Numéros / n° 4 - automne 2014

« Faust : contrôle et DSP sur le Web »

Laurent Pottier

Résumé

De par son haut degré de technicité, la création contemporaine et ses solutions informatiques ne sont pas toujours accessibles à la « pratique amateur ». Mais avec le développement du Web et de ses solutions embarquées, ainsi qu'avec la standardisation de nouveaux protocoles de communication, le contrôle distant d'applications et de DSP spécifiques sont désormais beaucoup plus accessibles. Les dernières versions de Faust, dont plusieurs applications sont présentées ici, permettent pleinement de profiter de ces technologies.

Introduction

Les technologies Web sont maintenant omniprésentes dans le monde du numérique. Le Web n'est plus seulement limité à l'affichage de pages HTML comportant des images et des textes, il permet également le streaming audio et vidéo. Mais les navigateurs ne se limitent plus à l'affichage, la lecture ou le transfert d'informations. Ils peuvent maintenant devenir le support de véritables applications, et d'applications multimédias.

Le réseau permet de connecter les hommes, via des ordinateurs mais aussi via des smartphones, des tablettes. De plus en plus d'équipements électroniques permettent d'envoyer des données par de simples connexions Wi-Fi.

Les dernières versions de Faust permettent de profiter de ces technologies. Nous présentons dans ce texte deux façons d'aborder la question : par le contrôle distant ou par du traitement de signal directement intégré au navigateur.

Ces deux technologies sont illustrées dans ce texte par des réalisations appliquées : la pièce *Interloper* (2014) de Thomas Cipierre et la borne interactive « Clavecin audio » (2014) réalisée pour le Musée d'arts et d'industrie de Saint-Étienne par Luc Faure et Laurent Pottier.

Ce travail s'inscrit dans le cadre du projet FEEVER, financé par l'Agence nationale de la Recherche, sous la référence ANR-13-BS02-0008-01 (1).

1. Faust : un outil de contrôle via le Web ? le spectateur artiste, par Thomas Cipierre

Interloper est un projet de création plaçant le public au centre du live musical. L'objectif était de questionner cette nouvelle dualité de l'être, où l'entité augmentée « Homme/avatar » confronte le simple consommateur anonyme à son véritable démiurge numérique. J'entends par là la question de cette permission d'action d'un avatar numérique interconnecté en permanence, face à l'espace-temps du spectacle vivant, où toute prise d'action/parole peut rester dans l'anonymat de la dimension humaine, tout en revêtant paradoxalement une responsabilité quant au déroulement du spectacle, jusqu'au statut même

d'égal à l'artiste.

D'un point de vue technique, il me fallait étudier la pertinence des solutions existantes en terme de contrôle temps-réel, tout en les rendant aisément accessibles. Le smartphone, véritable ordinateur de poche, ainsi que porte ouverte coutumière et confortable sur l'univers numérique, semblait alors tout indiqué. Pour ne pas bousculer « les rites de transition » propres à chaque personne, il ne me restait plus qu'à trouver une solution embarquée, ne nécessitant aucune installation préalable d'application. L'univers du libre étant particulièrement propice à l'intercommunication des programmes, mes solutions se sont donc naturellement développées autour de Faust [2], Pure Data [3] et Jack [4].

1. 1. Une liberté conditionnelle ? *routing* & DSP

Les contraintes sont motrices de la création et du lâcher-prise. Bien à l'abri derrière cette philosophie, l'artiste peut aisément mettre en place un *routing* et une gestion de DSP spécifique, cloisonnant de manière rassurante les interventions du public. Cette première étape de travail était d'autant plus simple pour ma part que cette liberté restreinte se basait sur une déconstruction sous forme de boucles synchronisées du morceau *The Greater Good* (2007) de Nine Inch Nails [5].

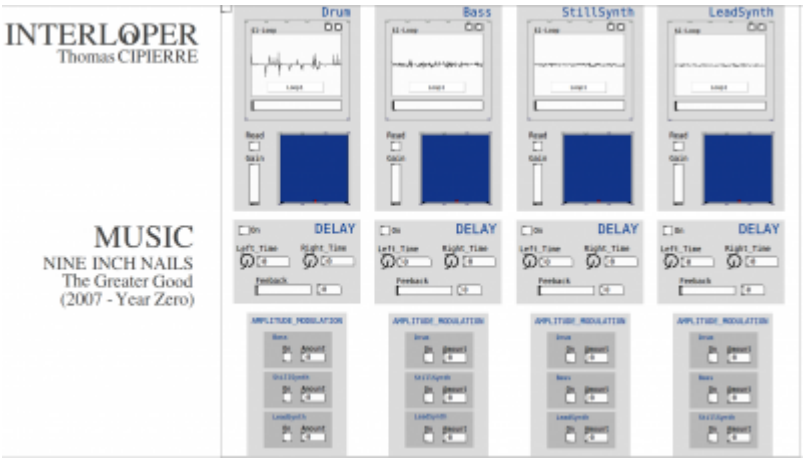
Pure Data a alors été choisi comme plateforme de centralisation des processus. Hormis son efficacité pour la lecture synchronisée des différentes boucles, il me permettait surtout d'agir comme centre névralgique des protocoles de communication inter-application via un port OSC unique de réception [6]. De plus, le travail d'Albert Graëf [7] m'a permis de développer et personnaliser toute la partie DSP en Faust pour la récupérer sous Pure Data, directement opérationnelle et cohérente avec sa description d'interface utilisateur. La syntaxe de compilation pour Pure Data après développement du code DSP Faust (et installation préalable de faust2pd via les archives de Pure) est la suivante :

faust2puredata <mondsp.dsp>

Figure 1. Compressor - faust2puredata

Un autre intérêt dans l'usage de Pure Data était de projeter sur un écran une interface globale regroupant les différents contrôles, proposant ainsi un rendu uniformisé, tout en fournissant un retour supplémentaire sur la bonne prise en compte des actions demandées par les spectateurs/artistes.

Figure 2. Vidéo-projection - patch Pure Data



Pour chaque piste, les paramètres de contrôle sont les suivants : le choix du *sample*, le volume, la spatialisation, le délai stéréo et les paramètres de modulation d'amplitude. Ces derniers ajoutaient une dimension intéressante de croisement des contrôles. En effet, un spectateur/artiste s'autorisait à tout moment à moduler une piste contrôlée par un autre. La projection vidéo permettait alors d'ajouter un retour visuel sur la piste contrôlée, tout en gardant sournoisement l'anonymat du perturbateur ! L'idée était ainsi de tenter de résoudre l'expérience sonore dans un monde numérique, via nos seuls sens et l'inspiration éphémère d'une rencontre de spectateurs acteurs de la représentation.

1. 2. Mise en oeuvre des protocoles communicants : Httpd/OSC

La phase technique la plus importante consistait à tenter de rendre facilement accessibles les éléments de contrôle via smartphone (quel que soit son OS), sans installation préalable d'application. Grâce à la librairie GNU libmicrohttpd [(8)], et à son utilisation optionnelle dans le fichier d'architecture à la compilation d'un objet Faust, nous pouvons créer un serveur HTTP incorporé à l'application générée. Elle peut alors être contrôlée à distance via un quelconque navigateur web (compatible HTML5) sur un port TCP choisi, tout en rendant l'application visible sous forme de page HTML (Javascript et SVG donnant par défaut une interface proche d'une application QT native). Pour profiter des options HTTPD à la compilation des DSP Faust, il faut au préalable installer Faust dans un terminal de la manière suivante :

make httpd && make && sudo make install

L'ajout de la fonction HTTPD lors de la compilation d'un objet Faust se fait alors via la commande :

faust2<environnement-compatible> -httpd <mondsp.dsp>

Le lancement de l'application compilée attribue par défaut le port TCP 5510 (ou le port suivant s'il est déjà utilisé) ou le port spécifié via la syntaxe :

./<mondsp> -port <TCPport>

Figure 3. Stéréo Delay ? HTTPD

Bien que garantissant la transmission des données, la relative lourdeur du protocole TCP (du fait de son typage connexion et son contrôle de redondance cyclique) a été palliée par l'ajout d'objets *line* [9] à l'arrivée des données dans Pure Data, pour éviter tout « *clipping* » en cas de changements brutaux.

Concernant l'étape de récupération des valeurs d'éléments contrôlés sur les applications HTTPD, Faust permet nativement [10] une compilation optionnelle, rendant sensibles les applications au protocole OSC [11] du CNMAT. Ce protocole permet l'envoi/réception de données sur le réseau, au travers une syntaxe type < URL valeur >. Pour profiter du contrôle distant par OSC sur une application Faust, il suffit de compiler le code Faust de la manière suivante :

faust2<environnement-compatible> -osc <mondsp.dsp>

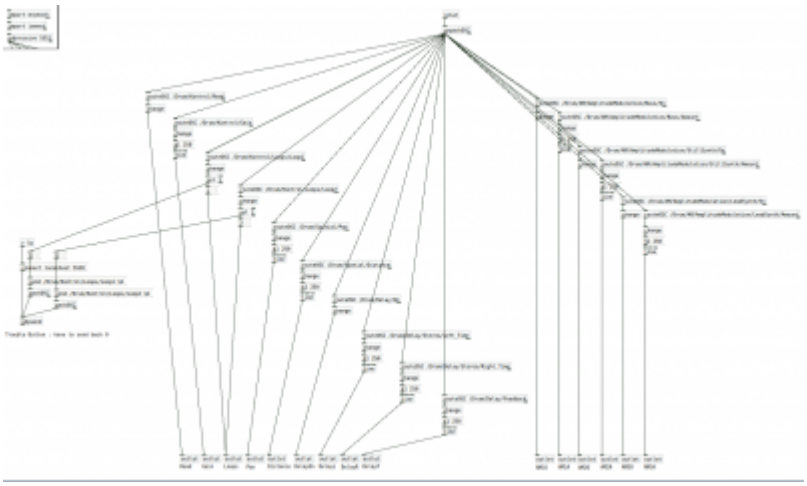
Le protocole OSC aura alors trois ports UDP ouverts pour la communication : un port d'écoute, un port d'envoi et un port d'erreur. Ils sont attribués automatiquement sur les ports 5510, 5511, 5512 (ou ports suivants en cas de port occupé) ou directement choisis à l'exécution via la syntaxe :

./<mondsp> -port <UDP-réception> -outport <UDP-envoi> -errport <UDP-erreur>

L'envoi des données sur le port UDP est par défaut désactivé, et doit clairement être énoncé à l'exécution de l'application via l'option ?xmit 1. Le protocole OSC permet aussi de spécifier l'IP de destination des messages OSC via l'option -desthost <host>, mais nous n'en avons pas l'utilité ici, tous les processus s'exécutant sur la même machine.

Finalement, le port de réception UDP étant verrouillé par la première application « écoutante », il est préférable de centraliser l'envoi de toute application Faust sur un même numéro de port, et la réception de tous les messages sur une seule application, en l'occurrence ici, Pure Data [12].

Figure 4. OSC ? Piste Drum ? patch Pure Data
(Cliquez sur la figure pour l'agrandir)



1. 3. Retour d'expérience

Concernant la partie administration du projet, la première étape consistait à configurer un routeur WiFi permettant aux spectateurs de se connecter au réseau, tout en attribuant une IP fixe à l'ordinateur opérant les processus. J'ai ainsi pu distribuer des QR codes, simplifiant encore la connexion aux interfaces de contrôles HTTPD [(13)]. Pour mon usage personnel, j'ai également produit une application HTTPD contrôlant le volume général de sortie (via Jack), permettant ainsi de couper tout signal en cas d'incident lors de l'exécution.

Figure 5. QR Codes ? 4 joueurs

Les applications bien compilées (avec les bonnes options activées) et le patch Pure Data finalisé correctement (avec adjonction d'un Compresseur/Limiter Faust en bout de chaîne pour éviter tout incident sonore sur des manipulations extrêmes de valeurs des délais), j'ai finalement créé un script Bash de lancement pour spécifier le bon ordre d'exécution des applications, avec leurs bonnes attributions de port TCP (pour l'accès à l'interface HTTPD) et de ports UDP (pour l'OSC). La compilation Faust combinant les propriétés HTTPD et OSC, le même numéro de port TCP/UDP est attribué en réception, puis le port d'envoi est activé, avec numéro d'envoi et d'erreur identiques un à un pour toutes les applications Faust via la syntaxe :

./<mondsp> -port <TCP/UDP-réception> -xmit 1 -outport <UDP-envoi-commun> -errport <UDP-erreur-commun>

Le script prévoyait un lancement pour 4 personnes (une interface globale par piste), pour 8 personnes (deux interfaces par piste, avec une pour le contrôle des *samples*, du volume et de la spatialisation, et l'autre pour l'activation et les réglages des effets, amenant une coopération de deux spectateurs/artistes sur une même piste), ou finalement pour 16 joueurs (une interface par onglet de contrôle, à savoir *samples* /volume, spatialisation, délai stéréo, et AM, soit quatre joueurs par piste).

Avec le recul, plusieurs améliorations seraient envisageables. Il faudrait notamment rendre l'interface utilisateur plus souple, en intégrant par exemple les données des accéléromètres issues des smartphones (pas encore proposées sous formes de métadonnées dans Faust). Cette représentation s'adressant de plus à

un public conscient de l'aspect expérimental du travail, tout développement dans un univers purement créatif demanderait quelques allègements, afin de gommer encore ce léger temps d'explications préalables. Il coupe à mon sens l'immersion par son aspect technique, découvrant les rouages de la création autant qu'il s'éloigne de la magie de la découverte autonome.

Cependant, cette rencontre fut pour ma part une réussite artistique et humaine. J'ai en effet été agréablement surpris par la prestation des spectateurs/artistes. Au-delà des rires, des réajustements techniques, et des premiers contrôles excessifs inhérents à toute compréhension de nouveaux modèles, la représentation a bien glissé dans un enfermement consciencieux de chaque spectateur/artiste, rivé sur son smartphone. Paradoxalement, la prise de risque décomplexée par l'anonymat du numérique a alors engendré une réelle symbiose de sensibilité humaine, pure de par son émotion et détachée de tout *a priori* relationnel. Malgré des bagages techniques très hétérogènes, l'investissement de chacun a permis l'émergence d'une « aura » collective, qui, bien loin du matériau sonore originel, a désacralisé l'icône de l'« artiste » dans un « final » créatif, inspiré et maîtrisé.

2. Du DSP sur le web - Faust et la Web Audio API, par Stéphane Letz

Disponible avec HTML5, l'interface de programmation Web Audio API comporte un ensemble de fonctions utilisables en JavaScript pour synthétiser, transformer et jouer des sons dans les navigateurs. Elle est principalement utilisée dans les jeux et pour les besoins de production et rendu sonore dans les applications Web récentes.

2. 1. Description de la Web Audio API

Avant l'apparition de l'élément <audio> dans la norme HTML5, l'utilisation de *plugins* (type Flash) spécialisés était nécessaire pour gérer le son dans les navigateurs. Avec HTML5, la Web Audio API a été créée pour donner au programmeur plus de contrôle sur la génération et transformation du son.

Le modèle utilisé est celui du graphe audio, constitué de noeuds audio (producteurs, transformateurs ou analyseurs de sons...), connectés les uns aux autres (en établissant un graphe arbitrairement complexe) et finalement aux entrées/sorties de la machine. Le graphe audio est alors exécuté par blocs, avec un *buffer* audio d'une taille donnée, à une fréquence d'échantillonnage donnée.

Deux types de noeuds audio peuvent être utilisés : un ensemble de *noeuds natifs* (gain, compresseur, générateur de signaux ou lecteurs de fichiers, etc.) codés en C/C++ ont été définis et implémentés dans la norme. Par ailleurs des *noeuds écrits en JavaScript*, peuvent être programmés et activés dans le graphe. C'est de cette manière que les fonctionnalités présentes dans la définition initiale de la norme peuvent être étendues de manière arbitraire. Nous allons voir comment cette fonctionnalité d'extension a été utilisée avec l'environnement Faust.

2. 2. Générer du JavaScript depuis le compilateur Faust

Un *backend* de génération de code JavaScript a été ajouté en 2012 à la branche de développement faust2. Il permet de traduire le code source DSP Faust en fichier JavaScript. Ce résultat est ensuite connecté à la Web Audio API grâce à un enrobage JavaScript générique qui instancie et active un noeud Web Audio (JavaScript node). Ce nouveau noeud peut alors être utilisé comme n'importe quel noeud audio dans la Web Audio API, et potentiellement connecté avec des noeuds natifs ou d'autres noeuds JavaScript pour définir un graphe de traitement audio plus complexe.

Le script nommé *faust2webaudio* est disponible pour créer des pages HTML à partir du code source DSP,

en enchaînant les étapes décrites précédemment et en créant au final un noeud JavaScript par page.

Le code JavaScript étant extrêmement lent à l'exécution, et à part pour des effets DSP assez simples, le résultat n'était pas réellement utilisable en l'état.

2. 3. Générer du code asm.js, un sous-ensemble typé et fortement optimisable de JavaScript

En 2011, pour faciliter le portage de code source C/C++ en JavaScript, des chercheurs de Mozilla ont développé *emscripten***,** un compilateur basé sur la technologie LLVM (14) qui produit du JavaScript à partir de code C/C++. Ils ont par ailleurs défini le langage *asm.js*, un sous-ensemble de JavaScript entièrement typé, qui peut être optimisé de manière très efficace par les compilateurs dynamiques embarqués dans les navigateurs Web. Il est possible d'atteindre alors des performances proches du code natif (15). Principalement destiné à manipuler des types simples (entiers et flottants), il est particulièrement adapté pour réaliser des calculs audio. Deux développements ont été réalisés avec cette approche :

- en utilisant la chaîne de compilation à base du compilateur *emscripten*. Dans le cas de Faust, il suffit alors d'utiliser la classe C++ générée par le compilateur à partir du fichier DSP, de compiler celle-ci grâce avec emscripten, pour produire au final un fichier JavaScript qui contient le code optimisé asm.js, ainsi que l'environnement d'exécution (*runtime*) ajouté automatiquement. Ce code est ensuite connecté à la Web Audio API grâce à un fichier JavaScript générique qui instancie et active un noeud Web Audio (JavaScript node). Ce nouveau noeud peut alors être utilisé comme n'importe quel noeud audio dans la Web Audio API.

- un *backend* de génération directe du code asm.js est en cours de développement dans la branche faust2. Il permettra de se passer totalement de la chaîne basée sur emscripten pour produire des pages HTML autonomes.

Le script nommé *faust2asm*génère uniquement le code JavaScript contenant le *runtime* emscripten, le code JavaScript de l'effet Faust, et l'enrobage pour connecter avec la Web Audio API. Par exemple :

faust2asm karplus.dsp

Une classe *faust.karplus* est alors définie. Des instances de cette classe peuvent alors être créées et utilisées comme des noeuds Web Audio habituels :

var DSP= faust.karplus(audio_context, buffer_size, update_handler);

DSP.start();

DSP.scriptProcessor.connect(DSP.scriptProcessor)

Update_handler est une fonction avec le prototype suivant (path_to_control, value) qui sera appelée par le code DSP lorsque les valeurs des contrôles en sortie (typiquement les « bargraphs ») ont changés.

L'interface utilisateur peut changer l'état interne de n'importe quel contrôle avec le code suivant, o) *path_to_control* désigne l'adresse du contrôleur, et *value* la valeur qui sera changée :

DSP.update(path_to_control, value);

Le script nommé *faust2webaudioasm*enchaîne les étapes nécessaires pour générer un noeud JavaScript

par page HTML.

Figure 6. Compressor - faust2webaudioasm

La description au format JSON de l'interface est accessible avec la fonction :

var json = DSP1.json() ;

La liste de tous les contrôleurs peut être connue avec la fonction suivante :

var control_list= DSP1.controls();

Pour les effets DSP avec entrées, le code client aura besoin d'activer les entrées dans le contexte audio et de lesconnecter connecter avec le noeud :

var audio_input= audio_context.createMediaStreamSource(device);

audio_input.connect(DSP1.scriptProcessor);

L'option *-comb* permet de générer plusieurs effets DSP Faust dans un seul fichier JavaScript, de manière à partager le *runtime* emscripten :

faust2asm -comb karplus.dsp freeverb.dsp zita_rev1.dsp

Modèle polyphonique : certains générateurs de signaux générés avec Faust (par exemple un modèle physique de code de piano) sont utilisés de manière polyphonique. Un fichier d'enrobage particulier a été développé dans cette perspective : plusieurs instances de DSP vont être automatiquement créées,

l'ensemble de tous les signaux générés seront mixés pour produire la sortie globale. Les générateurs individuels pourront alors être contrôlés de manière différenciée, par exemple par des commandes MIDI (keyOn, keyOff, contrôle). Voici la commande qu'il faut exécuter :

faust2webaudioasm -poly foo.dsp

L'effet DSP peut être alloué et utilisé avec le code suivant :

```
var DSP= faust.piano_poly(audio_context, buffer_size, update_handler);

DSP.start();

DSP.noteOn(channel, pitch, velocity);

DSP.noteOff(channel, pitch) ;
```

3. Simulation d'un clavecin, par Luc Faure et Laurent Pottier

3. 1. Le projet

Le Musée d'art et d'industrie de Saint-Étienne a entrepris la restauration d'un prestigieux clavecin qu'il possède dans ses collections depuis la fin du xix^{e} siècle. Selon les résultats d'études scientifiques préalables avant restauration, menées depuis 2007, il était impossible de le remettre en état de jeu. Ce clavecin est présenté au musée pour la première fois depuis 40 ans, du 20 septembre 2014 au 5 janvier 2015.

L'objectif de notre projet, par rapport à cette exposition, était de présenter au public, à travers une borne interactive, les phénomènes acoustiques produits par les différents éléments constitutifs d'un instrument à cordes pincées comme le clavecin et leurs interactions.

Cette étude a été organisée en trois axes :

- étude des mécanismes d'excitation des cordes (touche, sautereau, plectre, étouffoir) ;
- description du système de cordage, étude du comportement vibratoire d'une corde métallique, de cordes doublées, d'un ensemble de cordes couplées ;
- rôles de la table d'harmonie et de la caisse de résonance, couplage cordes-chevalet-table d'harmonie-caisse de résonance.

À ces trois axes ont été associés des dispositifs de synthèse par modèles physiques, présentés sur plusieurs pages, permettant de visualiser et d'entendre le résultat produit en temps réel selon les paramètres modifiés (caractéristiques du plectre, dimensions de la corde, dimensions du résonateur, matériaux utilisés, etc.) et les conditions initiales du dispositif.

Plusieurs techniques de modélisation ont été mise en oeuvre : synthèse par guides d'ondes, (université Stanford) et synthèse par modèles de résonance (IRCAM, CNMAT).

Le projet était destiné à être décliné en plusieurs versions :

- une version pour la borne interactive du musée exposée à côté du clavecin ;
- -une version pour smartphones et tablettes (application téléchargeable sur le Web) *en projet* ;
- -une version « HTML » pour le site internet du musée (en cours de finalisation [16]).

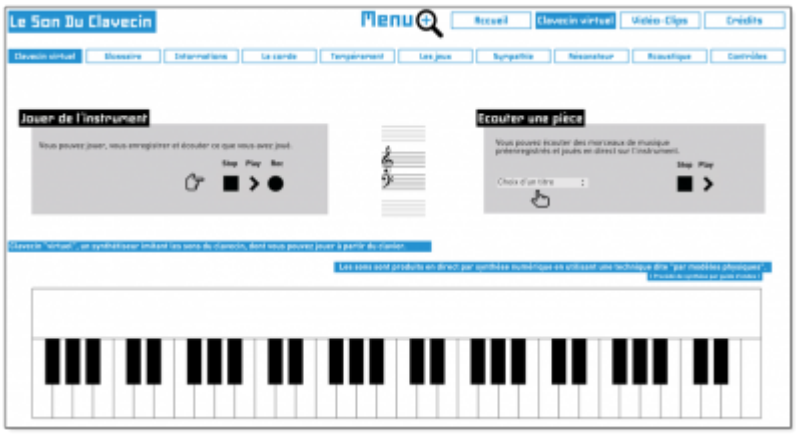
Ce travail a été réalisé en partenariat avec le Musée de la Musique de Paris [17] et le laboratoire Lutheries ? Acoustique ? Musique (LAM) de l'institut Jean le Rond d'Alembert [18]. Les séquences MIDI ont été enregistrées par Martial Morand [19], professeur de clavecin au conservatoire Massenet (CRR) de Saint-Étienne. Le GRAME a également été mis à contribution, notamment Stéphane Letz et Yann Orlarey (GRAME), pour leurs expertises sur l'environnement Faust et pour la réalisation de la version HTML-Javascript du clavecin.

3. 2. La version Faust pour Max/MSP

La borne interactive est basée sur un ordinateur PC [20], un écran 27 pouces tactile, un clavier MIDI 49 touches et une carte son. Elle utilise le langage Faust pour toute la synthèse et le traitement DSP et le programme Max/MSP pour l'interface utilisateur.

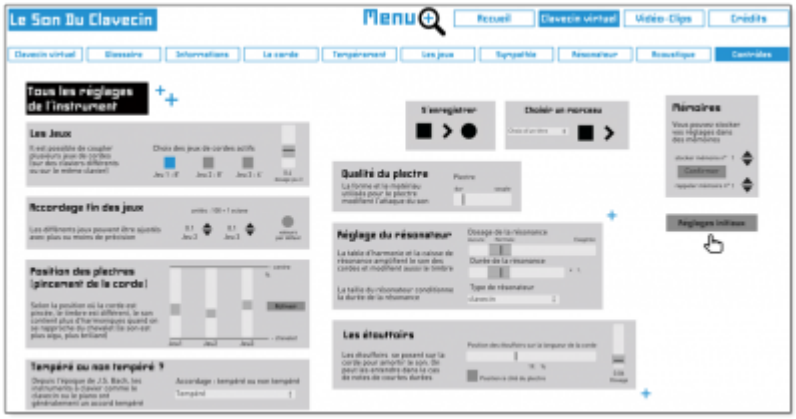
Le son du clavecin a été simulé par synthèse par modèle guide d'onde, mise au point dans les années 1980 au CNMAT par Julius Smith, en s'inspirant du modèle « harpsichord » développé dans les années 1990 dans le cadre du projet Synthbuilder et porté en 2011 pour Faust par Romain Michon [21].

Figure 7. La page d'accueil du clavecin simulé



La partie DSP est polyphonique et divisée en trois étages : excitation, vibration des cordes, résonance. La partie excitation, dispose des paramètres suivants : la fréquence fondamentale de vibration de la corde, l'angle d'attaque de la corde, la position de l'excitation (vers le chevalet ou vers le contre de la corde), et l'amortissement des étouffoirs. La partie vibration des cordes permet de combiner trois jeux de cordes, de les accorder, de régler le tempérament et de doser les amplitudes relatives des jeux. Enfin, la partie résonateur permet de doser l'amplitude du résonateur (cordage + table d'harmonie + caisse de résonance), d'en modifier la durée, et de choisir parmi plusieurs modèles de résonances analysés sur différents modèles d'instruments.

Figure 8. Page réunissant tous les contrôles sur l'instrument



Conclusion

Le Web peut devenir le centre névralgique du studio, permettant d'avoir à disposition à la fois tous les outils DSP nécessaires, d'y connecter toutes les interfaces de contrôles nécessaires et de communiquer à distance. Ces technologies sont amenées à se développer pour s'imposer comme de nouveaux standards pour le musicien.

Le langage Faust avait déjà montré toutes ses qualités pour la réalisation d'applications DSP professionnelles pour le temps réel (concision du code, efficacité CPU, documentation automatique offrant une garantie de préservation) lors du projet ANR ASTREE [(22)]. Il montre maintenant également une polyvalence d'applications, devenant disponibles sur tous supports, physiques et virtuels.

---

1. Projet regroupant ARMINES (Paris, responsable du projet), GRAME (Lyon), INRIA/IRISA (Rennes), CIEREC (Saint-Étienne).

2. http://faust.grame.fr/

3. http://puredata.info/

4. http://jackaudio.org/

5. Merci à ce groupe pionnier du master multitrack librement accessible : www.ninremixes.com/multitracks.php

6. Cf. 2. 2. Mise en oeuvre des protocoles communicants : HTTPD/OSC.

7. http://puredocs.bitbucket.org/faust2pd.html

8. http://www.gnu.org/software/libmicrohttpd/

9. Objet Pure Data pour interpolation de données sur une durée déterminée.

10. Via la libraire Oscpack de Ross Bencina.

11. http://opensoundcontrol.org/introduction-osc

12. Via l'activation des librairies de mrpeach et iemnet inclus dans Pure Data Extended.

13. Afin d'éviter d'entrer directement dans le navigateur la longue suite <`IP-machine-hôte`>:<`port-TCP-application`>, comme indiqué sur le bord droit des QR codes de l'illustration 5.

14. LLVM (*Low Level Virtual Machine*) est une infrastructure de compilateur. Elle est conçue pour compiler, éditer les liens et optimiser des programmes écrits dans un langage arbitraire.

15. Le code asm.js optimisé est alors environ 2 à 2.5 fois plus lent que le code natif équivalent.

16. Disponible à l'adresse : http://musinf.univ-st-etienne.fr/recherches/ClavecinHtml/index.html

17. Remerciements à Stéphane Vaiedelich (Musée de la Musique) pour ses réflexions pertinentes.

18. Remerciements à Jean-Loïc Le Carrou (LAM) pour son accueil et son expertise sur l'acoustique du clavecin.

19. Remerciements à Martial Morand pour ses conseils et la mise à disposition du clavecin du conservatoire.

20. Processeur Intel i7 - 4790k à 4.0Ghz // 4.4Ghz en Turbo // 4 coeurs // 8 threads // 8Go de Ram DDR3.

21. Romain Michon (2011), « Faust-STK : une bibliothèque de modèles physiques pour le langage FAUST », Actes des Journées d'informatique musicale (JIM-2011), Saint-Étienne, http://jim2011.univ-st-etienne.fr/html/actes/02_7_faust_stk_romainm.pdf.

22. 2007-2011, projet sur la préservation des outils temps réel, porté par l'IRCAM et faisant intervenir le GRAME, ARMINES et le CIEREC.

Pour citer ce document:
Laurent Pottier, « Faust : contrôle et DSP sur le Web », *RFIM* [En ligne], Numéros, n° 4 - automne 2014, Mis à jour le 27/10/2014
URL: http://revues.mshparisnord.org/rfim/index.php?id=343