DÉVELOPPEMENT MULTIPLATEFORME ET TEMPS-RÉEL D'UN CLAVECIN SYNTHÉTISÉ PAR MODÈLES PHYSIQUES AVEC FAUST

Thomas Cipierre
CIEREC – EA3068
Université Lyon-Saint-Etienne
thomas.cipierre@gmail.com

Laurent Pottier
CIEREC – EA3068
Université Lyon-Saint-Etienne
laurent.pottier@univ-st-etienne.fr

RÉSUMÉ

Dans le cadre du projet ANR FEEVER, portant sur la mise au point de solutions ubiquitaires pour le traitement audionumérique portable et multiplateforme en utilisant le langage Faust, nous avons réalisé une application interactive, disponible sous différentes formes, pour la synthèse par modèles physiques de sons de type « clavecin ». La première, installée au Musée d'Art et d'Industrie de Saint-Étienne est une application autonome, programmée sous *Max*, contrôlée à partir d'un clavier MIDI et d'un écran tactile. La seconde est une application Web, basée sur l'utilisation de la Web Audio API, et qui est disponible en téléchargement à partir de tout navigateur récent.

1. INTRODUCTION

Le Musée d'Art et d'Industrie de Saint-Étienne possède dans ses collections depuis la fin du XIX^{e} siècle un clavecin prestigieux. Après des études scientifiques préalables, menées notamment par le Musée de la musique à Paris, ce clavecin a été restauré entre 2013 et 2014 mais sans pouvoir être remis en état de jeu car il est trop fragile. Une exposition lui a été consacrée (sept. 2014 - janv. 2015) avant qu'il soit réintroduit dans les collections permanentes.

Nous présentons ici le travail réalisé au CIEREC pour produire pour le Musée une borne et une application Web, toutes deux interactives et destinées au public, permettant la synthèse par modèles physiques en temps réel de sons de type « clavecin », basée sur des mesures réalisées sur l'instrument ou d'autres clavecins similaires.

Ces recherches font suite à des travaux effectués par Romain Michon qui lors d'un contrat de recherche pour le CIEREC a effectué un stage au CCRMA à l'Université Stanford, entre janvier et avril 2011, au cours duquel il a porté les outils de synthèse par guides d'ondes vers le langage Faust [1]. Le clavecin virtuel a ensuite été réalisé avec Faust et *Max* par Luc Faure et Laurent Pottier entre janvier et août 2014. Thomas Cipierre a réalisé la version Web entre septembre et décembre 2014.

2. DÉVELOPPEMENT DU DSP SOUS FAUST

Nous avons segmenté la modélisation de l'instrument en trois parties : l'excitation, le modèle de corde et le résonateur. Les deux premières sont polyphoniques.

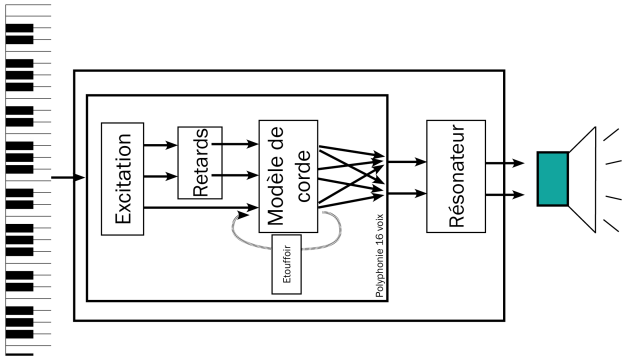

Figure 1. Schéma de câblage du clavecin virtuel.

2.1. Excitation

La partie excitation est constituée d'un bruit blanc dont l'amplitude est contrôlée par une enveloppe (très courte). Un premier filtre passe-bas modélise l'angle avec lequel le plectre attaque la corde. Un deuxième filtre (en peigne) prend en compte la position sur la corde où est réalisée l'excitation. Un troisième filtre (passe-bas à nouveau) contrôle le spectre de l'excitation en fonction de la hauteur de la note (la fréquence de vibration attendue de la corde). Bien que différents auteurs aient étudié les interactions plectre-corde dans le cas du clavecin [2,3], aucun d'entre eux ne proposent réellement une modélisation physique temps réel.

Figure 2. Les différents modules de synthèse pour l'excitation.

2.2. Modèle de corde

Le modèle de corde reçoit comme signal l'excitation produite par le module précédent. Dans un premier temps celui-ci est adressé à une enveloppe qui dépend de la fréquence attendue. L'excitation est ensuite envoyée à un guide d'ondes unidimensionnel formé d'un filtre de réflexion biquadratique, d'une ligne à retard (*FDelay*) et d'un filtre biquadratique (dont l'amortissement dépend de la fréquence) qui modélisent la propagation du son dans la corde. Un filtre passe-tout non-linéaire d'ordre six permet ensuite d'introduire une certaine distorsion dans le son imitant les perturbations introduites au niveau de la fixation sur le chevalet.

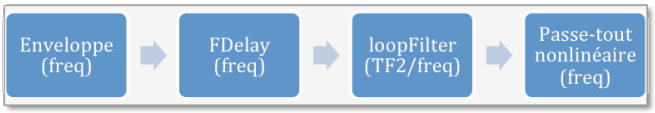

Figure 3. Les différents modules de synthèse pour le modèle de corde.

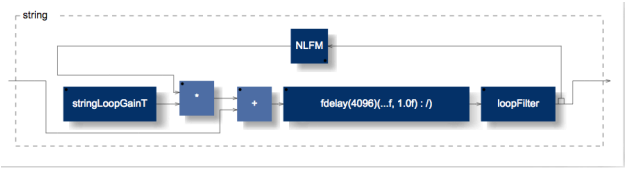

Figure 4. Le diagramme FAUST du modèle de corde.

2.3. Résonateur

La modélisation de la caisse de résonance et de la table d'harmonie est trop complexe pour pouvoir être obtenue par la méthode des guides d'ondes. Nous avons utilisé une réponse impulsionnelle de la caisse de résonance d'un clavecin, que nous avons analysée par la méthode des modèles de résonance [4]. Un banc de filtres permet ensuite de modéliser ces résonances. Le fait d'utiliser ces résonateurs permet ensuite de disposer de paramètres intuitifs pour contrôler la synthèse (modification des durées de résonances par exemple).

2.4. Désynchronisation des différents jeux

Nous avons réalisé des séances de tests de l'instrument avec Martial Morand, professeur de clavecin au Conservatoire à Rayonnement Régional de Saint-Étienne, au cours desquelles, suite à ses remarques, nous avons été amenés à introduire de légers décalages dans les temps d'attaque des différents jeux (20 à 40 ms) pour un meilleur ressenti au toucher lorsque plusieurs jeux sont activés simultanément. Lors de ces séances, nous avons également enregistré plusieurs séquences MIDI qui peuvent être lues par les utilisateurs pour tester et écouter les sonorités du clavecin virtuel.

2.5. Étouffoir

Pour modéliser l'action de l'étouffoir lorsqu'il retombe sur les cordes, introduisant des vibrations secondaires sur les deux tronçons de corde qu'il sépare, nous avons effectué une réinjection de quelques millisecondes du son vers l'entrée de la corde, en effectuant une modification de la fréquence de la corde selon la position de cet étouffoir. Cela permet d'imiter le son caractéristique du clavecin lors du jeu en notes piquées.

2.6. Accord et tempérament

En plus du tempérament égal, nous avons proposé différents tempéraments, dont l'accord selon les tempéraments Zarlino (XVI^{e} siècle) et Werkmeister III (XVIII^{e} siècle).

2.6.1. Zarlino

Avec le tempérament Zarlino, les principaux intervalles dans un mode donné (ici en *Do*) correspondent à des rapports de fréquences donnés par des fractions simples : *Do-Ré* = 9/8 (deux quintes pures transposées d'une octave : 3/2 × 3/2 ÷ 2) ; *Do-Mi* bémol = 6/5 ; *Do-Mi* = 5/4 ; *Do-Fa* = 4/3 ; *Do-Sol* = 3/2 ; *Do-La* = 5/3 (une tierce majeure + une quarte : 5/4 × 4/3 = 5/3) ; *Do-Si* = 15/8 (une tierce majeure + une quinte: 5/4 × 3/2 = 15/8).

Note	do	réb	ré	mib	mi	fa	fa#	sol	lab	la	sib	si
Rapport	1/1	16/15	9/8	6/5	5/4	4/3	45/32	3/2	8/5	5/3	9/5	15/8

2.6.2. Werkmeister III

Dans ce tempérament, le comma pythagoricien est réparti par quarts sur 4 quintes (qui deviennent ainsi un peu courtes) : *Do-Sol*, *Sol-Ré*, *Ré-La* et *Si-Sol* bémol.
Ainsi les tierces *Do-Mi* et *Fa-La* sont assez proches d'intervalles justes, les autres s'en éloignant progressivement. Ce tempérament, par le choix de la quinte tempérée *Si-Sol* bémol, favorise les tonalités en bémol.

Note	do	réb	ré	mib	mi	fa	fa#	sol	lab	la	sib	si	do
Cents	0	90.25	192.2	294.1	390.2	498	588.3	696.1	792.2	888.3	996.1	1092.2	1200

3. LES INTERFACES DE CONTRÔLE

3.1. La borne clavecin tactile sous Max

L'interface de la borne a été réalisée en utilisant le langage *Max*. La borne comporte deux parties. La première contient des vidéoclips présentant des élèves du CRR de Saint-Étienne interprétant des pièces de clavecin sur l'instrument du conservatoire. La seconde contient le clavecin virtuel que nous avons réalisé.

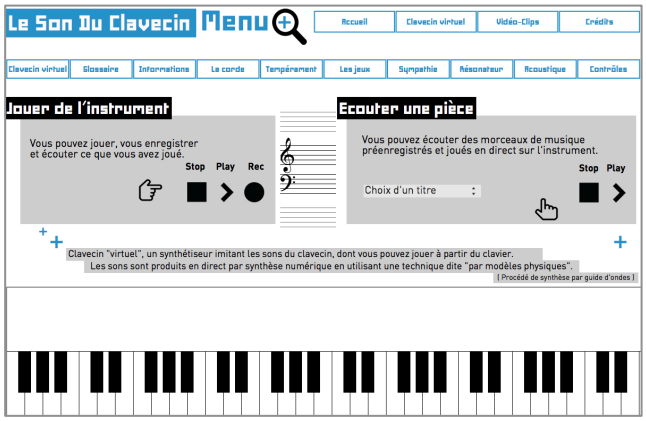

Figure 5. Page d'accueil du clavecin virtuel.

Cette partie est divisée en dix pages permettant d'aborder progressivement l'acoustique et le fonctionnement du clavecin.

La première page permet de jouer sur le clavier MIDI, de s'enregistrer en MIDI puis d'écouter à nouveau. Cette séquence pourra être réutilisée dans les différentes pages de la borne. Des séquences MIDI du répertoire sont également disponibles. Elles ont été enregistrées en MIDI sur le clavecin virtuel par Martial Morand. La dernière page récapitule l'ensemble des paramètres qu'il est possible de faire varier pour contrôler le son du clavecin virtuel.

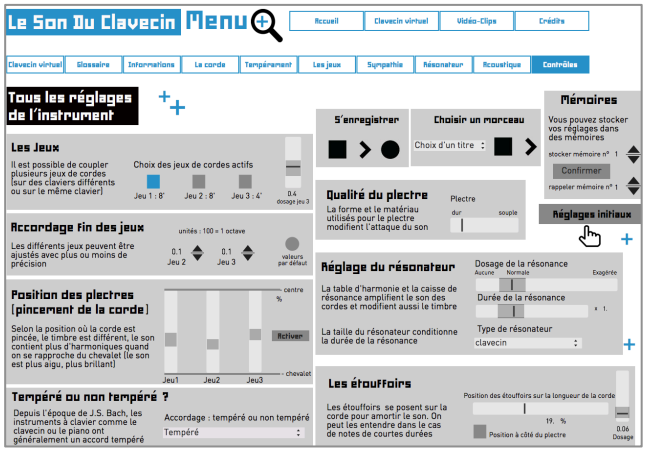

Figure 6. Page de contrôle des paramètres du clavecin.

3.2. Le clavecin Web

La version Web de l'application clavecin se base sur une reconstruction complète de l'interface utilisateur afin de développer les aspects intuitifs et ludiques des contrôleurs. La figure 7 présente une vue schématique de son interface Web. Elle est jouable à l'adresse : http://musinf.univ-st-etienne.fr/recherches/ClavecinHtml/web-harpsichord.html.

Le clavecin peut se contrôler à la souris, au clavier, ou via un contrôleur MIDI et intègre une gestion de *presets*. Il est de plus possible de jouer des pistes MIDI accessibles dans une liste.

Plusieurs modes de jeu sont proposés, allant du mode standard (aucune activation de paramètre avancé, avec réglage du clavecin par défaut) aux modes avancés. Ces derniers permettent le contrôle séparé des paramètres du plectre, des cordes et du résonateur, apportés par la synthèse par modèles physiques. Finalement, des curseurs de vélocité et de volume permettent de régler séparément l'attaque et le niveau de sortie.

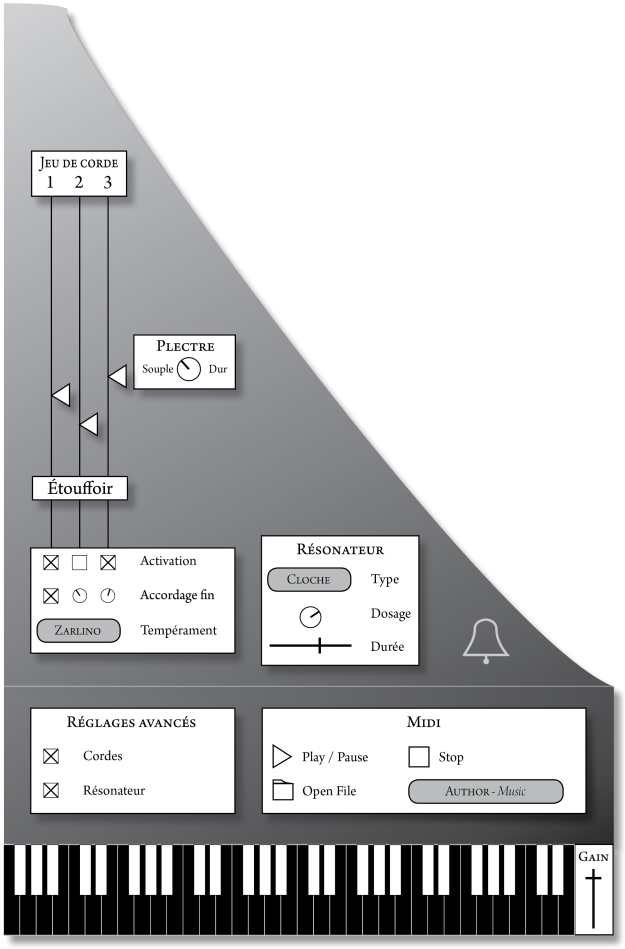

Figure 7. Interface utilisateur du clavecin Web.

Pour les paramètres avancés du plectre, la dureté ainsi que l'emplacement d'attaque sur chaque jeu de corde (du chevalet à mi-longueur) permettent de modifier l'excitation du système et, par conséquence, le timbre global de l'instrument.

Pour les paramètres avancés des cordes, chaque note jouée au clavecin peut potentiellement actionner trois jeux de cordes différents, activables indépendamment. En plus d'un réglage supplémentaire de volume, les jeux n°2 et n°3 peuvent être désaccordés par rapport à la note de référence (jeu n°1) par l'activation du réglage d'accordage fin (en

midicents). Le tempérament global du clavecin est accessible via un menu déroulant (tempérament égal, Zarlino, Werckmeister III).

Pour les paramètres avancés du résonateur, une liste déroulante permet de choisir entre plusieurs types de caisse de résonance (clavecin, piano, cloche, etc.), affectant le timbre du clavecin par filtrage des harmoniques. Le « mixage » du résonateur est réglable, ainsi que sa taille et durée de résonance.

4. LE DSP FAUST SUR LE WEB

Le Web se standardise autour des protocoles HTML, CSS et JavaScript, permettant dès aujourd'hui un développement quasi-ubiquitaire de pages Internet. Parallèlement, l'essor de JavaScript et de ses nombreux API/Frameworks permet de s'émanciper d'une simple visualisation de pages statiques pour le développement de véritables applications, dont les ressources peuvent intégralement se gérer côté client. Faust établit une passerelle simple dans le développement de DSP temps-réel sur le Web, en s'appuyant sur la Web Audio API[1], une API JavaScript pour le DSP et la synthèse audio aux normes W3C[2], et sur asm.js[3], un sous-ensemble de JavaScript hautement optimisant.

1.1.1. Compilation Faust pour le Web

À l'heure actuelle, Faust travaille conjointement, au choix de l'utilisateur, sur deux chaînes de compilation différentes afin de passer du code Faust au code asm.js. La première[4] permet une compilation du code C/C++ en asm.js par l'intermédiaire du compilateur LLVM Emscripten[5] (clang[6] convertissant le code C/C++ issu de Faust en bitcode LLVM). La deuxième[7] passe par la version librairie du compilateur Faust (libfaust) et le *backend* asm.js, afin de générer directement du code asm.js optimisé. Chaque chaîne de compilation apporte ses avantages (facilité d'utilisation, gestion des fonctions externes, compilation vectorielle, etc...) et reste donc développée conjointement. L'utilisateur devra choisir la méthode la plus adaptée à son projet.

Dans les deux cas, la compilation faust2webaudioasm permet d'obtenir un fichier HTML opérationnel, le DSP optimisé sous asm.js étant directement matricé via la Web Audio API. L'interface utilisateur est générée par JavaScript et SVG. Elle est proche d'une interface d'application Faust QT native.

Il est aussi possible de générer un fichier asm.js seul par la commande faust2asmjs. Pour construire l'interface utilisateur, il suffit alors d'utiliser les standards HTML et CSS pour le visuel de l'interface, et JavaScript pour l'aspect interactif et le renvoi des valeurs des GUI au DSP Faust. Pour l'interfaçage audio, il faut finalement créer les nœuds standards de la Web Audio API nécessaires, instancier le DSP Faust et le connecter. Plus concrètement, il faut donc au minimum créer un « *context audio* » de la Web Audio API ainsi qu'une instance du DSP Faust via le code JavaScript suivant :

```
var audioCtxExemple = new (window.AudioContext ||
window.webkitAudioContext)();
var DSP1 = faust.<nom-du-DSP>(audioCtxExemple,
<taille-buffer>);
```

Il faut ensuite matricer les entrées et sorties du DSP, comme par exemple pour une simple sortie sur le système audio assigné au navigateur :

```
DSP1.connect(audioCtxExemple.destination);
DSP1.start();
```

Finalement, JavaScript renverra les valeurs des interfaces HTML aux paramètres du DSP Faust en ajoutant aux « *event listener* » JavaScript le code :

```
DSP1.setValue(<chemin-accès>, <valeur>);
```

Pour accéder à ces paramètres, il est possible de récupérer la liste des paramètres Faust avec leurs chemins d'accès complets, ou l'architecture JSON via les commandes :

```
getPathTable<nom-du-DSP>();
getJSON<nom-du-DSP>();
```

5. CONCLUSION

Faust permet le développement de DSP complexes, sans la contrainte d'une quelconque architecture informatique dès leurs conceptions. Cette abstraction de type « API générique », couplée aux nombreuses possibilités de compilation offertes, permet d'utiliser le même « calcul » dans différents environnements, inscrivant d'emblée toute réalisation dans une optique ubiquitaire et pérenne. La création d'interfaces utilisateurs se consacre ainsi spécifiquement à une architecture précise et le développement innovant d'un clavecin par modèles physiques peut par exemple trouver des applications directes dans l'univers modulaire de *Max*, tout comme dans une application Web côté client.

---

1 http://webaudio.github.io/web-audio-api/
2 http://www.w3.org/
3 http://asmjs.org/
4 branche git « master ».
5 https://github.com/kripken/emscripten
6 http://clang.llvm.org/
7 branche git « faust2 ».

6. RÉFÉRENCES

[1] Michon, Romain, « Faust-STK : une Bibliothèque de Modèles Physiques pour le langage FAUST », *Actes des Journées d'Informatique Musicale (JIM-2011),* Saint-Étienne, 2011

[2] Chadefaux, Delphine ; Le Carrou, Jean-Loïc ; Le Conte, Sandy ; Castellengo, Michèle, « Analysis of the harpsichord plectrum-string interaction », *Proceedings of the Stockholm Music Acoustics Conference 2013*, SMAC 2013, Stockholm.

[3] Perng, Chao-Yu J. ; Smith, Julius ; Rossing, Thomas, « Physical Modeling of the Harpsichord Plectrum-String Interaction », *Proceedings of the 13th Int. Conference on Digital Audio Effects (DAFx-10)*, Graz, Autriche, 2010.

[4] Potard, Yves ; Baisnée, Pierre François ; Barrière, Jean-Baptiste, « Méthodologie de synthèse du timbre : l'exemple des modèles de résonance », *Le Timbre, Métaphore pour la Composition*, coll. Musique/Passé/Présent, Paris, C. Bourgois - IRCAM, 1991, p. 135-162.

[5] Cipierre, Thomas ; Faure, Luc ; Letz, Stéphane ; Pottier, Laurent, « Faust : contrôle et DSP sur le Web », *Revue Francophone d'Informatique Musicale* [En ligne], n°4, 2014, mis à jour : 27/10/2014, URL : http://revues.mshparisnord.org/rfim/index.php?id=343.

[6] O. Smith III, Julius, « Physical Modeling using Digital Waveguides », *Computer Music Journal*, Part I, Vol. 16, n° 4, MIT Press, Massachusetts, 1992, p. 74-91.

[7] Orlarey, Yann ; Fober, Dominique ; Letz, Stéphane, « Faust : an Efficient Functional Approach to DSP programming », *New Computationals Paradigms for Computer Music*, Delatour France, Sampzon, 2009.

[8] Kottick, Edward ; Marshall, Kenneth ; Hendrickson, Thomas, « L'acoustique du clavecin », *Les instruments de l'orchestre*, Bibliothèque Pour la science, Belin, Paris, 1995, p. 41-48.