FAUST-STK : UNE BIBLIOTHÈQUE DE MODÈLES PHYSIQUES POUR LE LANGAGE FAUST

Romain MICHON
Université Jean Monnet
CIEREC, EA 3068
F-42023, Saint-Etienne, France ;
Stanford University (CCRMA)
rmnmichon@gmail.com

RÉSUMÉ

Le *FAUST-Synthesis ToolKit* est une bibliothèque d'objets écrits dans le langage FAUST implémentant un certain nombre de modèles d'instruments de musique basés principalement sur des algorithmes utilisant les techniques de synthèse par guide d'onde et de synthèse modale. La plupart des objets créés sont inspirés de fonctions et de patches issus du *Synthesis ToolKit* et du programme *SynthBuilder*.

Une attention particulière a été portée sur la dimension pédagogique des objets créés. Ainsi, le code FAUST de chaque modèle a été optimisé afin d'être le plus expressif possible. Chaque algorithme est commenté de manière précise et fait régulièrement référence à des données bibliographiques.

Certains des modèles implémentés ont été modifiés dans le but d'être utilisés avec des données de suivi de geste. Une démonstration de ce type d'utilisation des objets du *FAUST-Synthesis ToolKit* est faite dans le programme *PureData*.

Enfin, les résultats d'un certain nombre de tests de performances des différents codes C++ générés par FAUST sont présentés.

1. INTRODUCTION

Le *FAUST Synthesis ToolKit* a été développé dans le cadre du projet ASTREE [1] portant sur la préservation des œuvres de musique électroacoustique. Il est constitué d'un ensemble d'objets programmés dans le langage FAUST [2] implémentant un certain nombre d'instruments basés pour la plupart sur des modèles physiques. En effet, les distributions actuelles de FAUST, bien que très fournies en filtres et en effets [3] ne proposaient aucune librairie d'objets constitués de synthétiseurs.

Les algorithmes utilisés dans le *FAUST Synthesis ToolKit* sont pour la plupart issus du *Synthesis ToolKit (STK)* [2] et du programme *SynthBuilder* [8].

Le *STK* est développé depuis 1996 par Perry R. COOK et Gary P. SCAVONE au CCRMA [4] de l'université de Stanford (USA) en collaboration avec le département d'informatique de l'université de Princeton (USA) et le CAML [5] de l'université McGill (Canada). Il est constitué d'un ensemble de classes écrites en C++, implémentant divers procédés de synthèse et de traitement de signaux sonores. Ces classes peuvent alors être intégrées à n'importe quel programme (plug-ins pour *MaxMSP* ou pour *PureData*, application pour Iphone, etc.).

De son côté, *SynthBuilder* a été l'un des principaux programme utilisé au CCRMA entre 1995 et 1999 pour la synthèse des sons. Ainsi, une grande partie des travaux de recherche dans le domaine de la modélisation physique d'instruments de musique menée à cette époque à l'université de Stanford a été implémentée dans ce programme. La plupart de ces modèles physiques sont basés sur la technique des guides d'ondes numériques bien que certains utilisent aussi la synthèse modale.

Une attention particulière a été portée sur la qualité du code FAUST. En effet, tous les algorithmes sont commentés et documentés en faisant régulièrement référence à des liens bibliographiques externes. Chaque modèle a par ailleurs été écrit de manière à rendre le code le plus compact et le plus efficace possible tout en gardant une cohérence au niveau de sa compréhension. Enfin, il faut préciser que les paramètres de chaque instrument ont été normalisés à l'échelle de l'ensemble du *FAUST-STK* et que l'interface utilisateur a été organisée de manière à rendre la manipulation des modèles la plus agréable et la plus simple possible.

Dans un premier temps, une présentation des différents instruments implémentés dans le *FAUST-STK* sera faite. Les difficultés rencontrées lors de la phase de programmation et les solutions adoptées seront alors exposées. Un exemple complet d'utilisation fera également l'objet d'une partie. Enfin, nous terminerons en faisant un bilan des

1 . Projet soutenu par l'Agence Nationale de la Recherche (ANR-08-CORD-003) : `http://www.grame.fr/Recherche/Programmes/` (lien vérifié le 02/03/2011).

2 . FAUST est un est un langage de programmation fonctionnel proposant une approche abstraite au traitement numérique du signal. Il est développé au GRAME (Groupe de Recherche en Acoustique et en Musique Electronique) de Lyon depuis 2002 : `http://faust.grame.fr/` (lien vérifié le 02/03/2011).

3 . Bibliothèques « filter.lib » et « effect.lib ».

4 . Center for Computer Research in Music and Acoustics.

5 . Computational Acoustic Modeling Laboratory.

performances des codes C++ générés par FAUST pour le *FAUST-STK*.

2. MODÈLES UTILISANT LA TECHNIQUE DES GUIDES D'ONDES NUMÉRIQUES

La synthèse par guide d'onde a été inventée par Julius O. SMITH dans les années quatre-vingt [14]. Cette variante de l'algorithme de synthèse des sons de cordes pincées de KARPLUS et STRONG [6] permet de décrire tout type de cordes, de tubes où de corps résonnants avec un réseau de lignes de délais et de filtres.

Le langage FAUST s'est révélé être particulièrement adapté pour l'implémentation d'algorithmes utilisant cette technique, notamment à cause de son caractère non séquentiel. En effet, elle est basée sur la création de réseaux de signaux audios sur lesquels un certain nombre de traitements (filtres, délais, etc.) est appliqué. Elle permet d'obtenir des résultats de bonne qualité pour la modélisation d'instruments à cordes et à vent. Les percussions peuvent également être modélisées avec cette technique.

Un certain nombre d'instruments utilisant la technique des guides d'ondes numériques a été implémentés dans le *FAUST-STK* :

- deux clarinettes ;
- deux flûtes ;
- un saxophone ;
- deux modèles de cuivres ;
- une bouteille en verre dans laquelle on souffle ;
- une barre métallique ;
- une barre en bois ;
- un bol tibétain ;
- une plaque en verre.

Une présentation des travaux effectués au niveau de l'implémentation de ces instruments dans le langage FAUST est faite dans les sous-parties suivantes.

2.1. Instruments à vent

Les algorithmes de synthèse utilisés dans le *FAUST-STK*, bien que très proches de ceux de *SynthBuilder* et du *Synthesis ToolKit*, ont subi un certain nombre de modifications dans le but de les rendre le plus optimum possible notamment vis-à-vis de la sémantique de FAUST (Cf. partie 7 concernant les performances et l'optimisation du code du *FAUST-STK*). Dans le cas des instruments à vent, une structure générique utilisable pour l'ensemble des modèles a été définie et est présentée dans la figure 1.

Les fonctions disponibles dans les différentes bibliothèques de FAUST ont été utilisées autant que possible dans les objets créés. Toutefois, la majorité des filtres du *STK* on dû être réécrits dans la mesure où ces derniers différaient légèrement de ceux de la bibliothèque `filter.lib` de la distribution de FAUST. Ce type de fonctions, commun à l'ensemble des instruments du *FAUST-STK* a été placé dans une bibliothèque : `instrument.lib`.

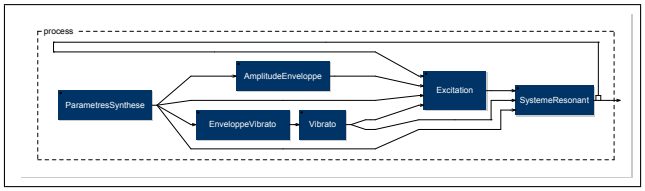

Figure 1: Structure générique des instruments à vent utilisant la synthèse par guide d'onde dans le *FAUST-STK*. Dans la pluspart des cas, les enveloppes d'amplitude sont du type Attaque-Déclin-Maintien-Relachement.

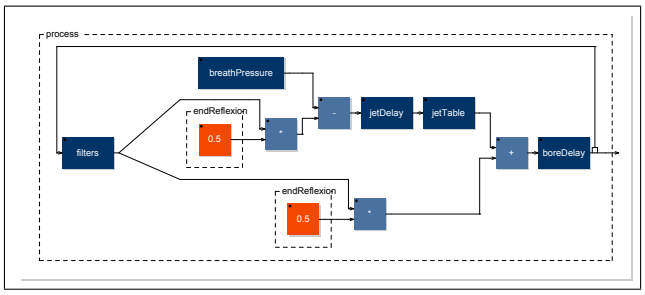

Figure 2: Schéma de l'algorithme de l'instrument `flutestk.dsp` généré dans FAUST avec le programme *faust2svg*. La boite *filters* contient un filtre passe-bas et un DC blocker connectés en série.

2.1.1. Flûtes

Deux modèles de flûte ont été implémentés dans le *FAUST-STK*. Le premier, issu du *Synthesis ToolKit* est basé sur une version très simplifiée du modèle présenté dans [16]. Deux lignes de délai connectées en série dans une boucle simple sont utilisées. L'effet d'atténuation du son dans cette boucle (damping) est produit par un filtre de type passe-bas du premier ordre connecté à un bloqueur DC (Cf. figure 2).

Les sons produits par cet algorithme étant peu satisfaisants, un certain nombre de modifications lui a été apporté (Cf. figure 3) :

- création d'une boucle autour de la deuxième ligne de délais ;
- utilisation d'un filtre passe-bas de type « butterworth » pour l'atténuation du son dans les lignes de délais [6] .

2.1.2. Clarinettes

Deux modèles de clarinettes basés sur ceux du *STK* ont été implémentés dans le *FAUST-STK*. Ils utilisent tous les deux l'algorithme présenté dans [12].

Le premier modèle (Cf. figure 4) implémente une simple ligne de délai dont le feedback est traité par un filtre passe-bas du premier ordre. Une fonction non-linéaire générant un signal reproduisant le son issu du bec de la clarinette est utilisée pour exciter le système.

Le deuxième modèle de clarinette implémenté dans le *FAUST-STK* est plus complet que celui présenté précé-

6 . Filtre « butterworth » du neuvième ordre défini dans la bibliothèque `filter.lib` de la distribution de FAUST.

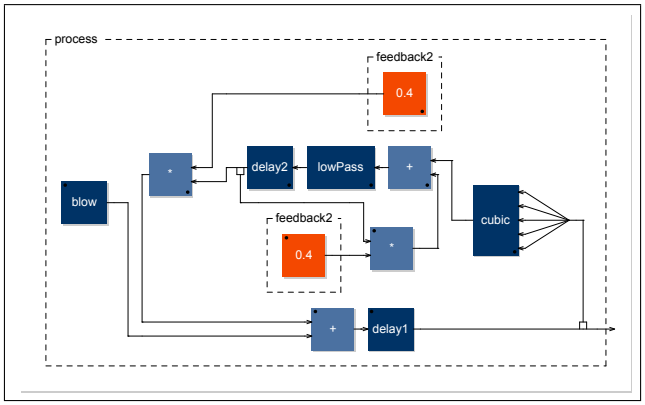

Figure 3: Schéma de l'algorithme de l'instrument `flute.dsp` généré dans FAUST avec le programme *faust2svg*.

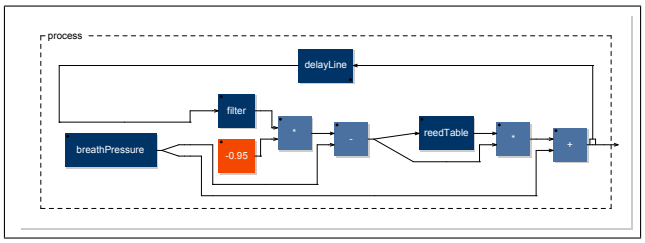

Figure 4: Schéma de l'algorithme de l'instrument `clarinet.dsp` généré dans FAUST avec le programme *faust2svg*.

demment. Il inclut un modèle de « trou » permettant de modifier la fréquence du son produit de façon plus naturelle. En effet, comme il est expliqué dans [14], la hauteur des sons générés dans les modèles simples de guide d'onde est contrôlée par la longueur des lignes de délais utilisée en nombre d'échantillons. En d'autres termes, dans le cas de la clarinette présenté dans la figure 4, la fréquence du son produit est contrôlée en modifiant dynamiquement la longueur de son tube, ce qui serait impossible dans la réalité.

2.1.3. Saxophone

Un modèle d'instrument pouvant être apparenté à un saxophone issu du *STK* a également fait l'objet d'une implémentation. En fait, d'un point de vue physique, il s'agit plutôt d'un instrument à corde excité par le son provenant d'un bec de clarinette. En déplaçant le point d'excitation le long de la corde, il est possible d'obtenir des sons allant du violon à la clarinette en passant par le saxophone. Ce modèle est très semblable à celui présenté dans la figure 4 à la différence qu'il utilise une ligne de délais supplémentaire.

2.1.4. Cuivres

Deux modèles de « cuivres » pouvant être apparentés à n'importe quel instrument de cette famille (trompette, trombone, cor, etc.) sont disponibles dans le *FAUST-STK*.

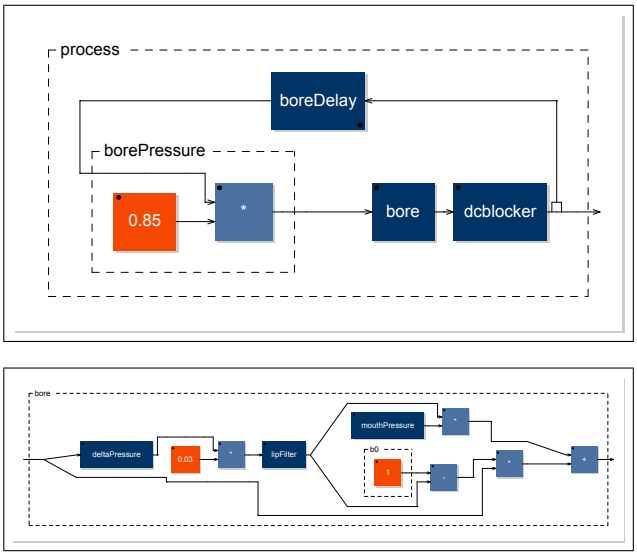

Figure 5: Schémas de l'algorithme de l'instrument `brass.dsp` généré avec le programme *faust2svg*. La figure supérieure représente l'algorithme dans son ensemble. La figure inférieure donne le détail de la boite « bore » de la figure supérieure.

Le premier est très similaire à celui implémenté dans le *Synthesis ToolKit*. Il est basé sur une simple ligne de délais mise en boucle dont le feedback est filtré par un bloqueur DC. Le son produit par la vibration des lèvres de l'instrumentiste dans l'embouchure est simulé en utilisant une version simplifiée de la technique présentée dans [10] qui consiste à mettre au carré et à saturer le signal de sortie d'un filtre biquadratique. Dans le modèle de cuivre dont il est ici question, ce filtre est alimenté par le signal de retour de la ligne de délais comme le montre la partie inférieure de la figure 5.

Une difficulté relative à ce modèle est que ses paramètres physiques (notamment la tension des lèvres et la pression du souffle) doivent-être modifiés dynamiquement en fonction de la nature de l'instrument (trompette, trombone, etc.) et de la fréquence de la note jouée. Si ce type de paramètres reste fixe, il est fort probable que l'instrument reste muet ou que le son généré ne soit pas vraiment celui d'un cuivre. Des travaux futurs pourraient permettre de résoudre ce problème en implémentant une règle de calcul qui ajusterait la tension des lèvres et la pression en fonction de la nature de l'instrument, de la fréquence et de l'amplitude du son désiré.

Un autre modèle de cuivre basé sur un patch écrit dans le programme *SynthBuilder* a été implémenté dans *FAUST-STK*. Il permet d'obtenir des sons moins agressifs que ceux de l'instrument présenté précédemment et peut-être utilisé sans avoir besoin de modifier les paramètres physiques dynamiquement. Son algorithme est basé sur deux boucles contenants chacune une ligne de délais. Le signal de retour de ces boucles est traité avec des filtres d'ordres plus élevés que ceux de l'autre modèle de cuivre.

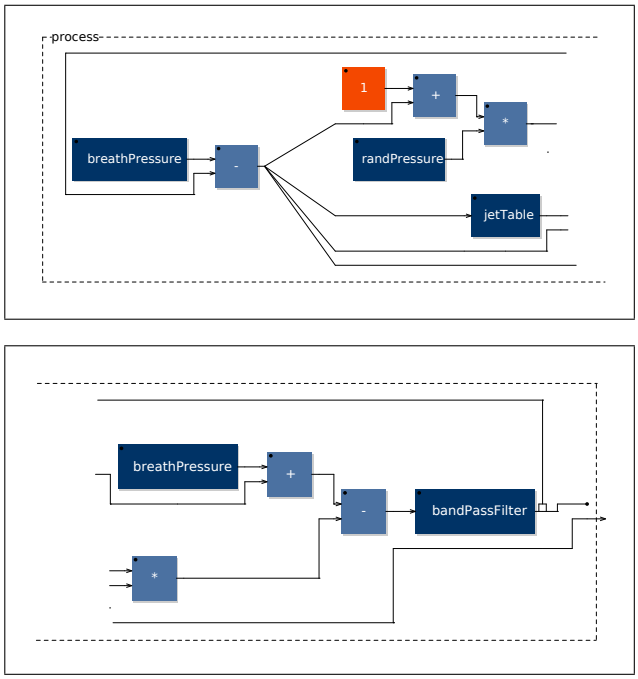

Figure 6: Schéma de l'algorithme de l'instrument `blowBottle.dsp` généré avec le programme *faust2svg*. Le schéma a été sectionné en deux parties : la suite de la première figure se trouve en dessous de celle-ci. La boite « randPressure » contient un générateur de bruit blanc sur lequel l'enveloppe d'amplitude désirée est appliquée. « jetTable » simule un effet « d'embouchure » sur le son produit.

2.1.5. *Autres instruments à vent*

Un instrument issu du *STK* permettant d'obtenir des sons allant de la flûte de pan à ceux d'une bouteille dans laquelle on soufflerait à travers le goulot a été implémenté. Dans ce modèle très simple, l'excitation est générée par du bruit blanc traité par la même fonction que celle simulant l'embouchure de la flûte dans la figure 3 (Cf. [16]). Le signal est ensuite passé dans un filtre passe-bande qui permet de déterminer la fréquence du son produit (Cf. figure 6).

Enfin, un certain nombre d'instruments à vent utilisant des filtres non-linéaires passifs au sein de réseaux de guides d'ondes est en cours de développement. Le *FAUST-STK* devrait donc être prochainement complété par un ensemble d'instruments à anches doubles tel qu'un hautbois, une cornemuse ou encore une bombarde.

2.2. Instruments à cordes

Julius O. SMITH ayant déjà travaillé sur des modèles avancés d'instruments à corde dans FAUST [13], seul un nombre réduit de ce type d'instrument a été implémentés dans le *FAUST-STK* qui contient donc une sitare, un piano (Cf. partie 4) et un instrument à cordes frottées.

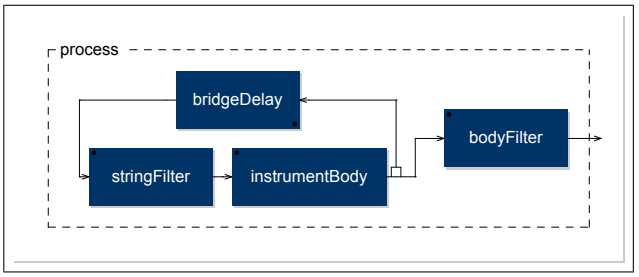

Figure 7: Schémas de l'algorithme de l'instrument `bowed.dsp` généré avec le programme *faust2svg*.

2.2.1. *Cordes frottées*

FAUST-STK dispose d'un instrument à cordes frottées basé sur celui disponible dans le *STK* lui même construit en s'inspirant de l'algorithme présenté dans [14]. Il permet d'imiter le son d'un certain nombre d'instruments : violon, violoncel, etc.

Ce modèle physique utilise deux boucles possédant chacune une ligne de délais. Le système est excité par une fonction générant un signal sonore semblable à celui d'un archet frotté sur une corde. Le signal de retour de la boucle est filtré par un filtre passe-bas du premier ordre qui permet de simuler l'effet d'atténuation du son de la corde. Un filtre biquadrique traite le son issu de la boucle afin d'imiter la réponse impultionnelle de la caisse de résonance de l'instrument (Cf. figure 7).

2.2.2. *Sitar*

Le modèle de Sitar présent dans le *FAUST-STK* est inspiré de celui du *Synthesis ToolKit* bien qu'il soit légèrement différent sur certains points.

Il est basé sur un simple Karplus-Strong (Cf. [6]) dont la taille de la ligne de délais est modulée de façon aléatoire dans le but de rendre le comportement du modèle de corde non-linéaire. Cette opération a nécessité l'utilisation de la fonction `smooth` de la bibliothèque `filter.lib` qui permet d'interpoler des valeurs :

```
delayLength = targetDelay*((1+(0.5*noise))
    : smooth(0.999));
delayLine = delay(4096,delayLength);
```

L'atténuation du son de la corde (damping) est simulée par un filtre de type passe-bas comme le montre la figure 8.

2.3. Percussions

Quatre percussions utilisant la technique des guides d'ondes par bandes sont implémentées dans le *FAUST-STK* :

- une barre en bois ;
- une barre métallique ;
- un bol Tibétain ;
- une plaque en verre.

Chacun de ces modèles peut-être excité soit par une mailloche (impulsions) soit par un archet (la fonction implémentant

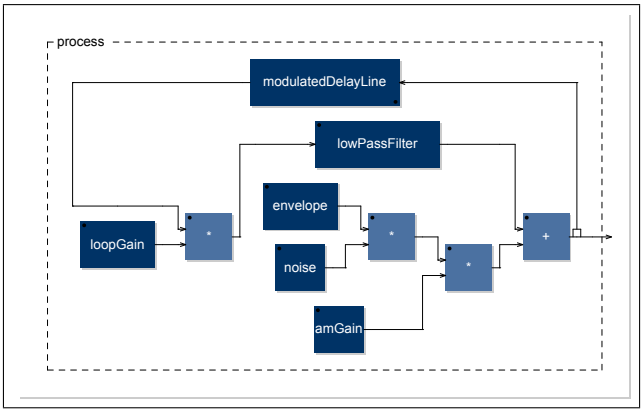

Figure 8: Schéma de l'algorithme de l'instrument `sitar.dsp` généré avec le programme *faust2svg*.

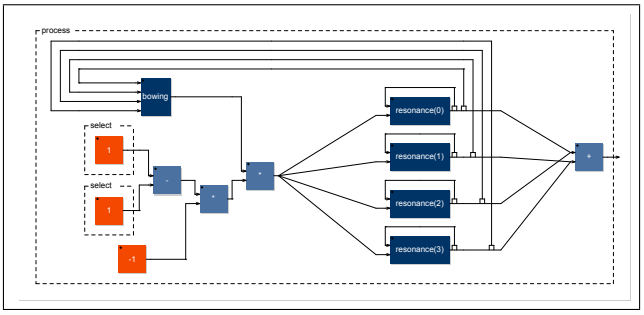

Figure 9: Schéma de l'algorithme de l'instrument `tunedBar.dsp` généré avec le programme *faust2svg*. Les boites « resonance(n) » contiennent chacune une ligne de délais et un filtre passe-bande connectés en série.

l'archet du violon présenté dans 2.2.1 est réutilisée). Les résonances sont créées par un ensemble de boucles contenant chacune une ligne de délais et un filtre passe-bande comme le montre la figure 9. Le nombre de boucles utilisé dépend donc du nombre de résonances désirées qui est différent pour les quatre instruments mentionnés ci-dessus.

3. MODÈLES UTILISANT LE SYNTHÈSE MODALE

Le *FAUST-STK* contient un modèle physique utilisant la technique de synthèse modale. Ce dernier permet d'imiter le son d'un nombre important d'instruments de musique.

La synthèse modale a été découverte par Jean-Marie ADRIEN dans les années quatre-vingt (Cf. [1]). Cette technique est assez similaire à celle présentée dans la partie 2.3. En effet, elle consiste à exciter une banque de résonateurs avec une impulsion (Cf. figure 10). L'implémentation de ce modèle dans FAUST a posé un certain nombre de problèmes et a nécessité l'utilisation du mecanisme de « fonctions étrangères ».

La première difficulté rencontrée a concerné l'importante quantité de paramètres à manipuler. En effet, chaque mode est caractérisé par une fréquence, une largeur de bande et une amplitude. Par conséquent, plus le nombre

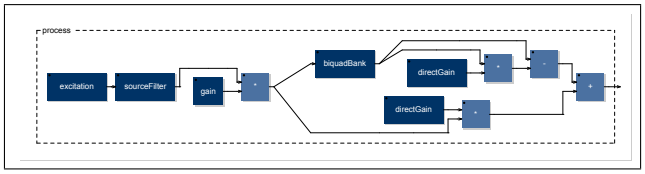

Figure 10: Schéma de l'algorithme de l'instrument `modalBar.dsp` généré avec le programme *faust2svg*. La boite « sourceFilter » implémente un filtre de type passe-bas. La boite « biquadBank » contient un nombre variable de filtres résonants.

de modes utilisés est grand, plus la quantité de paramètres à gérer est importante. Ainsi, les données modales utilisées ont été stockées dans un fichier C++ sous la forme d'une fonction dont les arguments sont plusieurs indexes et qui retourne la valeur correspondante.

L'autre problème rencontré a concerné l'importation d'un court fichier audio dans FAUST pour la création de tables de fonctions. En effet, dans le cas de l'instrument ici présenté, le banc de résonateur doit être excité par une impulsion dont la forme d'onde est enregistrée dans un fichier audio. Ainsi, plusieurs fonctions utilisant la bibliothèque `libsndfile` de Erik de Castro Lopo [9] ont d'abord été écrites. Néanmoins, celles-ci n'ont finalement pas été utilisées à cause des problèmes de compatibilité rencontrés avec les différentes architectures de FAUST.

Les tables d'ondes utilisées dans le *STK* ayant toutes une taille inférieure ou égale à 1024 échantillons, la solution retenue a consisté à extraire des fichiers audios les valeurs brutes de chaque échantillon et de les stocker dans des chaines de flottants C++. Par la suite, une fonction similaire à celle décrite précédemment permet l'importation de la valeur de ces échantillons dans FAUST. Ces derniers peuvent alors être utilisés avec la primitive `rdtable` qui crée et lit des tables de fonctions dans FAUST.

4. LE CAS PARTICULIER DU PIANO

4.1. Contexte

Bien que la plupart des algorithmes contenus dans les patchs de synthèse du programme *SynthBuilder* ait été porté (et bien souvent simplifié) dans le *Synthesis ToolKit*, certains d'entre-eux, la plupart du temps plus complexes, sont restés dormant dans les machines NeXT vieillissantes du CCRMA de Stanford. L'un d'entre eux implémente un modèle de piano utilisant la version « commutée » de la technique des guides d'ondes (Cf. [15]).

En 2006, ce patch a en partie été réécrit dans le *STK* par Stephen Sinclair à l'université McGill de Montréal [11]. Son travail a notamment permis d'extraire de *SynthBuilder* les importantes quantités de paramètres nécessaires à la synthèse (coefficients de filtres en fonction de la note jouée, etc.) et de les stocker sous la forme de fonctions C++.

Ainsi, la version du « piano commuté » disponible dans le *FAUST-STK* utilise ces banques de paramètres grace

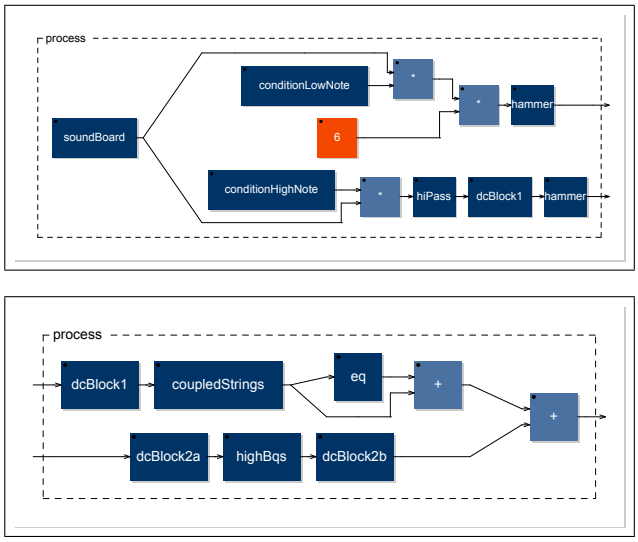

Figure 11: Schémas de l'algorithme de l'instrument `piano.dsp` généré avec le programme *faust2svg*. La figure supérieure représente la première partie de l'algorithme, la figure inférieure la suite. La boite « highBqs » contient les quatre filtres résonants connectés en série.

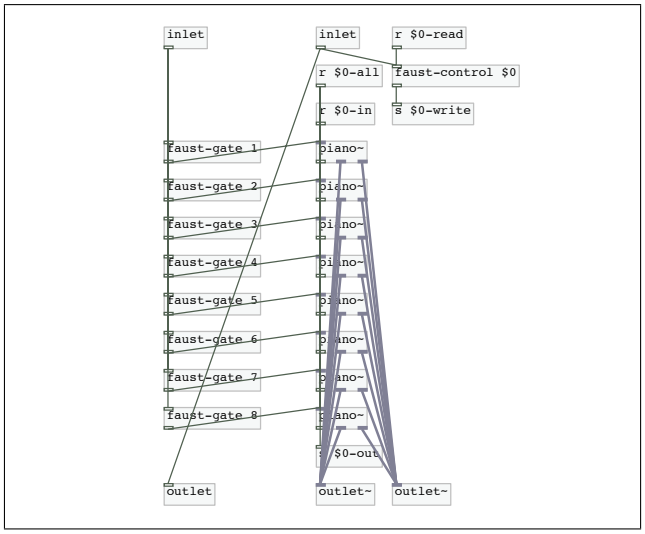

Figure 12: Extrait du sous-patch généré par *faust2pd* implémentant un piano polyphonique en utilisant le plug-in créé à partir de `piano.dsp`. Dans le cas présent, la polyphonie est de huit notes, il y a donc huit instances du plug-in « piano~ ».

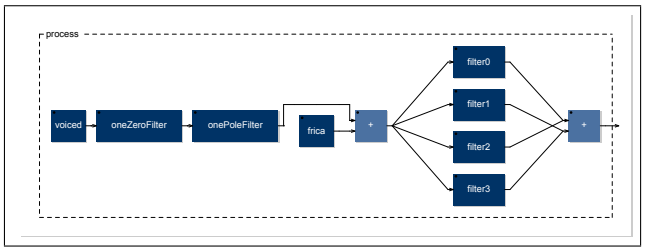

Figure 13: Schéma de l'algorithme de l'instrument `voiceForm.dsp` généré avec le programme *faust2svg*. La boite « voiced » contient la table d'onde lue en boucle produisant le son d'une source vocale. La boite « frica » contient le générateur de bruit blanc pour la production des sons de type fricatif.

au mécanisme de « fonction étrangère » de FAUST (de la même manière que dans le cas de la synthèse modale présenté dans la partie 3).

Les trois dernières machine NeXT du CCRMA de Stanford encore état de marche n'étaient hélas pas assez puissantes pour faire fonctionner le patch de *SynthBuilder* ce qui a fortement compliqué la phase de test.

4.2. Le modèle

Deux algorithmes différents permettent de calculer les sons pour l'ensemble de la tessiture du piano. En effet, les notes en dessous de *Mi6* (*Mi6* y compris) sont produites avec la technique des guides d'ondes commutées décrite dans [15]. De leur côté, les notes au-dessus de *Mi6* sont générées par un ensemble de quatre filtres résonants connectés en série (Cf. figure 11).

Le piano implémenté dans le *FAUST-STK* n'est pour l'instant que monophonique. Néanmoins, il est totalement compatible avec le programme *faust2pd* d'Albert GRAEF [4] qui permet de générer automatiquement des patchs pour *PureData* possédant une interface MIDI à partir de code FAUST. Une option donnée lors de la compilation permettant de créer des synthétiseurs polyphoniques peut être utilisée pour pallier au problème mentionné précédemment.

5. SYNTHÈSE DE LA VOIX CHANTÉE AVEC UN MODÈLE EXCITATION/FILTRES

Un modèle très simple de synthétiseur vocal s'inspirant d'un instrument du *Synthesis ToolKit* est implémenté dans le *FAUST-STK*. Une table contenant la forme d'onde d'un son imitant une source vocale est lue en boucle à différentes vitesses en fonction de la fréquence désirée. Le son produit est d'abord traité par un bloqueur DC avant d'être envoyé dans une banque de quatre filtres passe-bande qui permettent de créer des formants. Les sons produits de cette manière sont dits « voisés » et permettent de générer des voyelles. Les sons de type fricatif (consonnes) sont créés en traitant le son d'un générateur de bruit blanc dans la banque de filtre décrite précédemment (Cf. figure 13).

Afin de ne pas provoquer de discontinuités lors du passage d'un phonème à un autre, une interpolation linéaire des paramètres des filtres de formants est exécutée en utilisant la fonction `smooth` de la bibliothèque `filter.lib` (de la même manière que pour l'exemple de la Sitar donné dans la partie 2.2.2).

Les paramètres des formants sont stockés dans une fonction C++ pouvant être utilisée dans FAUST via le méchanisme de « fonction étrangère » (Cf. partie 3). Les données formantiques de plus de 32 phonèmes sont disponibles.

6. EXEMPLE D'UTILISATION : CONTRÔLE GESTUEL DES PARAMÈTRES DU VIOLON DANS *PUREDATA*

La gestion des paramètres est un élément d'une importance cruciale lors de l'utilisation de modèle physique. En effet, bien qu'il soit tout-à-fait possible d'utiliser les modèles présentés dans les parties précédentes avec des valeurs fixes pour les paramètres physiques, la qualité des sons générés peut-être largement améliorée en modifiant dynamiquement pour chaque note les valeurs de ces paramètres.

Esteban MAESTRE du MTG [7] de l'université Pompeu Fabra de Barcelone a travaillé dans le cadre de son doctorat sur la modélisation de la gestuelle instrumentale du violon [7]. Avec son aide, il a été possible de modifier légèrement l'algorithme de l'instrument à cordes frottées présenté dans la partie 2.2.1 afin de le rendre compatible avec des données gestuelles. Les modifications suivantes ont été exécutées :

- suppression de l'enveloppe ADSR contrôlant la vélocité de l'archet ;
- ajout d'un paramètre force ayant un effet sur la pente de la fonction utilisée pour générer le son du fortement de l'archer sur la corde ;
- ajout d'un interrupteur au niveau de la sortie de la table d'onde de l'archet ;
- création d'un instrument à quatre cordes avec possibilité de sélectionner la corde utilisée ;
- possibilité de contrôler indépendamment les paramètres physiques de chaque corde ;
- remplacement du filtre simple simulant la caisse de résonance de l'instrument par une banque de filtres permettant d'appliquer la réponse impulsionnelle d'un violon au son produit.

Le code FAUST du modèle de violon a par la suite permis de créer un plug-in pour *PureData* en utilisant l'architecture `puredata.cpp` de FAUST. Les données gestuelles pour chaque paramètre de l'instrument (fréquence des notes, position de l'archet sur la corde, vélocité de l'archet, force appliquée par l'archet sur la corde et numéro de la corde utilisée) ont alors été placées dans des fichiers textes séparés puis importés dans *PureData*. Ainsi, les valeurs de l'ensemble des paramètres sont modifiées dynamiquement toutes les 4,167 millisecondes. Les données gestuelles utilisées permettent de jouer un air traditionnel espagnol appelé *Muiñeira*.

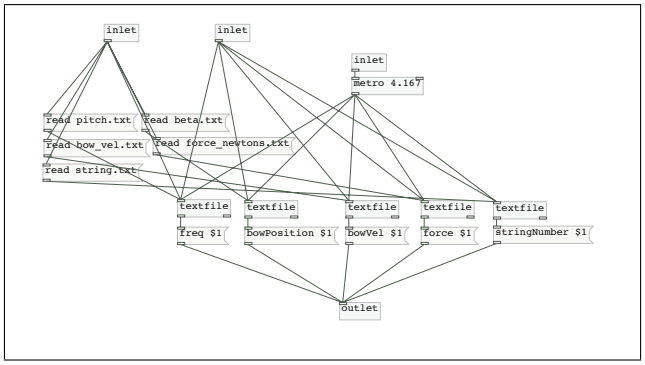

Figure 14: Sous-patch *PureData* utilisé pour envoyer les données gestuelles contenues dans les fichiers textes dans le plug-in créé par FAUST.

7. OPTIMISATION ET PERFORMANCES

7.1. Comparaison de la taille du code C++ du *STK* avec le code FAUST du *FAUST-STK*

La sémantique de FAUST permet d'écrire de façon très concise des algorithmes de traitement du signal. Une importance particulière a donc été accordée à cet aspect lors de la conception des instruments du *FAUST-STK* dont les algorithmes se veulent être les plus courts et les plus clairs possibles.

Il serait pratiquement impossible de comparer la taille des algorithmes écrits en C++ du *STK* avec ceux écrits en FAUST dans le *FAUST-STK*. En effet, dans le cas du C++, les modèles sont implémentés à travers plusieurs fonctions qui la plupart du temps contiennent également des informations ne concernant pas l'algorithme directement. Néanmoins, une comparaison du nombre de lignes du code FAUST avec le code C++ du *STK* pour certains instruments a pu être effectué à titre indicatif en prenant en compte en FAUST tout comme en C++ les éléments concernants la gestion des paramètres et l'algorithme lui-même. Notons que bien que l'exactitude d'une telle comparaison soit fort contestable, elle permet malgré tout de mesurer à quel point le langage FAUST peut-être concis.

Le résultat de cette confrontation est visible dans la table 1. En moyenne, le code FAUST est quatre fois plus court que le code C++.

Nom du fichier FAUST	Taille du code C++ du STK	Taille du code FAUST
blowBottle.dsp	237	54
blowHole.dsp	373	104
bowed.dsp	274	69
brass.dsp	272	63
clarinet.dsp	255	60
flutestk.dsp	309	70
modalBar.dsp *	217	78
saxophony.dsp	308	69
sitar.dsp	193	42
barres *	396	70
voiceForm.dsp *	325	109
piano.dsp *	750	246

Table 1: Tableau comparant le nombre de lignes du code C++ du *STK* avec le code FAUST du *FAUST-STK*. Dans le cas des fichiers dont le nom est suivi d'un astérisque, les banques de paramètres n'ont pas été prises en compte dans le comptage des lignes en C++ tout comme en FAUST.

7 . Music Technology Group.

7.2. Comparaison des performances

Le compilateur de *FAUST* permet de générer des codes C++ dont l'efficacité est optimisée. Ainsi, une comparaison des performances du code C++ d'origine avec le code généré par Faust a été effectuée dans le programme *PureData*.

Dans un premier temps les différents objets du *STK* ont été transformés en plug-ins pour *PureData* à l'aide du programme *stk2pd* développé au CCRMA de Stanford par Michael GUREVICH et Chris CHAFFE [5]. Les codes Faust ont de leur côté aussi été transformés en plug-ins pour *PureData* en utilisant l'architecture `puredata.cpp`.

Dans les deux cas, la compilation a été faite en 32bits avec un traitement du signal scalaire. La machine utilisée pour effectuer le test est un MacBook Pro avec un processeur intel core 2 duo de 2,2 GHz et possédant 2Go de mémoire vive DDR2. Les résultats présentés dans le tableau 2 démontrent que les plug-ins pour *PureData* générés par FAUST sont en moyenne environ 41 % plus efficace que ceux du *STK*.

Nom du fichier FAUST	*STK*	*FAUST*	Différence
blowBottle.dsp	3,23	2,49	-22%
blowHole.dsp	2,70	1,75	-35%
bowed.dsp	2,78	2,28	-17%
brass.dsp	10,15	2,01	-80%
clarinet.dsp	2,26	1,19	-47%
flutestk.dsp	2,16	1,13	-47%
saxophony.dsp	2,38	1,47	-38%
sitar.dsp	1,59	1,11	-30%
tibetanBowl.dsp	5,74	2,87	-50%

Table 2: Tableau comparant les performances des plug-ins pour *Puredata* générés à partir du *STK* avec ceux créés dans FAUST. Les valeurs contenues dans les colones « *STK* » et « *FAUST* » représentent la charge moyenne du plug-in sur le micro-processeur pendant une minute d'activité en pour-cent. La colonne *Différence* donne le pourcentage de gain des performances des plug-ins générés dans FAUST.

8. CONCLUSION

Bien que le *FAUST-STK* soit avant tout un outil pour la synthèse des sons, il est important de souligner sa dimension pédagogique. En effet, le langage FAUST, de par sa clarté et son efficacité est un outil privilégié pour l'enseignement des techniques de traitement du signal. Ainsi, un code FAUST épuré accompagné de commentaires et de nombreuses références bibliographiques est de notre point de vue la meilleure manière de documenter les objets créés.

Cet aspect s'intègre bien dans les objectifs du projet ASTREE sur la préservation des pièces de musique électronique. En effet, les travaux effectués dans le programme *SynthBuilder* sur la modélisation physique d'instruments de musique se devaient d'être préservés.

Avec sa communauté grandissante d'utilisateurs, FAUST est devenu un outil privilégié pour l'implémentation d'algorithmes de traitement du signal pour des applications musicales. Le nombre d'effets, de filtres mais aussi de synthétiseurs disponibles dans FAUST est en constante augmentation. Les forces combinées de FAUST et de JACK [8], renforcées récemment par la possibilité de contrôler les objets créés avec le système OSC [9] constituent une plateforme de travail pour la synthèse et le traitement du son d'une efficacité dont les limites ne sont contraintes que par l'imagination de son utilisateur.

9. REFERENCES

[1] ADRIEN, Jean-Marie, « The missing link : Modal Synthesis », *Representations of Musical Signals*, éd. sous la direction de Curtis Roads, Cambridge : The MIT press, 1991, p. 269-297.

[2] COOK, Perry R., *The Synthesis ToolKit (STK) : acte de la International Computer Music Conference, Pékin (Chine), 10/1999*, Pékin : Computer Music Association, 1999, p. 299-304.

[3] ESSL, Georg ; COOK, Perry R., *Banded Waveguides : Towards Physical Modeling of Bar Percussion Instruments : acte de la International Computer Music Conference, Pékin (Chine), 10/1999*, Pékin : Computer Music Association, 1999, p. 321-324.

[4] GRAEF, Albert, *faust2pd : Pd Patch Generator for Faust*, 2011, disponible en ligne à : `http://docs.pure-lang.googlecode.com/hg/faust2pd.html` (lien vérifié le 02/03/2011).

[5] GUREVICH, Michael ; CHAFE, Chris, *Stk2pd*, Stanford University : CCRMA, 2007, disponible en ligne à : `https://ccrma.stanford.edu/wiki/Stk2pd` (lien vérifié le 02/03/2011).

[6] KARPLUS, Kevin ; STRONG, Alex, « Digital Synthesis of Plucked String and Drum Timbres », *Computer Music Journal*, VII (1983), n° 2, p. 43-55.

[7] MAESTRE GOMEZ, Esteban, *Modeling Instrumental Gesture : An Analysis/Synthesis Framework for Violin Bowing*, Thèse de doctorat (inédite), Barcelone : UPF, 2009.

[8] JAFFE, David A. ; PORCARO, Nick ; SCANDALIS, Gregory P. ; SMITH, Julius O. ; STILSON, Tim, *SynthBuilder : a graphical real-time synthesis, processing and performance system : acte de la International Computer Music Conference, Banff (Canada), 1995*, Computer Music Association, 1995, p. 61-62.

[9] `http://www.mega-nerd.com/libsndfile/` (lien vérifié le 02/03/2011).

8 . JACK Audi Connection Kit : `http://jackaudio.org/`.

9 . Open Source Control est un protocole de communication entre ordinateurs, synthétiseurs ou tout autre appareil multimédia.

[10] RODET, Xavier, *One and two mass model oscillations for voice and instruments : acte de la International Computer Music Conference, Banff (Canada), 1995*, Computer Music Association, 1995, p. 207-214.

[11] SINCLAIR, Stephen, *Implementing the SynthBuilder Piano in STK*, Montreal : McGill University, 2006, disponible en ligne à : `http://www.music.mcgill.ca/~sinclair/content/stk_piano` (lien vérifié le 02/03/2011).

[12] SMITH, Julius O., *Efficient simulation of the reed-bore and bow-string mechanisms : acte de la International Computer Music Conference, La Hague (Hollande), 1986*, Computer Music Association, 1986, p. 275-280.

[13] SMITH, Julius O., *Making Virtual Electric Guitars and Associated Effects Using Faust*, Stanford University : CCRMA, 2009, disponible en ligne à : `https://ccrma.stanford.edu/realsimple/faust_strings/` (lien vérifié le 02/03/2011).

[14] SMITH, Julius O., « Physical Modeling using Digital Waveguides », *Computer Music Journal*, XVI (1992), n° 4, p. 74-91.

[15] SMITH, Julius O. ; VAN DUYNE, Scott, *Commuted Piano Synthesis : acte de la International Computer Music Conference, Banff (Canada), 1995*, Computer Music Association, 1995, p. 319-326.

[16] VERGE, Marc-Pierre, *Aeroacoustics of Confined Jets with Applications to the Physical Modeling of Recorder-Like Instruments*, Thèse de doctorat (inédite), Eindhoven University, 1995.