Numéros / n° 6 - Techniques et méthodes innovantes pour l'enseignement de la musique et du traitement de signal

Â«Â Des outils pour enseigner FaustÂ Â»

Yann Orlarey

Résumé

Bien que l?enseignement ne soit pas la vocation première de Faust, c?est un élément fondamental pour son développement et sa diffusion. Cependant, enseigner un langage de programmation, compilé de surcroît, dans un département de musicologie ou à des enfants, pose un certain nombre de problèmes spécifiques. Comme le montre l?article, des stratégies particulières doivent être mises en place, y compris au niveau de l?écosystème d?outils proposés. Un travail important a donc été réalisé dans cette direction et Faust est désormais enseigné dans plusieurs universités. Depuis 2018 un cours en ligne est proposé par l?université Stanford sur la plateforme kadenze.com.

Introduction

Bien que le langage de programmation Faust n?ait pas pour vocation principale de faire l?objet d?un enseignement, c?est aujourd?hui devenu un élément fondamental pour son développement et sa diffusion. Par exemple, les aspects pédagogiques de Faust sont l?un des composants du projet de recherche FEEVER (ANR-13-BS02-0008), dont le consortium comprend des équipes de Mines ParisTech, GRAME, INRIA et UJM.

Faust est désormais enseigné dans plusieurs universités, soit en tant que tel, soit comme support pour l?introduction de techniques de synthèse et de traitement du signal. FaustPlayground, une version graphique simplifiée de Faust, a également été utilisé avec succès chez les enfants dans les écoles secondaires.

Enseigner Faust aux étudiants en musicologie est une tâche difficile. Mais voir les étudiants, sans expérience en informatique ou traitement du signal, être capables de développer des plugins audio, des synthétiseurs de sons ou des instruments électroniques pour les smartphones est très gratifiant.

Certains des défis sont inhérents à l?enseignement de tout langage de programmation. Par exemple, les étudiants peuvent confondre les paramètres formels et les paramètres effectifs. Mais d?autres sont dus à la nature compilée de Faust, ce qui est inhabituel pour les environnements de programmation audio. Les outils habituels d?un langage compilé, l?interface de ligne de commande, la chaîne de compilation, les dépendances, etc., sont par nature plus complexes à maîtriser.

C?est pourquoi, de nombreux efforts de recherche ont récemment été consacrés à simplifier les outils Faust et à combler le fossé avec les environnements de programmation audio interprétés. L?objectif principal de cet article est de donner un aperçu de l?écosystème de Faust utilisé dans l?enseignement.

1. Aperçu de Faust

Avant de regarder l?écosystème, rappelons certaines caractéristiques importantes de Faust (Orlarey *et al.*, 2004) (Functional Audio Stream), un langage de programmation fonctionnel, synchrone, spécialement conçu pour le traitement et la synthèse des signaux en temps réel. Les lecteurs déjà familiarisés avec le langage peuvent ignorer cette section.

1.1. Dédié au traitement du signal

Les musiciens utilisent des langages de programmation spécialisés (DSL) depuis Music III (Mathews, 1959) et MUSICOMP (Hiller & Baker, 1963). Aujourd?hui, les DSL musicaux comme Chuck, Csound, OpenMusic, Max, Puredata, Faust ou Supercollider, pour n?en nommer que quelques-uns, sont habituellement utilisés comme moyens créatifs par des musiciens de musique électronique, des ingénieurs du son et des compositeurs d?avant-garde.

Certains DSL musicaux sont orientés traitement du signal, d?autres sont orientés composition, et certains tentent de combiner les deux approches. Faust est dans la première catégorie, en mettant l?accent sur la conception de synthétiseurs, d?instruments de musique électronique, d?effets audio, etc. Cette approche ciblée permet à Faust de reposer sur un modèle simple, la notion de *processeurs de signaux*. Tout dans Faust est un processeur de signal, et la programmation en Faust consiste à composer des processeurs de signaux en utilisant une algèbre de cinq opérations de *composition* :

```
<: :> : , ~.
```

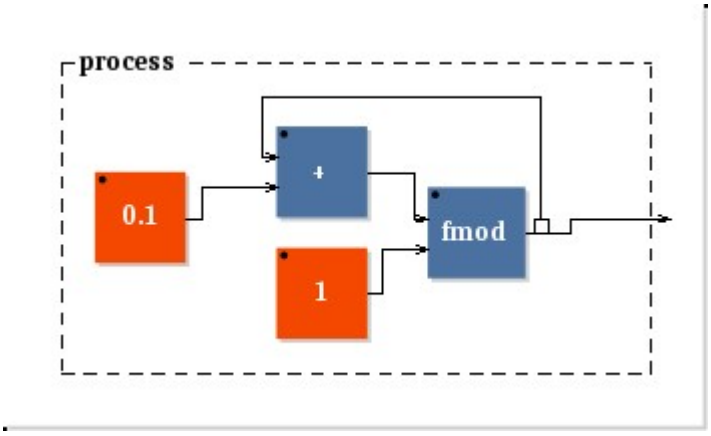
Afin de donner au lecteur un avant-goût du langage, voici la mise en ?uvre d?un oscillateur sinusoïdal en partant de zéro (évidemment, les oscillateurs existent prédéfinis dans les bibliothèques standards Faust).

1.1.1. Générateur de phase

Le composant de base est un générateur de phase. Un générateur de phase produit un signal de dents de scie périodique qui passe de « 0 » à « 1 » à chaque période. Par exemple, l?expression :

```
0.1 : (+,1.0:fmod) ~ _
```

où `fmod` est l?opération modulo en flottants, correspondant au schéma suivant :



produit le signal périodique :

{0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 0.0}...

Dans cette expression, `0.1` est l?incrément entre les échantillons successifs, et `fmod` est l?opération de reste à virgule flottante utilisée pour replier le signal entre `0` et `1`.

En contrôlant l?incrément, nous pouvons contrôler la fréquence du signal généré. Disons que le taux d?échantillonnage est de `48000` échantillons par seconde. En utilisant un incrément de `1/48000`, nous produirons un signal à `1` Hz, et en utilisant un incrément de `f/48000`, nous produirons à la fréquence `f` Hz.

Nous pouvons donc définir notre générateur de phase comme suit :

```
phasor(f) = f/48000 : (+,1.0:fmod) ~ _ ;
```

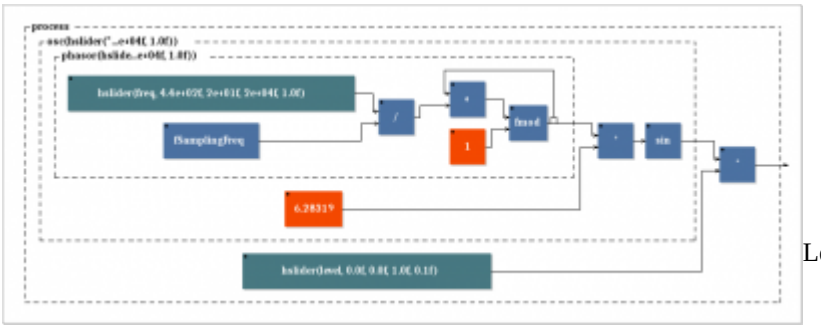
1.1.2. Oscillateur sinusoïdal

Une fois que nous avons un générateur de phase, il est facile de définir un oscillateur sinusoïdal en multipliant le signal de phase par 2 * PI et en prenant le sinus :

```
osc(f) = phasor(f) * 6.28318530718 : sin;
```

Nous pouvons compléter notre programme avec des curseurs pour contrôler la fréquence et le niveau de l?oscillateur :

```
process = osc(hslider("freq", 440, 20, 20000, 1))

            * hslider("level", 0, 0, 1, 0.1);
```



Le programme complet est représenté par le schéma de principe suivant :

1.2. Un langage pour les utilisateurs finaux

Les utilisateurs de DSL musicaux sont généralement des musiciens, des ingénieurs du son, des chercheurs, des réalisateurs en informatique musicale (RIM), etc. Ils ont parfois des connaissances en traitement du signal ou au moins une idée claire de la façon dont les effets audio et les systèmes de synthèse sonore devraient fonctionner. Mais ce ne sont pas nécessairement des informaticiens ou des développeurs professionnels. Le développement de logiciels audio en temps réel en C est habituellement hors de portée pour la plupart d?entre eux. L?ambition de Faust est de leur offrir une alternative de haut niveau, viable et efficace au langage C/C++, pour décrire et implémenter des instruments de musique performants, des effets audio et plus généralement des applications de traitement de signaux en temps réel.

1.3. Compiler au lieu d?interpréter

La plupart des DSL musicaux sont des langages interprétés. Pour amortir le coût d?interprétation, ces

langages fonctionnent sur un bloc d?échantillons audio au lieu d?échantillons individuels. Par conséquent, les opérations nécessitant de travailler au niveau de l?échantillon, par exemple les filtres IIR, ne peuvent pas directement être mises en ?uvre de manière efficace dans ces langages, et doivent être fournies en tant que primitives

1.4. Un langage de spécification de haut niveau

Faust est conçu pour être un langage de spécification de haut niveau plutôt qu?un langage permettant un contrôle fin de l?implémentation. Nous avons choisi de séparer clairement le rôle des utilisateurs, la spécification et le rôle du compilateur, chargé de la mise en ?uvre. La façon dont l?utilisateur écrit un programme Faust ne devrait pas être importante, seul son sens devrait compter. Deux programmes Faust différents, mais avec la même signification mathématique, devraient donner lieu à la même mise en ?uvre (bien que cela soit indécidable en général).

1.5. Sémantique simple et bien définie

Faust a une sémantique formelle simple et bien définie. Un programme Faust désigne un processeur de signal : une fonction continue qui transforme un n-uplet de signaux d?entrée à un m-uplet de signaux de sortie. Avoir une sémantique simple et bien définie n?a pas seulement un intérêt académique ; cela rend le langage plus facile à apprendre et à utiliser, et permet au compilateur Faust de faire des optimisations très avancées. En outre, cela permet des techniques de documentation et de préservation automatiques très utiles dans le domaine de l?informatique musique.

1.6. Approche purement fonctionnelle

La programmation fonctionnelle offre à Faust un haut niveau de modularité à la fois pour composer et comprendre les programmes Faust. En outre, cela offre un cadre très naturel pour le traitement du signal. Les signaux numériques périodiquement échantillonnés peuvent être modélisés en tant que fonctions du temps. Les processeurs de signaux sont des fonctions de second ordre opérant sur des signaux. L?algèbre de diagramme de blocs de Faust est un ensemble d?opérations de composition de troisième ordre sur les processeurs de signaux. Enfin, les fonctions définies par l?utilisateur sont des fonctions d?ordre supérieur sur les expressions du diagramme.

1.7. Syntaxe textuelle orientée blocs-diagrammes

Les DSL musicaux peuvent être grossièrement divisés en langages textuels (Csound, SuperCollider, Faust) et visuels (Max/MSP, PureData). Mais dans les deux cas, le concept de blocs-diagrammes, des blocs fonctionnels reliés par des signaux, est habituellement central. Faust est un langage textuel avec une syntaxe concise basée sur la notion de blocs-diagrammes. La syntaxe est construite sur une algèbre de cinq opérations de composition du processeur de signal. Elle est conçue pour favoriser la modularité et la composabilité des programmes. Ceux-ci peuvent être facilement traduits en blocs-diagrammes visuels.

1.8. Déploiement facile

Les musiciens doivent faire face à une grande variété de systèmes d?exploitation, d?environnements logiciels et d?architectures matérielles. Faust est conçu pour favoriser un déploiement simple de programmes sur toutes ces cibles en établissant une séparation claire entre le calcul lui-même, tel que décrit par le texte du programme, et la manière dont ce calcul devrait être lié au monde externe. Cette relation (avec les pilotes audio, l?interface graphique, les capteurs, etc.) est décrite dans les fichiers d?architecture spécifiques. Ces fichiers d?architecture sont essentiellement des enrobages pour le code généré par le compilateur. Actuellement, plus de quarante fichiers d?architecture sont fournis pour les principaux systèmes d?exploitation informatiques, ainsi que pour iOS, Android et certains systèmes embarqués.

2. Les outils Faust

Alors qu?à notre connaissance Faust était, en 2002, le premier langage de programmation audio entièrement compilé, il en existe désormais plusieurs autres, en particulier : Max/MSP Gen~, Heavy et Kronos (voir le texte de Vesa Norillo consacré à Kronos dans ce numéro). Mais, si les compilateurs ont l?avantage de l?efficacité, ils ont leurs propres inconvénients par rapport aux interprètes. Les compilateurs ont traditionnellement besoin d?une chaîne d?outils complète (compilateur, linker, bibliothèques de développement, etc.). Pour les non-programmeurs, utiliser l?ensemble de ces outils est une tâche complexe.

Le cycle de développement, depuis l?édition du code source vers une application en cours, est beaucoup plus long avec un compilateur qu?avec un interprète. Cela peut être un problème dans des situations créatives où une expérimentation rapide est essentielle. De plus, le code binaire n?est généralement pas compatible entre les plates-formes et les systèmes d?exploitation, alors qu?un patch Max/MSP s?exécute par exemple sur toutes les plateformes où Max/MSP est disponible.

Les deux tables suivantes résument les avantages et les inconvénients des langages interprétés et compilés :

Tableau . Langages interprétés

Avantages	Inconvénients
Faciles à installer	Exécution lente
Boucle Édition/Exécution rapide	Calcul par vecteurs
Code Source portable	Le code ne peut être déployé que là où existe un interprète

Tableau . Langages compilés

Avantages	Inconvénients
Exécution rapide	Boucle Édition/Compilation/ Exécution lente
Sémantique de calcul à l'échantillon	Code binaire non portable
Code généré facile à déployer	chaîne de compilation complexe à installer (avec beaucoup de dépendances)

Nous avons développé divers outils pour essayer de résoudre ces inconvénients :

- simplifier le flux de travail de compilation ;

- accélérer la boucle Édition/Compilation/Exécution ;
- avoir des outils autonomes faciles à installer ;
- proposer des outils Web basés sur l?installation à zéro.

Dans les sections suivantes, nous examinerons ces solutions.

2.1. Les outils en ligne de commande

Le compilateur Faust traduit un programme DSP source en un programme impératif équivalent. L?utilisateur peut choisir la langue cible parmi C, C++, Java, JavaScript, LLVM, asm.js et WebAssembly. En outre, l?utilisateur peut spécifier le *fichier d?architecture* à utiliser.

Un *fichier d?architecture* définit de quelle manière relier le calcul audio, décrit par le programme Faust, au monde externe. Il prend en charge les pilotes audio, l?interface utilisateur graphique, les protocoles de contrôle, etc. Cette séparation entre le calcul audio lui-même et sa relation avec le monde extérieur permet de déployer exactement le même code Faust sur une grande variété de plateformes matérielles et logicielles.

Mais l?ensemble du processus de compilation peut être assez complexe à maîtriser en raison de nombreux détails, options de compilation, bibliothèques à utiliser, etc. C?est pourquoi plusieurs scripts `faust2xxx` ont été développés. Ces scripts, construits « au dessus » de la commande `faust`, automatisent entièrement le processus de construction d?un fichier exécutable pour une cible donnée : des plugins VST aux applications Android natives. Plus de cinquante cibles différentes sont actuellement disponibles (voir les scripts de la Table 3).

Tableau . Scripts Faust

faust2alqt	faust2gen	faust2max6	faust2rpialsaconsole
faust2alsa	faust2graph	faust2md	faust2rpinetjackconsole
faust2alsaconsole	faust2graphviewer	faust2msp	faust2sig
faust2android	faust2ios	faust2netjackconsole	faust2sigviewer
faust2api	faust2iosKeyboard	faust2netjackqt	faust2supercollider
faust2asmjs	faust2jack	faust2octave	faust2svg
faust2au	faust2jackconsole	faust2owl	faust2vst
faust2bela	faust2jackinternal	faust2paqt	faust2vsti
faust2caqt	faust2jackserver	faust2pdf	faust2w32max6
faust2caqtios	faust2jaqt	faust2plot	faust2w32msp
faust2csound	faust2javaswing	faust2png	faust2w32puredata
faust2dssi	faust2ladspa	faust2puredata	faust2w32vst
faust2eps	faust2lv2	faust2raqt	faust2webaudio
faust2faustvst	faust2mathdoc	faust2ros	faust2webaudioasm
faust2firefox	faust2mathviewer	faust2rosgtk	faust2webaudiowasm

Par exemple, faust2caqt convertit un programme Faust en une application pour MacOS basée sur CoreAudio pour la partie audio, avec une interface utilisateur basée sur Qt.

Bien que ces scripts soient très utiles pour les développeurs, il est préférable de ne pas les utiliser avec les étudiants, car ces-derniers ont besoin d?une introduction minimale sur la façon d?utiliser le terminal. En outre, la boucle Edit/Compile/Run utilisant des scripts est lente. Par exemple, la production d?une application CoreAudio Qt utilisant faust2caqt wind.dsp prend 13 secondes sur un MacBook Air. Mais ils ont leur valeur pédagogique pour aider les élèves à comprendre ce qui se passe réellement sous le capot. Ils sont également très utiles lorsqu?il s?agit d?automatiser des tâches complexes de compilation.

2.2. FaustWorks

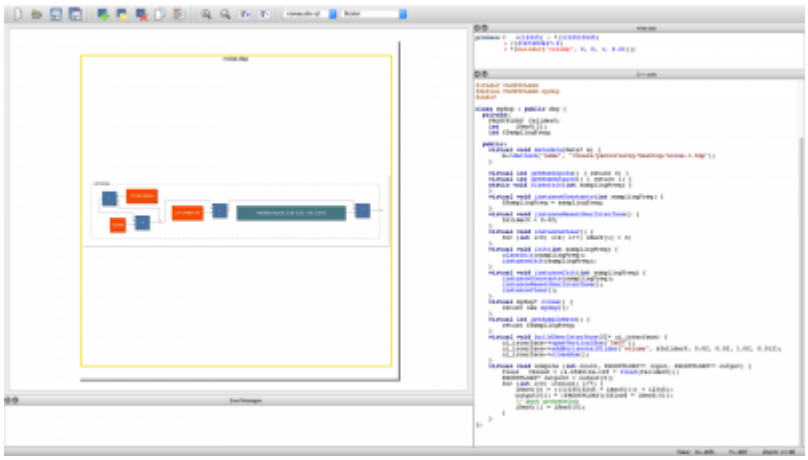
FaustWorks est un environnement de développement intégré (IDE) pour Faust. Il offre une alternative à la ligne de commande, simplifie grandement la boucle Édition/Compilation/Exécution et fournit des outils de visualisation avancés.

L?espace de travail de FaustWorks est organisé en quatre parties :

- un éditeur de code avec surlignage syntaxique où l?utilisateur peut éditer ses programmes Faust ;
- une « scène graphique » où tous les programmes édités de Faust sont visualisés sous la forme de diagrammes de blocs ;
- la visualisation du code C++ généré ;
- un volet d?erreur.

La représentation de blocs et la génération de code C++ sont calculées à la volée, à mesure que l?utilisateur écrit le programme. Cela rend FaustWorks très pratique pour débuter avec Faust. L?utilisateur peut mieux comprendre la syntaxe en regardant le diagramme de bloc généré et mieux comprendre la sémantique en regardant la traduction C++.

Figure . FaustWorks



La barre de menu permet de choisir entre les différentes architectures et les options de compilation. Elle est personnalisable : l?utilisateur peut créer de nouveaux éléments en les associant à des scripts externes.

En cliquant avec le bouton droit de la souris sur un diagramme dans la scène graphique, l?utilisateur peut sélectionner différentes actions. Par exemple, il peut ouvrir le diagramme correspondant dans un navigateur Web pour l?explorer en détail, générer de la documentation mathématique ou obtenir d?autres formes de représentations graphiques d?un programme Faust.

De nombreuses interactions avec FaustWorks se font par glisser-déposer. Faire glisser et déposer un fichier Faust externe dans FaustWorks déclenchera plusieurs actions. Le code source est ouvert dans l?éditeur de code, le diagramme est affiché dans la scène graphique, la traduction C++ est également affichée et le code binaire est généré selon l?architecture cible choisie.

FaustWorks est un excellent outil pour les élèves qui commencent avec Faust. C?est aussi un excellent outil pour expliquer le fonctionnement du compilateur Faust et le processus de génération de code. Mais le cycle Edit/Compile/Run est encore lent car il repose sur une chaîne de compilation traditionnelle Faust/C++. En outre, une telle chaîne de compilation peut être complexe à installer pour les utilisateurs inexpérimentés.

2.3. Libfaust, un compilateur Faust embarqué

Pour surmonter la complexité et le coût global d?une chaîne de compilation traditionnelle, nous avons développé une version embarquable du compilateur Faust : libfaust, avec un backend LLVM spécifique (Letz *et al.*, 2013). Cette bibliothèque est utilisée conjointement avec la bibliothèque de compilateurs JIT de LLVM pour créer une solution complète en mémoire qui transforme le code Faust en pointeurs vers du code machine exécutable.

Il y a trois étapes essentielles pour utiliser *libfaust* :

- compilation du code Faust dans une « usine » (factory) DSP ;
- instanciation d?un objet DSP à partir d?une « usine » DSP ;
- performance du processus DSP.

La première étape utilise l?une des deux fonctions de création, à partir d?un fichier ou d?une chaîne de caractères :

```
llvm_dsp_factory*

createDSPFactoryFromFile (

      const std::string& filename,

      int argc, const char *argv[],

      const std::string& target,

      std::string& error_msg, int opt_level = 3);

llvm_dsp_factory*

createDSPFactoryFromString (

      const std::string& name_app,

       const std::string& dsp_content,

      int argc, const char *argv[],

      onst std::string& target,

      std::string& error_msg, int opt_level = 3);
```

Dans les deux cas, une « usine » DSP est créée (l?équivalent de la classe en C++) qui contient le code machine exécutable en mémoire, prêt à être exécuté dans un programme. Pour ce faire, un objet DSP, avec son propre état interne, doit être instancié à partir de « l?usine » :

```
   llvm_dsp* createDSPInstance(llvm_dsp_factory* factory);
```

L?objet DSP dispose d?un certain nombre de méthodes publiques pour le manipuler :

```
class llvm_dsp : public dsp {
```

```
public:

      virtual int getNumInputs();

      virtual int getNumOutputs();

      virtual void init (int samplingFreq);

      virtual void buildUserInterface (UI* inter);

      virtual void compute (int count, FAUSTFLOAT** input,

                            FAUSTFLOAT** output);

};
```

Une fois qu?un objet DSP est créé, il peut être exécuté en invoquant sa méthode *compute()*. Le calcul produira un bloc de *count* échantillons pour chaque canal de sortie, tout en mettant à jour l?état interne de l?objet.

Libfaust est facile à utiliser et à intégrer. Plusieurs applications musicales audio s?appuient sur cette librairie pour permettre la programmation en Faust. Certains d?entre eux sont décrits dans les paragraphes suivants :

- *FaustLive*, une application multi-plate-forme autonome pour compiler et exécuter des programmes Faust.
- *Faustgen~*, un plugin externe pour compiler et exécuter des programmes Faust dans l?environnement de programmation Max/MSP.
- *Faust for Csound*, un ensemble d?opcodes Csound 6 pour compiler, exécuter et contrôler des programmes Faust dans des scores et instruments Csound.
- *Faust4Processing*, une bibliothèque permettant de compiler, d?exécuter et de contrôler les programmes Faust à partir de Processing, un IDE pour les artistes visuels basé sur Java.

D?autres éléments de composition/programmation musicale qui intègrent libfaust, ou fournissent des fonctionnalités similaires, méritent d?être mentionnés :

- *Antescofo* (Gubbins *et al.*, 2016), le moteur de suivi de partition de l?IRCAM, avec ses extensions expérimentales pour compiler, exécuter et contrôler les programmes Faust intégrés à la partition.
- *FaucK* (Wang, Michon, 2016), un Chugin pour l?environnement de programmation de musique ChucK. Il permet de compiler, d?exécuter et de contrôler les programmes de Faust en utilisant les mécanismes précis de synchronisation et de simultanéité de ChucK.
- *OpenMusic* (Bresson, Bouche, 2013), comprend des objets et des fonctions pour écrire, compiler et contrôler des programmes Faust.
- *pd-faust* (Graef, 2016) est un environnement dynamique pour exécuter des dsps Faust dans Pd en utilisant Pure, un langage de programmation fonctionnel basé sur la réécriture de termes.
- *pMIX* (https://github.com/olilarkin/pMix2), un interpolateur prédéfini, un encodeur plug-in et Faust IDE dans un plugin VST.

- *INScore* (Fober *et al.*, 2017), un environnement pour la conception de partitions musicales augmentées et interactives qui intègre notamment des signaux et leur représentation comme objet de premier niveau de la partition.

2.4. FaustLive

FaustLive essaie de réunir la simplicité d?utilisation d?un langage interprété autonome avec l?efficacité d?un langage compilé. Avec son compilateur Faust intégré, basé sur la bibliothèque libfaust, FaustLive propose un cycle de compilation et d?exécution ultra-court, tout en étant simple à installer et ne nécessitant aucun compilateur externe, chaîne d?outils de développement ou SDK à exécuter.

Les programmes Faust peuvent être compilés et exécutés à la volée simplement par glisser-déposer, non seulement concernant les fichiers .dsp de Faust, mais aussi les fragments de code, les fichiers audio et même les URL indiquant les fichiers source Faust publiés sur le Web.

FaustLive fournit des fonctionnalités avancées pour accélérer le cycle de développement. Par exemple, pendant qu?une application Faust est en cours d?exécution, il est possible d?éditer et de recompiler son code à la volée, sans interruption du son. FaustLive réalisera un fondu enchaîné audio entre l?ancienne et la nouvelle version. En outre, si l?application utilise JACK comme pilote, toutes les connexions audio sont maintenues.

Une autre caractéristique intéressante est la possibilité de migrer une application en cours d?exécution d?une machine à l?autre via le réseau, même dans tous les systèmes d?exploitation. Les applications peuvent également être contrôlées à distance, en utilisant MIDI, HTTP ou OSC.

FaustLive intègre son propre serveur Web qui permet de contrôler tous les programmes en cours d?exécution. Chaque programme a sa propre URL qui vient d?ouvrir dans un navigateur Web pour voir une reproduction de l?interface de l?application permettant de contrôler toutes ses fonctions à distance.

Figure . FaustLive

FaustLive fournit également un support MIDI avec des capacités polyphoniques. Cette fonctionnalité simplifie grandement la mise en ?uvre d?un synthétiseur polyphonique, car il suffit de décrire une seule voix. Le gestionnaire polyphonique prend automatiquement en charge l?allocation dynamique des voies, ainsi que la réception et l?interprétation des messages de contrôle MIDI.

Lorsqu?une connexion Internet est disponible, FaustLive peut utiliser le service de compilation en ligne

Faust pour produire des applications et des plugins natifs pour les différentes plateformes prises en charge. L?utilisateur choisit parmi les différentes options du gestionnaire d?exportation celles qui lui conviennent et appuie sur le bouton d?exportation. Le code Faust est envoyé au service de compilation, qui renvoie la version binaire du programme à FaustLive.

FaustLive est un excellent outil pour commencer avec Faust. Il est facile à déployer en classe ou en atelier. Il comble l?écart entre les langages interprétés et compilés.

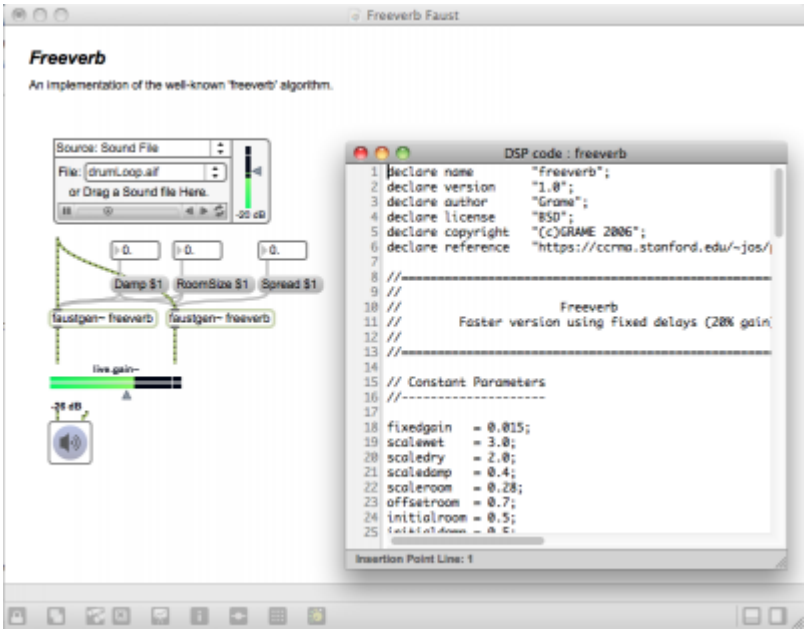
2.5. Faustgen

Faustgen~ est un objet externe pour Max/MSP qui incorpore le compilateur Faust. En s?appuyant sur libfaust et LLVM, faustgen~ permet de compiler et d?exécuter dynamiquement les programmes Faust dans un plugin externe. Le code source des programmes peut être édité directement dans un éditeur intégré à faustgen~. L?exécutable natif est compilé automatiquement lorsque l?éditeur est fermé. Il est ensuite inséré dans le graphe de calcul audio de Max/MSP.

Les noms et les caractéristiques des paramètres de contrôle sont imprimés dans la console Max et l?utilisateur doit ensuite créer et connecter les éléments graphiques (curseurs, boutons, champs numériques?) nécessaires pour contrôler le processus audio. L?utilisateur peut être assisté dans cette tâche par un script qui permet de générer automatiquement une interface de contrôle simplifiée qui peut ensuite être éditée manuellement.

Chaque objet faustgen~ contient potentiellement un programme Faust différent. Pour partager le même algorithme, l?objet faustgen~ doit être nommé et il peut alors être dupliqué pour créer autant d?instances différentes. Si le programme d?un faustgen ~ object nommé est modifié, toutes les instances de même nom sont mises à jour.

Figure . FaustGen~, compilateur Faust embarqué dans Max



Le programme source en format texte ainsi que les options de compilation utilisées sont enregistrés dans le patch. Pour optimiser les temps de rechargement en sauvegardant certaines des étapes de compilation, une version binaire (format Bitcode LLVM) du code LLVM intermédiaire est également enregistrée.

Faustgen~ est un outil très utile avec les étudiants en particulier s?ils apprennent déjà Max/MSP. Il fournit un outil de prototypage très puissant avec une boucle Edit/Compile/Run ultrarapide qui bénéficie de toute

l?infrastructure Max/MSP pour tester et valider les programmes Faust.

2.6. Opcodes Faust pour Csound

Avec Csound 6, un ensemble de quatre opcodes construits sur libfaust est disponible pour compiler, exécuter et contrôler le code Faust directement à partir de Csound. (Lazzarini, 2014)

2.6.1. Faustcompile

L?opcode `faustcompile` invoque le compilateur just-in-time pour produire un processus DSP instantiable à partir d?un programme Faust. Il compilera un programme Faust à partir d?une chaîne, contrôlée par différents arguments. Les chaînes multi-lignes sont acceptées, en utilisant `{{}}` pour délimiter la chaîne.

2.6.2. Faustaudio

L?opcode `faustaudio`instancie et exécute un programme Faust compilé. Cela fonctionnera avec un programme compilé avec `faustcompile`

2.6.3. Faustctl

L?opcode `faustctl` ajuste les contrôles UI dans une instance Faust DSP. Il mettra un contrôle donné dans un programme Faust en cours d?exécution.

2.6.4. Faustgen

L?opcode `faustgen` compile, instancie et exécute un programme Faust compilé. Il invoquera le compilateur just-in-time en i-time, et instantiera le programme DSP. Au temps de perf-time, il exécutera le code Faust compilé.

Voici un exemple impliquant des opcodes `faustgen` et `faustctl`, un programme simple qui ajuste le gain d?une entrée :

```
idsp,a1 faustgen {{

      gain = hslider("vol",1,0,1,0.01);

      process = _ * gain;

}}, ain1)

faustctl idsp, "vol", 0.5
```

Ces opcodes Faust sont un outil très utile avec les étudiants surtout s?ils apprennent déjà Csound. La combinaison de Csound pour la composition musicale et audio avec Faust pour la mise en ?uvre effective de nouveaux algorithmes DSP offre un environnement de musique informatique très riche et pédagogiquement significatif.

2.7. Faust4processing

Processing est un langage de programmation basé sur Java et disposant d?un environnement de développement intégré (IDE) spécialement conçu pour les arts électroniques et visuels. La bibliothèque Faust4Processing, construite avec libfaust, permet aux utilisateurs de modifier la compilation et exécuter des programmes Faust à la vitesse native à partir de l?environnement de traitement.

L?API est assez simple. Après avoir importé les bibliothèques nécessaires

```
import faustProcessing.*; // to compile a Faust programs

import controlP5.*;  // to build the user interface
```

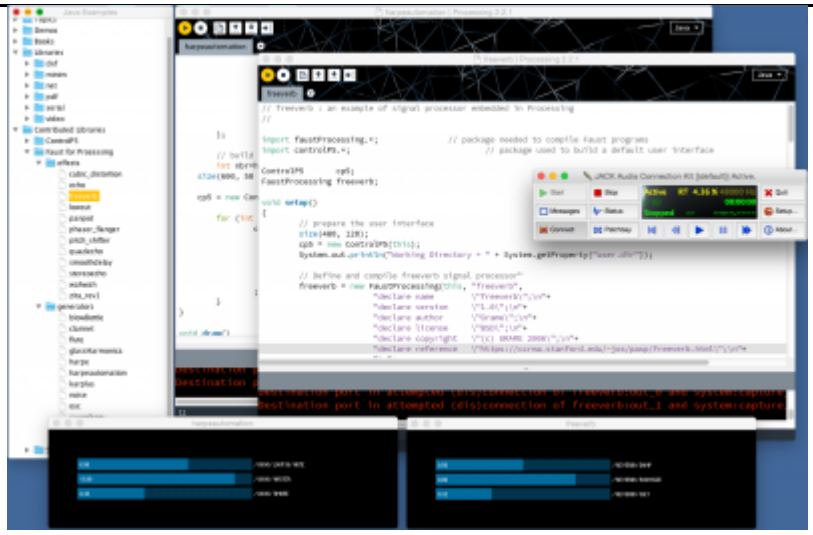
l?utilisateur crée des objets FaustProcessing en passant une chaîne de caractères représentant un programme Faust :

```
FaustProcessing echo;

echo = new FaustProcessing ( this, "echo",

                "process = +~(@(44100):*(0.75);"

            );
```

Ceci est suffisant pour compiler automatiquement et exécuter le programme Faust correspondant en tant que client JACK indépendant. L?activité du programme peut être contrôlée en utilisant les méthodes `start ()`et `stop ()`. Le programme peut être interrogé pour découvrir ses paramètres de contrôle en utilisant `String getJSON ()`pour avoir une description globale, ou par paramètre, en utilisant une série de méthodes get/set.

L?image suivante est une capture d?écran d?une session de traitement avec deux programmes Faust en action : harpeautomation et freeverb. Les deux programmes sont fournis dans le dossier des exemples de faust4processing. L?interface utilisateur des deux programmes est générée automatiquement, en utilisant controlP5, en interrogeant les objets. Les deux programmes fonctionnent comme des clients JACK et sont interopérables avec d?autres clients JACK. Nous n?avons pas suffisamment de points de vue pour juger de la pertinence pédagogique de Faust4Processing. Néanmoins, nous sommes convaincus que disposer de la puissance de Faust pour synthétiser et traiter du son manipulé par ailleurs dans l?environnement Processing devrait intéresser cette communauté.

Figure . Faust4Processing en action



2.8. Online compiler

Tous les outils que nous avons présentés jusqu?à présent nécessitent une installation plus ou moins complexe. L?installation à zéro d?outils basés sur le Web, même s?il existe des problèmes mineurs d?incompatibilité, constitue une alternative intéressante pour les utilisateurs occasionnels, les ateliers et les classes, à condition qu?une bonne connexion Internet soit disponible.

The Faust Online Compiler (http://faust.grame.fr/oninecompiler) (Michon & Orlarey, 2012) est le premier outil que nous avons développé dans cette direction. C?est une application Web basée sur PHP/JavaScript qui permet de modifier et de compiler les programmes Faust pour la plupart des plates-formes prises en charge directement dans un navigateur.

Le cycle de travail est organisé autour de cinq onglets :

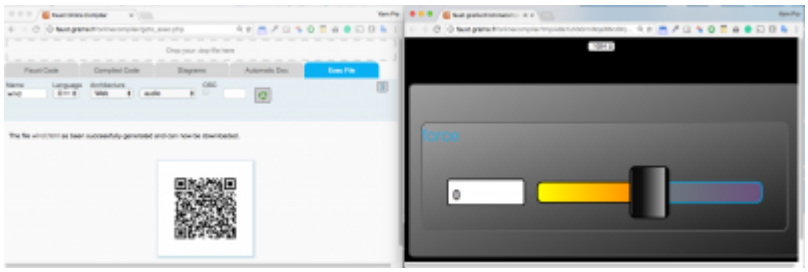
- Faust Code, l?éditeur de code Faust avec la mise en surbrillance de syntaxe est le point de départ. L?utilisateur peut glisser et déposer les fichiers de code Faust existants à partir de son bureau ou démarrer un nouveau fichier à partir de zéro.
- Le code compilé permet de visualiser le résultat de la compilation. Par défaut, le compilateur produit un code C++ mais l?utilisateur peut choisir d?autres langues comme C, Java, JavaScript et LLVM.
- Ces diagrammes sont utilisés pour visualiser une représentation graphique du programme. Par défaut, il fournit une représentation par blocs, mais d?autres options sont disponibles. Ces diagrammes sont très utiles pour comprendre les programmes Faust, spécialement pour les débutants, mais pas seulement.
- La génération automatique de documentation est une fonctionnalité unique de Faust. Elle permet de générer une documentation PDF de la sémantique d?un programme Faust dans un ensemble d?équations mathématiques. Ce document peut être utilisé à des fins de préservation.
- Exec File est utilisé pour créer des applications exécutables ou des plugins du code Faust. L?utilisateur a le choix entre plates-formes et architectures, des plugins Windows VST, des pages Web ou des applications Android.

Figure . Faust Onlinecompiler, compilateur Faust en ligne

La sortie des pages Web est une extension récente du compilateur en ligne. Les pages Web générées utilisent l?API Web Audio et asm.js (un sous-ensemble efficace de JavaScript) pour produire une expérience audio raisonnablement temps réel sur le Web.

L?illustration suivante est une copie d?écran montrant la compilation d?un simulateur de vent avec une page Web sur le côté droit et l?exécution de ce simulateur de vent comme une application audio sur le côté droit.

Figure . Exécution du code compilé directement dans le web



Il est même possible de créer des synthétiseurs polyphoniques basés sur le Web qui peuvent être joués avec un clavier MIDI externe. Dans ce cas, l?utilisateur ne décrit qu?une seule voix du synthétiseur et le fichier d?architecture prend en charge la gestion des messages MIDI et l?attribution vocale dynamique selon la polyphonie.

Le compilateur en ligne est très utile dans les salles de classe et pour les ateliers, car il fonctionne sur tout navigateur récent prenant en charge HTML 5 et l?API Web Audio. Mais l?efficacité du code généré, pour la cible web/audio, dépend de la façon dont asm.js est pris en charge dans la machine virtuelle du navigateur.

Lorsque asm.js est bien pris en charge, par exemple dans Firefox, les programmes Faust ne sont que 3 à 4 fois plus lents que le code natif, ce qui est tout à fait acceptable. Mais dans tous les navigateurs testés, l?exécution est loin d?être en temps réel. Les enregistrements audio se produisent assez souvent, spécialement lorsque le code est « froid ».

Avec les versions récentes de Chrome, il n?est plus possible d?utiliser le microphone sur une connexion non https. Il s?agit actuellement d?un problème car le compilateur Faust en ligne et FaustPlayground ne sont accessibles que par des connexions non sécurisées. De l?autre côté, MIDI n?est actuellement pris en charge qu?en mode natif dans Chrome.

2.9. FaustPlayground

FaustPlayground (http://faust.grame.fr/faustplayground) est une plate-forme Web conçue pour permettre aux enfants d?apprendre une programmation audio de base de manière simple et agréable. En particulier, il leur permet de développer des instruments de musique pour les smartphones Android en assemblant et en connectant des modules audio.

La mise en ?uvre de FaustPlayground est basée sur l?API Web Audio, asm.js et une version JavaScript de libfaust appelée libfaust.js produite en utilisant le compilateur emscripten C ++ vers asm.js. Cette bibliothèque permet de compiler les programmes Faust directement dans le navigateur de l?utilisateur, sans nécessiter de serveur (un serveur sera utilisé uniquement pour exporter des programmes vers d?autres architectures). L?application comprend une collection intéressante de modules prédéfinis que l?utilisateur peut sélectionner dans le menu biblio. Ces exemples prédéfinis sont organisés en trois catégories :

- instruments et générateurs de sons ;
- les effets, comme les réverbérations, les filtres, etc.
- des exemples prédéfinis combinant des instruments et des effets.

Figure . FaustPlayground

FaustPlayground peut également compiler et exécuter des programmes externes. Il suffit de supprimer le code source d?un programme Faust dans FaustPlayground pour qu?il soit compilé et exécuté comme les modules prédéfinis. Si nécessaire, le code d?un module peut être édité en appuyant sur l?icône du stylo en bas à gauche du module. Le module est recompilé à la volée dès que l?éditeur est fermé.

L?interface utilisateur du module FaustPlayground est simplifiée par rapport à celle qu?on trouve habituellement dans les applications Faust. Tous les curseurs sont des curseurs horizontaux et la seule disposition est verticale. Mais les couleurs, rouge, vert, bleu des curseurs indiquent qu?un curseur est associé à un accéléromètre particulier. Le rouge correspond à l?axe X, vert à l?axe Y, bleu à l?axe Z. La couleur blanche indique les curseurs non assignés.

Ces assignements peuvent être facilement édités en choisissant EDIT dans la barre de menus et en cliquant sur l?un des curseurs. Une boîte de dialogue s?ouvre pour décrire précisément le mappage entre les accéléromètres et le curseur. Lorsque FaustPlayground s?exécute sur une tablette, les curseurs peuvent être directement animés par les mouvements de la tablette. En outre, l?application a été optimisée pour exploiter les dispositifs multitouch.

La fonctionnalité EXPORT donne accès aux mêmes cibles et architectures que FaustLive car les deux utilisent le même service de compilation à distance.

Figure . Export d?une application depuis FaustPlayground

Comme avec FaustLive, l?utilisateur sélectionne la cible et l?architecture à utiliser et appuie sur le bouton d?exportation. Le code source est envoyé au service de compilation à distance et après un certain temps (l?exportation d?Android peut prendre plusieurs minutes), un QRCode apparaît indiquant que la compilation est terminée. Dans le cas d?Android, l?utilisateur doit simplement faire un clic sur le QRCode pour télécharger et installer l?application résultante sur son smartphone.

FaustPlayground a été utilisé avec succès depuis 2015 avec des enfants de cours de musique dans l?enseignement secondaire en France. Mais il est également utile pour les étudiants et les utilisateurs expérimentés comme un outil de prototypage rapide. Il est souvent utilisé conjointement avec le compilateur en ligne.

2.10 Faust service

Faustservice (http://faustservice.grame.fr/) est un compilateur distant. Il n?est pas utilisé directement, mais il est essentiel pour que FaustLive et FaustPlayground exportent des programmes Faust pour des cibles et des architectures spécifiques.

Le service est basé sur une API de type *restful*. Le client démarre habituellement la session en obtenant la liste des cibles possibles avec l?aide d?une URL spécifique, http://faustservice.grame.fr/targets. Il reçoit une réponse au format JSON décrivant les cibles et les architectures disponibles. Le client envoie ensuite le code source à compiler et reçoit en retour une clé SHA unique pour le programme. Avec cette clé, le client peut forger plusieurs URL en fonction du service à utiliser. Par exemple, l?URL http://faustservice.grame.fr/shakey/osx/vst/binary.zip sera utilisée pour compiler le programme correspondant en tant que plugin VST pour OSX.

Conclusion

Nous avons présenté un aperçu de l?écosystème de Faust. Cet écosystème a évolué au cours des années pour combler l?écart avec les environnements de programmation audio interprétée sur trois aspects clés :

- simplification du processus de compilation ;
- accélération de la boucle Édition/Compilation/Exécution ;
- mise à disposition d?outils autonomes et faciles à installer.

Grâce à la technologie libfaust, nous avons maintenant des outils tels que FaustLive qui sont faciles à installer et qui offrent une boucle d?édition/compilation/exécution plus rapide qu?une chaîne de

compilation traditionnelle. Plus récemment, nous avons investi dans la technologie Web, qui a permis le développement d?outils en ligne comme FaustPlayground, particulièrement adapté à des utilisations éducatives. Nous espérons que ces outils contribueront efficacement à la diffusion de Faust.

Pour citer ce document:
Yann Orlarey, Â« Des outils pour enseigner Faust Â», *RFIM* [En ligne], Numéros, n° 6 - Techniques et méthodes innovantes pour l'enseignement de la musique et du traitement de signal, Mis Ã jour le 10/07/2018
URL: http://revues.mshparisnord.org/rfim/index.php?id=566