien que les primitives fournies par les deux API ne soient pas strictement identiques, la même logique est mise en oeuvre et les mêmes possibilités sont offertes.

# 3 Cahier des charges

## 3.1 Fonctionnalités

- Les fonctionnalités que devra impérativement implémenter le plugin sont :
  - Pouvoir charger une réponse impulsionnelle depuis un fichier audionumérique.
  - Pouvoir traiter un signal entrant par convolution avec le fichier chargé précédemment.
  - Pouvoir régler le taux entre les signaux traités et non traités en sortie du plugin.
  - Pouvoir passer sur calcul par GPU à celui par CPU (soit avec un bouton à deux états, pendant la lecture, soit en proposant de changer le comportement à la compilation, et donc en proposant deux binaires).
  - Pouvoir utiliser plusieurs instances du plugin simultanément.
- Les fonctionnalités que pourra implémenter le plugin, dans un ordre de priorité décroissant sont :
  1. Pouvoir mettre un pré-délai sur le traitement, pour simuler une pièce de plus grande taille.
  2. Pouvoir rogner le signal impulsionnel en durée.
  3. Pouvoir inverser le signal impulsionnel.
  4. Pouvoir changer l'enveloppe du signal impulsionnel (attack, decay, sustain, release).

5. Pouvoir égaliser le signal impulsionnel.

### 3.2 Performances

Le plugin se devra d'avoir de bonnes performances. De part la complexité du calcul d'une convolution, la lecture risque d'être retardée par le temps de calcul de l'effet. Ce phénomène est appelé latence en audionumérique. La latence, induite par le traitement, devra être minimale, la lantence nulle étant impossible.

Un humain ne pouvant distinguer la latence qu'à partir d'environ 10 millisecondes à 440Hz, nous devrions idéalement être en-dessous de ce seuil pour rendre le jeu en direct à travers le plugin possible.

Le SDK VST propose de plus des fonctions de compensation de latence, ce qui pourrait nous permettre d'augmenter la latence (par exemple pour diminuer la charge CPU ou GPU, et donc lancer plusieurs instances du plugin), sans pour autant entendre un quelconque décalage lors de la lecture.

### 3.3 Interface

- L'interface du plugin devra pouvoir permettre à l'utilisateur de :
  - Choisir le signal impulsionnel à utiliser, par l'intermédiaire d'une liste de fichiers.
  - Proposer des contrôles pour les différentes fonctionnalités du plugin, tels que des faders (potentiomètre linéaires), des potentiomètres rotatifs, des boutons.
- L'interface du plugin pourra en plus incorporer les éléments suivants :
  - Un visualiseur pour le fichier d'impulsion.
  - Des vu-mêtres pour les signaux d'entrée et de sortie.
  - Un moyen plus visuel d'éditer les éventuels courbes de niveau, en dessinant la courbe, par exemple.