Modélisation physique d'instruments de musique par guides d'ondes numériques – enjeux, environnements existants, implémentation, perfectionnements et utilisation

Remerciements

Tout d'abord, je souhaite remercier Laurent POTTIER qui a dirigé mes travaux tout au long de mon Master II. Au cours des six dernières années, il a su me transmettre sa passion pour l'informatique musicale, je lui dois énormément pour cela. Son dévouement pour son métier et ses étudiants a toujours forcé mon admiration. Enfin, sa disponibilité, sa gentillesse et ses conseils éclairés m'ont permis à moi et à d'autres de travailler dans les meilleures conditions possibles.

Je tiens également à remercier tout particulièrement les personnes suivantes :

- Yann ORLAREY, pour la confiance qu'il m'a accordé, pour m'avoir introduit dans la communauté Faust et pour l'aide qu'il m'a apporté lors de la mise en place du FAUST-STK ;
- Julius SMITH, pour m'avoir accueilli au CCRMA de l'université Stanford et pour avoir partagé avec moi ses nombreuses bonnes idées et son expérience de la modélisation physique d'instruments de musique par guides d'ondes ;
- Esteban MAESTRE, pour avoir partagé ses travaux avec moi et pour toutes ses suggestions éclairées ;
- Junji KURODA, pour son aide en traitement du signal, notamment dans MATLAB ;
- John CHOWNING, pour m'avoir apporté son appui pour me rendre au CCRMA.

Je dois beaucoup à mes parents et mes amis pour leur soutien constant, leurs relectures et pour les nombreux moments de détente qu'ils m'ont offert. Merci tout particulièrement à Clément, Léo, Benni, Rafa, Steffen et Florian.

Je souhaite aussi exprimer toute ma gratitude à l'ensemble de l'équipe du département de musicologie de l'université Jean Monnet pour son dynamisme, sa disponibilité, son en-

thousiasme et avec qui j'ai beaucoup appris ces six dernières années en échange de très peu.

Pour terminer, je souhaite rendre hommage à Max MATHEWS que j'ai eu l'honneur de rencontrer lors de mon séjour à Stanford et qui m'a aidé à mettre en place avec Julius SMITH le filtre passe-tout non-linéaire passif d'ordre arbitraire présenté dans le second chapitre de ce mémoire. De manière plus générale, nous lui devons tous énormément pour ses travaux pionniers dans le domaine de l'informatique musicale.

Introduction

> « L'instrument de musique, achevé, suppose que l'homme ait maîtrisé des problèmes d'ordre acoustique et technologique, aussi sophistiqués soient-ils. Il y aura toujours un compromis entre le but recherché, une échelle sonore, un timbre, et le résultat obtenu. Ce n'est que dans les mains du musicien-utilisateur que l'instrument manifestera toutes ses possibilités ; il jouera alors son rôle et recevra sa véritable signification. » [1]

Ces problématiques signalées par Josiane BRAN-RICCI et qui font partie de celles du facteur d'instruments de musique et du musicien, ont transcendé toute évolution technologique depuis l'apparition des premières musiques. Les instruments de musique virtuels, fruits de l'avènement des technologies numériques dans le domaine de la musique n'échappent pas à cette règle.

Des instruments de musique virtuels

La découverte des premières techniques de synthèse a ouvert la voie à la création de sons inouïs, purement synthétiques. Néanmoins, chercheurs et musiciens ont également rapidement été fascinés par la possibilité de reproduire le son d'instruments réels avec des machines. Bien qu'un certain nombre d'expériences dans ce sens aient été tentées à une époque où l'informatique n'existait pas encore, ce n'est que lorsque les premiers ordinateurs et la synthèse sonore numérique ont fait leur apparition que les premiers instruments de musique virtuels, modèles d'instruments « réels », ont pu être construits.

Julius SMITH, inventeur de la modélisation physique par guides d'ondes, définit de façon assez générale ce qu'il qualifie « d'instrument de musique virtuel » (au sens d'équivalent virtuel d'un instrument réel) :

> « Les instruments de musique virtuels sont des modèles informatiques efficaces, implémentant *l'essence sonore* d'un instrument de musique et de ses

1. BRAN-RICCI, Josiane, « Instruments de musique », *Dictionnaire des Musiques*, Paris : Universalis, 2009, p. 554.

> effets associés en temps réel. En plus d'avoir le même son que la *chose réelle*, ces derniers doivent également être en mesure de jouer comme elle (du point de vue d'un interprète humain interfacé au modèle par l'intermédiaire de contrôleurs spécialisés et de capteurs). Evidemment, quand cela est possible, la qualité et la jouabilité d'un instrument virtuel peuvent surpasser celles de leur équivalent dans le monde réel ce qui est déjà arrivé à plusieurs reprises. » [2]

Il confie d'ailleurs à un journaliste lors d'une interview télévisée en 1983 qu'il se considère comme un facteur d'instruments de musique [3].

La notion de *modèle informatique* (ou *modèle physique*) mentionnée dans cette définition constitue le fil conducteur des travaux qui sont ici présentés.

Intérêts de la modélisation physique d'instruments de musique

Avant toute chose, il est intéressant de se questionner sur l'utilité de modéliser des instruments de musique. Un premier élément de réponse est donnée dans le *Computer music tutorial* de Curtis ROADS dans le chapitre sur les techniques de modélisation physique :

> « Les objectifs de la synthèse par modélisation physique sont doubles : scientifiques et artistiques. Tout d'abord, la modélisation physique étudie de quelle manière les équations mathématiques et la logique algorithmique permettent de simuler les mécanismes permettant de produire des sons d'instruments existants. Cette approche est fondée sur l'idée que plus la simulation est proche de la réalité, plus le système étudié est compris. [...] Le second but de la modélisation physique est artistique. La simulation avec des modèles physiques peut permettre la création d'instruments fantaisistes qu'ils seraient impossible de construire autrement. » [4]

2. SMITH, Julius, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, Palo Alto (USA) : Stanford University, 2010, vol. 3, p. 3, citation traduite de l'anglais : « Virtual musical instruments are fast, computational models that implement the "audible essence" of a musical instrument and associated effects in real time. The model should not only sound like the real thing, it should also play like the real thing (from the perspective of a human performer interfaced to the model via specialized controllers and sensors). Of course, a virtual instrument is allowed to surpass the quality and playability of its real-world counterpart, when possible, and this as already occured in a number of instances. ».

3. NORDIN, Hud, *HighTech Heroes n° 6 : Julius Smith and David Jaffe*, Montain View (US), 1983, disponible sur YoutTube à l'adresse suivante : `http://youtu.be/15jG1zfx-IM` et sur le CD dans le fichier `High-Tech-Hereos-6.mp4` contenu dans le dossier `/videos/`.

4. ROADS, Curtis, *The computer music tutorial*, Cambridge (USA) : MIT, 1996, p. 265-266, citation traduite de l'anglais : « The goals of physical modeling synthesis are twofold : one scientific, and one artistic. First, physical modeling investigates the extent to which mathematical equations and algorithmic logic can simulate the sound-producing mechanisms of existing instruments. This approach is based on the premise that the closer the simulation, the better understood the system is. [...] The second goal of physical modeling is more artistic. Simulation by physical models can create sounds of fanciful instruments that would otherwise be impossible to build. ».

David MALHAM approfondit ces observations en expliquant : « Ce n'est peut être qu'en apprenant à réaliser une simulation complète du réel et du naturel que nous pouvons pleinement apprécier ce qu'il est possible de faire avec la synthèse. » [5].

Ce thème est approfondi dans le chapitre quatre qui présente des exemples d'utilisation de la technique de synthèse par guides d'ondes dans le répertoire des musiques contemporaines et des musiques actuelles.

Premières découvertes et circuits analogiques de modèles physiques

De manière indirecte, comme l'explique Julius SMITH, les premiers travaux à avoir contribué au domaine de la modélisation physique d'instruments de musique sont ceux d'Isaac NEWTON que ce dernier présente dans son œuvre majeure : *Philosophiae Naturalis Principia Mathematica* [6] :

> « Dans la mesure où il n'est pas nécessaire de se soucier des effets relativistes ou quantiques pour des instruments de musique (du moins, pour l'instant), les bons vieux mécanismes Newtoniens suffiront pour la modélisation physique d'instruments de musique. » [7]

Ce n'est qu'au dix-neuvième siècle que des traités plus spécifiques sur les mécanismes vibratoires de plaques, de barres ou de tubes ont vu le jour [8]. Le plus important d'entre-eux est certainement *The Theory of Sound* du Baron John William Strutt RAYLEIGH [9]. Il est également possible de mentionner les travaux de Hermann VON HELMHOTZ [10] et de John POYNTING et Joseph THOMSON [11] sur des modèles mécaniques pour simuler la physique d'instruments de musique.

L'invention du tube à vide (ou lampe) au début du vingtième siècle va permettre la mise en place de modèles primitifs analogiques d'instruments de musique [12]

5. MALHAM, David, « Toward Reality Equivalence in Spatial Sound Diffusion », *Computer Music Journal*, XXV (2001), n° 4, p. 31-38, citation traduite de l'anglais : « It is perhaps only by learning how to achieve a full simulation of the real and natural that we can appreciate the full extent of what is possible in the synthetic. ».

6. NEWTON, Isaac, *Philosophiae Naturalis Principia Mathematica*, Londres : Royal Society, 1687.

7. SMITH, Julius, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, Palo Alto (USA) : Stanford University, 2010, vol. 3, p. 6, citation traduite de l'anglais : « Since there are no relativistic or quantum effects to worry about in musical instruments (at least not yet), good old Newtonian mechanisms will suffice for musical instruments physical modeling purposes. ».

8. ROADS, Curtis, *The Computer Music Tutorial*, Cambridge (USA) : MIT, 1996, p. 267-268.

9. RAYLEIGH, John William Strutt, *The Theory of Sound*, Londres : Maquilla, 1894.

10. HELMHOTZ, Hermann (Von), *On the Sensations of Tone as a Physiological Basis for the Thoery of Music*, New York : Dover, 1863, R/1954.

11. POYNTING, John ; THOMSON, Joseph, *Sound*, Londres : Charles Griffin, 1900.

12. MILLER, Dayton, *Anecdotal History of the Science of Sound*, New-York : MacMillan, 1935.

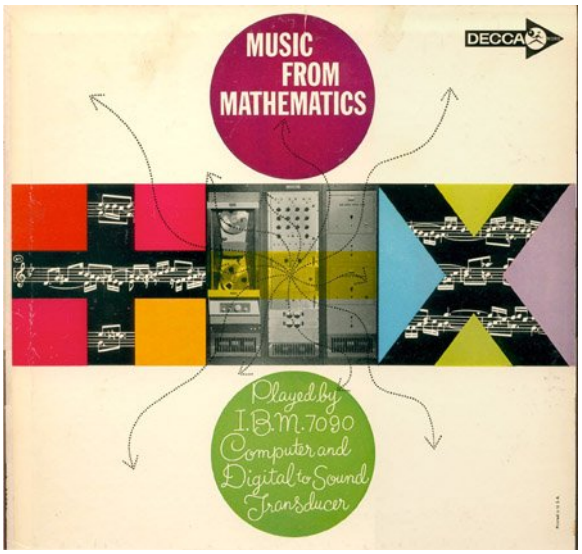

FIGURE 1 – Couverture du disque *Music from Mathematics* des laboratoires Bell.

et de l'appareil phonatoire [13] [14]. Les travaux les plus avancés sur ce sujet ont été effectués par Harry OLSON dans les années soixante qui a mis en place des circuits analogiques de modèles de percussions et d'instruments à cordes pincées et à vent [15].

L'avènement du numérique

A partir du début des années soixante, tout comme dans le cas d'un très grand nombre de domaines, les travaux autour de la modélisation physique d'instruments de musique se sont multipliés avec l'arrivée de l'informatique. Le premier modèle physique numérique a été mis en place par John KELLY et Carol LOCHBAUM en 1962 et permettait de simuler les sons produits par un appareil phonatoire humain [16]. Ce dernier a été utilisé par Max MATHEWS en 1960 aux laboratoires Bell pour générer le très célèbre *Bicycle Built for Two* [17], aujourd'hui connu du grand public.

Les premiers modèles physiques d'instruments de musique ont vu le jour une dizaine d'années plus tard avec les travaux Lejaren HILLER et de Pierre-Michel RUIZ sur la synthèse d'objets vibrants tels que des cordes, des plaques, des barres et des

13. STEWARD, John, « An Electrical Analogue of the Vocal Organs », *Nature*, 1922, n° 110, p. 311-312.
14. STEVENS, Ken ; FANT, Gunnar, « An Electrical Analog of the Vocal Tract », *Journal of the Acoustical Society of America*, XXV (1953), p. 734-742.
15. OLSON, Harry, *Music, Physics, and Engineering*, New York : Dover, 1967.
16. KELLY, John ; LOCHBAUM, Carol, *Speech Synthesis : actes de Fourth International Congress on Acoustics, Copenhague, 21-28/08/1962*, Copenhague, 1962.
17. Cet enregistrement a été diffusé en 1962 dans le disque *Music from Mathematics* : MATHEWS, Max, *Music from Mathematics*, 1CD Decca DL 9103 (enreg. : 1960 ; R/1962).

membranes [18] [19].

La modélisation physique d'instruments de musique aujourd'hui

A partir du milieu des années soixante-dix, avec la diversification des projets sur ce sujet, le nombre de techniques développées pour la modélisation physique d'instruments de musique s'est mis à croître. Ainsi, selon la catégorisation faite par Nicolas CASTAGNE et Claude CADOZ, il est aujourd'hui possible de distinguer cinq techniques différentes de ce type [20].

La plus ancienne a été mentionnée précédemment et est basée sur les travaux de Lejaren HILLER et de Pierre-Michel RUIZ. Elle consiste à modéliser les comportements vibratoires d'un objet en adoptant une analyse numérique. Il est alors question de construire dans un premier temps un modèle en temps continu en utilisant les lois de la physique traditionnelle. Lors d'une seconde étape, des techniques d'analyse numérique sont employées pour discrétiser le modèle théorique qui n'est alors qu'une approximation de l'objet réel. Cette technique est de loin la plus précise mais elle n'en demeure pas moins la plus complexe à implémenter. De plus, l'importante quantité de calculs qu'elle requiert la rend difficile à utiliser en temps réel dans un contexte musical. Ses fins sont donc principalement scientifiques. Des travaux plus récents utilisant cette technique ont été menés à bien lors des vingts dernières années [21] [22].

Une autre technique pour la modélisation physique d'instruments de musique a été mise au point à la fin des années soixante-dix par Claude CADOZ. Elle consiste à représenter tout corps vibrant par un ensemble de masses connectées entre elles par des ressorts [23]. Cette technique, appelée modélisation physique particulaire, est très intuitive et facile à utiliser puisqu'elle ne nécessite pas d'avoir de connaissances particulières en physique. Elle est donc assez adaptée pour des applications musicale. Elle

18. RUIZ, Pierre-Michel, *A Technique for Simulating the Vibrations of Strings With a Digital Computer*, Mémoire de Master inedit, University of Illinois School of Music, 1970.

19. HILLER, Lejaren ; RUIZ, Pierre-Michel, « Synthesizing Sounds by Solving the Wave Equation for Vibrating Objects », *Journal of the Audio Engineering Society*, XIX (1971), n° 7, p. 542-551.

20. CASTAGNE, Nicolas ; CADOZ, Claude, *10 Criteria for Evaluating Physical Modelling Schemes for Musical Creation : actes de Conference on Digital Audio Effects (DAFx-03), Londres, 8-11/09/2003*, Londres : Queen Mary University, 2003.

21. CHAIGNE, Antoine, « Numerical Simulations of Stringed Instruments - Today's Situation and Trends for the Future », *Catgut Acoustical Society Journal*, IV (2002), n° 5, p. 12-20.

22. MCINTYRE, Michael ; SHUMACHER, Robert ; WOODHOUSE, James, « On the Oscillations of Musical Instruments », *Journal of the Acoustical Society of America*, LXXIV (1983), n° 5, p. 1325-1345.

23. CADOZ, Claude, *Synthèse sonore par simulation de mécanismes vibratoires, application aux sons musicaux*, Thèse de doctorat inédite, Institut Polytechnique de Grenoble, 1979.

a été rendue accessible au début des années quatre-vingt-dix au travers du programme CORDIS-ANIMA [24] puis plus tard dans le programme GENESIS [25].

A la même époque, Jean-Marie ADRIEN a mis au point la synthèse « modale » qui consiste à représenter un corps vibrant à l'aide d'une série d'oscillateurs indépendants accompagnés de données de couplage. Chacun d'entre-eux modélise un mode de résonance de la structure et les données de couplage permettent de représenter la forme modales pour chaque mode [26]. Cette technique a été et continue d'être beaucoup utilisée à l'IRCAM par l'équipe d'acoustique instrumentale au travers du programme MODALYS [27] qui a succédé à MOSAIC [28] au milieu des années quatre-vingt-dix.

Une technique se trouvant aux frontières de la modélisation physique d'instruments de musique basée sur des modèles de signaux a été mise en place par l'équipe de Xavier RODET à l'IRCAM au début des années quatre-vingt-dix [29]. Elle a été utilisée plus tard pour l'implémentation de modèles de trompettes [30] [31].

La synthèse par guides d'ondes numériques

La cinquième technique pour la modélisation physique d'instruments de musique est appelée synthèse par guides d'ondes numériques. Elle utilise le même type d'outils mathématiques que ceux utilisés pour la construction de fours à micro ondes et a été inventée par Julius SMITH au CCRMA [32] de l'université Stanford au début des années

24. CADOZ, Claude ; LUCIANI, Annie ; FLORENS, Jean-Loup, « CORDIS-ANIMA : A Modeling and Simulation System for Sound and Image Synthesis - the General Formalism », *Computer Music Journal*, XVII (1993), n° 4, 1993, p. 19-29.

25. CASTAGNE, Nicolas ; CADOZ, Claude, *GENESIS : A Friendly Musician-Oriented Environment for Mass-Interaction Physical Modeling : actes de International Computer Music Conference (ICMC), Gothenburg (Suède), 2002*, Gothenburg : Computer Music Association, 2002.

26. ADRIEN, Jean-Marie, « The Missing Link : Modal Synthesis », *Representations of Musical Signals*, éd. sous la direction de Curtis Roads, Cambridge : MIT Press, 1991, p. 269-297.

27. MORISON, Joseph ; WAXMAN, David, *Modalys Introduction*, Paris : IRCAM, 1997 (troisième édition).

28. MORISON, Jospeh ; ADRIEN, Jean-Marie, « MOSAIC : a Framework for Modal Synthesis », *Computer Music Journal*, XVII (1993), n° 1, p. 45-56.

29. RODET, Xavier ; DEPALLE, Philippe ; FLEURY, Gilles ; LAZARUS, Francis, *Modèles de signaux et modèles physiques d'instruments, études et comparaisons : actes du colloque sur les modèles physiques, la création musicale et les ordinateurs, 1990, Grenoble*, Paris : Edition de la maison des sciences de l'homme, 1994.

30. RODET, Xavier ; VERGEZ, Christophe, *Physical Models of Trumpet-like Instruments Detailed Behavior and Model Improvements : actes de International Computer Music Conference (ICMC), Hong-Kong, 1996*, Hong-Kong : University of science and technology, 1996.

31. VERGEZ, Christophe ; RODET, Xavier, « New Algorithm for Nonlinear Propagation of a sound Wave, Aplication to a Physical Model of a Trumpet », *Journal of Signal Processing*, IV (2000), n° 2, p. 79-87.

32. Center for Computer Research in Music and Acoustics. Plus d'information sur cette institution sont données dans le glossaire à la page 120.

quatre-vingt. Elle est basée sur l'amélioration d'un algorithme plus ancien mis au point par Alexander STRONG et Kevin KARPLUS dans les années soixante-dix permettant de modéliser le comportement vibratoire d'une corde simplement avec une ligne de retards mise en boucle et quelques opérations mathématiques très simples.

Dans sa forme la plus simple, un guide d'ondes est donc un simple ensemble *ligne de retards récursive/coefficient multiplicateur de dispersion* pouvant modéliser une corde ou un tube. Il est possible de décrire la forme de presque n'importe quel instrument de musique en assemblant plusieurs guides d'ondes entre eux.

Le partenariat entre le CCRMA de l'université Stanford et la firme Yamaha

La synthèse par guides d'ondes a connu un succès très rapide par rapport aux techniques mentionnées précédemment à cause de sa simplicité et de son efficacité et a même été commercialisée par la firme Yamaha à partir de 1994 sous la forme d'un synthétiseur qui lui était entièrement dédié : le *VL1*. En effet, sous le couvert d'un brevet qui donnait de manière exclusive les droits de développement de la technique de synthèse par guides d'ondes au CCRMA de l'université Stanford et à la firme Yamaha, un certain nombre de projets autour de cette dernière a vu le jour.

Yamaha a développé toute une gamme de synthétiseurs dits *VL* et de son côté, le CCRMA a implémenté cette technique dans le programme SynthBuilder entre 1994 et 1996 et dans le Synthesis ToolKit à partir de 1995. Celui-ci a la forme d'un ensemble de classes dans le langage C++ implémentant divers algorithmes d'instruments de musique en utilisant la technique des guides d'ondes. Il est totalement libre et *Open Source*.

Enjeux actuels

Durant ces quinze dernières années, l'intérêt du public de musiciens pour la modélisation physique par guides d'ondes a diminué de manière progressive. Il est possible d'expliquer ce phénomène de plusieurs façons. Tout d'abord, comme cela est expliqué dans le chapitre un, l'échec commercial de la gamme de synthétiseur *VL* de Yamaha a eu un impact fort sur l'utilisation par des musiciens de cette technique. La licence qui régissait les développements autour de cette dernière a certainement aussi limité dans une certaine mesure la recherche dans ce domaine. Enfin, avec le temps, le type de sons produits par les instruments implémentés dans le Synthesis ToolKit est devenu assez connoté.

Néanmoins, il est possible d'observer un regain d'intérêt depuis quelques années pour cette technique et la modélisation physique en général, certainement à cause du phénomène de prise de conscience des limites des techniques par échantillonnage par le public de musiciens. De plus, l'expiration de la licence de la synthèse par guides d'ondes cette année a probablement accéléré le développement de logiciels à but commercial utilisant cette technique comme *Pianoteq* de la firme Modart [33].

Modélisation physique d'instruments de musique par guides d'ondes numériques – enjeux, environnements existants, implémentation, perfectionnements et utilisation : définition du sujet

C'est dans ce contexte d'ouverture que les travaux présentés dans ce mémoire se placent. En effet, il est évident que le domaine du logiciel libre a son épingle à tirer du jeu dans cet engouement récent pour la modélisation physique. Ainsi, le travail exposé ici gravite autour d'une réimplémentation complète de l'ensemble des instruments du Synthesis ToolKit dans le langage de programmation Faust : le Faust-STK.

Les objectifs sont multiples. La première dimension du projet est pédagogique avec la volonté de fournir un code le plus commenté possible et en tirant partie de la syntaxe épurée et intuitive de Faust qui permet d'implémenter en quelques lignes des algorithmes qui nécessitent en C++ parfois plusieurs dizaines de déclarations. L'autre objectif de ce travail est l'accessibilité, là aussi, grâce à Faust qui permet de générer des objets pour un ensemble de plateformes, d'applications et de standards.

En plus de cette réimplémentation, il est également question dans cette étude de donner différents exemples d'extensions possibles à l'utilisation standard des modèles physiques par guides d'ondes numériques d'instruments du Synthesis ToolKit notamment par le perfectionnement et le développement des modèles existants, la mise en place de nouveaux modèles, l'introduction de composantes nouvelles dans le son de certains instruments grâce à l'utilisation de filtres non-linéaires et enfin, le contrôle de la synthèse avec des données de suivis de gestes d'instrumentistes.

Dans un premier temps, un état des lieux du domaine de la modélisation physique par guides d'ondes numériques sera fait et les fondements théoriques de cette technique, nécessaires à la compréhension des modèles présentés par la suite, seront rap-

33. *Pianoteq* est un plug-in VST modélisant différents types de pianos. Il est commercialisé depuis 2006. Plus d'informations sur ce dernier peuvent être trouvées sur le site suivant : `http://www.pianoteq.com/`.

pelés. Lors de cette étape, les outils disponibles actuellement, aussi bien dans le domaine commercial que libre, seront détaillés.

Comme mentionné précédemment, une manière d'étendre les possibilités de modèles physiques par guides d'ondes d'instruments de musique est d'introduire des éléments non-linéaires dans leur algorithme. Ainsi, dans un second chapitre, les possibilités permettant de mener à bien ce type d'opérations seront explorées. Toutes les techniques présentées feront l'objet d'implémentations dans FAUST et de démonstrations. Un nouveau type de filtre non-linéaire d'ordre arbitraire offrant la possibilité de moduler ses coefficients à chaque échantillon sera aussi décrit.

Les modèles implémentés dans le FAUST-STK, riches de ce nouveau type de filtre, seront présentés dans un troisième chapitre dans lequel d'autres modèles physiques, extraits du programme SYNTHBUILDER seront également décrits. Un modèle inédit d'instruments de la famille des percussions pouvant être utilisé pour produire des sons de plaques, de barres, etc., basé sur un réseau de lignes de retards récursives, sera aussi détaillé. Un modèle de violon amélioré et contrôlé avec des données de suivis de gestes d'instrumentiste fera aussi l'objet d'une description détaillée.

Enfin, un panorama de l'utilisation de la modélisation physique par guides d'ondes numériques dans le répertoire des musiques actuelles et des musiques contemporaines depuis sa création au début des années quatre-vingt sera fait dans un quatrième et dernier chapitre. Une discussion sur les problématiques de l'utilisation de la technique des guides d'ondes dans un contexte musical sera alors engagée.

Chapitre 1

Principe de fonctionnement et outils existants

A la différence d'autres techniques de synthèse, la modélisation physique par guides d'ondes repose sur des fondements théoriques d'acoustique et de mécanique importants. Une bonne connaissance de ces derniers est nécessaire pour comprendre pleinement les différents modèles et algorithmes présentés dans les chapitres deux et trois.

Dans ce premier chapitre, les mécanismes vibratoires d'un instrument de musique en lien avec la théorie autour de la synthèse par guides d'ondes sont brièvement rappelés. L'algorithme de KARPLUS-STRONG et ses différentes évolutions sont présentés et mis en œuvre dans FAUST. Enfin, un bref historique de l'évolution et de l'utilisation de la synthèse par guides d'ondes est donné afin de comprendre l'impact que cette dernière a pu avoir sur la production musicale depuis sa création au début des années quatre-vingt. Ce dernier aspect est développé dans le chapitre quatre.

1.1 Modèle théorique d'une corde en vibration avec la technique des guides d'ondes numériques

La première théorie sur les modes de vibration des cordes sur les instruments de musique a été énoncée par Jean le Rond d'ALEMBERT au dix-huitième siècle [34]. Il explique alors que la solution de l'équation d'onde d'une corde en vibration correspond à la superposition des ondes se déplaçant vers la gauche et vers la droite le long de la

34. ALEMBERT, Jean le Rond (d'), *Recherche sur les cordes vibrantes*, 1747.

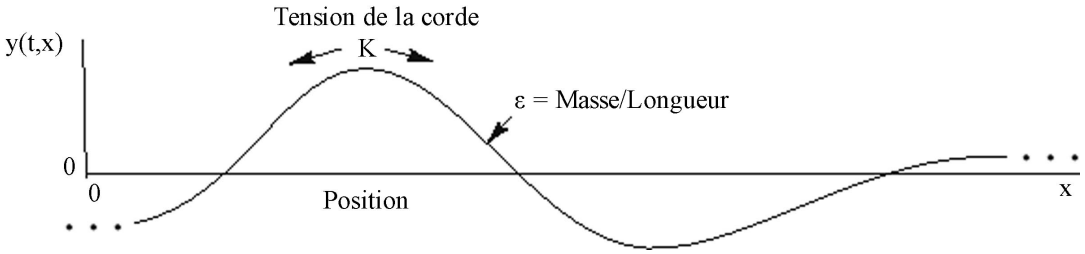

FIGURE 1.1 – Corde idéale en vibration.

corde : $y(m,n) = y^+(m-n) + y^-(m+n)$ avec y^+, les ondes se déplaçant vers la droite, y^-, les ondes se déplaçant vers la gauche, m, la position sur la corde et n, le temps.

Julius SMITH a mis en évidence dans les années quatre-vingt que la technique des guides d'ondes numériques permet d'implémenter informatiquement la solution de l'équation d'onde d'une corde en vibration.

Dans cette partie, une brève introduction à la théorie sur les guides d'ondes (nécessaire à la compréhension des modèles exposés tout au long de ce mémoire) est donnée. Elle est basée sur les travaux de Julius SMITH [35].

1.1.1 Equation d'onde d'une corde idéale de longueur infinie

Le modèle de corde dont il est question dans cette partie, bien que hautement idéaliste, permet de modéliser de façon assez juste le comportement d'une corde « réelle ». En effet, la corde considérée est totalement flexible et élastique. Ainsi, une fois excitée, elle vibre pendant une durée infinie (cf. figure 1.1).

Il est possible de décrire l'onde se propageant dans cette corde avec l'équation suivante [36] :

$$Ky'' = \epsilon\ddot{y}$$

où :

$$\begin{array}{lcl@{\qquad}lcl} K & \triangleq & \text{tension de la corde} & y & \triangleq & y(t,x) \\ \epsilon & \triangleq & \text{densité linéaire de la masse} & \dot{y} & \triangleq & \frac{\partial}{\partial t}y(t,x) \\ y & \triangleq & \text{déplacement de la corde} & y' & \triangleq & \frac{\partial}{\partial x}y(t,x). \end{array}$$

A une échelle microscopique, un lien de parenté peut être établi entre cette dernière et la seconde loi de NEWTON [37] :

35. SMITH, Julius, 2010, *op. cit.*
36. SMITH, Julius, « Ideal Vibrating String », 2010, *op. cit*, p. 204-207.
37. `http://www.phy6.org/stargaze/Fnewt2.htm`.

$$force = masse \times accélération.$$

Les ondes se propageant dans la corde étant des ondes transversales, la force de rappel est donnée par la tension multipliée par la courbure de la corde : $K \times y''(t,x)$.

L'équation donnée au début de ce paragraphe peut être résolue pour toute forme prise par la corde en mouvement et se déplaçant vers la gauche ou la droite à une vitesse $c \triangleq \sqrt{\frac{K}{\epsilon}}$. Si les ondes se déplaçant vers la droite sont représentées avec une fonction générale $y_r(t - x/c)$ et les ondes se déplaçant vers la gauche avec $y_l(t + x/c)$ où y_r et y_l sont des fonctions différentielles doubles arbitraires, alors la solution générale de l'équation d'onde peut être exprimée de la façon suivante :

$$y(t,x) = y_r(t - x/c) + y_l(t + x/c).$$

1.1.2 Corde avec des terminaisons rigides

La corde présentée dans la paragraphe précédent a permis d'introduire les fondements théoriques nécessaires à la compréhension de la synthèse par guides d'ondes. Cette corde ne pourrait exister dans le monde réel dans la mesure où sa longueur est infinie. Il est donc nécessaire de compléter le modèle en terminant cette corde par des terminaisons rigides ce qui implique que cette dernière ne peut pas bouger à ses extrémités. Ce type de terminaisons est le plus simple à décrire et à implémenter, il n'est donc question que de celui-ci dans ce chapitre qui a pour objectif de donner un aperçu de la technique de synthèse des guides d'ondes numériques [38].

S'il est considéré que les terminaisons à chaque extrémité de la corde sont parfaitement rigides, l'onde qui se propage dans la corde lorsque celle-ci est excitée, est réfléchie lorsqu'elle atteint l'une des extrémités et sa phase est inversée (le signal est multiplié par -1).

Ainsi, si une corde idéale de longueur L est terminée à $x = 0$ et $x = L$, alors les conditions aux limites sont remplies [39] :

$$y(t,0) \equiv 0 \qquad y(t,L) \equiv 0.$$

Il est possible de calculer la durée N du déplacement de l'onde d'une extrémité à l'autre de la corde et de son retour au point d'excitation avec $N \triangleq 2L/X$.

38. Plus de détails au sujet des types de terminaisons de cordes et de tubes sont donnés dans : SMITH, Julius, « Acoustic Guitars », 2010, *op. cit*, p. 343-358.
39. SMITH, Julius, « Rigid Terminations », 2010, *op. cit*, p. 210-212.

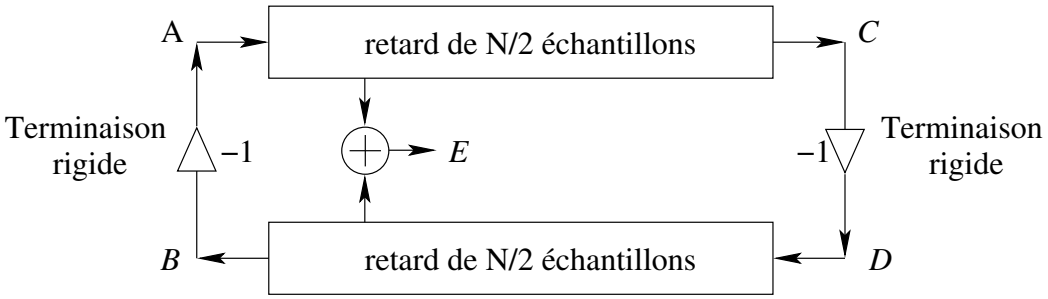

FIGURE 1.2 – Implémentation numérique d'une corde possédant une terminaison rigide à chacune de ses extrémités et dont le signal est extrait à la position $x = \xi$. Les terminaisons rigides reflètent les ondes qui les touchent avec une inversion de signe. Sur la figure, $A = y^+(n)$, $B = y^-(n)$, $C = y^+(n - N/2)$, $D = y^-(n + N/2)$ et $E = y(nT, \xi)$.

Si l'on applique la solution donnée à l'équation d'onde de la corde de taille infinie décrite dans le paragraphe précédent, on obtient :

$$y(nT, 0) = y^+(n) + y^-(n) \equiv 0$$
$$y(nT, NX/2) = y^+(n - N/2) + y^-(n + N/2) \equiv 0.$$

Par conséquent,

$$y^+(n) = -y^-(n)$$
$$y - (n + N/2) = -y^+(n - N/2).$$

L'implémentation numérique de cet algorithme consiste donc à placer de manière parallèle deux lignes de retards de longueur $N/2$, de les connecter entre elles et d'effectuer une inversion de signe à chacune de leurs extrémités (cf. figure 1.2). Il est possible de déduire la fréquence f du son produit à partir de la longueur combinée (N) de ces deux lignes de retards :

$$f = \frac{SR}{N} \times \frac{1}{2}$$

où SR est la fréquence d'échantillonnage. Il est précisé dans la partie 1.2 qui traite de l'algorithme de KARPLUS-STRONG qu'il est également possible de modéliser une corde en utilisant une seule ligne de retards.

Comme cela a été expliqué précédemment, ce modèle reste assez simpliste dans la mesure où des terminaisons parfaitement rigides ne peuvent exister dans la réalité. Il est montré dans la partie 1.3 qu'il est possible de simuler un comportement plus naturel en remplaçant l'inversion de signes aux extrémités du guide d'ondes par un filtre.

1.1.3 Amortissement de l'énergie contenue dans la corde

Le modèle présenté dans la figure 1.2 implémente une corde de longueur finie et terminée à chacune de ses extrémités par des éléments rigides. Si cette corde est excitée, elle vibre à l'infinie et émet alors un son constant assez ennuyeux :

> « Sans amortissement, lorsque « la corde idéale » est pincée, son son est plus proche de celui d'un orgue électronique peu couteux que d'une corde dans la mesure il est parfaitement périodique et ne décroit jamais. » [40]

Dans « le monde réel », l'énergie introduite dans la corde lors de l'excitation est progressivement atténuée jusqu'à devenir nulle. Ceci est principalement dû au frottement de la corde avec l'air qui l'entoure et à la souplesse des terminaisons de la corde qui ne peuvent être totalement rigides et qui absorbent donc une partie de l'énergie.

La manière la plus simple de simuler ce comportement est d'introduire un terme supplémentaire [41] proportionnel à la vélocité, à l'équation donnée au début de la partie 1.1.1 :

$$Ky'' = \epsilon\ddot{y} + \mu\dot{y}.$$

Dans le cas présent, $\mu > 0$ peut être vu comme un simple cœficient de friction. Les déperditions alors engagées au niveau du son produit ne sont pas dépendantes du spectre du son ce qui est un comportement peu naturel. En effet, il est expliqué dans la partie 1.6 que lors de la vibration d'une corde, les partiels dont les fréquences sont les plus hautes sont les premiers à disparaitre. Une solution est alors donnée pour simuler ce type de comportement.

1.2 L'algorithme de Karplus-Strong

La synthèse par guides d'ondes est basée sur une technique utilisée pour la reproduction de sons de cordes pincées mise au point au début des années quatre-vingt par Kevin KARPLUS et Alex STRONG. Afin de comprendre les différents modèles de guides d'ondes décrits par la suite, il est important d'étudier le fonctionnement de cette technique.

40. SMITH, Julius, « The Damped Plucked String », 2010, *op. cit*, p. 220, citation traduite de l'anglais : « Without damping, the ideal plucked string sounds more like a cheap electronic organ than a string because the sounds is perfectly periodic and never decays. ».
41. SMITH, Julius, « The Damped Plucked String », 2010, *op. cit*, p. 220-223.

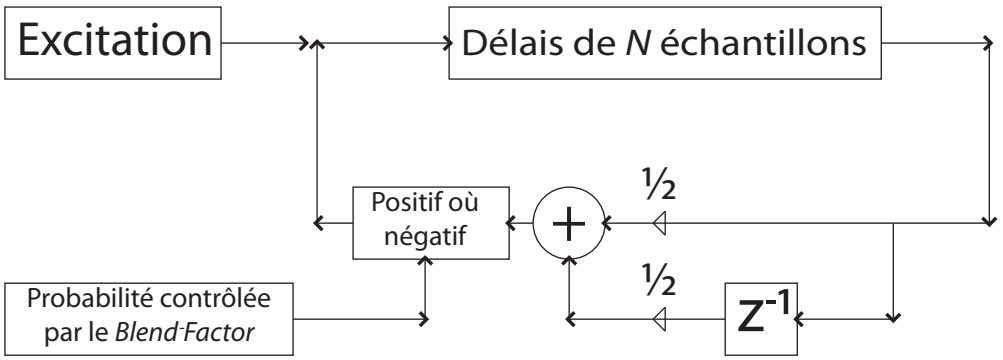

FIGURE 1.3 – Algorithme de *Karplus-Strong*

Les premières méthodes de synthèse par modèle physique étaient particulièrement coûteuses en calculs ce qui les rendaient très difficiles à implémenter empêchant alors toute application musicale en temps différé comme en temps réel. La découverte de l'algorithme KARPLUS-STRONG (aussi appelé *Digitar*) [42] au début des années quatre-vingt, va permettre, en raison de la faible quantité de calculs demandée, la démocratisation de la synthèse par modélisation physique autrefois réservée à des applications purement scientifiques. En 1981, une expérience fut menée aux laboratoires Bell pour comparer l'algorithme KARPLUS-STRONG à de la synthèse additive. Il était alors nécessaire d'utiliser simultanément plus de trente oscillateurs pour atteindre des résultats similaires à ceux obtenus par le modèle physique qui demandaient un taux de calculs équivalent à l'utilisation de seulement deux oscillateurs.

L'algorithme de KARPLUS-STRONG est principalement basé sur l'utilisation d'une ligne de retards d'une durée de N échantillons, mise en boucle, dont le son est amorti en faisant la moyenne de deux échantillons successifs comme le montre la figure 1.3. Cette idée peut être résumée de la façon suivante [43] :

$$y_n = \frac{1}{2}(y_{n-N} + y_{n-N-1})$$

où n est un échantillon à un instant t.

Afin de simuler un pincement de corde, le système est excité par l'introduction d'un grain de son riche en harmonique, typiquement du bruit blanc. La longueur N

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

43. SMITH, Julius, « The Karplus-Strong Algorithm », *Physical Audio Signal Processing*, Stanford University : CCRMA, 2007, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/pasp/Karplus_Strong_Algorithm.html`.

de la ligne de retards permet de contrôler la hauteur du son produit dont la fréquence est donc égale à :

$$f = \frac{SR}{N}$$

où SR correspond à la fréquence d'échantillonnage. L'amplitude du son produit dépend de celle du grain de bruit blanc introduit dans la boucle et décroît donc au cours du temps.

Dans le cas d'une corde en vibration, les ondes se propagent sur une seule dimension. Pour cette raison, on peut voir qu'une seule ligne de retards est utilisée dans l'algorithme de KARPLUS-STRONG à la différence de la technique présentée dans la partie 1.1. En effet, si les ondes se propagent sur une dimension dans un guide d'ondes, il est possible de substituer les deux lignes de retards de longueur $N/2$ par une simple ligne de longueur N et de supprimer les inversions de signes intervenant à la sortie de chaque ligne.

L'algorithme de KARPLUS-STRONG dont il est ici question est implémenté dans le fichier FAUST[44] `ks.dsp`[45]. Ce dernier a été compilé pour fonctionner dans le programme PUREDATA[46] et est implémenté dans le patch présenté dans la figure 1.4 dans lequel le fichier MIDI `take5.mid`[47] est utilisé pour contrôler la synthèse.

Bien que les sons obtenus avec l'algorithme de KARPLUS-STRONG soient très réalistes, l'utilisateur n'a qu'un contrôle très réduit sur le résultat final. Ainsi, en 1979, Kevin KARPLUS a découvert qu'une simple modification de l'algorithme permet d'élargir ses capacités sonores. Celle-ci consiste en l'utilisation d'une formule de probabilités modifiant de façon alternative le signe du résultat de l'opération faisant la moyenne de la valeur d'un échantillon avec celle du précédent (cf. figure 1.5). Cette alternance est contrôlée par un paramètre appelé *blend-factor* déterminant la récurrence de chaque signe. Si *blend-factor* est égale à un, aucun changement de signe n'est opéré, le son est donc celui d'une corde pincée. S'il est égale à zéro, le son obtenu rappelle celui d'une bouteille que l'on frapperait. S'il est égale à $1/2$, on obtient des sons de percussions.

Malgré cette modification astucieuse, le contrôle des caractéristiques des sons produits par l'algorithme de KARPLUS-STRONG reste très limité. De plus, il est important de noter qu'un problème de justesse apparaît au niveau des notes lorsque les

44. FAUST est défini dans le glossaire à la page 120.
45. Fichier disponible sur le CD dans le dossier `/ks/` et décrit dans l'annexe B.
46. PUREDATA est défini dans le glossaire à la page 120.
47. `take5.mid` est disponible dans le dossier `/util/` du CD. Plus d'informations sur ce fichier sont données dans l'annexe A.

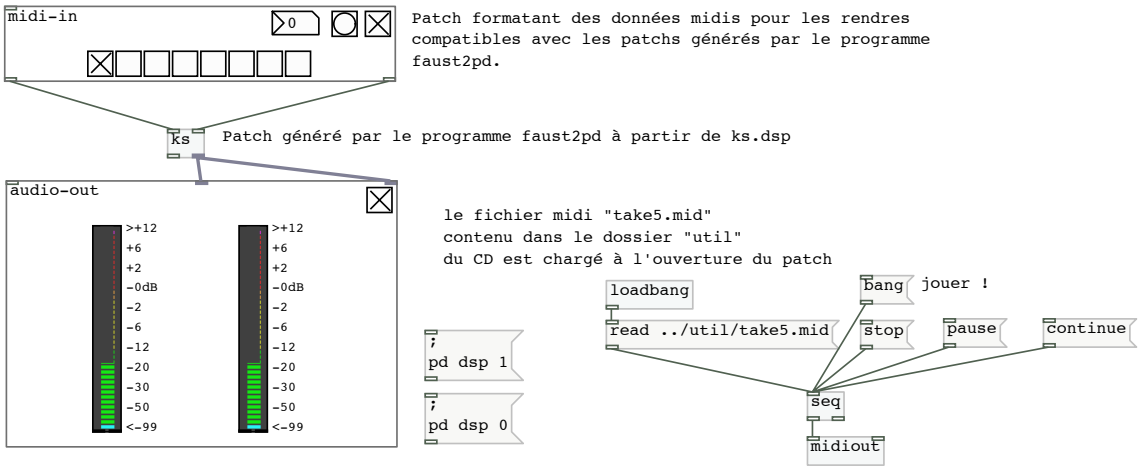

FIGURE 1.4 – Capture d'écran du patch PUREDATA `take5-ks.pd` implémentant l'algorithme de KARPLUS-STRONG grâce à l'objet `ks~` généré dans FAUST – Le soon obtenu avec ce patch a été enregistré dans le fichier `take5-ks.aif` (disponible sur le cd dans le dossier `/ks/`).

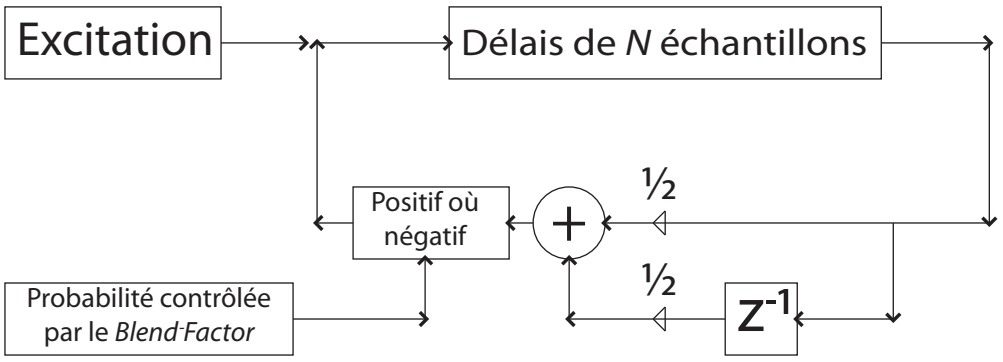

FIGURE 1.5 – Algorithme de *Karplus-Strong* utilisant l'amélioration de Kevin KARPLUS permettant d'obtenir un plus grand nombre de sons

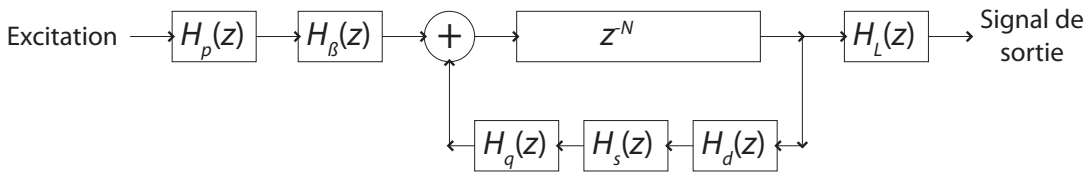

FIGURE 1.6 – Version améliorée de l'algorithme de *Karplus-Strong* par Julius SMITH et David JAFFE

sons produits sont très aigus. Ceci est dû à la quantification générée par la ligne de retards qui permet de contrôler la fréquence de la fondamentale. Il est possible de résoudre ce problème en augmentant la fréquence d'échantillonnage mais cette technique demeure très rudimentaire. Une solution est apportée à ce problème dans la partie suivante.

1.3 Améliorations apportées à l'algorithme de KARPLUS-STRONG dans le cadre des travaux sur la synthèse par guides d'ondes

Julius SMITH et David JAFFE ont effectué au début des années quatre-vingt un certain nombre de modifications sur l'algorithme de KARPLUS-STRONG dans le but de rendre les sons produits plus naturels et afin d'avoir un contrôle accru sur ses caractéristiques [48]. Pour ceci, un certain nombre de filtres de différents types a été ajouté à l'algorithme présenté dans la figure 1.3. Ces modifications sont visibles dans la figure 1.6.

Le rôle de chaque nouvel élément de l'algorithme va être ici brièvement présenté (le détail des formules des filtres utilisés ne sera toutefois pas donné). En effet, ces mécanismes font déjà l'objet d'une description détaillée dans l'article de Julius SMITH *Making Virtual Electric Guitars and Associated Effects Using Faust* [49].

Contrôle de la direction du pincement de la corde

Le filtre $H_p(z)$ visible dans la figure 1.6 est un filtre de type *passe-bas*. Il prend en argument un indice indiquant la direction du pincement de la corde dont la

48. JAFFE, David ; SMITH, Julius, « Extensions of the Karplus-Strong Plucked String Algorithm », *Computer Music Journal*, VII (1983), n° 2, p. 56-69.

49. SMITH, Julius, *Making Virtual Electric Guitars and Associated Effects Using Faust*, Stanford University : CCRMA, 2005, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/`.

valeur doit être comprise entre 0 (la corde est tirée vers le haut) et 0,9 (la corde est tirée vers le bas).

Position du point d'excitation de la corde

Le filtre $H_\beta(z)$ visible sur la figure 1.6 est un filtre en peigne. Il prend en argument un indice indiquant la position du point d'excitation sur la corde. Sa valeur doit être comprise entre 0 (le chevalet sur lequel la corde prend appui) et 1 (le point d'attache à l'autre extrémité de la corde).

Contrôle de la durée de vibration de la corde

Dans l'algorithme de KARPLUS-STRONG présenté dans la partie 1.2, l'atténuation progressive du son est effectuée en faisant la moyenne de la valeur d'un échantillon avec celle du précédent. Cette étape est ici remplacée par le filtre $H_d(z)$ (passe-bas). Celui-ci prend en argument un paramètre p permettant de contrôler la durée de la phase d'atténuation du son (on considère que le son est terminé lorsque son amplitude atteint -60dB). Il est nécessaire d'utiliser la formule suivante pour convertir une durée en secondes en une valeur de p compatible avec $H_d(z)$ (dans la mesure où p permet de contrôler un gain $p \in (0, 1)$) :

$$p = (0.001)\frac{PT}{t_{60}}$$

où t_{60} est la durée de vibration de la corde en secondes et P la durée d'une période.

Il a été expliqué dans 1.1.3 que lorsque l'amplitude du son d'une corde qui a été pincée décroît, les partiels hauts du spectre du son produit sont les premiers à disparaître. Dans la mesure où le filtre $H_d(z)$ est un filtre de type basse-bas, il permet ici de simuler ce comportement.

Contrôle de la dynamique du son

Dans le cas d'une corde réelle, plus on applique une force importante sur celle-ci, plus le spectre du son est riche. Afin d'obtenir des sons les plus naturels possibles, il est important de reproduire cette propriété. Celle-là est simulée dans l'algorithme présenté dans la figure 1.6 par le filtre $H_L(z)$. C'est un filtre de type *passe-bas* prenant en argument une amplitude linéaire dont la valeur est donc comprise entre 0 et 1.

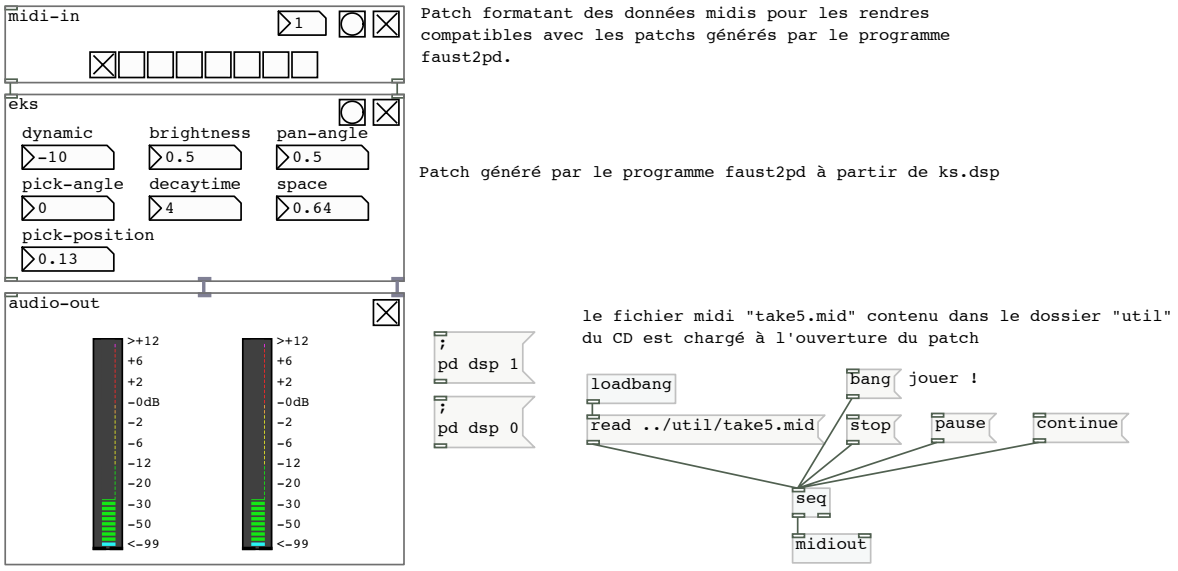

FIGURE 1.7 – Utilisation de l'objet `eks.dsp` dans le programme PUREDATA.

Résolution des problèmes de justesse de l'algorithme de KARPLUS-STRONG

Il a été expliqué dans la partie 1.2 que la hauteur des sons produits avec l'algorithme de KARPLUS-STRONG est contrôlée par la durée en échantillons de la ligne de retards utilisée. Il a aussi été indiqué que l'une des plus grosse faiblesse de cet algorithme est qu'il ne permet pas de générer des notes très aiguës à cause d'un problème de quantification au niveau de la ligne de retards. Ce problème est résolu dans l'algorithme présenté dans la figure 1.6 en utilisant un ensemble de filtres ($H_q(z)$ et $H_s(z)$) implémentant une interpolation linéaire entre chaque échantillon. De manière plus générale, cette opération peut être effectuée en utilisant une ligne de retards fractionnelle [50].

Implémentation dans FAUST et PUREDATA

Le fichier `eks.dsp` [51] implémente l'algorithme présenté précédemment dans le langage FAUST. Il est utilisé dans le patch PUREDATA visible dans la figure 1.7.

Des exemples de sons obtenus avec ce patch sont disponibles dans le dossier `eks` du cd :

– `take5-eks.aif` : le fichier MIDI `take5.mid` [52] est ici joué. Les paramètres

50. LAAKSO, Timo ; VALIMAKI, Vesa ; KARJALAINEN, Matti ; LAINE, Unto, « Splitting the Unit Delay – Tools for Franctional Delay Filter Design », *Signal Processing Magazine*, XIII (1996), p. 30-60.
51. Fichier disponible sur le CD dans le dossier `/eks/` et décrit dans l'annexe C.
52. Fichier disponible dans le dossier `/util/` du CD. Plus d'informations sur ce fichier sont données dans l'annexe A.

par défaut de `eks.dsp` sont utilisés (`pickangle` = 0, `pick_position` = 0.13, `decaytime_T60` = 4, `brightness` = 0.5, `dynamic_level` = -10) ;

- `eks1.aif` : les paramètres par défaut de `eks.dsp` sont utilisés (`pick_angle` = 0, `pick_position` = 0.13, `decaytime_T60` = 4, `brightness` = 0.5, `dynamic_level` = -10) ;
- `eks2.aif` : la valeur du paramètre `brillance` est très élevée (`pick_angle` = 0, `pick_position` = 0.13, `decaytime_T60` = 4, `brightness` = 0, `dynamic_level` = -10) ;
- `eks3.aif` : la corde est pincée près du sillet à l'opposé du chevalet (`pick_angle` = 0, `pick_position` = 0.9, `decaytime_T60` = 4, `brightness` = 0.5, `dynamic_level` = -10).

1.4 Application à d'autres types d'instruments

L'utilisation de la synthèse par guide d'ondes peut être étendue à la modélisation d'un grand nombre de types d'instruments de musique comme le souligne Julius SMITH :

> « L'équation d'onde de la corde en vibration peut être appliquée à tout corps élastique déplacé sur une dimension. Par exemple, la colone d'air d'une clarinette ou le tuyau d'un orgue peuvent être modélisés par cette équation en substituant le déplacement de la corde par la déviation de la pression de l'air et la vélocité des ondes transversales se propageant dans la corde par la vélocité du volume longitudinal. Il est possible de faire référence à ce type de structure en parlant de guide d'ondes à une dimension. Des extensions vers des modèles en deux ou trois dimensions (et plus, pour les plus curieux) sont aussi possibles. » [53]

Il est assumé dans cette formulation que les ondes se déplaçant dans un tube acoustique sont planes et voyagent dans un espace à une dimension (vers la gauche ou la droite du tube). Ceci est dû au fait que la longueur d'onde des signaux qui se déplacent dans le tube est beaucoup plus importante que son diamètre [54].

53. SMITH, Julius, « Ideal Vibrating String », 2010, *op. cit*, p. 205, citation traduite de l'anglais : « The ideal-string wave equation applies to any perfectly elastic medium which is displaced along one dimension. For example, the aire column of a clarinet or organ pipe can be modeled using one-dimensional wave equation by substituting air-pressure deviation for string displacement, and longitudinal volume velocity for transverse string velocity. We refer to the general class of such media as *one-dimensional waveguides*. Extensions to two and three dimensions (and more for the mathematically curious), are also possibles. ».
54. SMITH, Julius, « Ideal Acoustic Tube », 2010, *op. cit*, p. 208.

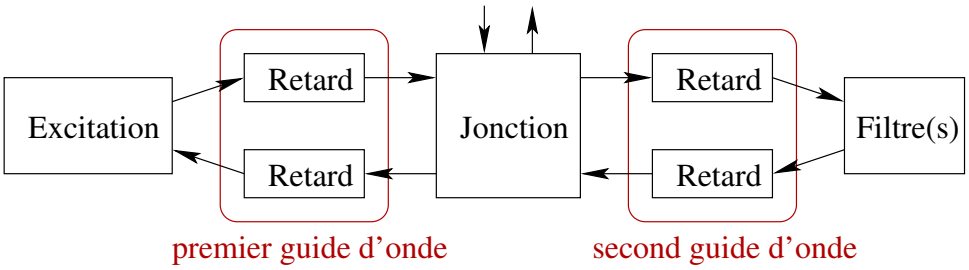

FIGURE 1.8 – Modèle physique générique d'un instrument de musique en guide d'ondes. Figure inspirée de la figure n° 27 du mémoire de master d'Aline HUFSCHMITT.

Il est possible de modéliser le corps de n'importe quel instrument en assemblant plusieurs guides d'ondes en les connectant à l'aide de modules de jonctions. Ces derniers peuvent aider à la simulation de frottements, de trous, de changements de forme, etc. comme le montre la figure 1.8 [55] dans laquelle il est possible de visualiser le schéma d'un modèle physique générique par guides d'ondes d'un instrument de musique. Le modèle est composé de deux guides d'ondes connectés par un module de jonctions pouvant par exemple simuler un trou dans le cas d'une clarinette ou d'une flûte. Il est terminé à l'une de ses extrémités par un filtre modélisant les effets d'un pavillon, d'un chevalet, d'une caisse de résonance, etc. Enfin, il peut être excité à son autre extrémité par un générateur de bruit blanc (comme c'est le cas dans l'algorithme de KARPLUS-STRONG présenté dans la partie 1.2) simulant du souffle, un pincement, etc. ou par une fonction non-linéaire reproduisant alors le signal généré par une anche ou encore la vibration des lèvres dans une embouchure.

La combinaison de guides d'ondes et de modules de jonctions peut permette de modéliser de façon précise des structures très complexes : tubes, plaques de différentes formes, etc. Il est par exemple possible de citer les travaux de Perry COOK sur la modélisation de conduits vocaux [56] en développant la technique de synthèse de la voix introduite par John KELLY et Carol LOCHBAUM [57]. Des tubes de diamètres différents sont modélisés à l'aide de guides d'ondes et sont assemblés en utilisant des modules de jonctions dans le but de créer un tube acoustique de diamètre irrégulier (cf. figure 1.9).

55. HUFSCHMITT, Aline, *La synthèse par modèle physique*, Mémoire de master, Université de Paris Sorbonne (Paris IV), 2000, disponible en ligne à l'adresse suivante : `http://alinehuf3.free.fr/`.

56. COOK, Perry, « SPASM, a Real-Time Vocal Tract Physical Model Controller and Singer », *Computer Music Journal*, XVII (1993), n° 1, p. 30-44.

57. KELLY, John ; LOCHBAUM, Carol, *Speech Synthesis : actes de Fourth International Congress on Acoustics, Copenhague, septembre 1962*, article G42, Stockholm : Speech Transmission Lab (Royal Institute of Technology), 1962.

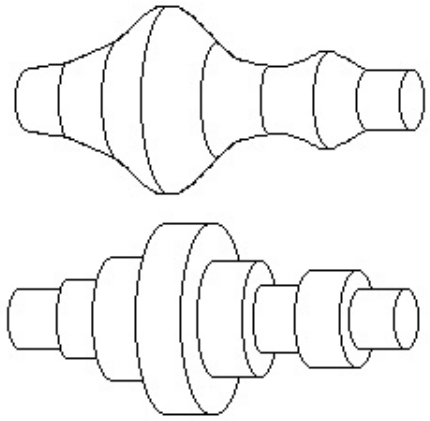

FIGURE 1.9 – Modélisation d'un tube acoustique irrégulier à l'aide de segments de tubes cylindriques. Le tube présent dans la partie supérieure de la figure est le tube à modéliser découpé en section régulière. Le tube présent dans la partie inférieure est sa version numérique modélisée avec la technique employée par Perry COOK (figure de la page trente deux de l'article de Perry COOK).

L'instrument `clarinette.dsp`[58] dont le fonctionnement est détaillé dans l'annexe D permet d'illustrer de façon assez simple l'utilisation de la technique des guides d'ondes pour modéliser un instrument à vent. En effet, comme le montre la figure 1.10, le corps de la clarinette est décrit à l'aide d'un guide d'ondes à une dimension. Ce dernier est excité par une fonction non-linéaire (`reedTable`) qui simule les interactions entre le son produit par l'anche vibrant dans le bec et les ondes de réflexions renvoyées par la fin du guide d'ondes. Tout comme dans le cas de l'algorithme de KARPLUS-STRONG (cf. partie 1.2), la longueur de la ligne de retards du guide d'ondes permet de déterminer la fréquence du son généré.

Le fichier `take5-clarinette.wav`[59], qui a été produit à l'aide du patch PUREDATA `take5-clarinette.pd`[60] contient un exemple sonore dans lequel le fichier `take5.mid`[61] est joué par le modèle de clarinette. Un autre exemple sonore a été généré avec le même patch en utilisant le fichier `voi-che.mid`[62] : `voiChe-clarinette.wav`[63].

Les exemples de modèles d'instruments de musique par guides d'ondes donnés dans ce chapitre restent d'une très grande simplicité. En effet, il est possible de

58. Fichier disponible sur le CD dans le dossier `/clarinette/`.
59. *Ibid.*
60. *Ibid.*
61. Fichier disponible sur le CD dans le dossier `/util/`. Plus d'informations sur ce fichier sont données dans l'annexe A.
62. *Ibid.*
63. Fichier disponible sur le CD dans le dossier `/clarinette/`.

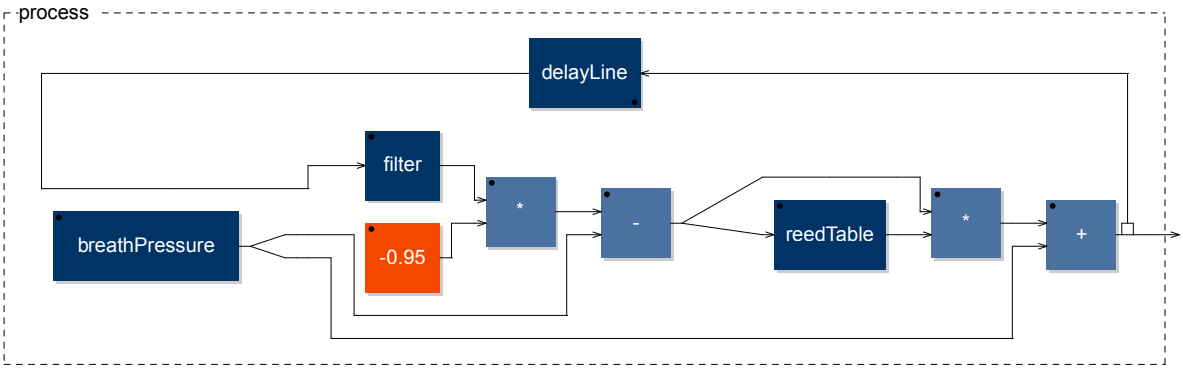

FIGURE 1.10 – Diagramme de l'algorithme de l'instrument `clarinette.dsp` (disponible sur le CD dans le dossier `/clarinette/`) généré avec l'outil `faust -svg` de FAUST. Plus d'informations sur la fonction des différentes « boîtes » qui le constitue sont données dans l'annexe D.

construire des modèles à plusieurs dimensions très complexes en assemblant un nombre important de guides d'ondes entre eux. Des modèles physiques de ce type sont présentés dans la partie 3.5 portant sur l'implémentation de réseaux de guides d'ondes numériques.

1.5 Outils existants

En 1986, la technique de synthèse par guides d'ondes fut présentée à la Haye (Hollande) lors de la *International Computer Music Conference* (ICMC). Cet évènement a eu un impact très important sur l'avenir de la recherche et de l'utilisation de la synthèse par guides d'ondes numériques pour les vingts années suivantes.

En effet, à cette occasion, Julius SMITH a eu l'opportunité de prendre contact avec l'entreprise Yamaha comme il le raconte :

> « La technique de synthèse par guides d'ondes fut présentée avec des exemples sonores à ICMC en 1986 lors de laquelle l'ingénieur en chef de Yamaha faisait partie du public. Il faut alors noter que le brevet de la synthèse FM touchait à sa fin. Yamaha employa rapidement quelques consultants afin d'évaluer la synthèse par guides d'ondes, en conséquence de quoi ils entamèrent en 1989 un important effort de développement qui engendra la famille de synthétiseurs *VL1* (« Virtual Lead ») qui fut introduite en 1994 à Los Angeles lors des NAMM (National Association of Music Merchants). » [64]

64. SMITH, Julius, « Digital Waveguide Models », 2010, *op. cit*, p. 458, citation traduite de l'anglais : « The waveguide synthesis technique was presented, with sound exemples, at the 1986 ICMC, and it so happened that Yamaha's chief engineer was in the audience. Perhaps significantly, the FM synthesis patent was nearing the end of his life. Yamaha soon hired some consultants to evaluate waveguide synthesis, and in 1989 they began a strenuous development effort culminating in the *VL1* ("Virtual Lead") synthesizer family, introduced in the U.S at the 1994 NAMM show in LA. ».

William COAKLEY qui était à l'époque journaliste pour le *Keyboard Magazine* était présent lors de cet événement et témoigne :

> « Lorsque je me suis dirigé vers la zone Yamaha, j'ai entendu un groupe jouer (chose typique lors des NAMM shows) et mon attention a été retenue par le piano numérique produit par la firme. Avant mon départ, j'ai entendu le solo du saxophone et j'ai pensé que l'instrumentiste était assez bon. Je remarquais qu'une foule de personnes subjuguées s'était amassée devant le groupe que je ne pouvais toujours pas voir. Je me rapprochais et découvris alors qu'il n'y avait pas de groupe. Un seul homme avec un contrôleur de souffle produisait ce super son de sax avec une telle expressivité que je ne pouvais pas croire qu'il provenait d'un clavier. [...] Je me renseignais précipitamment sur le prix – plus de six milles dollars. « Bien », pensais-je alors que je partais, « peut être un jour à l'occasion ». » [65]

De la rencontre entre Julius SMITH et l'ingénieur en chef de Yamaha est donc né un partenariat entre le CCRMA [66] de l'université de Stanford (USA) et la firme Yamaha ainsi qu'un brevet pour la synthèse par guides d'ondes. Pour cette raison, la plupart des outils créés dans les années quatre-vingt-dix basés sur cette technologie ont été développés par l'une de ces deux entités. En effet, une équipe de chercheurs a travaillé au CCRMA entre 1994 et 1996 dans le cadre du projet *Sondius* [67] sur un programme nommé SynthBuilder implémentant toute une collection d'instruments utilisant la technique des guides d'ondes. De son côté, Yamaha a travaillé sur la famille de synthétiseurs *VL1* comme cela a été mentionné précédemment.

Dans cette partie, les outils développés sur la base de SynthBuilder sont décrits avant de présenter les différentes applications commerciales de la synthèse par guides d'ondes faite par la firme Yamaha dans un premier temps, puis par d'autres par la suite.

---

65. COAKLEY, William, « Amazing Instruments : Yamaha VL1, VL7 and VL70m », septembre 2001, article en ligne à l'adresse suivante : `http://williamcoakley.com/articles.php?article=yamaha.php`, citation traduite de l'anglais : « When I walked into the Yamaha suite to browse I heard a band playing (typical at NAMM shows) and my attention was drawn to the digital piano they had produced. Before I left, I heard the saxophone solo and thought the guy was pretty good and I noticed a crowd of people standing around the band in amazement... but I couldn't see the band. So I took a closer look and found no band at all. Here was a guy with a breath controller playing this super sax sound with all its expressiveness that I just couldn't believe that it was coming from a keyboard. [...] I hastily inquired about the price - nearly six thousands dollars ! Well, I thought as I left, "maybe someday soon". ».

66. Center for Computer Research in Music and Acoustics. Plus d'informations sur ce la laboratoire sont données dans le glossaire à la page 120.

67. `http://www.sondiusxg.com/`.

1.5.1 De SynthBuilder au Synthesis ToolKit

Le programme SynthBuilder

Au début des années quatre-vingt-dix, le CCRMA de l'université Stanford qui co-détenait avec Yamaha les droits sur le brevet de la synthèse par guides d'ondes, décida de lancer son propre projet de recherche afin de diffuser et d'étendre les possibilités de cette technique. L'équipe de recherche rassemblée autour du projet *Sondius* a ainsi travaillé entre 1994 et 1996 sur le développement d'un certain nombre de modèles physiques « prêts à l'emploi ». Ces derniers étaient implémentés dans le programme SynthBuilder [68] développé par Nick PORCARO et commercialisé par la société Stacato.

SynthBuilder permettait d'implémenter de façon très simple et ludique des algorithmes pour le traitement numérique du signal orienté vers la synthèse des sons au travers d'une interface graphique. De la même manière que dans les programmes Max/MSP ou PureData, les différents éléments d'un algorithme (filtres, lignes de retards, etc.) étaient assemblés sous la forme d'un patch à l'aide de câbles (cf. figure 1.11).

Un ensemble d'outils permettait alors d'interfacer les algorithmes créés avec des contrôleurs MIDIs pour pouvoir les utiliser dans un contexte musical.

SynthBuilder fonctionnait sur des machines NeXT [69] à l'aide du NeXT MusicKit [70] qui tirait partie de la puissance des modules Motorola 56k DSP qui effectuaient les tâches de traitement du signal. Ces derniers avaient une fréquence d'horloge de 80MHz et pouvaient être combinés entre eux rendant alors possible l'utilisation des algorithmes créés dans SynthBuilder en temps réel [71].

A l'issu du projet *Sondius*, une vingtaine de modèles physiques utilisant pour la plupart la technique de synthèse par guides d'ondes (quelques modèles étaient également basés sur la synthèse modale) étaient implémentés dans le programme SynthBuilder : flûte, clarinette, clavecin, piano, trompette, cor, guitare électrique et acoustique, guitare basse, ensemble de percussions, etc. Des exemples sonores [72] de certains de ces instruments sont disponibles sur le CD dans le dossier `/synthbuilder/`. Le détail du contenu de chaque

68. PORCARO, Nick ; JAFFE, David ; SCANDALIS, Gregory ; SMITH, Julius ; STILSON, Tim ; VAN DUYNE, Scott, « SynthBuilder : A Graphical Rapid-Prototyping Tool for Development of Music Synthesis and Effect Patches on Multiple Platforms », *Computer Music Journal*, XXII (1998), n° 2, p. 35-43.

69. Plus d'informations sur cette entreprise sont données dans le glossaire à la page 120.

70. Plus d'informations sur ce système sont données dans le glossaire à la page 120.

71. SCANDALIS, Gregory, « Music Technology », 2004, article disponible sur le site internet de l'auteur : `http://www.scandalis.com/Jarrah/`.

72. Exemples sonores issus du site internet de Gregory SCANDALIS à la page suivante : `http://www.scandalis.com/Jarrah/PhysicalModels/index.html#StanfordCCRMA`.

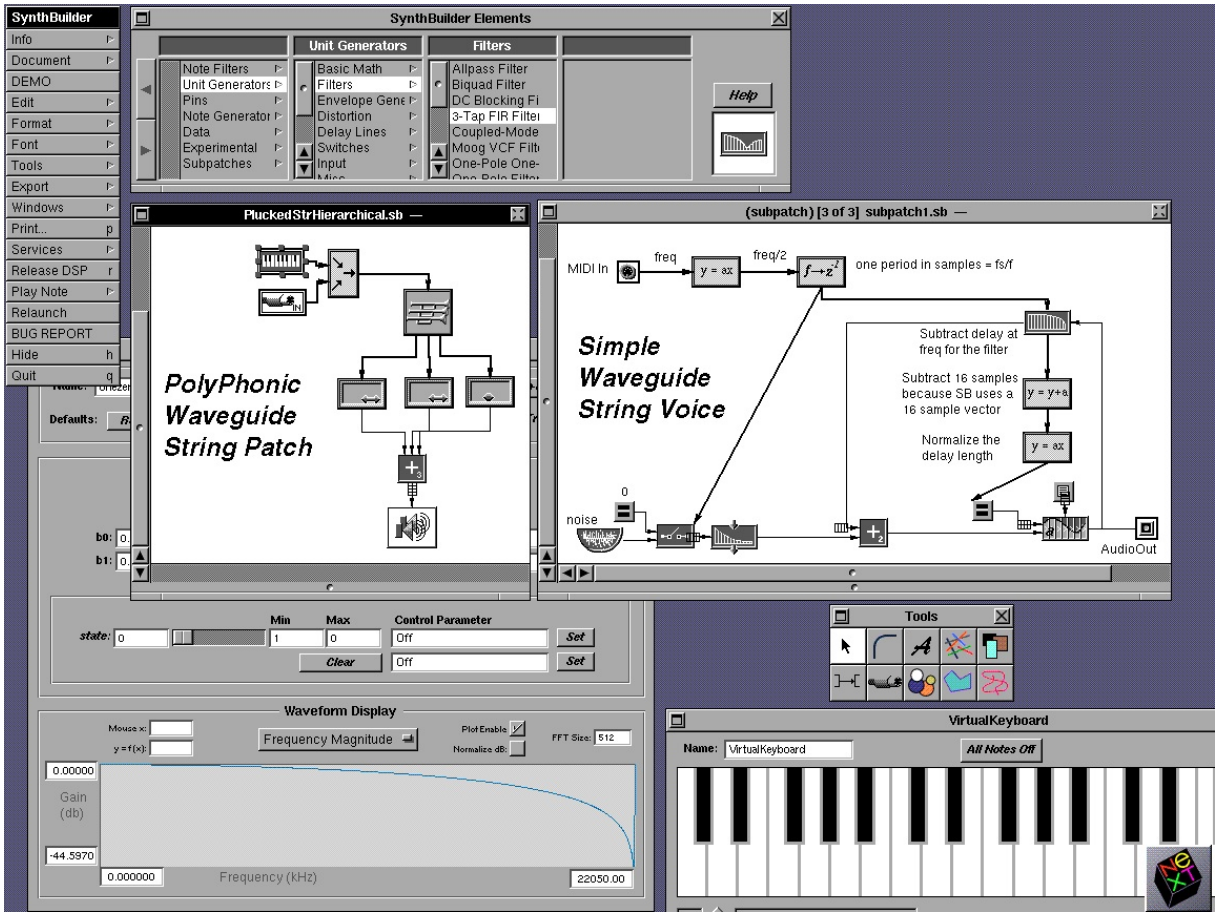

FIGURE 1.11 – Capture d'écran du programme SYNTHBUILDER.

fichier audio est donné dans l'index des fichiers du CD à la page 186.

Bien que SynthBuilder était un programme de grande qualité avec un fort potentiel, il n'a pas vraiment rencontré le succès espéré par ses créateurs. Il a ainsi eu une durée de vie assez courte de moins de trois ans (son développement a été stoppé en 1996).

Toutefois, l'importante quantité de travail effectuée au niveau de la synthèse par guides d'ondes dans SynthBuilder n'a pas été totalement perdue grâce aux efforts de Perry COOK et de son Synthesis ToolKit ou *STK*.

Le Synthesis ToolKit et ses applications

Les premiers travaux autour du *STK*[73] ont eu lieu en 1995 au département d'informatique de l'université de Princeton aux USA et étaient conduits par Perry COOK[74]. Le Synthesis ToolKit se présente sous la forme d'un ensemble de classes et de fonctions écrites dans le langage `C++`[75], implémentant un nombre important d'algorithmes de traitement du signal pour la synthèse et le traitement du son.

Beaucoup d'efforts ont été déployés pour le rendre le plus portable et le plus compatible possible en limitant au maximum les dépendances avec d'autres bibliothèques. La qualité et la clareté du code du *STK* en font un outil pédagogique d'une très grande efficacité et a été par conséquent utilisé par des centaines d'étudiants dans le cadre de projets d'ordre académique.

Il existe un lien très fort entre le Synthesis ToolKit et le programme SynthBuilder. En effet, Perry COOK, qui par la suite a été rejoint par Garry SCAVONE, étaient tout deux anciens doctorants de Julius SMITH au CCRMA de l'université Stanford. Ainsi, un grand nombre des algorithmes implémentés dans SynthBuilder ont servi de base à la mise en place du *STK*. Ce dernier contient donc la plupart des modèles physiques par guides d'ondes implémentés dans SynthBuilder quelques années plus tôt. Il est toutefois important de préciser que ces derniers ont été modifiés et parfois simplifiés.

Il est possible de dire dans une certaine mesure que le *STK* est aujourd'hui « victime de son succès ». En effet, il a tellement été utilisé dans un grand nombre d'applications que ses sons sont devenus au cours du temps assez connotés. Ceci est renforcé

73. `https://ccrma.stanford.edu/software/stk/`, site internet du *Synthesis ToolKit.*

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

75. Plus d'informations sur ce langage sont données dans le glossaire à la page 120.

par le fait que l'utilisation qui en est faite est dans beaucoup de cas assez basique et que le potentiel des algorithmes du *STK* n'est par conséquent pas assez exploité comme cela est montré dans la partie 3.3.3 portant sur l'utilisation de données de suivis de gestes d'instrumentiste avec un modèle physique de violon.

Dans la mesure où le *STK* se veut être le plus ouvert possible, il a été utilisé pour mettre en place des bibliothèques d'objets pour MAX/MSP, PUREDATA (à l'aide du programme `stk2pd`[76] qui permet de convertir toute classe du *STK* en un objet pour PUREDATA) ou encore d'opcodes pour CSOUND[77][78]. Dans ce dernier cas, quelques travaux ont également été effectués afin d'implémenter certains des algorithmes du *STK* directement sous la forme de code CSOUND[79].

A la date de l'écriture de ce mémoire (le *STK* est toujours en cours de développement), les modèles suivant utilisant la technique des guides d'ondes y sont implémentés :

- une flûte,
- un saxophone,
- deux clarinettes,
- un ensemble de modèles de percussions,
- une bouteille dans laquelle on souffle,
- un cuivre (modèle pouvant être utilisé pour produire des sons de trompette, trombone, etc.),
- une mandoline,
- un sitar.

Le détail du fonctionnement de chacun de ces modèles sera donné dans le chapitre 3 traitant de l'implémentation d'algorithmes de modèles physiques par guides d'ondes dans le langage FAUST.

76. GUREVICH, Michael ; CHAFE, Chris, « Stk2pd », Stanford University : CCRMA, 2007, article disponible en ligne à : `https://ccrma.stanford.edu/wiki/Stk2pd`.

77. Plus d'informations sur ce programme sont données dans le glossaire à la page 120.

78. VERCOE, Barry, « Waveguide Physical Modeling », *The canonical CSOUND Reference Manual, Version 5.09, éd. sous la direction de Barry Vercoe*, Cambridge : MIT, 2005, p. 92.

79. NEBOT, Josep, « Csound Implementation of Physical Models of Woodwind Instruments », *The Csound Magazine*, 1999, article de journal en ligne disponible à l'adresse suivante : `http://www.csounds.com/ezine/autumn1999/woodwinds/`.

FIGURE 1.12 – Le synthétiseur *VL1* de la firme Yamaha.

FIGURE 1.13 – Contrôleur de souffle pour le *VL1*.

1.5.2 Applications commerciales

La plupart des applications commerciales de la synthèse par guides d'ondes ont été faites par la firme Yamaha qui détient les droits sur cette technique avec l'université Stanford. Comme il a été mentionné précédemment, le fruit du travail mené sur l'exploitation de cette technique a été le synthétiseur *VL1*, premier modèle dans son genre.

Il a été expliqué précédemment que la synthèse par guides d'ondes implique (tout comme la plupart des techniques basées sur la modélisation physique) la manipulation d'un nombre important de paramètres. Il est montré dans la partie 3.3.3 que la qualité du son produit par un modèle physique dépend fortement de la cohérence des données utilisées pour contrôler chacun de ses paramètres.

Yamaha a tout de suite compris l'importance de cet aspect en offrant un

contrôle inégalé sur le son produit à l'aide d'un ensemble d'outils tel qu'un contrôleur de souffle (cf. figure 1.13), des pédales ou encore des roues placées à côté du clavier (cf. figure 1.12). Dans un article du magazine *Sound on Sound* de 1984, le journaliste Martin RUSS écrit :

> « Le *VL1* n'est pas seulement un autre synthétiseur. Il représente un changement majeur dans la manière dont les instruments de musique électronique sont perçus et construits. La différence est vraiment de cette ampleur ! » [80]

Le *VL1* se démarquait ainsi en beaucoup de points de la plupart des synthétiseurs produits à l'époque qui utilisaient presque tous des techniques d'échantillonnage [81] pour reproduire les sons d'instruments du « monde réel ». Le *VL1* était le premier synthétiseur depuis la période analogique à demander un important effort d'apprentissage au musicien qui en joue. Il fallait en effet un certain temps à l'instrumentiste avant de pouvoir produire des sons de bonne qualité semblables à ceux des exemples impressionnants donnés par Yamaha à l'époque [82].

Le *VL1* a été le premier synthétiseur de toute une série dans laquelle on pouvait retrouver dans la seconde moitié des années quatre-vingt-dix le *VL1-m* ou encore le *VL70*. Le *VL1-m* était un simple boitier utilisant les mêmes processeurs de traitement du signal et les mêmes algorithmes que le *VL1* (cf. figure 1.14). Il était possible d'y connecter toute une collection de contrôleurs MIDIs dont le plus célèbre est certainement le WX5 (cf. figure 1.15). Enfin, le VL70 regroupait une grande partie des fonctionnalités du *VL1* dans un clavier au prix plus abordable.

Malgré son potentiel énorme, le *VL1* n'a jamais connu le succès espéré par Yamaha. Ceci s'explique notamment par son prix élevé (presque dix mille dollars lors de sa sortie) mais aussi par le fait qu'il était un instrument difficile à maîtriser qui demandait un vrai travail d'apprentissage le rendant très peu populaire auprès des musiciens amateurs :

> « Que puis-je dire ? Je n'ai pas les moyens de m'en payer un, je n'ai pas le temps d'apprendre à en jouer correctement, mais il n'en reste pas moins

80. RUSS, Martin, « Yamaha VL1, Virtual Acoustic Synthesizer », *Sound Of Sound*, Juillet 1994, article disponible en ligne : `http://www.soundonsound.com/sos/1994_articles/jul94/yamahavl1.html`, citation traduite de l'anglais : « The *VL1* is not just another synthesizer. It represents a major change in the way that electronic musical instruments are made and perceived. It really is that different ! ».

81. Plus d'informations sur cette technique sont données dans le glossaire à la page 120.

82. La vidéo `VL1-comp.m4v` est disponible sur le CD dans le dossier `/vl1/`. Elle contient une comparaison effectuée entre un synthétiseur utilisant la technique de l'échantillonnage et le *VL1*. Dans la première partie de la vidéo, un son de saxophone fait l'objet de la comparaison. Par la suite, une démonstration de l'utilisation du *VL1* est faite avec un son de trompette. Il est interessant de noter l'utilisation du contrôleur de souffle.

FIGURE 1.14 – Synthétiseur *VL1-m* de la firme Yamaha.

FIGURE 1.15 – Contrôleur MIDI *WX5* pour la gamme de synthétiseur *VL* de Yamaha.

l'instrument le plus désirable que j'ai eu l'occasion d'entendu. » [83]

L'échec commercial du *VL1* a eu un impact important sur l'avenir des techniques de synthèse par modélisation physique. En effet, après 1994 et pendant presque dix ans, aucun constructeur n'a osé s'engager dans le voie de la synthèse par modélisation physique faisant la part encore plus belle aux synthétiseurs par échantillonnage pour la reproduction de sons d'instruments du monde réel. Toutefois, la licence de la synthèse par guides d'ondes arrivant à terme en 2012, il est possible d'observer depuis quelques années l'apparition sur le marché de synthétiseurs commerciaux développés par d'autres firmes que Yamaha utilisant très certainement cette technique. Bien qu'il ne s'agisse là que de suppositions (ces entreprises ne publient bien évidement pas leurs travaux de recherche), on peut imaginer que les plug-ins *Pianoteq* [84] de la firme Modartt ou *Brass* [85] de Arturia utilisent en partie la synthèse par guides d'ondes dans leurs modèles physiques. Il est également possible de citer les claviers *Oasys* de Korg [86] [87], *sx-WSA1* de Technics [88] et *Fusion* d'Alesis [89] [90]. A la différence du *VL1*, ces instruments ne sont pas entièrement basés sur la synthèse par modélisation physique et offrent des modèles empruntés sous licence à la firme Yamaha.

Depuis sa découverte au milieu des années quatre-vingt, la synthèse par guides d'ondes numériques a fait l'objet d'un certain nombre de travaux effectués pour la plupart au CCRMA de l'université de Stanford et par la firme Yamaha sous le couvert d'une licence partagée entre ces deux institutions. Cette licence arrivant à terme en 2012,

83. *Ibid.*, citation traduite de l'anglais : « What can I say ? I can't afford one ; I haven't got the time to learn how to play it properly ; and yet this is one of the most desirable instrument I have ever heard. ».

84. Site internet commercial de la gamme de programmes *Pianoteq* : `http://www.pianoteq.com/`. Des exemples sonores sont disponibles dans la rubrique « Listen » du site.

85. Site internet de commercial de la gamme de programmes *Brass* : `http://www.arturia.com/evolution/en/products/brass/intro.html`. Des exemples sonores sont disponibles dans la rubrique « Media » du site.

86. Site internet commercial de la gamme de clavier *Oasys* : `http://www.korg.com/oasys`.

87. Critique de Gordon REID du clavier Korg *Oasys* dans le magasine *Sound on Sound* : REID, Gordon, « Korg OASYS – Workstation Synth », *Sound on Sound*, Novembre 2005, article disponible en ligne : `http://www.soundonsound.com/sos/nov05/articles/korgoasys.htm`.

88. Ce synthétiseur n'étant plus commercialisé, il est assez dure de trouver des références venant directement du constructeur. Toutefois une critique intéressante de ce dernier a été faite par Martin RUSS en 1995 dans le magasine *Sound on Sound* : RUSS, Martin, « Technics SX-WSA1 – Acoustic Modeling Synthesizer », *Sound on Sound*, Décembre 1995, article disponible en ligne : `http://www.soundonsound.com/sos/1995_articles/dec95/technicswsa1.htm`.

89. Site internet commercial du clavier *Fusion* : `http://www.alesis.com/fusion6hd`.

90. Critique de Nicholas ROWLAND du clavier Alesis *Fusion* dans le magasine *Sound on Sound* : ROWLAND, Nicholas, « Alesis Fusion – Synth Workstation », Sound on Sound, Mai 2006, article disponible en ligne : `http://www.soundonsound.com/sos/may06/articles/alesisfusion.htm`.

on peut tout-à-fait imaginer qu'un important regain d'intérêt pour cette technique se fasse dans les années à venir, aussi bien dans le monde de l'informatique « libre » que dans le monde des synthétiseurs commerciaux comme le prouve *Pianoteq* et *Brass*.

Chapitre 2

Implémentation de guides d'ondes non-linéaires

La plupart des instruments de musique ont un certain nombre de comportements non-linéaires qui se traduisent principalement par un effet d'enrichissement dynamique du spectre du son produit.

Le rôle et l'importance des non-linéarités dans le son varient fortement d'un instrument à un autre. En effet, chez les instruments à vent et à cordes frottées, les non-linéarités sont associées au maintien des oscillations [91]. De manière différente, dans le cas des instruments à cordes frappées tels que le piano et les percussions, la compression non-linéaire générée lors du choc avec la corde ou le corps de l'instrument engendre la création d'un déplacement dynamique d'énergie entre les différents modes de résonance du son généré au cours du temps. Ceci a pour effet de rendre les sons produits plus brillants et de leur donner un caractère très naturel [92] [93].

Les travaux de Neville FLETCHER et de Thomas ROSSING [94] ont permis de démontrer que lorsqu'une cymbale, un gong chinois ou un tam-tam sont frappés, les partiels de la zone supérieure du spectre du son produit deviennent plus intenses au détriment de ceux de la zone inférieure. Ceci est dû au fait que les non-linéarités transfèrent

91. SMITH, Julius, *Rapport de recherche n° STAN-M-39 : Music Applications of Digital Waveguides*, Stanford University : Center for Computer Music and Acoustics, 1987.

92. CHAIGNE, Antoine ; ASKENFELT, Anders, « Numerical Simulations of Piano Strings, a Physical Model for a Struck String Using Finite Difference Methods », *Journal of Acoustical of America*, II (1994), n° 95, p. 1112-1118.

93. VAN DUYNE, Scott ; PIERCE, John ; SMITH, Julius, *A Passive Nonlinear Mode-coupling Circuit and the Wave Digital Hammer : actes de la International Computer Music Conference, Århus (Danemark), 1994*, Århus : Danish Institute of Electroacoustic Music, 1994.

94. FLETCHER, Neville ; ROSSING, Thomas, *The Physics of Musical Instruments*, New-York : Springer-Verlag, 1991, p. 133-151.

l'énergie présente au niveau des partiels graves aux partiels plus aigus.

Enfin, il a été prouvé que les non-linéarités jouent un rôle très important dans la caractérisation du timbre de certains instruments à cordes dont le chevalet modifie dynamiquement la longueur de la corde lors de la performance (sitar, tampura, etc.).

La technique de modélisation physique par guide d'ondes présentée dans le chapitre précédent ne permet pas d'imiter ce type de comportement. Il est néanmoins assez aisé de remédier à ce problème à l'aide d'un certain nombre de méthodes [95].

Dans ce chapitre, les techniques existantes permettant de rendre non-linéaires des modèles de cordes basés sur la technique des guides d'ondes sont décrites dans un premier temps. Par la suite, une technique innovante basée sur l'utilisation de filtres passifs passe-tout non-linéaires d'ordre arbitraire dans un algorithme de guide d'ondes est introduite.

2.1 Techniques existantes

2.1.1 Filtre passe-tout du premier ordre avec modulation binaire du coefficient

Définition

Le filtre passif passe-tout du premier ordre décrit par John PIERCE et Scott VAN DUYNE [96] est basé sur l'idée de remplacer la terminaison rigide d'une corde modélisée avec la technique des guides d'ondes par deux ressorts de coefficients différents (cf. figure 2.1). La corde n'est reliée qu'à un seul des deux ressorts à la fois, ainsi ces derniers sont utilisés de manière alternative. Afin de conserver le caractère passif de ce type de système, il est primordial d'effectuer le passage d'un ressort à un autre lorsque le déplacement du ressort utilisé est égal à zéro. Par conscéquent, l'énergie potentielle contenue dans un ressort k_i déplacée sur une distance de x_i mètres est donnée par :

$$k_i x_i^2/2$$

95. LEGGE, Karl ; FLETCHER, Neville, « Nonlinear Generation of Missing Modes on a Vibrating String », *Journal of the Acoustical Society of America*, II (1984), n° 76, p. 5-12.

96. PIERCE, John ; VAN DUYNE, Scott, « A Passive Non-linear Filter Design Which Facilitates Physics-based Sound Synthesis of Highly Nonlinear Musical Instruments », *Journal of Acoustical Society of America*, II (1997), n° 101, p. 1120-1126.

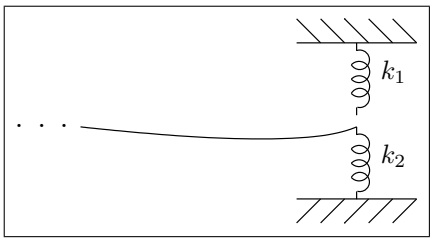

FIGURE 2.1 – Corde terminée par deux ressorts différents : k_1 et k_2. Un seul ressort est utilisé à la fois (dans le cas présent k_2).

On considère une corde idéale en vibration dont l'impédance des ondes est dénotée par R. Terminer cette corde par un ressort idéal k_i a pour effet de réfléchir les ondes se propageant dans la corde et de les filtrer en mettant en valeur une bande de fréquence spécifique, ce qui revient à utiliser un filtre de type passe-tout. En effet, les ondes progressives $y^-(t)$ réfléchies à l'extrémité de la corde sont liées aux ondes d'incidence $y^+(t)$ par

$$Y^-(s) = Y^+(s)H_i(s)$$

où $H_i(s)$ correspond à la fonction de transfert du filtre passe-tout dans le domaine de Laplace :

$$H_i(s) = \frac{s - k_i/R}{s + k_i/R}.$$

Il est possible de créer un modèle numérique du système décrit précédemment en implémentant la corde à l'aide d'un guide d'ondes et en numérisant la réflexion $H_i(s)$ du ressort en effectuant une transformation bilinéaire. La réflexion numérique obtenue peut alors être notée de la façon suivante :

$$H_i(z) = -\frac{a_i + z^{-1}}{1 + a_i z^{-1}} \qquad avec \qquad a_i = \frac{k_i - 2Rf_s}{k_i + 2Rf_s}$$

où f_s est la fréquence d'échantillonnage exprimée en Hertz.

A cause des propriétés induites par la transformation bilinéaire, la reflexion numérique correspond à un filtre passe-tout du premier ordre. Ainsi, les conditions de conservation de l'énergie sont respectées sauf dans le cas où la variable d'état du filtre passe-tout devient égale à zéro, c'est-à-dire au moment où le changement de ressort est effectué.

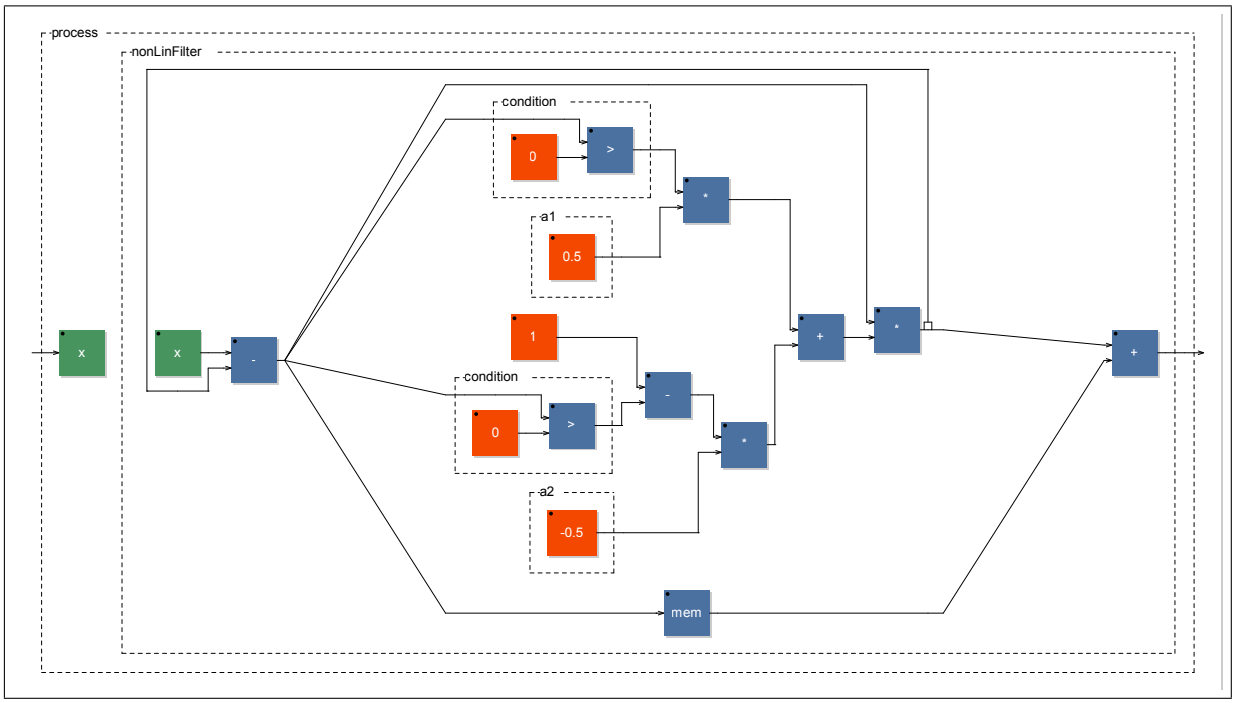

FIGURE 2.2 – Diagramme du filtre passe-tout non-linéaire à modulation binaire du coefficient généré dans FAUST à l'aide de l'outil `faust -svg`, à partir de la fonction `apnl` disponible sur le CD dans le fichier `ksk1k2.dsp` (dossier `/nlfk1k2/`) et dans la bibliothèque `effect.lib` de la distribution de FAUST.

Implémentation

Le filtre décrit précédemment a fait l'objet d'une implémentation dans le langage FAUST et a été intégré à la bibliothèque `effect.lib` de la distribution de ce dernier dans la fonction `apnl` qui prend en argument les deux coefficients a_1 et a_2 utilisés alternativement par le filtre. La variable `condition` permet de sélectionner le coefficient à utiliser :

```
apnl(a1,a2,x) = nonLinFilter
with{
  condition = _ > 0;
  nonLinFilter = (x - _ <: *(condition*a1 + (1-condition)*a2),_')~_
      :> +;
};
```

L'outil `faust -svg` a permis de créer un diagramme de cette fonction visible dans la figure 2.2.

Le filtre passe-tout passif non-linéaire décrit précédemment peut être utilisé avec l'instrument implémenté dans `ks.dsp`[97] basé sur la technique de synthèse de KARPLUS-STRONG présenté dans la partie 1.2. Il suffit de placer le filtre juste avant que

97. Fichier disponible sur le CD dans le dossier `/ks/`.

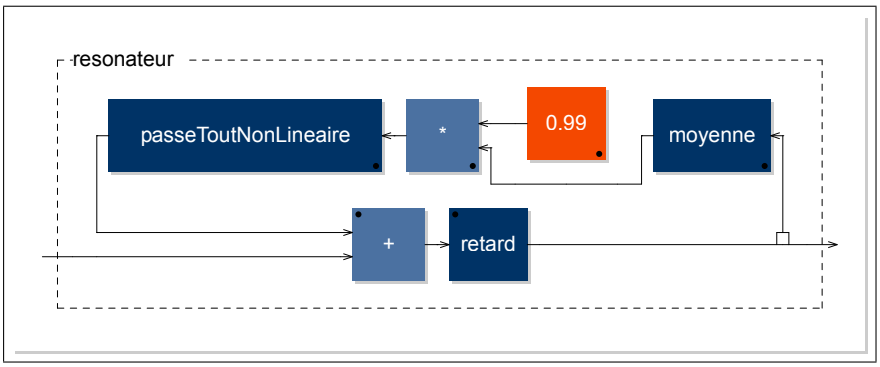

FIGURE 2.3 – Diagramme de la corde de l'instrument implémenté dans le fichier `ksk1k2.dsp` (dossier `/nlfk1k2/` du CD) généré avec l'outil `faust -svg` de FAUST. Le filtre passe-tout non-linéaire est placé dans la boucle juste avant que le signal soit ré-injecté. Le signal issu de la fonction `moyenne` est multiplié par 0,99 afin de diminuer le temps de la résonance.

le signal soit réintroduit dans la boucle du guide d'ondes comme le montre la figure 2.3 qui présente le résonateur de l'instrument implémenté dans le fichier `ksk1k2.dsp`[98]. Afin de faciliter la compréhension et la lecture de la fonction `apnl`, cette dernière a été re-déclarée dans ce fichier.

Dans le fichier audio `ksk1k2Test.wav`[99], une succession de cinq notes de même fréquence (440Hz) est jouée avec des valeurs différentes pour a_1 et a_2 par l'instrument `ksk1k2.dsp`. Dans le cas de la première note, $a_1 = 0$ et $a_2 = 0$ ainsi, aucune non-linéarité n'est introduite dans le son. Pour les notes suivantes, a_1 est incrémenté de 0.25 et a_2 est diminué de 0.25 à chaque nouvelle note jusqu'à arriver à la cinquième note où $a_1 = 1$ et $a_2 = -1$. Le spectrogramme de ce son est visible dans la figure 2.4. Son analyse permet de noter que plus la quantité de non-linéarités introduite dans le son est importante, plus le spectre du son généré est dense. Il est également possible d'observer une répartition dynamique et non-linéaire de l'énergie dans le spectre. Ce phénomène est très visible au niveau de la quatrième note.

Le patch PUREDATA `jouer_ksk1k2.pd`[100] joue le fichier MIDI `voi-che.mid`[101] avec l'instrument décrit ci-dessus et enregistre le résultat dans le fichier audio `voi-Cheksk1k2.wav`[102]. Au début de l'enregistrement, aucune non-linéarité n'est introduite dans le son ($a_1 = 0$ et $a_2 = 0$), puis les valeurs de a_1 et de a_2 sont incrémentées pour l'un

98. Fichier disponible sur le CD dans le dossier `/nlfk1k2/`.
99. *Ibid.*
100. *Ibid.*
101. Fichier disponible sur le CD dans le répertoire `/util/`. Plus d'informations sur ce fichier sont données dans l'annexe A.
102. Fichier disponible sur le CD dans le dossier `/nlfk1k2/`.

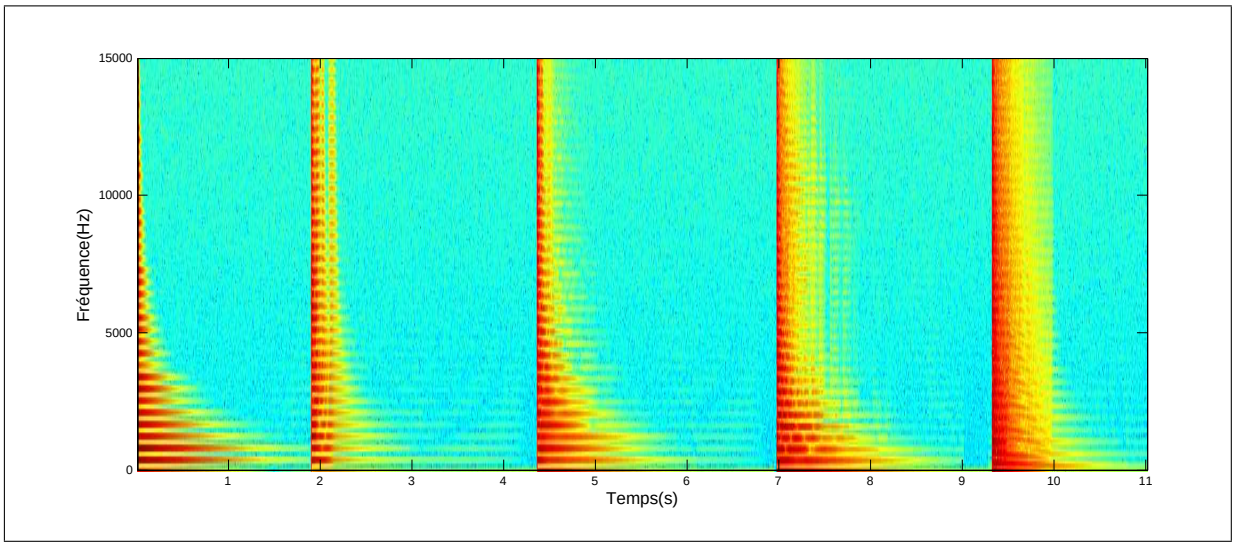

FIGURE 2.4 – Spectrogramme du son `ksk1k2Test.wav` (disponible sur le CD dans le dossier `/nlfk1k2/`) généré avec le programme MATLAB.

et diminuées pour l'autre jusqu'à atteindre respectivement 1 et -1 à la fin de l'air. Dans cet exemple, il est donc possible d'entendre l'effet du filtre non-linéaire sur le son dans un contexte plus musical.

Le filtre passe-tout passif non-linéaire présenté dans cette partie permet d'obtenir des résultats interessants lorsqu'il est utilisé avec des modèles physiques d'instruments à cordes. Il peut néanmoins être aussi employé avec des modèles d'instruments à vent et des percussions bien que les non-linéarités produites soient souvent trop faibles pour avoir un réel intérêt musical.

Dans la section suivante, une technique basée sur la modulation aléatoire de la longueur de la ligne de retards du guide d'ondes permettant de produire des effets similaires est présentée.

2.1.2 Modulation aléatoire de la longueur de la ligne de retards

Le comportement des filtres passe-touts est totalement caractérisé par leurs durées de retards à chaque fréquence. Ainsi, le filtre présenté dans la section précédente peut être assimilé à une modulation non-linéaire de la longueur de la ligne de retards lors de laquelle chaque valeur est interpolée linéairement.

Cette technique a été utilisée dans le passé pour simuler des comportements non-linéaires de cordes. Par exemple, Tero TOLONEN, Vesa VÄLIMÄMI et Matti KAR-

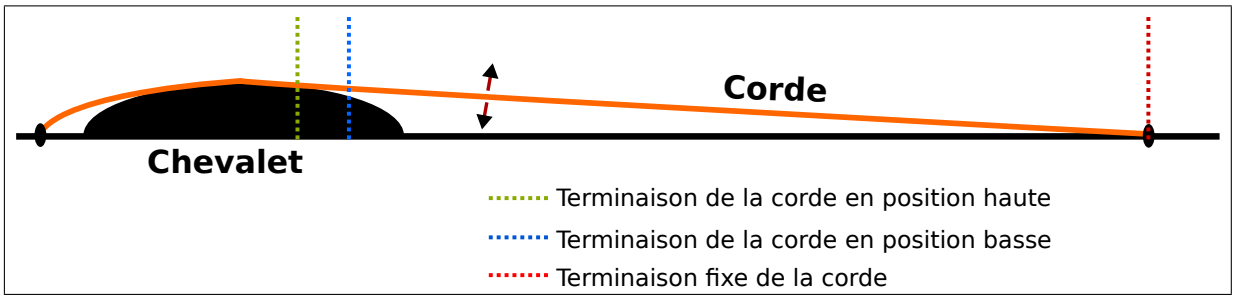

FIGURE 2.5 – Modulation de la longueur de la corde d'un sitar lorsque celle-ci est en vibration. Les proportions réelles n'ont pas été respectées afin de mieux mettre en valeur le phénomène.

JALAINEN ont pu modéliser les variations de la longueur d'une corde en fonction des modulations de sa tension lorsque celle-ci est mise en vibration [103].

Le sitar est un instrument à cordes produisant des sons aux caractéristiques non-linéaires très importantes. Ceci est principalement dû au fait que le chevalet auquel les cordes sont reliées à l'une de leurs extrémités a une forme très recourbée ce qui à pour effet de changer légèrement la longueur des cordes à chaque vibration [104] (cf. figure 2.5). Le tambura reproduit le même effet à cause d'un fil de coton placé près du chevalet.

Il est possible d'imiter de façon assez grossière ce type de comportement en modulant aléatoirement la longueur de la ligne de retards d'une boucle de guide d'ondes. Ceci peut être implémenté dans FAUST de la manière suivante :

```
targetDelay = SR/freq;
delayLength = targetDelay*((1+(0.5*noise)) : smooth(0.9992));
delayLine = delay(4096,delayLength);
```

où `freq` est la fréquence de la note à jouer, `SR` la fréquence d'échantillonnage et `delayLength` la longueur de la ligne de retards en nombre d'échantillons. Cette dernière est modulée par un générateur de bruit blanc (fonction `noise`) et ses valeurs sont interpolées linéairement à l'aide de la fonction `smooth` de la bibliothèque `filter.lib`. La figure 2.7 résume cette opération sous la forme d'un diagramme.

Cette technique a été implémentée dans l'instrument `sitar.dsp` [105] dont le fonctionnement est décrit dans l'annexe G. Il est possible d'entendre l'effet de la modulation aléatoire de la longueur de la ligne de retards sur le son produit dans l'exemple

103. TOLONEN, Tero ; VÄLIMÄKI, Vesa ; KARJALAINEN, Matti, « Modeling of Tension Modulation Nonlinearity in Plucked Strings », *IEE Transaction on Speech and Audio Processing*, SAP-8 (2000), p. 300-310.
104. FLETCHER, Neville ; ROSSING, Thomas, *The Physics of Musical Instruments*, New-York : Springer-Verlag, 1991.
105. Fichier disponible sur le CD dans le dossier `/sitar/`.

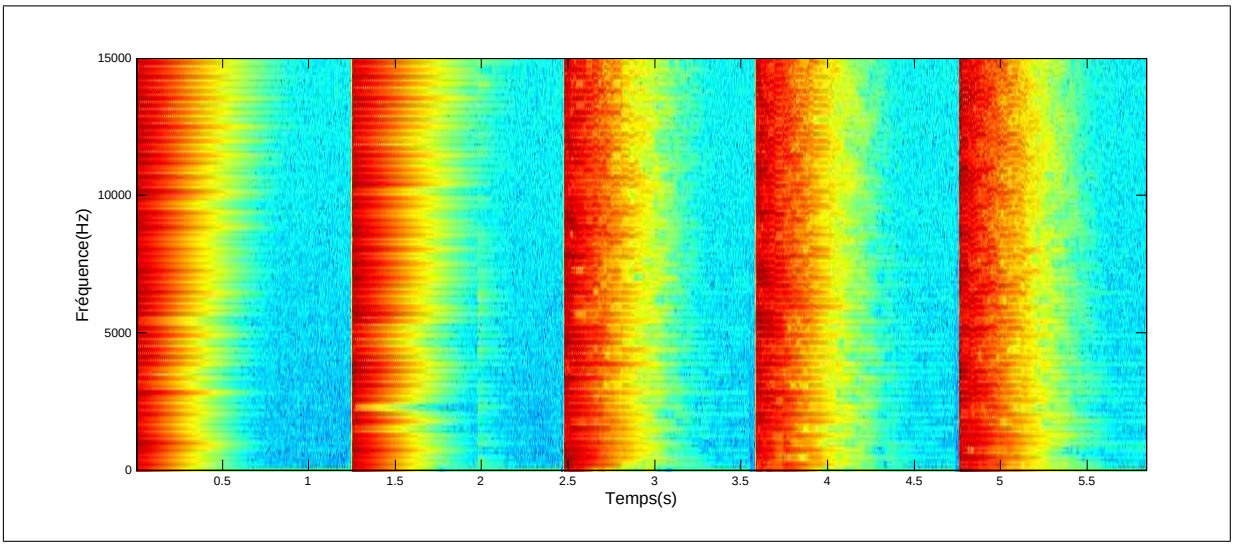

FIGURE 2.6 – Spectrogramme du son `sitarTest.wav` (disponible sur le CD dans le dossier `/sitar/`) généré avec le programme MATLAB.

sonore contenu dans le fichier `sitarTest.wav` [106] dont le spectrogramme est visible dans la figure 2.6. Cinq notes de fréquence égale (440Hz) sont jouées avec des taux de non-linéarités différents. La première note ne contient aucune non-linéarité. Par la suite le taux de non-linéarités est augmenté de vingt-cinq pour cent à chaque note par rapport à la première pour atteindre son maximum à la dernière note.

L'étude du spectrogramme du son permet de tirer des conclusions similaires à celles de la partie précédente : plus le taux de non-linéarités est important, plus le spectre est dense et plus l'énergie présente au niveau de chaque partiel est déplacée de façon dynamique.

Un autre exemple sonore : `sitarVoiChe.wav` [107], plus musical, a été généré à l'aide du patch PUREDATA `sitar.pd` [108] dans lequel est joué un extrait de l'air *Voi Che Sapete* des *Noces de Figaro* de MOZART.

2.2 Filtre passe-tout passif non-linéaire d'ordre arbitraire

Les deux techniques présentées précédemment permettent d'obtenir des comportements non-linéaires intéressants d'un point de vue musical avec les instruments à

106. *Ibid.*
107. *Ibid.*
108. *Ibid.*

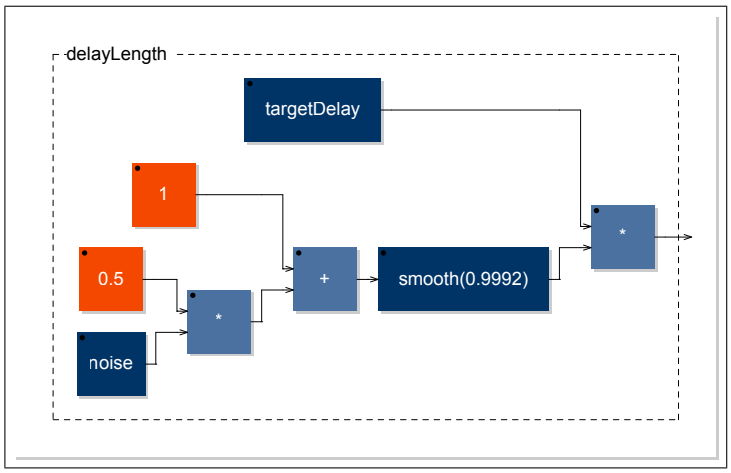

FIGURE 2.7 – Modulation de la longueur de la ligne de retards de l'instrument `sitar.dsp` (disponible sur le CD dans le dossier `/sitar/`). Diagramme généré avec l'outil `faust -svg`.

cordes pincées. Toutefois, les résultats sont beaucoup moins convaincants lorsque ces dernières sont utilisées avec des modèles d'instruments à vent et de percussions par exemple.

La technique présentée dans cette section permet d'augmenter de manière significative les effets du filtre passe-tout passif non-linéaire à modulation binaire du coefficient (cf. partie 2.1.1) en rendant possible le choix de l'ordre du filtre avant la compilation ainsi que l'utilisation de n'importe quel type de modulation pour les coefficients (pas seulement une modulation entre deux valeurs lorsque la variable d'état du filtre le permet).

2.2.1 Définition

Modèle de base

Il a été montré dans la partie 2.1.1 qu'il est possible d'introduire des comportements non-linéaires dans un guide d'ondes en utilisant un filtre passe-tout du premier ordre, stable énergétiquement parlant, dont le coefficient oscille entre deux valeurs lorsque la variable d'état le permet. Le filtre présenté ici permet de dépasser cette condition en autorisant la modulation à chaque échantillon de ses différents coefficients tout en assurant la stabilité énergétique du système.

La méthode utilisée est basée sur la version normalisée du filtre « Ladder » décrit par Augustine GRAY et John MARKEL en 1975 [109]. Ce type de filtre peut être

109. GRAY, Augustine ; MARKEL, John, « A Normalized Digital Filter Structure », *IEE Transaction on Acoustics, Speech and Signal Processing*, ASSP-23 (1975), n° 3, p. 268-277.

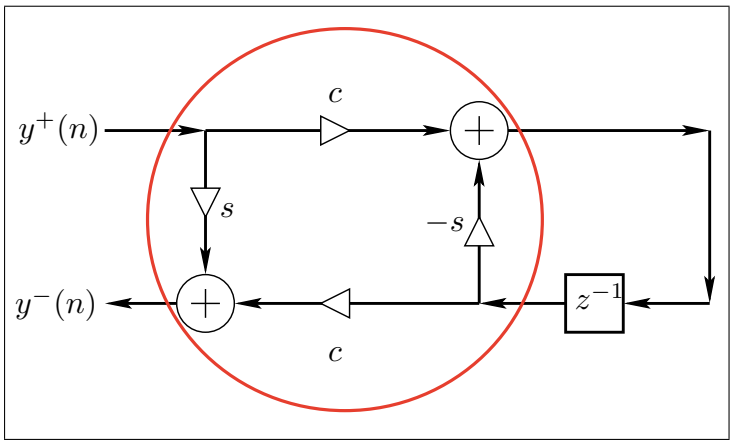

FIGURE 2.8 – Représentation d'un filtre passe-tout normalisé « Ladder » du premier ordre avec les coefficients $c = \cos(\theta)$ et $s = \sin(\theta)$ où $\theta \in [-\pi, \pi]$. Le cercle rouge délimite l'emplacement de la jonction de dispersion normalisée.

apparenté à un réseau de guides d'ondes où les ondes progressives physiques ordinaires sont normalisées [110] tel que le décrit Julius SMITH [111]. Cette normalisation peut être effectuée de plusieurs manières différentes. La plus utilisée, qui fait l'objet de ce premier paragraphe, consiste à s'assurer que le carré de l'amplitude des ondes progressives soit égal à la puissance associée à l'échantillon concerné à travers le réseau de guides d'ondes. Il est possible de voir dans la figure 2.8 la représentation sous forme de diagramme de cette version du filtre passe-tout normalisé « Ladder » (premier ordre). Les coefficients c et s ont alors pour valeur respective $c = \cos(\theta)$ et $s = \sin(\theta)$ avec $\theta \in [-\pi, \pi]$.

Cet algorithme peut être écrit aisément dans le langage FAUST de la manière suivante [112] :

```
c = cos(take(n,tv));
s = sin(take(n,tv));
process = _ <: *(s),(*(c) : (+ : _)~(*(-s))) : _,mem*c : +;
```

Le diagramme généré par le compilateur de FAUST à partir de cette courte ligne de code est visible dans la figure 2.9.

110. SMITH, Julius, « Normalized Scattering Junctions », 2010, *op. cit*, p. 572-574, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/pasp/Normalized_Scattering_Junctions.html`.
111. SMITH, Julius, « Digital Waveguide Filters », 2010, *op. cit*, p. 576-582, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Filters.html`.
112. Code complet disponible dans le fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` du CD.

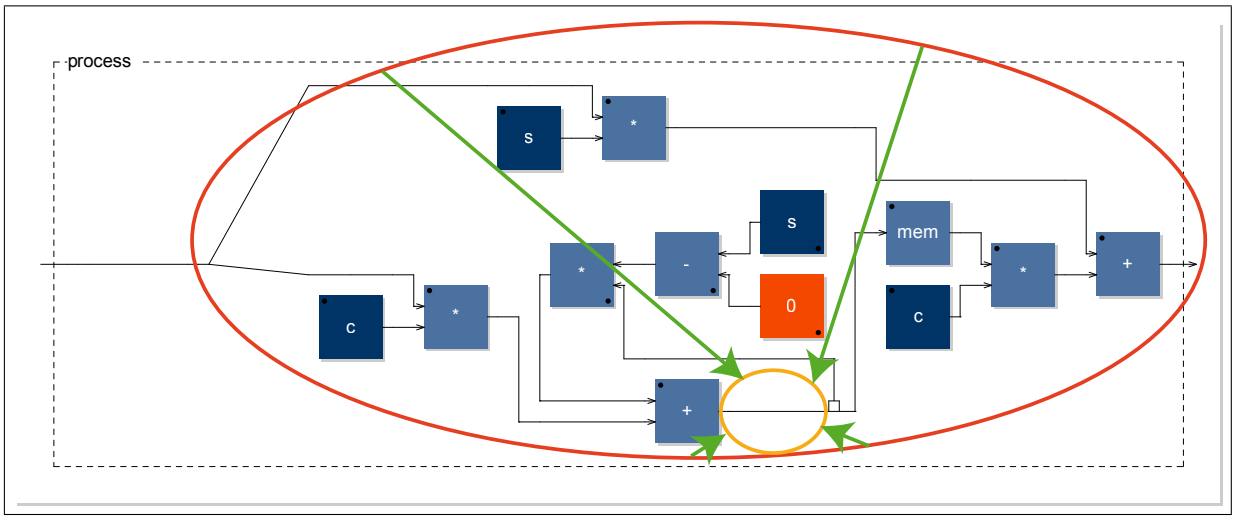

FIGURE 2.9 – Filtre passe-tout passif non-linéaire du premier ordre avec normalisation de la puissance des ondes progressives. Le cercle orange indique l'emplacement où l'algorithme contenu dans le cercle rouge peut être copié afin d'augmenter l'ordre du filtre. Diagramme généré avec l'outil `faust -svg` à partir du fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` du CD.

Choix de l'ordre du filtre

Le code FAUST présenté précédemment peut être significativement amélioré en ajoutant la possibilité de choisir l'ordre du filtre avant la compilation. En effet, une des propriétés des filtres passe-tout est de pouvoir remplacer chaque élément créant un retard dans l'algorithme par un autre filtre passe-tout [113]. Cette opération permet de modifier de façon très simple l'ordre du filtre comme l'illustre les figures 2.9 et 2.10. Le cercle orange présent dans la figure 2.9 indique l'emplacement où l'algorithme contenu dans le cercle rouge sera copié. La figure 2.10 montre le résultat de cette opération, il est donc possible d'y voir le diagramme d'un filtre passe-tout passif non-linéaire du second ordre.

Le système de filtrage par motif (*Pattern Matching*) [114] [115] du langage FAUST a permis de tirer partie de cette propriété en donnant à l'utilisateur le choix de l'ordre du filtre décrit précédemment avant la compilation. Ainsi, le code présenté au début de cette section a été amélioré de la manière suivante [116] (la variable `n` définie l'ordre du filtre) :

```
allpassnnT(0,tv) = _;
```

113. SMITH, Julius, « Nested Allpass Filters », 2010, *op. cit*, p. 71-74, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/pasp/Nested_Allpass_Filters.html`.
114. ORLAREY, Yann ; FOBER, Dominique ; LETZ, Stéphane, *FAUST Quick Reference*, Lyon : GRAME, 2012, p. 16-17.
115. Le terme « filtrage par motif » est défini dans le glossaire à la page 120.
116. Code complet disponible dans le fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` de CD.

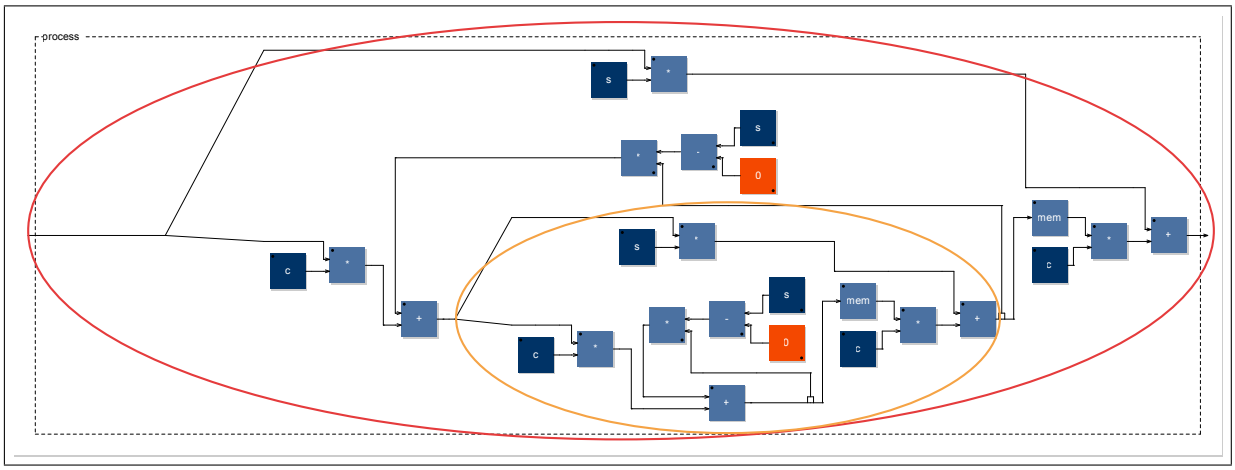

FIGURE 2.10 – Filtre passe-tout passif non-linéaire du second ordre avec normalisation de la puissance des ondes progressives. Le cercle orange contient la partie de l'algorithme copiée contenue dans le cercle rouge de la figure 2.9. Diagramme généré avec l'outil `faust -svg` à partir du fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` du CD.

```
allpassnnT(n,tv) = _ <: *(s), (*(c) : (+
    : allpassnnT(n-1,tv))~(*(-s))) : _, mem*c : +
with{
    c = cos(take(n,tv));
    s = sin(take(n,tv));
};
```

Autres formes

Le type de jonction de dispersion utilisé dans le filtre présenté précédemment permet de normaliser la puissance des ondes progressives qui se déplacent dans le réseau de guides d'ondes assurant alors la stabilité énergétique de ce dernier. D'autres types de jonction de dispersion pour la normalisation des ondes progressives peuvent être utilisés.

La jonction de *Kelly-Lochbaum* est la première à avoir été découverte et est très similaire à celle présentée précédemment (cf. figure 2.11) bien qu'elle soit légèrement moins stable numériquement. Elle était utilisée pour connecter les différentes sections du modèle physique d'appareil phonatoire mentionné dans la partie 1.4 et décrit par John KELLY et Carol LOCHBAUM [117]. La théorie autour de ce type de jonction a été formalisée et étendue à l'utilisation dans des réseaux de guides d'ondes dans les années quatre-vingt par Julius SMITH [118].

117. KELLY, John ; LOCHBAUM, Carol, *Speech Synthesis : actes du Fourth International Congress on Acoustics, Copenhague, septembre 1962*, article G42, Stockholm : Speech Transmission Lab (Royal Institute of Technology), 1962.

118. SMITH, Julius, « Kelly-Lochbaum Scattering Junctions », 2010, *op. cit*, p. 569-570, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/pasp/Kelly_Lochbaum_`

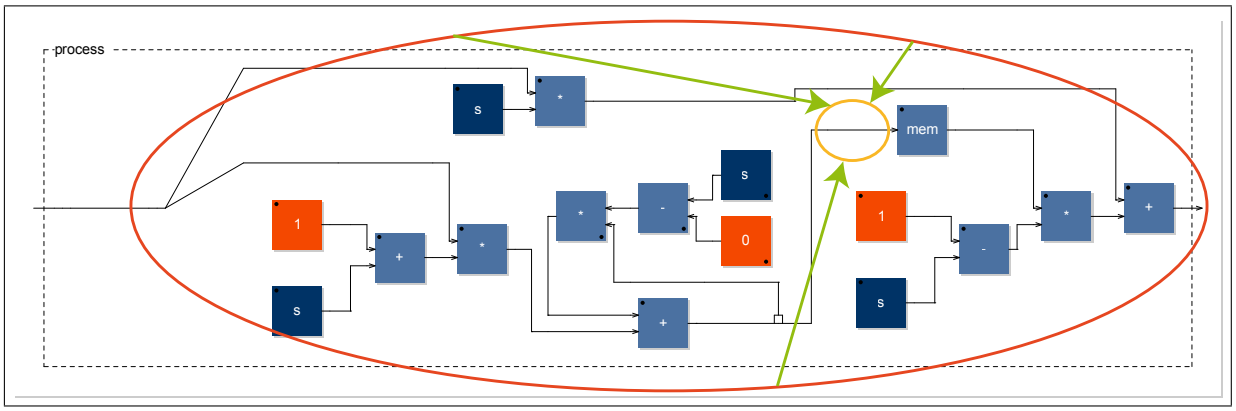

FIGURE 2.11 – Filtre passe-tout passif non-linéaire du premier ordre utilisant une jonction de *Kelly-Lochbaum* pour le normalisation des ondes progressives. Le cercle orange indique l'emplacement où l'algorithme contenu dans le cercle rouge peut être copié afin d'augmenter l'ordre du filtre. Diagramme généré avec l'outil `faust -svg` à partir du fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` du CD.

Il est possible de modifier le code FAUST [119] du filtre présenté précédemment afin d'utiliser des jonctions de *Kelly-Lochbaum* :

```
allpassnklT(0,sv) = _;
allpassnklT(n,sv) = _ <: *(s),(*(1+s) : (+
    : allpassnnT(n-1,sv))~(*(-s))) : _, mem*(1-s) : +
with{ s = take(n,sv); };
```

Dans ce cas, la liste de coefficients `sv` du filtre doit vérifier $\texttt{sv} \in [-1, 1]$. Le diagramme généré par le compilateur de FAUST à partir de ce code est disponible dans la figure 2.11.

Ce type de jonction a été fortement simplifié par John MARKEL et Augustine GRAY dans les années soixante-dix [120] en réduisant le nombre de multiplications nécessaires à une (la jonction de *Kelly-Lochbaum* implique l'utilisation de quatre multiplications). Julius SMITH a là aussi étendu cette théorie aux réseaux de guides d'ondes numériques et parle alors de *one-multiply scattering junction* [121]. Il est également important de mentionner les travaux de formalisation de Dana MASSIE sur le sujet [122].

Le code FAUST du filtre passe-tout passif non-linéaire utilisant ce type de

`Scattering_Junctions.html`.

119. Code complet disponible dans le fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` de CD.

120. MARKEL, John ; GRAY, Augustine, *Linear Prediction of Speech*, New York : Springer Verlag, 1976.

121. SMITH, Julius, « One-Multiply Scattering Junctions », 2010, *op. cit*, p. 570-572, artcile disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/pasp/One_Multiply_Scattering_Junctions.html`.

122. MASSIE, Dana, « An Engineering Study of the Four-Multiply Normalized Ladder Filter », *Journal of Audio Engineering Society*, XLI (1993), n° 7/8, p. 564-582.

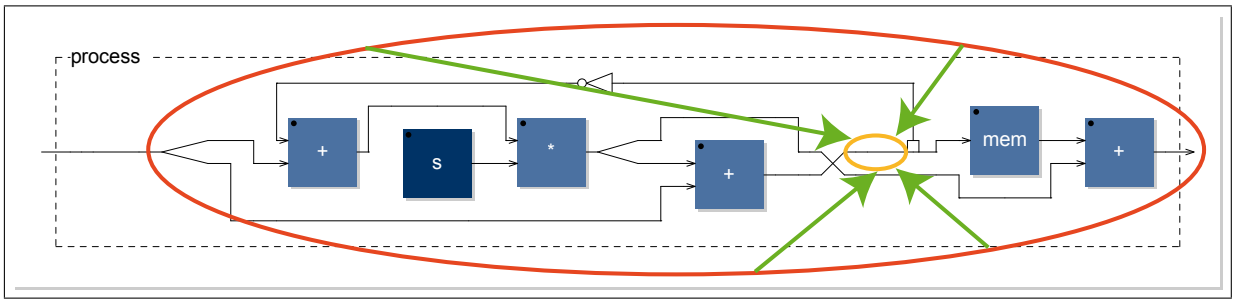

FIGURE 2.12 – Filtre passe-tout passif non-linéaire du premier ordre utilisant la version simplifiée à une multiplication de la jonction de *Kelly-Lochbaum* pour le normalisation des ondes progressives. Le cercle orange indique l'emplacement où l'algorithme contenu dans le cercle rouge peut être copié afin d'augmenter l'ordre du filtre. Diagramme généré avec l'outil `faust -svg` à partir du fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` du CD.

jonction peut être écrit de la manière suivante [123] :

```
allpassn1mT(0,sv) = _;
allpassn1mT(n,sv)= _ <: _,_ : ((+:*(s) <: _,_),_ : _,+ : cross
     : allpassn1mT(n-1,sv),_)~(*(-1)) : _',_ : +
with{
     s = take(n,sv);
     cross = _,_ <: !,_,_,!;
};
```

Le diagramme généré par le compilateur de FAUST à partir de ce code permet de constater l'ampleur de la simplification de l'algorithme (cf. figure 2.12).

L'ensemble de ces fonctions a été intégré à la distribution de FAUST au travers des fonctions `allpassn`, `allpassnn`, `allpassnkl` et `allpassn1m` dans la bibliothèque `filter.lib`.

2.2.2 Stabilité énergétique

La stabilité énergétique du filtre présenté au début de la partie précédente (filtre passe-tout passif non-linéaire utilisant des jonctions de dispersions normalisant la puissance des ondes progressives) lorsque ses coefficients sont modulés à chaque échantillon a été testée dans FAUST et le programme MATLAB [124] avec le fichier `allpassnn.dsp`. Le filtre utilisé pour le test est d'ordre trois (`N = 3`). Ses coefficients sont déterminés par la variable `theta` (cf. section 2.2.1) dont la valeur est modulée de façon aléatoire à chaque échantillon :

123. Code complet disponible dans le fichier `allpassnn.dsp` contenu dans le dossier `/allpassnn/` de CD.
124. Plus d'informations sur ce langage de programmation sont données dans le glossaire à la page 120.

```
N = 3;
theta = par(i,N,PI*noise@(i+1));
```

Le filtre est alimenté par une source de bruit blanc et son amplitude de sortie est convertie en amplitude RMS.

```
rms = ^(2) : smooth(0.999) : sqrt;
uvwnoise = sqrt(3)*noise;

process = uvwnoise : allpassnnT(N,theta) : rms;
```

Les 2100 premiers échantillons issus des signaux d'entrée et de sortie du filtre ont par la suite été enregistrés dans deux fichiers textes : `allpassnnAmpTestValIn.txt` [125] et `allpassnnAmpTestValOut.txt` [126] à l'aide du script `faust2plot` [127]. Ces données ont alors été importées dans le programme MatLab afin de les placer dans un graphique à l'aide du fichier `allpassnnAmpTest.m` [128]. Ce graphique est visible dans le figure 2.13.

On peut voir qu'il est impossible de distinguer à l'oeil nu les courbes d'amplitude du signal d'entrée et du signal de sortie du filtre. On déduit de cette experience que le filtre passe-tout normalisé « Ladder » n'a aucun effet sur le signal traité même lorsque ses coefficients évoluent rapidement de façon aléatoire.

Des résultats similaires ont été obtenus en effectuant la même comparaison avec les autres types de jonction de dispersion.

2.3 Utilisation du filtre passe-tout passif non-linéaire d'ordre arbitraire

Le filtre décrit précédemment offre un nombre important d'applications pour la synthèse des sons. Il permet d'améliorer de façon significative les modèles physiques d'instruments de musique utilisant la technique des guides d'ondes numériques. Le type de modulations applicables sur les coefficients du filtre étant très nombreux, seul un nombre assez restreint de possibilités ont été testées à l'heure actuelle. Ces dernières sont présentées dans cette partie.

125. Fichier disponible sur le CD dans le dossier `/allpassnn/testStabilite/`.
126. *Ibid.*
127. Plus d'informations sur le script `faust2plots` peuvent être trouvées dans le glossaire à la page 120.
128. Fichier disponible sur le CD dans le dossier `/allpassnn/testStabilite/`.

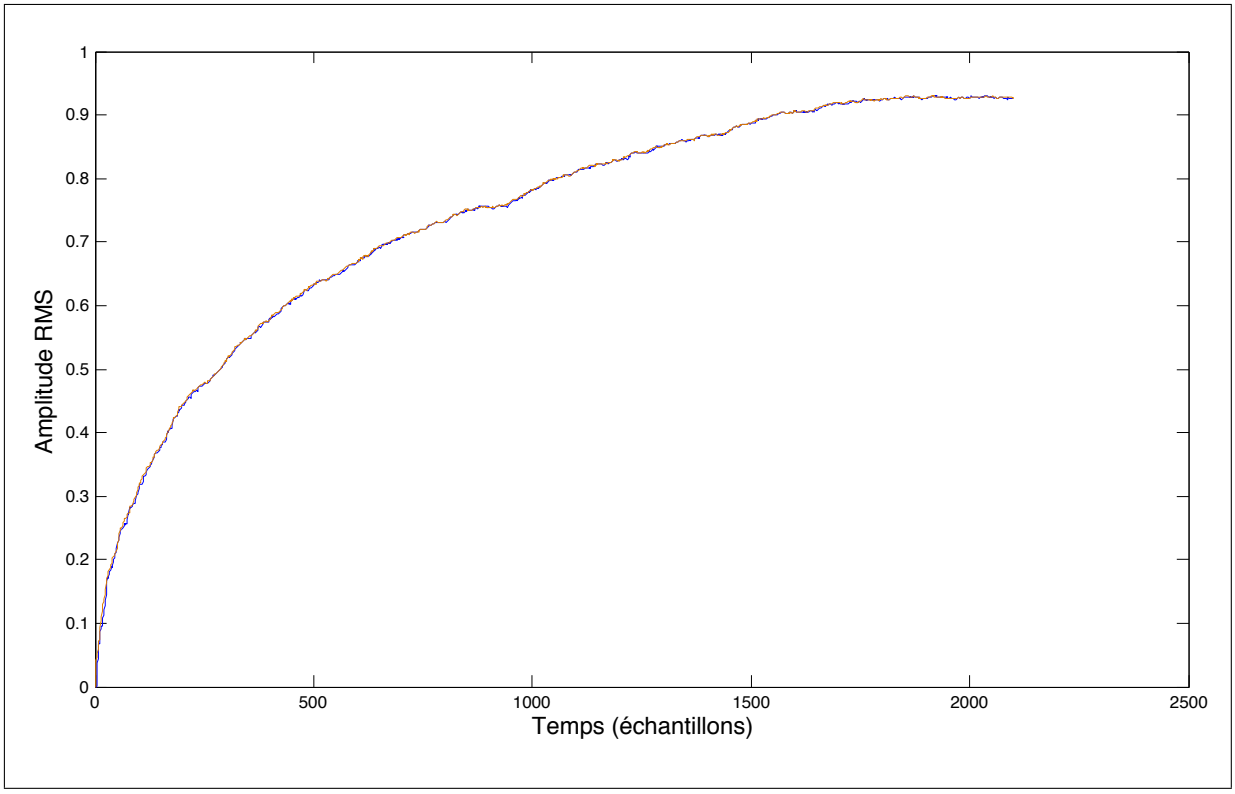

FIGURE 2.13 – Amplitudes RMS des signaux d'entrée et de sortie du filtre passe-tout normalisé « Ladder » lorsque celui-ci est alimenté par un générateur de bruit blanc et que ses coefficients sont modulés à chaque échantillon. La courbe bleue correspond à l'amplitude du signal de sortie du filtre et la courbe orange à l'amplitude du signal d'entrée. Il est impossible de les distinguer à l'oeil nue.

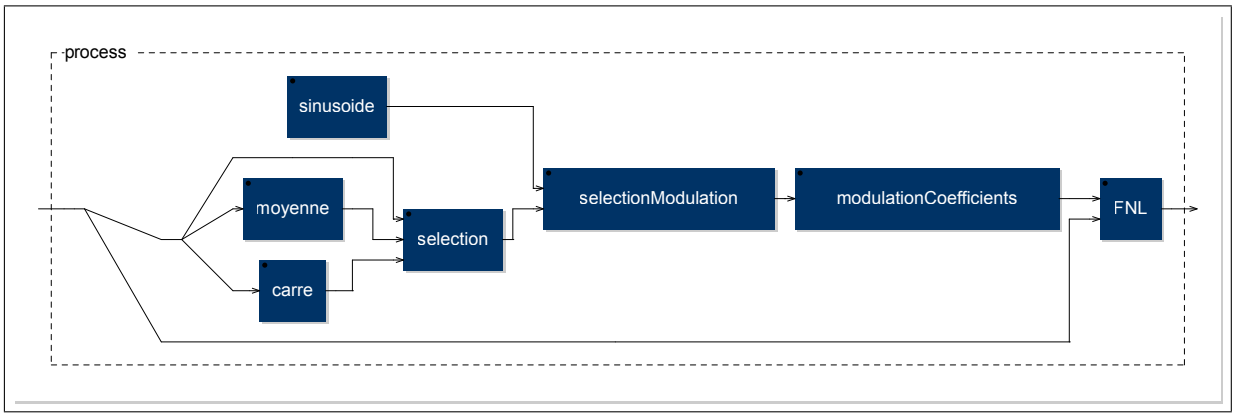

FIGURE 2.14 – Diagramme du mode de fonctionnement de la fonction `nonLinearModulator` de la bibliothèque `instrument.lib` (disponible sur le CD dans le dossier `/faust-stk/`). La boite *FNL* représente le filtre passe-tout passif non-linéaire. Diagramme généré avec l'outil `faust -svg`.

2.3.1 La fonction `nonLinearModulator` du FAUST-STK

Il a été montré précédemment que les coefficients du filtre passe-tout passif non-linéaire (FNL) peuvent être modifiés à chaque échantillon par n'importe quel type de signal. La fonction `nonLinearModultor` de la bibliothèque `instrument.lib` [129] dont le fonctionnement est détaillé dans l'annexe Q tire ainsi partie de cette propriété en offrant la possibilité de moduler les coefficients du filtre soit par un signal sinusoidal, soit par le signal entrant dans le filtre. Pour ce dernier cas, il est possible de choisir entre différentes versions du signal :

- la valeur de chaque échantillon est mise au carré : $coefficient = x(n)^2$;
- on fait la moyenne de la valeur d'un échantillon avec celle du précédent est faite : $coefficient = \frac{x(n)+x(n-1)}{2}$;
- le signal lui-même est utilisé : $coefficient = x(n)$.

Si les coefficients du filtre sont modulés par le signal sinusoidal, il est possible de choisir si la fréquence de ce dernier est la même que celle de la note jouée ou bien si elle est différente.

Le mode de fonctionnement de la fonction `nonLinearModulator` est résumé dans la figure 2.14.

129. Fichier disponible sur le CD dans le dossier `/faust-stk/`.

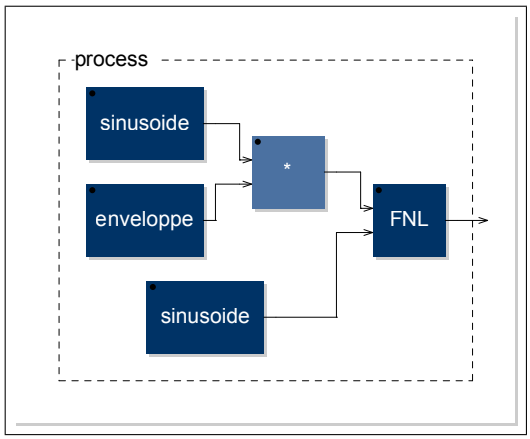

FIGURE 2.15 – Diagramme de l'algorithme de l'instrument `sineAllpasnn.dsp` (disponible sur le CD dans le dossier `/allpassnn/utilisation/sineAllpassnn/`) généré avec l'outil `faust -svg`.

2.3.2 Test de base sur un signal sinusoidal

Les effets du filtre FNL ont été mesurés sur un signal sinusoidal afin de voir son impact sur le spectre du son. Pour ceci, l'instrument `sineAllpassnn.dsp`[130] dont l'algorithme est résumé dans le figure 2.15 a été utilisé dans PUREDATA avec le patch `jouer_sineAllpassnn.pd`[131]. Un filtre passe-tout passif non-linéaire avec des jonctions de dispersion normalisant la puissance des ondes progressives a été utilisé dans cet exemple.

Plusieurs enregistrements utilisant différentes configurations ont été effectués lors de ces tests et ont été répertoriés dans le tableau 2.1. Tous sont disponibles dans le dossier `/allpassnn/utilisation/sineAllpassnn/` du CD.

n°	Nom du fichier	Ordre du filtre	Taux de non-linéarités	Fréquence de modulation
1	`sineNLEqFreqOrd2.wav`	2	croissant de 0 à 1	constante : 440Hz
2	`sineNLEqFreqOrd4.wav`	4	croissant de 0 à 1	constante : 440Hz
3	`sineNLDifFreq.wav`	2	constant : 1	croissante de 440Hz à 880Hz

TABLE 2.1 – Caractéristiques des sons enregistrés avec le patch PUREDATA `jouer_sineAllpasnn.pd` (disponible sur le CD dans le dossier `/allpassnn/utilisation/sineAllpassnn/`). Chacun d'entre-eux ont une fréquence principale de 440Hz.

Le fichier audio numéro un permet de montrer l'effet de l'augmentation du taux de non-linéarités sur le spectre de la sinusoide. Le test a été effectué avec une fréquence de modulation égale à celle du son traité. Il est possible de noter à l'observation du

130. Fichier disponible sur le CD dans le dossier `/allpassnn/utilisation/sineAllpassnn/`.
131. *Ibid.*

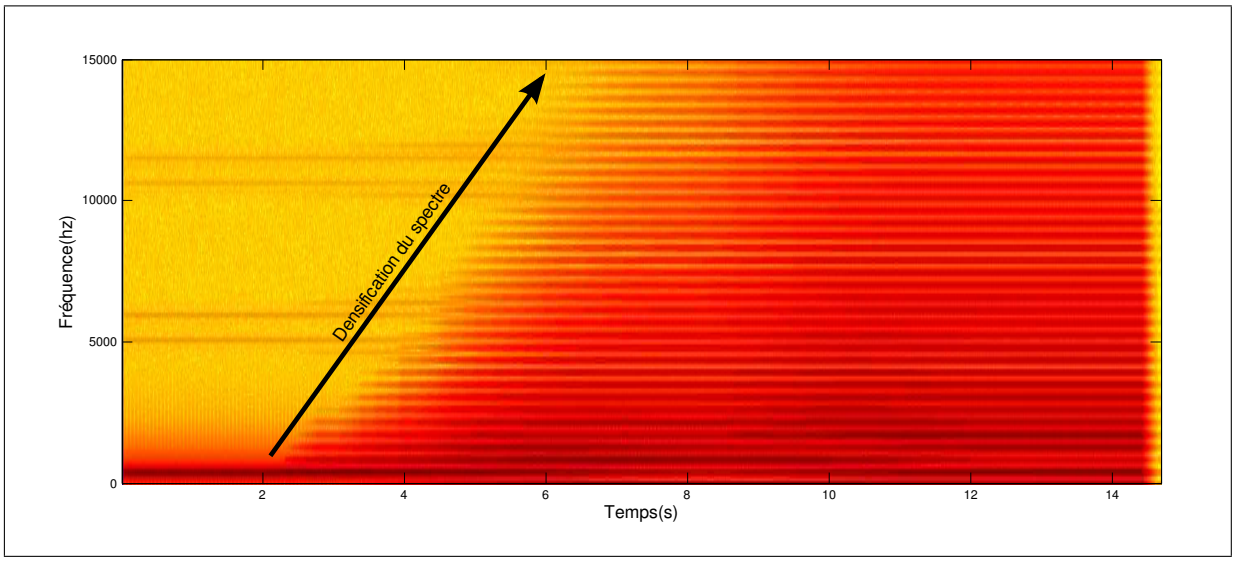

FIGURE 2.16 – Spectrogramme du son `sineNLeqFreqOrd2.wav` (disponible sur le CD dans le dossier `/allpassnn/utilisation/sineAllpassnn/`) généré avec le programme MatLab. Le taux de non-linéarités est augmenté de façon linéaire entre *temps* = 2s (*taux de non-linéarités* = 0) et *temps* = 14s (*taux de non-linéarités* = 1).

spectrogramme de ce son (cf. figure 2.16) que plus le taux de non-linéarités est important, plus le spectre du son est enrichi par des partiels dont la fréquence peut être calculée de la manière suivante :

$$f_n = f_0 + f_0 \times n$$

où f_0 est la fréquence de la fondamentale et f_n la fréquence du partiel n. A l'écoute, on reconnait aussi très facilement le son produit par un synthétiseur à modulation de fréquence (série harmonique standard).

Le fichier audio numéro deux permet de montrer que l'utilisation d'un filtre d'ordre plus élevé a pour effet de renforcer l'amplitude de l'ensemble des partiels du son.

Dans le fichier audio numéro trois, le taux de non-linéarités du filtre est réglé au maximum et la fréquence de modulation de ses coefficients est progressivement augmentée pour être doublée. D'après les observations faites sur le spectrogramme de ce son (cf. figure 2.17), il est possible de compléter la formule donnée précédemment de la manière suivante :

$$f_n = f_0 + (f_0 \times n + (f_{mod} - f_0))$$

où f_{mod} est la fréquence de l'oscillateur modulant. Toutefois, ceci n'est vrai que pour les partiels compris entre f_1 et f_{36}. En effet, d'après les observations faites sur le spectrogramme de la figure 2.17, les partiels au dessus de cette limite semblent obéir à la règle

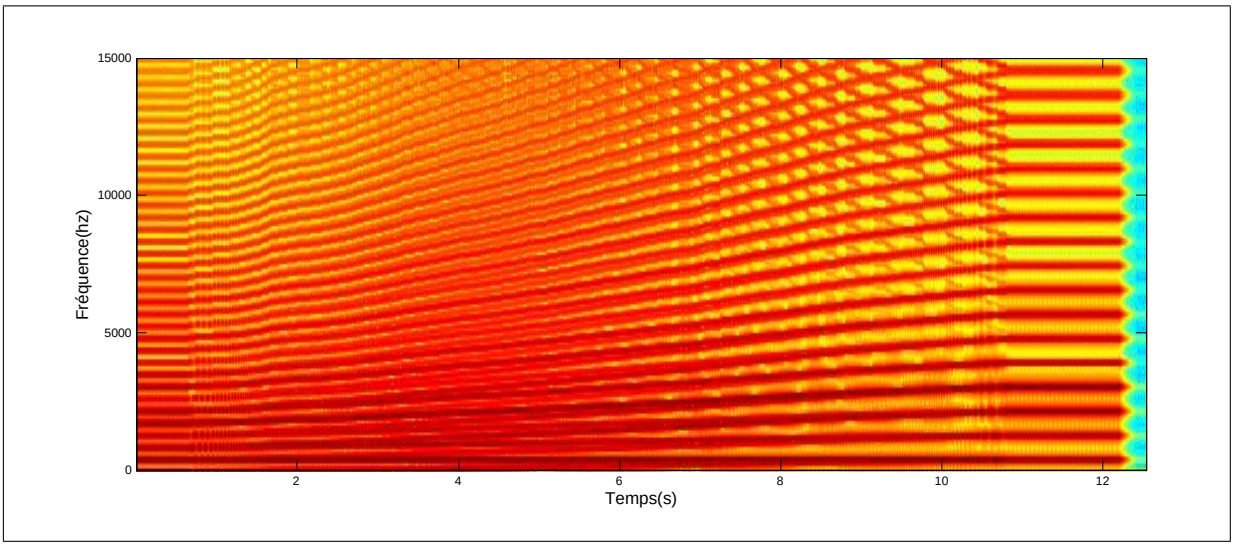

FIGURE 2.17 – Spectrogramme du son `sineNLeqFreqDifFreq.wav` (disponible sur le CD dans le dossier `/allpassnn/utilisation/sineAllpassnn/`) généré avec le programme MATLAB. La fréquence de l'oscillateur modulant les coefficients du filtre est augmentée de façon linéaire entre $temps = 0.7$s ($f_{mod} = 440$Hz) et $temps = 11.8$s ($f_{mod} = 880$Hz).

suivante :

$$f_n = f_0 + (f_0 \times n - (f_{mod} - f_0))$$

2.3.3 Utilisation sur des modèles physiques par guides d'ondes

Le type d'utilisation du filtre passe-tout passif non-linéaire présenté dans la partie précédente, bien qu'intéressant, est en revanche assez peu utile et n'apporte rien de très innovant puisqu'il est possible d'acquérir le même type de résultat avec la synthèse par modulation de fréquence [132], connue depuis maintenant une trentaine d'années. Une utilisation plus intéressante de ce type de filtre peut être faite pour la création de comportements non-linéaires dans des modèles physiques d'instruments par guides d'ondes tels que ceux décrits dans le chapitre 3. Pour ceci, il suffit de placer le FNL dans la boucle du guide d'ondes de l'instrument comme le montre les figures 2.18 et 2.20 dans lesquelles l'emplacement du FNL est indiquée par un cercle rouge.

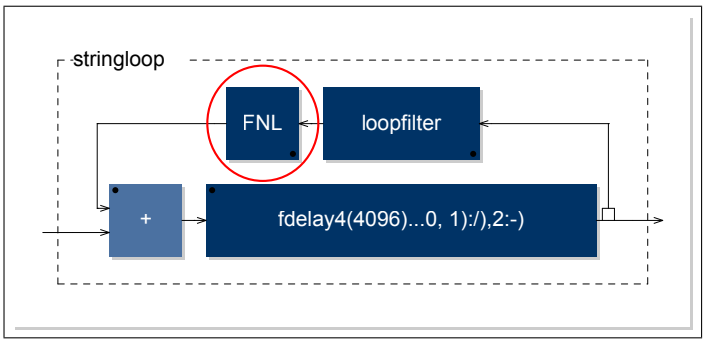

FIGURE 2.18 – Diagramme de l'instrument `eksAllpassnn.dsp` (disponible sur le CD dans le dossier `/allpassnn/utilisation/eksAllpassnn`) généré avec l'outil `faust -svg`.

Exemple d'utilisation avec un modèle d'instrument à corde

L'instrument présenté dans la figure 2.18 qui est implémenté dans le fichier `eksAllpassnn.dsp` [133] est une version modifiée de celle présentée dans la partie 1.3 portant sur la version étendue de l'algorithme de KARPLUS-STRONG. Une série de tests sonores ont été effectués avec différents types de modulations pour les coefficients du filtre à l'aide du patch PUREDATA `jouer_eksAllpassnn.pd` [134]. Dans la liste suivante, le contenu de chaque fichier est décrit et comparé [135] :

- `eksAllpassnnTest.wav` : Une série de quatre notes de fréquence égale (261Hz : C4) sont jouées de manière successive. Les coefficients du FNL (sixième ordre) sont modulés par le signal entrant dans le filtre. Le taux de non-linéarités est augmenté à chaque note de +0.25 en partant de 0 pour la première pour atteindre 1 lors de la dernière. Cet exemple permet de montrer l'effet sur le son de l'augmentation du taux de non-linéarités. Si ce dernier prend des valeurs élevées (supérieures à 0.9), alors le son produit est assez semblable à celui du sitar présenté dans la partie 2.1.2 bien qu'il soit légèrement moins « granuleux ».
- `eksAllpassnnVoiCheSigMod.wav` : Le fichier MIDI `voi-che.mid` [136] est joué par le modèle. Les coefficients du FNL (sixième ordre) sont modulés par le signal entrant dans le filtre. Le taux de non-linéarités est constant

132. CHOWNING, John ; BISTROW, David, *FM Thoery and Applications*, Tokyo : Yamaha Corporation, 1986.
133. Fichier disponible sur le CD dans le dossier `/allpassnn/utilisation/eksAllpassnn/`.
134. *Ibid.*
135. Tous sont disponibles dans le dossier `/allpassnn/utilisation/eksAllpassnn/` du CD.
136. Fichier disponible sur le CD dans le dossier `/util/`. Plus d'informations sur ce fichier sont données dans l'annexe A.

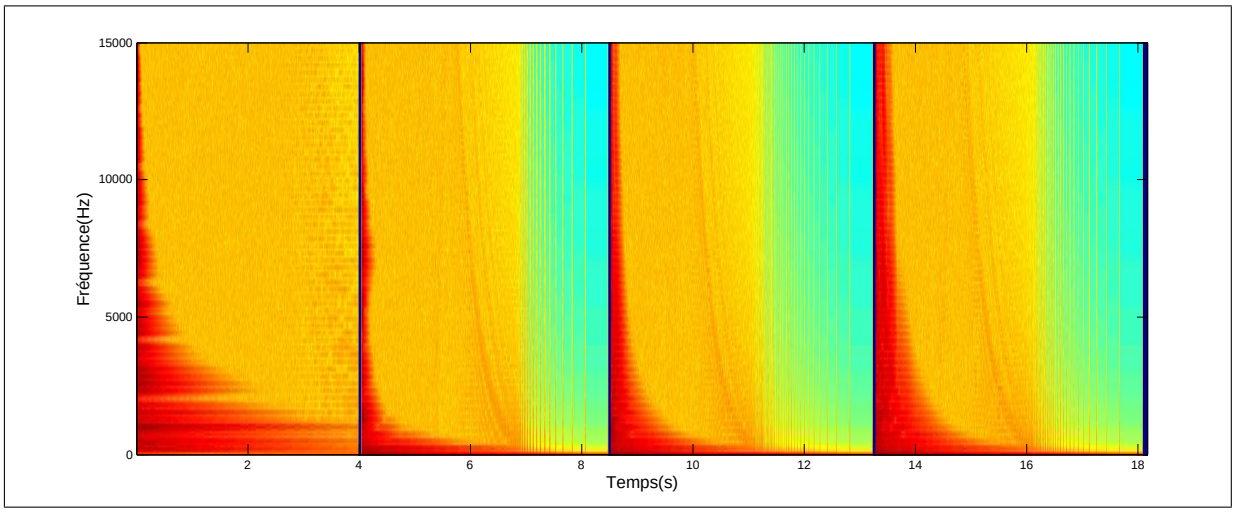

FIGURE 2.19 – Spectrogramme du son `eksAllpassnnTest.wav` (disponible sur le CD dans le dossier `/allpassnn/utilisation/eksAllpassnn/`) généré avec le programme MATLAB.

et est égal à 1. Dans cet exemple, l'utilisation du modèle dans un contexte plus musical est donnée.

– `eksAllpassnnVoiCheSineMod.wav` : Le fichier MIDI `voi-che.mid` est joué par le modèle. Les coefficients du FNL (sixième ordre) sont modulés par un signal sinusoidal dont la fréquence est égale à celle de la note jouée. Le taux de non-linéarités est augmenté de façon linéaire entre le début (*taux de non-linéarités* = 0) et la fin de l'enregistrement (*taux de non-linéarités* = 1). La modulation des coefficients du FNL par un signal sinusoidal est une méthode beaucoup plus radicale que celle utilisée précédemment et produit des sons assez peu naturels bien que très intéressants. Un effet de modulation de fréquence du modèle physique, inédit, est donc introduit ici.

Un spectrogramme du son enregistré dans le fichier `eksAllpassnnTest.wav` est visible dans la figure 2.19. Il permet d'observer les effets du FNL sur le spectre du son en comparant la première note jouée avec les autres. Il est ainsi possible de noter les phénomènes suivants :

– réduction du temps de résonance du modèle ;
– densification et homogénéisation du spectre ;
– allongement du temps de résonance des partiels hauts.

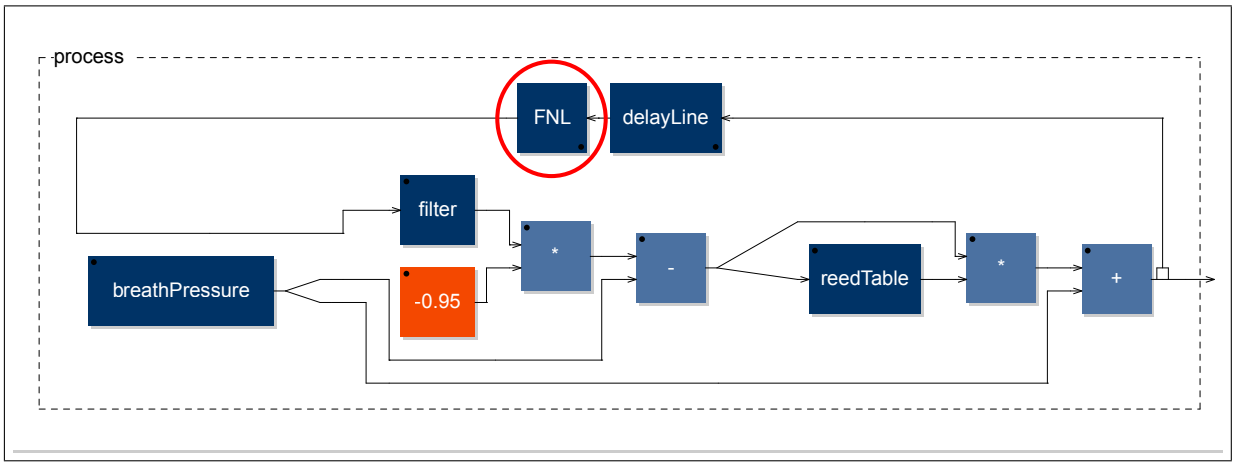

FIGURE 2.20 – Diagramme de l'instrument `clarAllpassnn.dsp` (disponible sur le CD dans le dossier `/allpassnn/utilisation/clarAllpassnn`) généré avec l'outil `faust -svg`.

Exemple d'utilisation avec un modèle d'instrument à vent

Les mêmes types de tests ont été effectués avec l'instrument `claAllpassnn.dsp` [137] dont le diagramme est présenté dans la figure 2.20. C'est une version modifiée de l'algorithme de clarinette présenté dans la partie 1.4. Les tests ont été menés à bien grâce au patch PUREDATA `jouer_clarAllpassnn.pd` [138]. Dans la liste suivante, le contenu de chaque fichier est décrit et comparé avec les autres [139] :

- `clarAllpassnnSigMod.wav` : une note de fréquence constante de 440Hz est jouée par le modèle pendant une période de vingt-et-une secondes. Les coefficients du FNL (sixième ordre) sont modulés par le signal entrant dans le filtre. Le taux de non-linéarités est augmenté progressivement en débutant à 0 au début de l'enregistrement et en terminant à 1 à la fin.
- `clarAllpassnnSineModE.wav` : une note de fréquence constante de 440Hz est jouée par le modèle pendant une période de dix-neuf secondes. Les coefficients du FNL (sixième ordre) sont modulés par un signal sinusoidal de fréquence fixe égale à celle de la note jouée. Le taux de non-linéarités est augmenté progressivement en débutant à 0 au début de l'enregistrement et en terminant à 1 à la fin.
- `clarAllpassnnSineModD.wav` : une note de fréquence constante de 440Hz est jouée par le modèle pendant une période de dix-huit secondes. Les

137. Fichier disponible sur le CD dans le dossier `/allpassnn/utilisation/clarAllpassnn`.
138. *Ibid.*
139. Tous sont disponibles dans le dossier `/allpassnn/utilisation/clarAllpassnn/` du CD.

coefficients du FNL (sixième ordre) sont modulés par un signal sinusoidal dont la fréquence est augmentée de façon linéaire en passant de 440Hz au début de l'enregistrement à 880 à la fin. Le taux de non-linéarités reste constant et est égal à un.

- `clarAllpassnnVoiCheSigMod0_1.wav` : le fichier `voi-che.mid` est joué par le modèle. Les coefficients du FNL (sixième ordre) sont modulés par le signal entrant dans le filtre. Une enveloppe du type *Attaque – Maintien – Chute* est appliquée sur le taux de non-linéarités dont la valeur maximale atteinte lors du maintien est de 0.1. La durée de l'attaque pour les non-linéarités est de 0.3 secondes pour chaque note. Dans le cas présent, les non-linéarités introduites dans le son produit par le modèle de clarinette sont presque imperceptibles à l'écoute, le but recherché ici étant de générer un son le plus naturel possible. Cet exemple sonore peut-être comparé au fichier `voiChe-clarinette.wav`[140] présenté dans la partie 1.4 qui ne contient aucune non-linéarités. L'aspect naturel du son est renforcé par l'utilisation d'une enveloppe pour contrôler le taux de non-linéarités.
- `clarAllpassnnVoiCheSigMod0_5.wav` : afin d'aider à distinguer les non-linéarités présentes dans l'exemple précédent, ce dernier a été rejoué en utilisant un taux de non-linéarités plus important d'une valeur de 0.5.

Il est possible de comparer le spectrogramme du fichier `clarAllpassnnVoiCherSigMod0_5.wav` visible dans la figure 2.22 avec celui du fichier `voiChe-clarinette.wav` de la partie 1.4 disponible dans la figure 2.21 afin d'observer les effets du FNL sur le spectre du son. Il est alors possible de noter que ces derniers sont assez similaires à ceux décrits dans le cas du modèle de corde pincée : spectre plus dense et partiels hauts renforcés. Il semble également que le vibrato soit beaucoup plus marqué dans le cas où le FNL est utilisé. Enfin, il est interessant d'observer l'effet du générateur d'enveloppe sur le taux de non-linéarités au début de chaque note avec la mise en place progressive du renforcement spectral observé précédemment.

Le filtre passe-tout passif non-linéaire d'ordre arbitraire permet, lorsqu'il est utilisé dans un modèle d'instrument de musique par guide d'ondes d'élargir considérablement le panel de sons productibles par ce dernier. Les tests présentés ici n'ont été

140. Fichier disponible sur le CD dans le dossier `/clarinette/`.

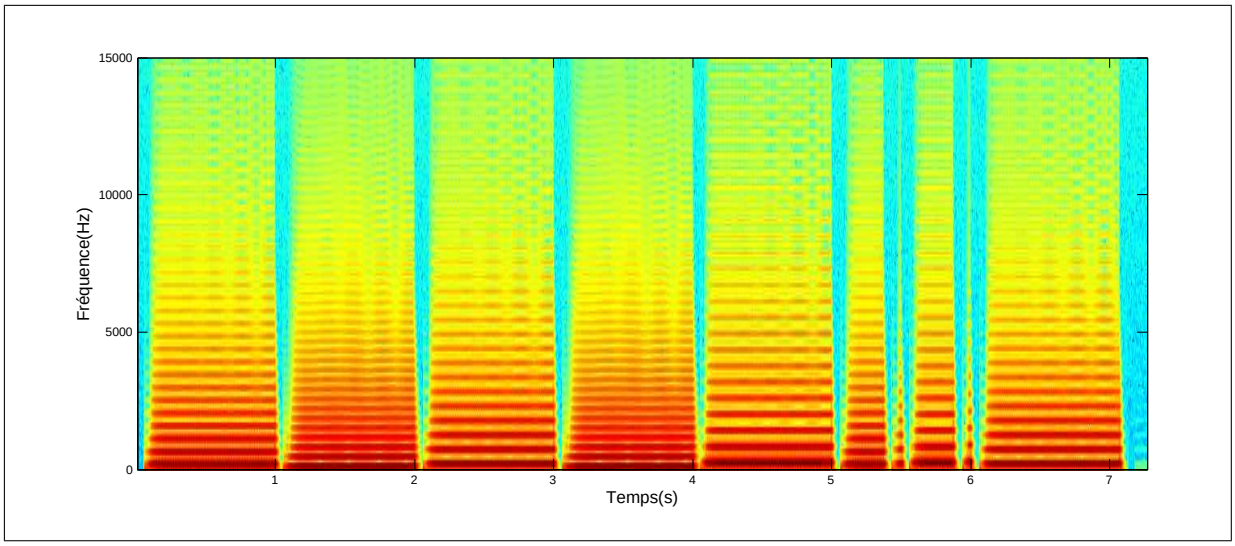

FIGURE 2.21 – Spectrogramme du son `voiChe-clarinette.wav` (disponible sur le CD dans le dossier `/clarinette/`) généré avec le programme MATLAB.

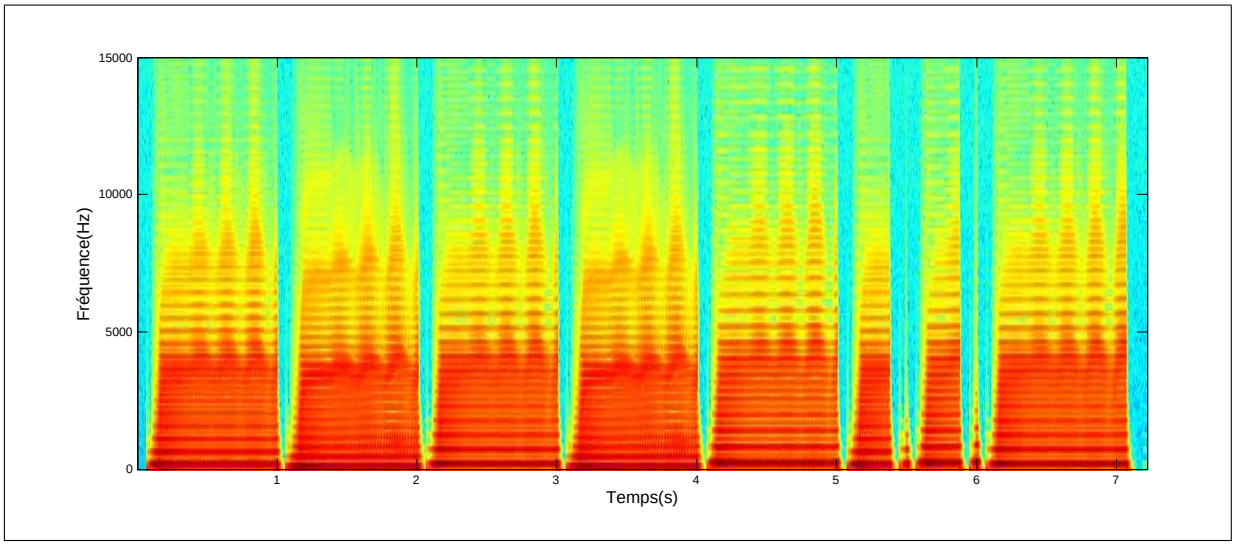

FIGURE 2.22 – Spectrogramme du son `calAllpassnnVoiCherSigMod0_5.wav` (disponible sur le CD dans le dossier `/allpassnn/utilisation/clarAllpassnn`) généré avec le programme MATLAB.

effectués qu'avec un nombre très réduit de types de modulations pour les coefficients du filtre. Un important travail de test et de découverte de nouveaux sons reste donc à être effectué ici et pourrait faire l'objet de travaux futurs.

Les deux exemples d'application donnés précédemment, bien que très significatifs, possèdent toutefois un certain nombre de limites. En effet, il s'agit de modèles d'instruments basés sur un seul guide d'ondes.

Chapitre 3

Implémentation de modèles physiques d'instruments de musique

Le détail de l'implémentation d'un ensemble de modèles physiques par guides d'ondes d'instruments de musique et les techniques qui leur sont associées sont présentés dans ce chapitre. Ces modèles sont issus de diverses sources tels que le SYNTHESIS TOOLKIT et le programme SYNTHBUILDER. Quelques uns d'entre-eux sont inédits. Ils ont tous été regroupés dans une bibliothèque d'objets qui fait maintenant partie intégrante de la distribution de FAUST : le FAUST-STK.

Bien qu'une description assez détaillée de chaque algorithme soit faite, il n'est pas question de donner ici trop de détails sur les différents modèles implémentés puisque cela a déjà fait dans les articles qui leurs sont associés [141].

3.1 Gérer les modèles d'instruments polyphoniques dans FAUST

Certains des instruments présentés dans cette partie, à la différence de ceux dont il a été question précédemment, sont des instruments polyphoniques. Cette caractéristique peut dans une certaine mesure poser des problèmes d'implémentation et d'optimisation dans FAUST. En effet, bien qu'il soit tout à fait possible de mettre au point un système qui permettrait de rendre un instrument polyphonique, il est nécessaire de s'interroger sur la position de FAUST par rapport à d'autres langages de programmation

141. Les références de ces articles seront données au cours de ce chapitre au cas pas cas.

pour la synthèse des sons et sur ses objectifs.

Si l'on se place dans le contexte de MAX/MSP, PUREDATA, CSOUND, etc., les plug-ins générés par FAUST pour ces différents programmes n'ont pas pour objectif de remplacer l'ensemble des éléments qui constitue un patch « `.pd` » ou un orchestre « `.sco` » mais plutôt d'apporter une alternative à C++ pour l'implémentation d'objets de bas niveau pour le traitement du signal. Ainsi, dans la mesure où un grand nombre de ces programmes propose des objets permettant de rendre un processus de synthèse polyphonique, il paraît plus judicieux d'utiliser ces outils que de mettre en place un système dans FAUST qui effectuerait le même type de travail. Les modèles physiques présentés par la suite sont donc tous monophoniques et sont rendus polyphoniques, si nécessaire, pour PUREDATA, grâce à *faust2pd* qui permet de générer un patch polyphonique d'un synthétiseur implémenté dans FAUST.

Il n'est pas question ici de détailler le fonctionnement de *faust2pd* puisque que cela est déjà fait dans la documentation de ce programme [142]. Néanmoins, il est nécessaire de comprendre les patchs générés pour pouvoir les utiliser correctement.

Si lors de l'utilisation de *faust2pd* l'option `-n` est spécifiée, le patch généré sera un synthétiseur polyphonique avec une interface MIDI. Le nombre de voix de cette polyphonie est déterminé par le chiffre qui suit l'option `-n` et est limité à huit. Le patch PUREDATA produit lors de cette opération contiendra autant d'instances du plug-in généré par FAUST que de voix de polyphonie nécessaires (cf. figure 3.1). Le patch `midi-in.pd` [143], fourni avec *faust2pd* se charge en suite d'envoyer le signal MIDI reçu alternativement à chaque instance du plug-in comme c'est la cas dans l'exemple présenté sur la figure 1.7 à la page 20.

3.2 Instruments à cordes pincées et frappées

Dans la partie 1.2 traitant de l'algorithme de KARPLUS-STRONG, il a été montré que les premières implémentations de modèles physiques d'instruments de musique utilisant la technique des guides d'ondes numériques étaient des instruments à cordes pincées. Le cas de la guitare ayant servi d'illustration (cf. annexes B et C) aux propos théoriques tenus dans 1.1, ce dernier n'est pas traité à nouveau dans cette partie.

142. `http://pure-lang.googlecode.com/svn/docs/faust2pd.html`.
143. Fichier contenu dans le dossier `/util/` du CD.

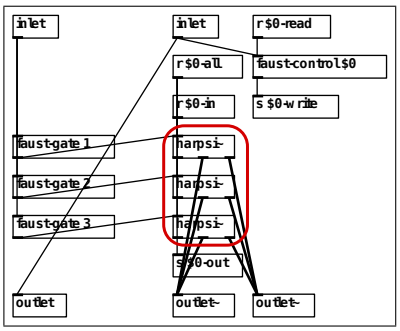

FIGURE 3.1 – Zone d'un patch généré par *faust2pd* dans laquelle est déclaré les différentes instances d'un plug-in de synthétiseur afin de le rendre polyphonique. Il s'agit ici d'une polyphonie à trois voix, les trois instance de `harpsi` se trouvent dans l'encadré rouge.

3.2.1 Clavecin

Présentation du modèle

Le modèle de clavecin présent dans le FAUST-STK (`harpsi.dsp`[144]) dont l'implémentation est détaillée dans l'annexe H est basé sur le modèle de cet instrument dans le programme SYNTHBUILDER bien qu'un certain nombre de modifications lui aient été apportées.

Son fonctionnement diffère légèrement des autres modèles par guides d'ondes d'instruments à cordes présentés précédemment dans la partie 1.2. En effet, il utilise la technique des guides d'ondes commutés qui permet de faciliter grandement l'implémentation de modèles complexes de ce type d'instrument.

La technique des guides d'ondes commutés a été élaborée au début des années quatre-vingt-dix [145] [146]. Elle est basée sur le fait, que si les éléments qui produisent le son d'un instrument à corde sont presque linéaires et ne varient pas (ou très peu) au cours du temps, il est possible d'intervertir la place des cordes et du résonateur dans l'algorithme du modèle physique (cf. figure 3.2). Il devient alors possible de convoluer l'excitation avec la réponse impulsionnelle du résonateur. Le résultat de cette opération peut en suite être enregistré dans une table de fonction pouvant être appelé à chaque fois que cela est nécessaire, réduisant par conséquent de manière significative la quantité de calculs demandés lors de la synthèse du son.

144. Disponible sur le CD dans le dossier `/harpsi/`.
145. KARJALAINEN, Matti ; LAINE, Unto ; LAAKSO, Timo ; VALIMAKI, Vesa, *Transmission-Line Modeling and Real-Time Synthesis of String and Wind Instruments : actes de International Computer Music Conference, Montreal, 1991*, Computer Music Association, 1991, p. 293-296.
146. SMITH, Julius, *Efficient Synthesis of Stringed Musical Instruments : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993.

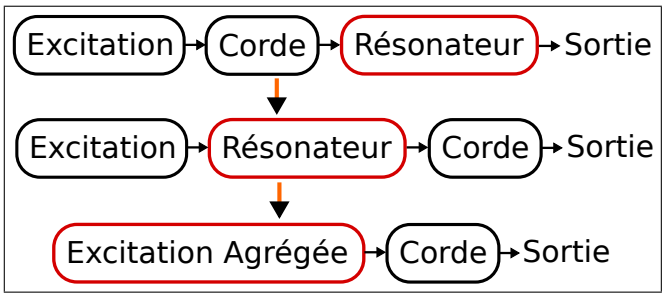

FIGURE 3.2 – Passage d'un modèle physique par guides d'ondes conventionnel à un modèle par guides d'ondes commutés. L'*excitation agrégée* correspond à la fusion des boîtes *Excitation* et *Résonateur*.

Le modèle de clavecin présenté ici reste très simple et n'utilise pas de réponse impulsionnelle de sa table d'harmonie à convoluer avec l'excitation. Ainsi, *l'excitation agrégée* n'est produite qu'avec un simple générateur de bruit blanc sur lequel une enveloppe d'amplitude est appliquée (cf. annexe H). La durée de la chute de cette enveloppe varie en fonction de la fréquence de la note jouée. Les valeurs de temps correspondants à une fréquence sont extraites d'un tableau contenu dans une fonction C++ appelée à l'aide du système de fonction étrangère de FAUST [147]. Des détails sur la technique employée sont donnés dans la section suivante.

Les cordes sont modélisées avec un guide d'ondes unidimensionnel utilisant un filtre de réflexion biquadratique. Le filtre passe-tout non-linéaire du sixième ordre présenté dans 2.2 est utilisé pour introduire des nonlinéarités dans le son produit. Le temps de résonance de la corde varie en fonction de la fréquence de la note jouée. Ainsi, un tableau placé dans une fonction C++ dans lequel des coefficients de dispersions sont associés à certaines gammes de fréquences est consulté (voir section suivante).

Un exemple audio dans lequel *la Marche turque* de Wolfgang Amadeus MOZART joué par ce clavecin peut être entendu dans le fichier `turc-clavecin.wav` [148]. Il a été généré par le patch PUREDATA `marche-turc-clavecin.pd` [149].

Import de tableaux de données externes dans un objet FAUST

L'implémentation de modèles physiques d'instruments de musique peut parfois impliquer l'utilisation de banques de données contenant différents types d'informations

147. ORLAREY, Yann ; FOBER, Dominique ; LETZ, Stéphane, *FAUST Quick Reference*, Lyon : GRAME, 2012, p. 27-29.
148. Fichier disponible sur le CD dans le dossier `/harpsi/`.
149. *Ibid.*

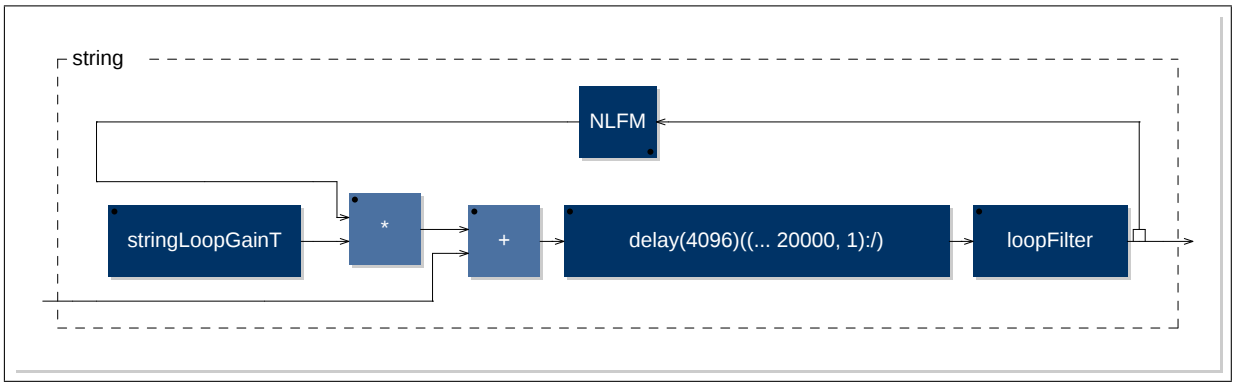

FIGURE 3.3 – Diagramme de la corde du clavecin contenu dans le fichier `harpsi.dsp` (disponible sur le CD dans le dossier `/harpsi/`) généré avec l'outil `faust -svg`. Plus d'informations sur la fonction de chaque boîte peuvent être trouvées dans l'annexe H.

(coefficients de dispersions dans le cas précédent, coefficients de filtres, etc.). Or, le langage FAUST ne possède aucune fonction permettant d'incorporer d'importantes quantités de données au sein d'un objet.

La solution trouvée pour pallier ce problème passe par l'utilisation du système de fonction étrangère de FAUST. En effet, il est possible d'appeler une fonction C++ dans un objet FAUST avec une formulation du type :

```
ffunction(float getValueLoopFilter(float), <harpsichord.h>,"");
```

où le premier `float` correspond au type de données retournées par la fonction, `getValueLoopFilter` est le nom de la fonction C++, le second `float` est le type de données prises en argument par la fonction et `<harpsichord.h>` est le nom de la fonction C++.

La fonction C++ appelée lors de l'opération précédente est de la forme :

```
float getValueLoopFilter(float index){
  return loopFilter.getValue(index);
}
```

où `loopFilter` est la tableau contenant les informations à récupérer et `index`, l'index permettant de consulter ce tableau. Dans le cas présent, il s'agira d'une fréquence.

Cette technique peut également permettre d'importer de courts échantillons de sons dans un objet FAUST pouvant être chargé dans une table de fonction par exemple.

3.2.2 Piano

Le piano implémenté dans le FAUST-STK est basé sur le modèle de cet instrument du programme SYNTHBUILDER. Dans la mesure où il s'agit d'une reproduction

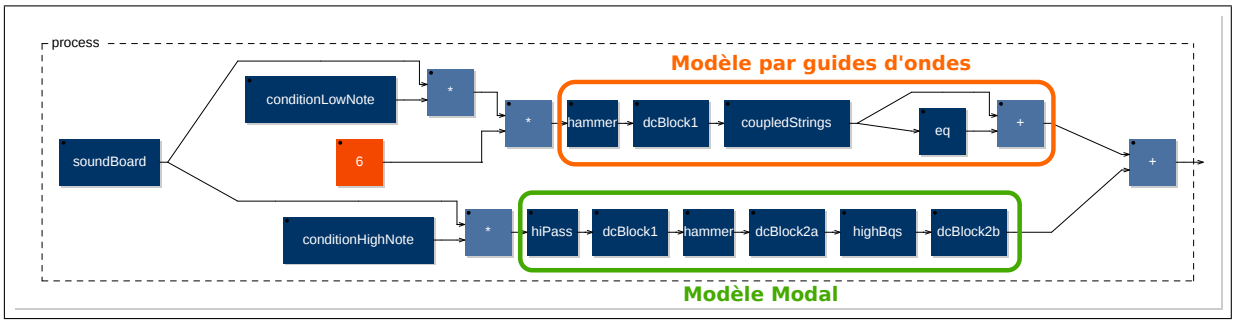

FIGURE 3.4 – Diagramme de haut niveau du modèle de piano du FAUST-STK généré avec l'outil `faust -svg`. Les fonctions de chacune des « boîtes » du diagramme sont présentées dans l'annexe I.

à l'identique de la version originale de l'algorithme utilisé, seule une description très brève de cet instrument est ici faite. De plus, il est important de préciser que c'est de très loin le modèle physique par guides d'ondes le plus complexe implémenté dans le FAUST-STK. Le décrire en détail prendrait trop de temps et cela a déjà fait l'objet de plusieurs publications [150] [151].

Le modèle de piano dont il est ici question est disponible dans le fichier `piano.dsp` [152]. Il utilise deux techniques de synthèse différentes en fonction de la hauteur des notes à jouer. En effet, les notes en dessous de Mi6 sont synthétisées avec un algorithme avancé de synthèse par guides d'ondes commutés du même type que celui utilisé pour le clavecin présenté dans la partie précédente. Dans l'autre cas, les notes au dessus de Mi6 sont générées par un algorithme simple de synthèse modale (cf. figure 3.4).

Cette différenciation est faite pour deux raisons. La première est relative à la justesse des notes produites qui peut être altérée dans le cas de sons très aigus à cause du phénomène de quantification de la longueur de la ligne de retards du guide d'ondes si la fréquence d'échantillonnage utilisée n'est pas assez élevée (cf. partie 1.2 sur l'algorithme de KARPLUS-STRONG). Cette différenciation est également faite dans un but d'économie de calculs. En effet, il est montré par la suite que le modèle de corde utilisé est assez complexe et « gourmand » en opérations à la différence de l'algorithme de synthèse modale qui n'implique l'utilisation que de quatre filtres du premier ordre.

150. SMITH, Julius ; DUYNE (VAN), Scott, *Commuted Piano Synthesis : actes de International Computer Music Conference, Banff (Canada), 1995*, Banff : Computer Music Association, 1995, p. 319-326.
151. BANK, Balázs ; ZAMBON, Stefano ; FONTANA, Frederico, « A Modal-Based Real-Time Piano Synthesizer », *Transactions on Audio, Speech, and Language Processing*, XVIII (2010), n° 4, p. 809-821.
152. Fichier disponible sur le CD dans le dossier `/piano/` et décrit dans l'annexe I.

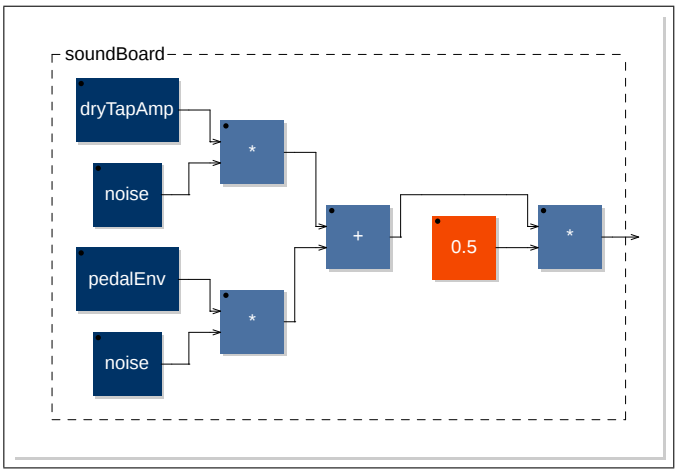

FIGURE 3.5 – Diagramme (produit avec l'outil `faust -svg`) détaillant le contenu de la fonction `soundBoard` qui implémente l'algorithme utilisé pour générer l'excitation appliquée sur les cordes du piano.

Excitation agrégée

L'excitation est produite par les signaux additionnés de deux générateurs de bruit blanc sur lesquels une enveloppe exponentielle complexe, distincte pour chacun d'entre-eux, est appliquée (cf. figure 3.5). Le premier générateur de bruit modélise l'excitation générée par l'impact du marteau sur la corde ainsi que la résonance qui s'en suit. L'autre générateur permet de continuer à introduire de l'énergie dans le résonateur lorsque la pédale de maintien est enfoncée.

Dans le cas présent, il est légitime de se questionner sur la nécessité de continuer à introduire de l'énergie dans la partie résonnante du modèle physique alors que celui-ci est dans une étape de résonance menant progressivement à sa stabilisation. Ceci est dû au fait qu'il s'agisse ici de synthèse par guides d'ondes commutés. En effet, comme la réponse impulsionnelle de la caisse de résonance du piano est appliquée au niveau de l'excitation et non à la fin de l'algorithme comme c'est le cas dans la réalité, il est nécessaire de continuer à exciter le modèle avec l'excitation agrégée pour que le son produit contienne les propriétés spectrales de la caisse de résonance du piano (cf. partie 3.2.1).

L'étape de convolution mentionnée précédemment est menée à bien par une série de quatre filtres biquadratiques connectés en série et s'effectue au niveau des boites *hammer* de la figure 3.4. Cette opération est expliquée de manière plus précise dans l'annexe I.

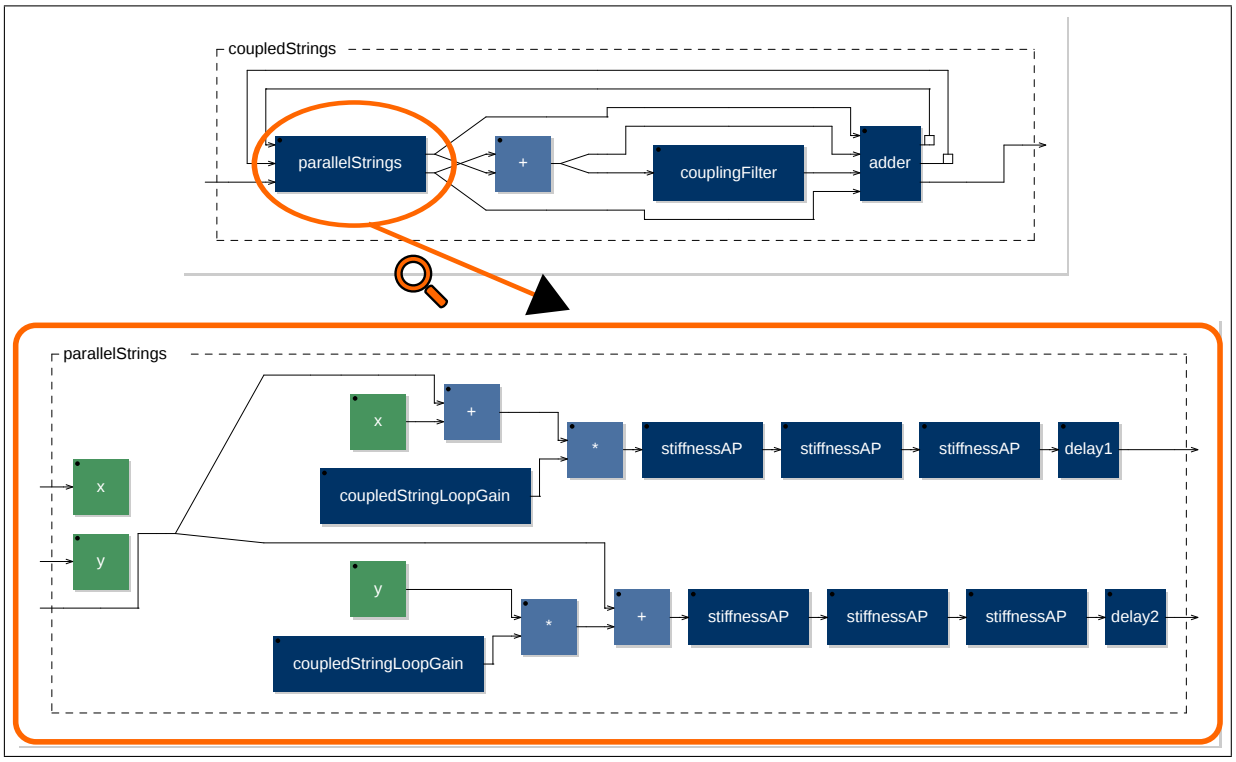

FIGURE 3.6 – Digramme (produit avec l'outil `faust -svg`) de l'algorithme utilisé pour modéliser les cordes couplées du piano. Le diagramme dans la partie inférieure de la figure permet de visualiser le contenu de la boite *parallelStrings*.

Cordes couplées

Les pianos mettent en œuvre plusieurs cordes pour produire une seule note. Bien que certains modèles assez rares utilisent quatre cordes, leur nombre est généralement de trois pour une même hauteur, on parle donc de « cordes couplées ». La présence de ces trois cordes permet de créer un effet de déphasage dans le son produit.

Ces différents facteurs ont été pris en compte pour la mise en place du modèle de cordes de `piano.dsp`. En effet, le modèle par guides d'ondes utilisé pour les notes en dessous de Mi6 (cf. figure 3.6) implémente deux cordes couplées et simule leurs différentes interactions. Il est intéressant de noter le nombre important de filtres utilisés dans le cas présent par rapport aux exemples d'instruments donnés précédemment. Plus de détails sur cet algorithme sont donnés par son inventeur, Gabriel WEINREICH [153].

153. WEINREICH, Gabriel, « Coupled Piano Strings », *Journal of the Acoustical Society of America*, LXII (1977), p. 1474-1484.

Notes aiguës

Comme cela a été brièvement expliqué précédemment, les notes aiguës du piano au dessus de Mi6 sont synthétisées avec un algorithme simple de synthèse modale. [154] Une série de quatre filtres biquadratiques connectés en série et dont les coefficients varient en fonction de la fréquence de la note jouée est utilisée. Ils sont mis en résonance avec la même excitation agrégée que celle utilisée pour la synthèse par guides d'ondes. La seule différence vient de l'utilisation d'un filtre passe-haut juste avant l'application de la réponse impulsionnelle de la caisse de résonance du piano sur l'excitation. Ce dernier permet d'augmenter l'effet de la forte tension appliquée sur les cordes pour les notes hautes du piano.

Le modèle présenté précédemment permet de modéliser un seul couple de cordes de piano. Afin de le rendre polyphonique, il est donc nécessaire d'utiliser la technique présentée dans la partie 3.2.1. Ce modèle étant assez coûteux en calculs à cause du nombre important de filtres biquadratiques qu'il utilise, il est nécessaire de posséder une machine assez puissante pour pouvoir l'utiliser en temps réel dans le cas d'une polyphonie de plus de six notes.

Enfin, il est important de souligner le fait qu'un certain nombre de facteurs n'est pas pris en compte dans ce modèle. En effet, bien que les différentes interactions possibles entre les cordes couplées soient modélisées, il serait intéressant de prendre également en compte les interactions ayant lieu entre l'ensemble des cordes du piano qui peuvent parfois être aussi importantes.

Un exemple sonore polyphonique de cet instrument dans lequel *la Marche Turque* de Wolfang Amadeus MOZART est jouée peut être entendu dans le fichier `turc-piano.wav` [155].

3.2.3 Guitare basse

Un modèle de guitare basse est implémenté dans le FAUST-STK. Il est basé sur le modèle de cet instrument implémenté dans le programme SYNTHBUILDER et utilise un simple algorithme de KARPLUS-STRONG (cf. partie 1.2), amélioré avec un certain

154. ADRIEN, Jean-Marie, « The Missing Link : Modal Synthesis », *Representations of Musical Signals*, éd. sous la direction de Curtis Roads, Cambridge : MIT Press, 1991, p. 269-297.
155. Disponible sur le CD dans le dossier `/piano/`.

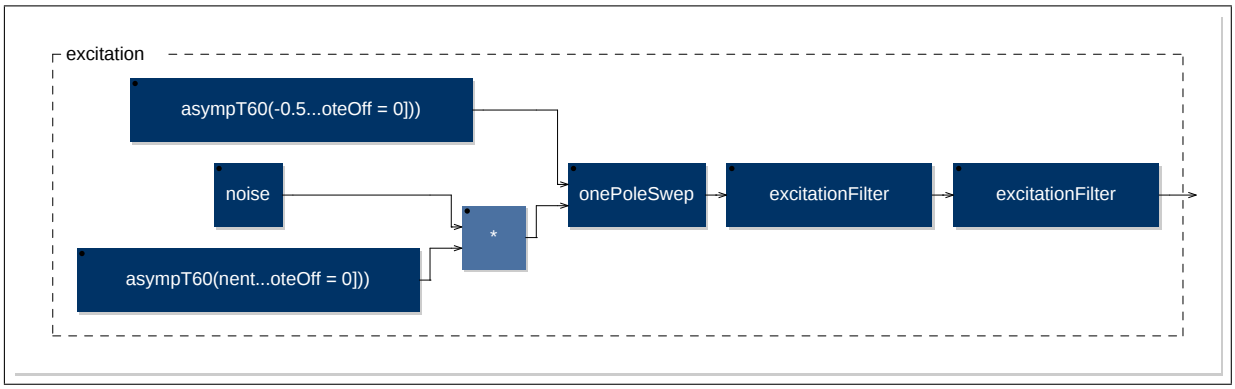

FIGURE 3.7 – Diagramme (produit avec l'outil `faust -svg`) de l'algorithme utilisé pour générer l'excitation des cordes de la guitare basse. La fonction de chaque « boîte » est donnée dans l'annexe J.

nombre de filtres. Il est disponible dans le fichier `bass.dsp` [156] dont le fonctionnement est expliqué dans l'annexe J.

L'excitation est modélisée avec un générateur de bruit blanc sur lequel une enveloppe d'amplitude exponentielle est appliquée. Le son produit lors de cette opération est alors filtré avec une série de trois filtres simples du premier ordre (cf. figure 3.7). Le pôle de l'un d'entre-eux « glisse » de manière exponentielle au cours de l'excitation. Plus de détails sont donnés sur cette opération dans l'annexe J.

Le son produit par une guitare basse contient légèrement plus de non-linéarités que celui d'une simple guitare acoustique. Cela est dû au fait que les cordes utilisées sont plus lourdes et plus épaisses. Elles appliquent donc une force plus importante au niveau du chevalet qui, même s'il est très rigide, réfléchira les ondes qui se propagent dans les cordes avec plus de non-linéarités que dans le cas d'une guitare acoustique.

Dans le cas du modèle ici présenté, les non-linéarités sont modélisées à l'aide du filtre passe-tout passif non-linéaire décrit dans la partie 2.2 qui est placé dans le guide d'ondes modélisant les cordes de la basse (cf. figure 3.8).

Comme cela a été montré précédemment dans le cas du sitar par exemple (cf. partie 2.1.2), ce filtre introduit de très fortes non-linéarités, il est donc important d'utiliser dans le cas présent un coefficient de non-linéarités très faible afin que le son reste le plus naturel possible.

Enfin, un simple filtre passe-bande est utilisé pour modéliser la réponse impulsionnelle du corps de la guitare basse. Ce dernier est placé à la sortie du guide d'ondes produisant le son de la corde.

156. Fichier disponible sur le CD dans le dossier `/basse/`.

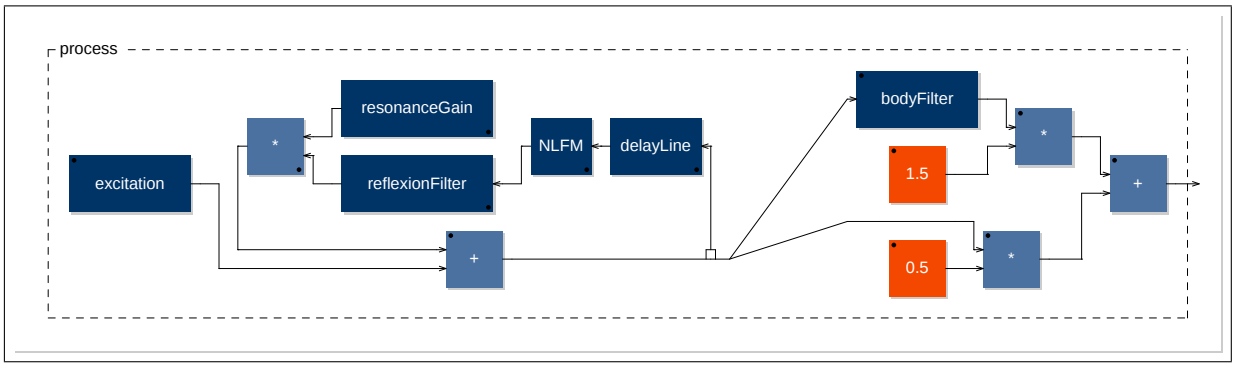

FIGURE 3.8 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `bass.dsp`. La fonction de chaque « boîte » est donnée dans l'annexe J.

Le fichier audio `funk-bass.wav` [157] contient un exemple d'utilisation de cet instrument généré avec la patch PUREDATA `funk-bass.pd` [158] qui joue le fichier MIDI `funk.mid` [159].

3.3 Instruments à cordes frottées

3.3.1 Modèle physique de base d'instruments à cordes frottées

Le fichier `bowed.dsp` [160] implémente un modèle simple d'instrument à cordes frottées avec la technique des guides d'ondes numériques. Il peut être utilisé pour imiter le son d'un grand nombre d'instruments appartenant à cette famille tels que le violon, l'alto, le violoncelle, etc.

Il est basé sur le modèle de violon disponible dans le SYNTHESIS TOOLKIT implémenté par Julius SMITH [161] et n'a subit que très peu de modifications. Comme la plupart des instruments du FAUST-STK, il utilise le filtre passe-tout passif non-linéaire présenté dans la partie 2.2.

A la différence des autres modèles physiques présentés dans ce mémoire, les instruments à cordes frottées nécessitent l'utilisation d'au moins deux guides d'ondes pour modéliser une corde en vibration. En effet, pour que le son soit produit, l'archet doit être

157. Fichier disponible sur le CD dans le dossier `/basse/`.
158. *Ibid.*
159. Fichier disponible sur le CD dossier `/util/`. Plus d'informations sur ce fichier sont données dans l'annexe A.
160. Disponible sur le CD dans le dossier `/violon/` et décrit dans l'annexe N.
161. SMITH, Julius, *Efficient Simulation of the Reed-bore and Bow-string Mechanisms : actes de International Computer Music Conference, La Haye, 1986*, La Haye : Computer Music Association, 1986, p. 275-280.

frotté de manière continue sur la corde ce qui a pour effet de la partager en deux parties distinctes. Dans la mesure où l'archet est alors considéré comme une terminaison de la corde, il devient nécessaire d'utiliser deux guides d'ondes distincts pour modéliser la partie supérieure et la partie inférieure de la corde. Ces deux zones sont alors reliées à l'aide de jonctions de dispersion [162] entre lesquelles est introduit le signal d'excitation généré à l'aide d'une fonction non-linéaire. Le même principe est utilisé avec le modèle d'instrument à anche simple du FAUST-STK (cf. partie 3.4.2). Tout comme pour ce dernier, il est donc possible de contrôler la position du point d'excitation en augmentant et en réduisant de manière proportionelle la longueur des deux lignes de retards utilisées dans les guides d'ondes du modèle. Plus d'informations sur ce sujet peuvent être trouvées dans la partie 3.4.3 portant sur l'implémentation de modèles d'instruments génériques à anche simple.

La caisse de résonance joue un rôle très important dans la détermination du timbre du son produit chez les instruments à cordes :

> « Il est probable que le facteur le plus important dans la détermination de la qualité et de la malléabilité du son des instruments à cordes soit le comportement vibratoire de leur caisse de résonance. » [163]

Il est donc nécessaire de reproduire son effet au niveau du modèle. Dans le cas de `bowed.dsp`, cela est fait de manière très basique en utilisant simplement un filtre passe-bande du premier ordre dont la fréquence de résonance est réglée à cinq cent hertz (`bodyFilter` sur la figure 3.9). Ce filtre est alors placé à la sortie du guide d'ondes juste avant que le son soit émis. Dans le modèle présenté dans la partie suivante, il est remplacé par une convolution de la réponse impulsionnelle de la caisse de résonance d'un violon ce qui permet d'améliorer de manière significative le réalisme du son produit.

Un simple filtre passe-bas du premier ordre est utilisé comme filtre de réflexion. Dans le cas présent, il sert à modéliser l'effet combiné du chevalet et des doigts sur les ondes qui se déplacent dans la corde. Il est représenté par la boite `stringFilter` sur la figure 3.9.

Tout comme dans les cas de `piano.dsp` [164] et `harpsi.dsp` [165] qui sont eux

162. VALIMAKI, Vesa ; KARJALAINEN, Matti ; LAAKSO, Timo, *Modeling of Woodwind Bores with Finger Holes : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993, p. 32-29.

163. FLETCHER, Neville ; ROSSING, Thomas, *op. cit*, p. 285, citation traduite de l'anglais : « Probably the most important determinant of the sound quality and playability of a string instrument is the vibrational behavior of its body. ».

164. Fichier disponible sur le CD dans le dossier `/piano/`.

165. Fichier disponible sur le CD dans le dossier `/harpsi/`.

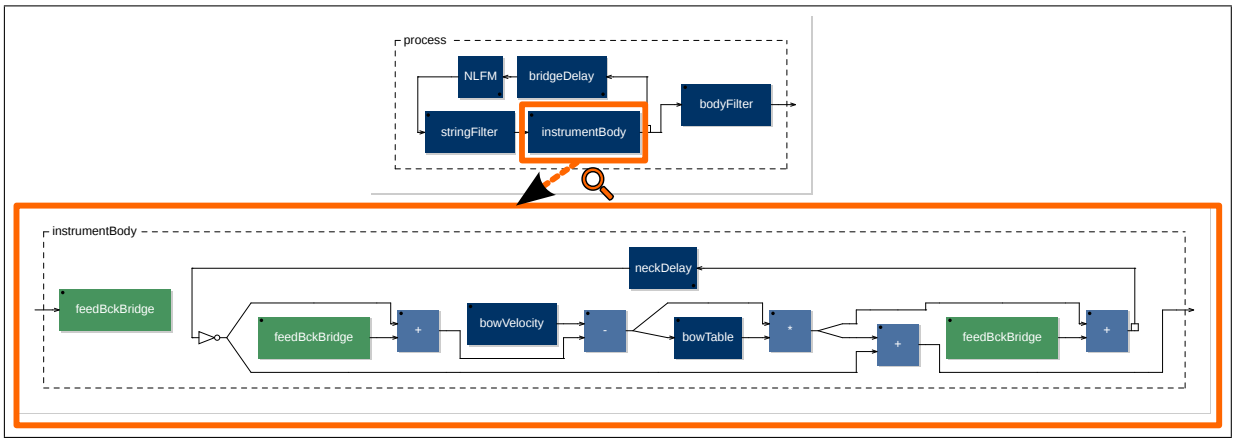

FIGURE 3.9 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `bowed.dsp`. Le contenu de la boîte `instrumentBody` est détaillé dans la partie inférieure de la figure. Plus de détails sur la fonction de chaque « boîte » sont donnés dans l'annexe N.

aussi des instruments polyphoniques, seule une corde d'instrument à cordes frottées est implémentée dans `bowed.dsp`. Il est toutefois possible de jouer sur plusieurs cordes en utilisant la technique décrite dans la partie 3.2.1.

Le fichier `voi-che-bowed.wav` [166] contient un exemple utilisant ce modèle. Il a été produit avec le patch PUREDATA `voi-che-bowed.pd` [167]. Il est évident à l'écoute de cet exemple que le résultat obtenu n'est que très peu convaincant. On reconnaît parfois des grincements sur l'attaque de certaines notes propres au violon, mais lorsque des notes longues sont jouées, il est très difficile d'assimiler le son produit à celui d'un violon. Plusieurs raisons permettent d'expliquer ces mauvais résultats.

Tout d'abord, l'utilisation d'un archet rend la reproduction du son du violon très complexe dans la mesure où celle-ci engendre une augmentation considérable du nombre de paramètres physiques nécessaires pour contrôler le modèle : vélocité de l'archet, pression exercée par ce dernier sur les cordes, etc. L'absence de frettes sur le manche de l'instrument rend aussi plus subtil l'effet du contact des doigts sur la corde qui prennent alors le rôle de terminaisons.

Dans le cas d'un modèle de clarinette, une bonne configuration avec des valeurs constantes pour la pression de l'air au niveau du bec et de la position de la bouche de l'instrumentiste permet d'obtenir simplement des résultats convaincants. Acquérir un meilleur rendu avec le modèle de violon présenté ici passe donc par un contrôle accru des différents paramètres physiques déterminant les propriétés du son produit. Une solution

166. Fichier disponible sur le CD dans le dossier `/violon/`.
167. *Ibid.*

à ce problème est donnée dans la partie 3.3.3.

Enfin, il est fort probable que notre perception du violon soit plus exigente de celle que nous avons de beaucoup d'autres instruments de musique. En effet, au travers des musiques de films, des publicités, de la radio, etc., notre oreille est plus entraînée à entendre des sons de violons que des sons de hautbois par exemple nous rendant donc plus sensible à une mauvaise qualité du son pour cet instrument.

Le fichier `muneira-bowed.wav`[168] contient un autre exemple d'utilisation de `bowed.dsp`. Il a été produit avec le patch PureData `muneira-bowed.pd`[169]. Il est interessant de le comparer avec l'exemple audio présenté dans la partie suivante dans laquelle un modèle amélioré de cet instrument est présenté.

3.3.2 Modèle physique avancé de violon

Contexte

Dans la cadre de mon séjour au CCRMA[170] de l'université Stanford de Janvier à Mai 2011, j'ai eu la chance de travailler au côté d'Esteban MAESTRE[171] qui effectuait durant cette même période un post-doctorat dans l'équipe de traitement numérique du signal dirigée par Julius SMITH et Jonathan ABEL. Esteban travaillait alors sur la modélisation de suivi de données de gestes d'instrumentiste pour différents types d'instruments dont le violon. A nos « heures perdues », nous avons travaillé ensemble sur un modèle physique expérimental de violon basé sur celui présenté dans la partie précédente que nous avons rendu compatible pour qu'il puisse utiliser en temps réel des données de suivi de gestes dans le programme PureData. Il était donc question dans cette approche de considérer le modèle physique comme un véritable instrument manipulé par un musicien, le but étant de reproduire de la manière la plus fidèle possible le résultat obtenu par l'instrumentiste sur son violon et cela pour un ensemble de pièces.

Esteban s'est occupé du collectage de données de suivi de gestes et du traitement de la réponse impulsionnelle du violon qui a servi de modèle. De mon côté, j'ai travaillé sur son implémentation dans le langage Faust et dans PureData.

Ce modèle est présenté dans cette partie et utilisé dans la suivante.

168. *Ibid.*
169. *Ibid.*
170. Plus d'informations sur cette institution sont données dans le glossaire à la page 120.
171. `https://ccrma.stanford.edu/~esteban/`.

Améliorations apportées à `bowed.dsp`

Les deux principales modifications apportées à l'instrument `bowed.dsp` concernent le filtre de réflexion et le filtre modélisant la caisse de résonance du violon. En effet, la même fonction d'excitation est utilisée et la corde est toujours modélisée avec deux guides d'ondes unidimensionnels connectés à l'aide de jonctions de dispersions à deux ports.

Deux étudiants d'Esteban MAESTRE au MTG [172] de l'université Pompeu Fabra de Barcelone : Panos PAPIOTIS et Marco MARCHINI ont enregistré dans le cadre de nos travaux des réponses impulsionnelles de caisses de résonance et de chevalets de violons à l'aide de capteurs piézoélectriques. Elles ont été utilisées pour remplacer le filtre de réflexions et le filtre modélisant la caisse de résonance du violon.

Pour pouvoir être utilisée dans un instrument FAUST en temps réel, l'étape de convolution nécessaire pour appliquer une réponse impulsionnelle sur un son donné a dû être contournée. La solution choisie a alors été de remplacer cette dernière par une série de filtres biquadratiques dont les coefficients ont été calculés à partir de la réponse impulsionnelle souhaitée en utilisant la technique de *Bark Frequency Warping* [173]. Un programme MATLAB (contenu dans le fichier `IRfilterdesign.m` [174]) prenant en argument un fichier audio contenant la réponse impulsionnelle et l'ordre du filtre modélisé par l'ensemble de filtres biquadratiques a été implémenté pour mener à bien cette opération.

Plusieurs réponses impulsionnelles contenues sur le CD dans le dossier `/muneira/matlab/` dans les fichiers dont le nom commence par « `imp_r_11` » ont été testées. Celle finalement retenue pour être utilisée dans le modèle de violon peut être trouvée dans le fichier `admitanceIR_cut2.wav`. Un filtre d'ordre seize est donc simulé dans FAUST à l'aide de six filtres biquadratiques pour la caisse de résonance du violon et de cinq pour le traitement du signal de réflexion.

Les coefficients des filtres calculés par `IRfilterdesign.m` ont été stockés dans le fichier C++ `instrument.h` [175] dans des fonctions appelées dans l'objet FAUST à l'aide du système de *fonction étrangère* (cf. partie 3.2.1). Comme la même action de déclaration est répétée à plusieurs reprises, l'outil de filtrage par motif de FAUST a été utilisé. Ainsi, la déclaration de la fonction modélisant la caisse de résonance du violon

172. Music Technology Group.
173. STRUBE, Hanz, « Linear Prediction on Warped Freqeuncy Scale », *Journal of the Acoustical Society of America*, LXVIII (1980), n° 4, p. 1071-1076.
174. Disponible sur le CD dans le dossier `/muneira/matlab/`.
175. Disponible sur le CD dans le dossier `/muneira/`.

est de la forme (avec `tf2`, un filtre biquadratique et `fC`, la fonction C++ contenant les coefficients des filtres) :

```
bodyFilter = seq(i,6,tf2(fC(i,0),fC(i,1),fC(i,2),fC(i,3),fC(i,4)))
   : *(0.0637)
with{
  fC = ffunction(float violinImpRes2(int,int), <instrument.h>,"");
};
```

La même technique est employée pour la série de filtres utilisée pour le traitement du signal de réflexion. La seule différence vient de l'utilisation de coefficients différents pour chaque corde du violon. En effet, il est montré dans la sous-partie suivante sur l'utilisation de données de suivie de gestes que quatre cordes sont déclarées pour ce modèle, comme dans le cas d'un vrai violon.

Travaux futurs

Un nombre important d'améliorations peut encore être apporté au modèle présenté précédemment. Il serait par exemple très intéressant de pouvoir modéliser de manière précise les effets des doigts de l'instrumentiste sur les ondes qui se propagent dans la corde.

En effet, comme cela a été mentionné précédemment, lorsqu'un doigt est placé sur la corde pour déterminer la hauteur de la note jouée, ce dernier fait office de terminaison et remplace donc le frette placé au bout du manche de l'instrument. Or, la « rigidité » de la peau en contact avec la corde est bien moindre que celle du frette ce qui a pour effet de filtrer les ondes qui se déplacent dans la corde. Une série de filtres biquadratiques complexes du même type que celle présentée précédemment pourrait permettre de mener à bien cette opération.

Un travail conséquent au niveau de la fonction modélisant l'excitation de la corde avec un archet reste également encore à être fait. Une table de fonction basée sur des courbes exponentielles évoluant donc de manière plus progressive pourrait être mise en place. Il serait aussi important de prendre en compte l'effet d'échauffement produit par le frottement de l'archet sur la corde qui engendre de manière plus ou moins significative un allongement de la longueur de la corde et donc une légère baisse de la fréquence du son généré.

3.3.3 Utilisation de données de suivi de gestes

Adaptations de compatibilité du modèle physique

Les modèles physiques du FAUST-STK présentés dans ce mémoire ont été conçus pour être contrôlés facilement avec des informations MIDIs. Ainsi, les trois principaux paramètres utilisés pour les faire fonctionner sont la fréquence de la note jouée, son amplitude et les messages `note-on/note-off`. Ces paramètres de très haut niveau qui ont une signification cohérente d'un point de vue musical n'ont rien à voir avec les paramètres purement physiques utilisés par les modèles du FAUST-STK. Dans le cas du violon dont il est question dans cette partie, les données MIDI concernant l'amplitude et les messages `note-on/note-off` envoyés au modèle doivent être converties en vélocité de l'archet lors de sa friction avec une corde et en force appliquée par l'archet sur la corde au niveau du point de frottement. Comme cela l'a été démontré précédemment, cette opération est menée à bien dans la plupart des cas en utilisant une enveloppe de type *ADSR*[176] qui a donc pour but de modéliser de manière très grossière le geste de l'instrumentiste et son impact sur certains des paramètres physiques de l'instrument.

Par conséquent, pour utiliser des données de suivis de geste avec le modèle physique de violon présenté précédemment, il est tout d'abord nécessaire de rendre accessible l'ensemble des paramètres physiques de l'instrument dans l'interface utilisateur et de retirer le générateur d'enveloppe de l'algorithme.

Le paramètre *force* mentionné précédemment nécessite un traitement particulier. En effet, il est primordial que toutes ses valeurs soient interpolées afin d'éviter toutes discontinuités dans le son produit. La fonction `smooth` de la bibliothèque `filter.lib` permet de mener à bien cette tâche.

Comme cela a été dit précédemment, le modèle physique de violon dont il est ici question utilise quatre cordes. Cela permet d'éviter les discontinuités possibles dans le son produit lors d'un jeux rapide, de modéliser la courte résonance qui survient lorsque la corde n'est plus excitée par l'archet et surtout d'appliquer des filtres de réflexions différents pour chacune d'entre elles (cf. sous-partie précédente). Il est important de noter que des améliorations pourraient être apportées au modèle afin qu'il prenne en compte les interactions de résonance par sympathie qui peuvent survenir entre les différentes cordes de l'instrument et la caisse de résonance.

176. Attack – Decay – Sustain – Release : Attaque – Relâchement – Maintien – Chute.

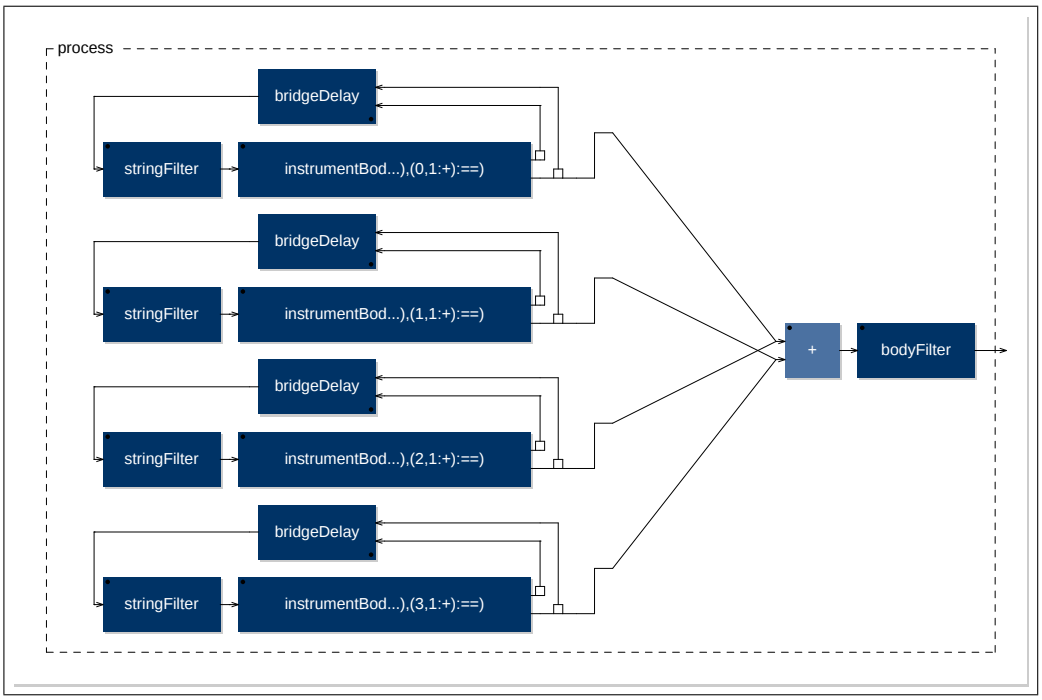

FIGURE 3.10 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `violin.dsp`. Quatre instances de corde sont déclarées avec des filtres de réflexion (`stringFilter`) différents pour chacune d'entre-elles. Le filtre modélisant la caisse de résonance du violon est appliqué à la somme des signaux générés par les quatre cordes.

L'utilisation de plusieurs cordes nécessite d'apporter quelques modifications au niveau de la gestion des paramètres du modèle. En effet, lorsqu'une corde n'est plus utilisée, il est indispensable que la longueur de la ligne de retards calculée en fonction de la fréquence de la note jouée et que la position du point de frottement de l'archet sur la corde, conservent les dernières valeurs qui leurs ont été envoyées. Cela permet d'éviter toutes discontinuités dans le son produit lors de l'utilisation suivante de la corde et surtout de conserver la bonne note lorsque la corde vibre librement une fois qu'elle n'est plus excitée par l'archet.

Une description détaillée du fichier `violin.dsp` [177] qui implémente ce modèle est faite dans l'annexe O.

Utilisation des données de suivi de gestes d'instrumentiste

La technique utilisée pour le collectage de données de suivis de gestes pour le violon est décrite en détail dans la thèse de doctorat d'Esteban MAESTRE [178]. Elle

177. Disponible sur le CD dans le dossier `/muneira/`.
178. MAESTRE, Esteban, *Modeling Instrumental Gestures : an Analysis/Synthesis Framework for Violin Bowing*, Thèse de doctorat inédite, Université Pompeu Fabra (Barcelone), 2009.

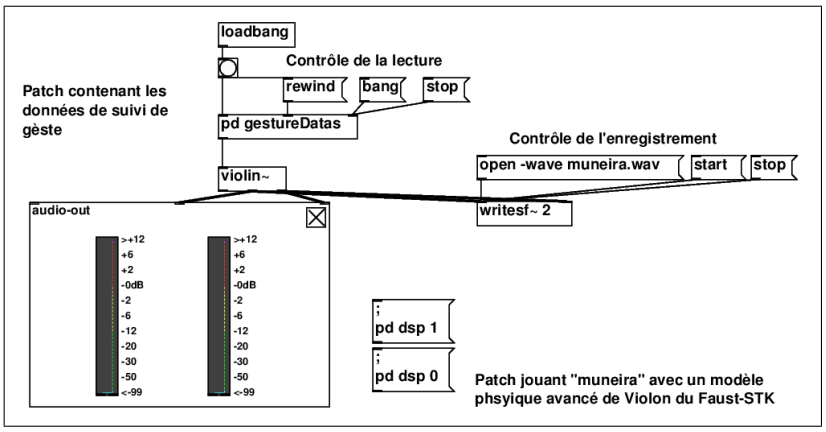

FIGURE 3.11 – Capture d'écran du patch PUREDATA `jouerMuneira.pd`.

consiste à placer un ensemble de capteurs sur le violon et sur l'archet afin d'enregistrer les mouvements de l'instrumentiste lors de son jeu. A partir des données enregistrées, les paramètres suivants (donnés avec leur nom dans le programme FAUST) qui sont par la suite directement utilisés avec le modèle physique sont calculés :

- `bowPosition` : la position du point de frottement de l'archet sur la corde ;
- `bowVel` : la vélocité de l'archet lorsque celui-ci est utilisé pour exciter la corde ;
- `force` : la force en Newtons appliquée par l'archet sur la corde ;
- `stringNumber` : le numéro de la corde mise en vibration.

En même temps que les données de suivis de gestes sont enregistrées, la fréquence de la fondamentale de la note jouée est calculée en temps réel en effectuant une analyse FFT [179] sur le son enregistré par un microphone placé près du violon.

Afin de ne pas surcharger le modèle physique avec un surplus d'informations inutiles, l'ensemble des flux de données collectés lors des opérations précédentes ont été synchronisés et quantifiés avec une période de 4,167 millisecondes. Les listes de valeurs obtenues ont alors été enregistrées dans des fichiers textes formatés pour être facilement utilisés dans un patch PUREDATA : `jouerMuneira.pd` [180] dont une capture d'écran est visible dans la figure 3.11.

Chaque valeur successive pour un paramètre donné a été placée sur une ligne différente. Le sous-patch `gestureData` (cf. figure 3.12) lit alors de manière synchronisée l'ensemble des fichiers textes contenant les flux de valeurs ligne par ligne à raison d'une valeur toutes les 4,167 millisecondes. Ces valeurs sont en suite formatées en temps réel

179. *Fast Fourier Transform* : Transformée de Fourier Rapide.
180. Fichier disponible sur le CD dans le dossier `/muneira/`.

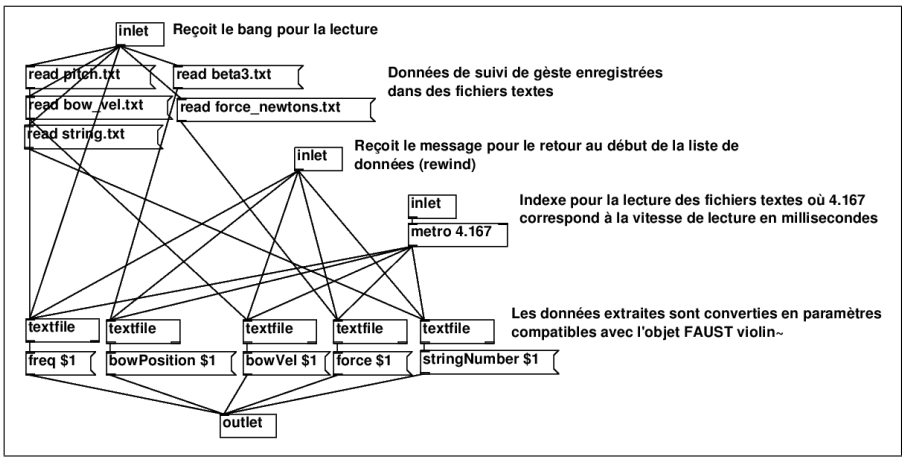

FIGURE 3.12 – Capture d'écran du sous-patch `gestureDatas` du patch PUREDATA `jouer-Muneira.pd`.

pour être reconnues par le plug-in produit par FAUST qui est placé dans le patch principal.

Lorsque le bouton `bang` est pressé, les flux de données sont envoyés au modèle physique et la lecture est lancée. Le résultat obtenu peut être entendu dans le fichier audio `muneira.wav` [181].

A l'écoute de ce dernier, il est évident que le son obtenu est beaucoup plus fidèle à celui d'un violon réel que celui de l'exemple de l'instrument `bowed.dsp` (`muneira-bowed.dsp` [182]) présenté dans la partie 3.3.1. En dépit des améliorations apportées au modèle physique de `bowed.dsp`, il est aisé de s'apercevoir qu'un contrôle accru sur les différents paramètres physiques du modèle permet d'acquérir des résultats de très haute qualité, même avec un modèle très simple. Il est intéressant de mettre en parallèle ces conclusions avec la technique choisie par les ingénieurs de la firme Yamaha pour contrôler le synthétiseur *VL1*. En effet, il a été expliqué dans la partie 1.5.2 que pour jouer des différents instruments à vent de ce synthétiseur, il était possible d'utiliser un contrôleur de souffle. Ce dernier offrait donc le même type de contrôle au niveau des paramètres physiques du modèle que celui obtenu en utilisant des données de suivi de gestes d'instrumentistes dans le cas du modèle de violon présenté dans cette partie.

La conclusion qui peut être donnée à ce travail est que la complexité du modèle physique utilisé n'a finalement qu'une importance moindre face à l'exactitude des valeurs utilisées pour chacun de ses paramètres dans un contexte de jeu donné.

181. *Ibid.*
182. Fichier disponible sur le CD dans le répertoire `/violon/`.

Esteban MAESTRE travaille actuellement au CCRMA de l'université Stanford sur la constitution d'une base de données de comportements gestuels de violonistes en fonction de différents styles musicaux et de différents modes de jeu en s'inspirant des travaux du même type menés par Jordi BONADA au MTG de l'université Pompeu Fabra de Barcelone sur la synthèse de la voix chantée [183]. Son but est d'implémenter un programme qui donnerait la possibilité à un utilisateur d'entrer des informations de très haut niveau (informations MIDI par exemple) qui seraient automatiquement traduites en listes de valeurs de paramètres physiques en fonction des informations contenues dans la base de données pour un mode de jeu particulier qui serait également défini par l'utilisateur. En d'autres thermes, le programme d'Esteban fonctionnerait comme un échantillonneur de données de suivi de gestes qui seraient envoyées à un modèle physique. Naturellement, cette technique est applicable à la plupart des modèles physiques d'instruments de musique. A l'écoute des résultats expérimentaux obtenus, les travaux d'Esteban semblent très prometteurs.

3.4 Instruments à vent

3.4.1 Flûtes traversières

Deux modèles de flûtes traversières sont implémentés dans le FAUST-STK. Tous deux utilisent un guide d'ondes bi-dimensionnel pour modéliser le tube constituant le corps de l'instrument. Le premier, `flutestk.dsp` [184], reproduit de manière presque identique le modèle de flûte disponible dans le SYNTHESIS TOOLKIT [185]. La seule différence vient de l'utilisation du filtre passe-tout passif non-linéaire présenté dans la partie 2.2 qui permet d'ajouter des non-linéarités dans le son produit.

L'autre modèle, `flute.dsp` [186], est basé sur un modèle implémenté dans un programme CSOUND par Hank MIKELSON dans lequel un ensemble d'instruments utilisant la technique des guides d'ondes numériques est utilisé pour jouer une courte pièce. L'orchestre, la partition CSOUND correspondante et le son qu'ils permettent de

183. BONADA, Jordi ; SERRA, Xavier, « Synthesis of the Singing Voice by Performance Sampling and Spectral Models », *Signal Processing Magazine*, XXIV (2007), n° 2, p. 67-79.
184. Fichier disponible sur le CD dans le dossier `/fluteSTK/`.
185. COOK, Perry, *A Meta-wind-instrument Physical Model, and a Meta-controller for Real Time Performance Control : actes de the International Computer Music Conference, San Jose (US), 1992*, San Jose : Computer Music Association, 1992.
186. Fichier disponible sur le CD dans le dossier `/flute/`.

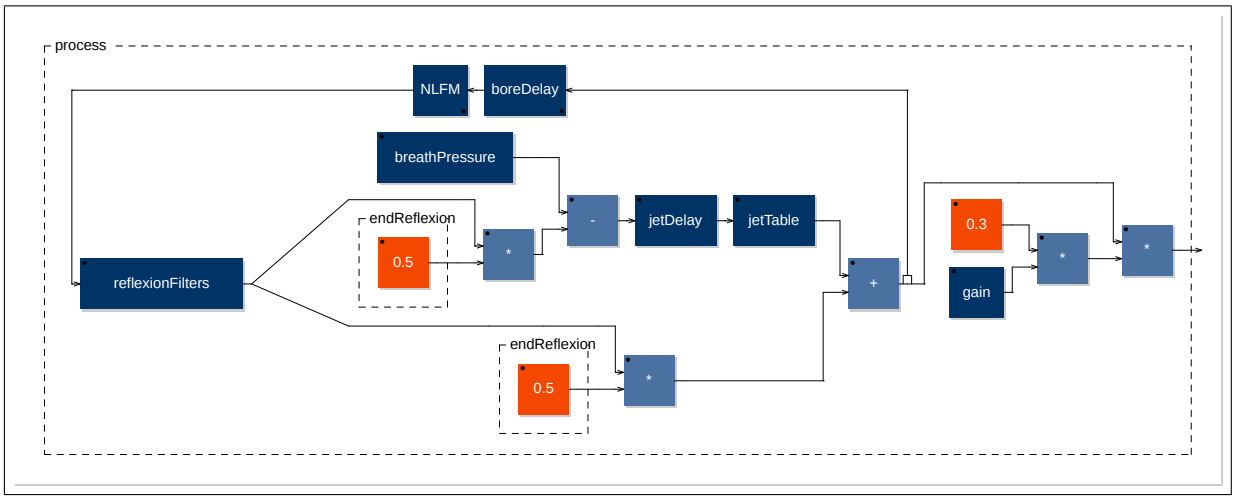

FIGURE 3.13 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `flutestk.dsp`. La fonction de chaque « boîte » est donnée dans l'annexe K.

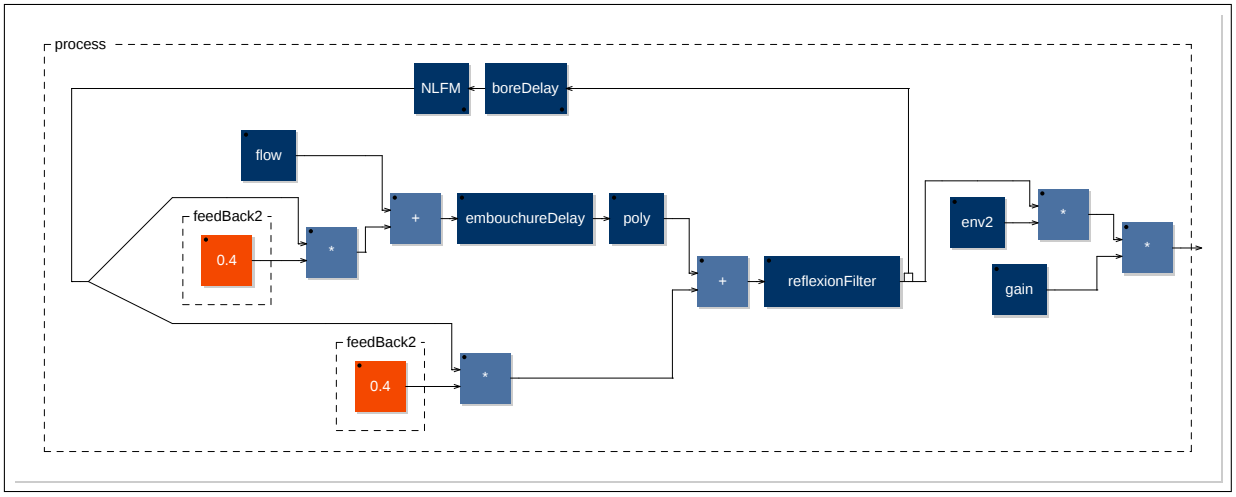

FIGURE 3.14 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `flute.dsp`. La fonction de chaque « boîte » est donnée dans l'annexe L.

produire sont disponibles sur le CD dans les fichiers `physmodl.orc`, `physmodl.sco` et `physmodel.wav` [187].

Les diagrammes de ces deux instruments sont visibles dans les figures 3.13 et 3.14.

Excitations

Les deux modèles de flûtes traversières présentés dans cette partie utilisent des algorithmes semblables pour synthétiser le son de l'excitation qui permet au corps

187. MIKELSON, Hank, *Waveguide Physical Modeling : physmodl.orc et physmodl.sco*, fichiers disponibles en ligne à l'adresse suivante `http://csounds.com/mikelson/#waveguide` et sur le CD dans le dossier `/csound/`.

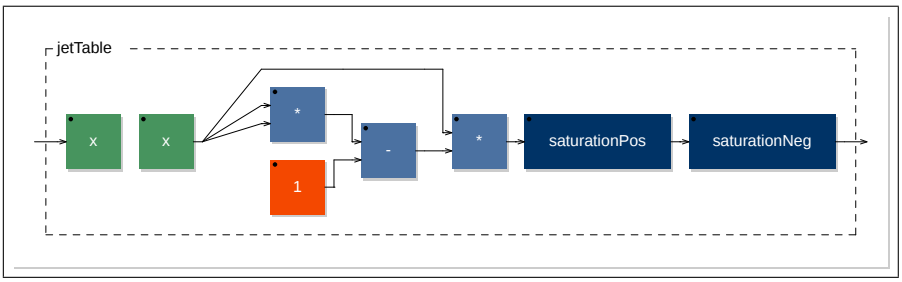

FIGURE 3.15 – Diagramme (produit avec l'outil `faust -svg`) de l'algorithme modélisant les interactions entre l'excitation et le signal de réflexion dans l'instrument `flutestk.dsp`. La boite *saturationPos* permet de faire saturer le signal positivement et la boite *saturationNeg* permet de le faire saturer négativement.

de l'instrument de rentrer en vibration. Ainsi, un simple générateur d'enveloppe de type *Attaque – Relâchement – Maintien – Chute* est utilisé pour introduire dans le tube acoustique l'énergie nécessaire pour que ce dernier se mette à vibrer. Le signal produit par ce générateur est légèrement modulé avec un oscillateur de basse fréquence afin de créer un effet de vibrato. Enfin, un générateur de bruit blanc est utilisé pour imiter le son du souffle de l'instrumentiste.

Le signal produit lors de l'opération précédente est introduit dans le guide d'ondes de la flûte en l'ajoutant aux ondes réfléchies par la fin du tube de l'instrument. Les interactions entre ces deux signaux créent des non-linéarités qui sont alors modélisées avec un polynôme du type[188] :

$$y = x - x^3$$

Sa fonction peut être comparée à celle de la fonction `reedTable` de l'instrument `clarinette.dsp` présenté dans l'annexe D. Dans le cas du modèle de flûte issu du SYNTHESIS TOOLKIT, d'autres non-linéarités sont créées dans le signal en le faisant saturer (cf. figure 3.15).

L'algorithme utilisé pour produire l'excitation dans le cas de `flute.dsp` peut parfois entraîner l'apparition de légères irrégularités lors des phases d'attaque et de chute. Pour pallier ce problème, une seconde enveloppe d'amplitude est appliquée sur le son généré par le modèle (cf. boîte *env2* sur la figure 3.14).

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

Tubes acoustiques

Les corps des deux flûtes sont très similaires et sont composés d'un guide d'ondes bi-dimensionnel. Deux lignes de retard sont donc utilisées et des coefficients de réflexion sont placés à chaque extrémité du tube. Un filtre de réflexion de type passe-bas est utilisé à la fin du tube et est appliqué sur le signal réfléchi et renvoyé à l'autre extrémité du guide d'ondes. Dans le cas de `flutestk.dsp`, le filtre est un simple filtre récursif à un pôle tandis que `flute.dsp` utilise un filtre passe-bas possédant une réponse impulsionnelle légèrement plus complexe (cf. annexe L) qui permet d'améliorer de manière significative la qualité du son produit.

Dans la mesure où `flute.dsp` ne possède pas de système permettant de faire saturer le signal à une certaine valeur lorsque les interactions entre l'énergie introduite dans le tube acoustique et les ondes de réflexions se produisent, ce modèle est plus instable que celui implémenté dans `flutestk.dsp`. De ce fait, il est nécessaire de limiter la quantité d'énergie introduite dans le guide d'ondes afin d'éviter que ce dernier devienne instable. Cela a pour effet de limiter la vitesse d'attaque du son rendant l'enchaînement de notes rapides pratiquement impossible avec ce modèle. Les exemples sonores `voi-che-flute.wav` [189] et `voi-che-flutestk.wav` [190] mettent assez bien en valeur cette propriété.

3.4.2 Clarinettes

Deux modèles physiques utilisant la technique des guides d'ondes numériques sont implémentés dans le FAUST-STK. Le premier, `clarinette.dsp` [191], a déjà fait l'objet d'une description dans la partie 1.4 et dans l'annexe D, son fonctionnement n'est donc pas détaillé à nouveau ici. Il s'agit d'un modèle très simple utilisant un guide d'ondes unidimensionnel pour modéliser le corps de l'instrument et qui n'utilise pas le filtre passe-tout passif non-linéaire décrit dans la partie 2.2. L'autre modèle de clarinette du FAUST-STK est implémenté dans le fichier `blowHole.dsp` [192] dont le fonctionnement est détaillé dans l'annexe E. Il intègre deux modèles différents de clefs pouvant être utilisés respectivement pour simuler l'effet de la clef d'octave sur le son produit ou celui de toute

189. Fichier généré avec le patch PUREDATA `voi-che-flute.pd` contenus dans le dossier `/flute/` du CD.
190. Fichier généré avec le patch PUREDATA `voi-che-flutestk.pd` contenus dans le dossier `/fluteSTK/` du CD.
191. Fichier disponible sur le CD dans le dossier `/clarinette/`.
192. Fichier disponible sur le CD dans le dossier `/clarinette2/`.

autre clef. Bien qu'ils soient parfaitement fonctionnels, ces derniers ne peuvent pas être utilisés en pratique pour changer la hauteur du son produit et sont implémentés à titre purement expérimental. Ainsi, dans le cas de l'exemple sonore `voi-che-blowHole.wav` [193], la fréquence du son généré est déterminée par la longueur de la ligne de retards du guide d'ondes principal du modèle et non par l'ouverture ou la fermeture des clefs placées sur le corps de l'instrument comme c'est la cas dans la réalité [194] [195].

Le modèle physique

Le signal d'excitation est produit avec une fonction non-linéaire dont le fonctionnement est décrit dans l'annexe R à la page 175. Cette dernière prend en compte les interactions possibles avec les ondes de réflexions renvoyées par la fin du tube acoustique.

Le filtre passe-tout passif non-linéaire décrit dans la partie 2.2 est utilisé pour provoquer des comportements non-linéaires assez récurrents dans le son produit par une clarinette. Ainsi, le taux de non-linéarités utilisé peut prendre parfois des valeurs assez importantes ce qui aura un effet direct sur la justesse de l'instrument. Il est donc nécessaire d'ajuster la longueur de la ligne de retards déterminant la fréquence du son joué en fonction du taux de non-linéarités et de l'ordre du filtre en utilisant la formule suivante :

$$rL = SR/freq - nlT.fO$$

où rL est la longueur de la ligne de retards en nombre d'échantillons, SR est la fréquence d'échantillonnage, $freq$ est la fréquence de la note en Hertz, nlT est le taux de non-linéarités et fO est l'ordre du filtre.

Modèles de clefs

Les clefs, qui sont constituées d'un système trou / tampon sont modélisées à l'aide de guides d'ondes utilisant une ligne de retards très courte et de longueur constante grâce à la technique décrite par Maarten VAN WALSTIJN et Gary SCAVONE [196]. En

193. *Ibid.*
194. SMITH, Julius, *Efficient Simulation of the Reed-bore and Bow-string Mechanisms : actes de International Computer Music Conference, La Haye, 1986*, La Haye : Computer Music Association, 1986, p. 275-280.
195. SMITH, Julius, *Waveguide Simulation of Non-cylindrical Acoustic Tubes : actes de International Computer Music Conference, Montreal, 1991*, Montreal : Computer Music Association, 1991, p. 303-307.
196. VAN WALSTIJN, Maarten ; SCAVONE, Gary, *The Wave Digital Tonehole : actes de Computer Music Conference, Berlin, 2000*, Berlin : Computer Music Association, 200, p. 465-468.

effet, un trou au niveau de la clef sur le corps d'une clarinette peut être considéré comme un tube de très petite taille et peut par conséquent être modélisé avec un guide d'ondes. Parallèlement, le tampon peut être implémenté sous la forme d'un filtre de réflexion aux effets très radicaux. Plus la distance entre le trou et le tampon est faible, plus le gain du filtre de réflexion du guide d'ondes est important et a de l'effet sur le son produit.

Le guide d'ondes modélisant une clef de la clarinette est relié au corps de l'instrument à l'aide d'une jonction de dispersion à deux ports dans le cas de la clef d'octave et d'une jonction de dispersion à trois ports dans le cas d'une clef normale [197]. Ce type d'assemblage peut être visualisé sur la figure 3.16.

Le fait de lever le tampon a pour effet de disperser l'énergie contenue dans le modèle à son niveau et donc de réduire la taille du tube de l'instrument. Cela se traduit par une hausse de la fréquence du son produit.

Implémenter un modèle de clarinette où la fréquence des notes générées serait totalement contrôlée par des clefs constituerait un travail futur de grand intérêt. En effet, la hauteur des notes est pour l'ensemble des instruments du FAUST-STK contrôlée en modulant la longueur de la ligne de retards du guide d'ondes utilisé pour modéliser le corps de l'instrument. Bien que les résultats obtenus en employant cette technique soient tout à fait corrects, ils pourraient être nettement améliorés en utilisant des modèles de clefs. De plus, le comportement et le maniement des instruments ainsi créés seraient beaucoup plus en corrélation avec ce qu'il se passe dans la réalité. Il est néanmoins important de noter qu'un tel modèle serait beaucoup plus coûteux d'un point de vue computationnel qu'un modèle simple comme celui implémenté dans `clarinette.dsp`.

3.4.3 Saxophone

Un modèle hybride d'instrument à anche simple a été implémenté dans le fichier `saxophony.dsp` [198] du FAUST-STK. D'un point de vue physique, il s'agit d'ailleurs plus d'un violon sur lequel les cordes seraient excitées en les connectant à un bec de clarinette ce qui n'est pas réalisable dans le monde réel. Il est basé sur un modèle récent

197. VALIMAKI, Vesa ; KARJALAINEN, Matti ; LAAKSO, Timo, *Modeling of Woodwind Bores with Finger Holes : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993, p. 32-29.

198. Fichier disponible sur le CD dans le dossier `/saxophone/`. Son fonctionnement est détaillé dans l'annexe F.

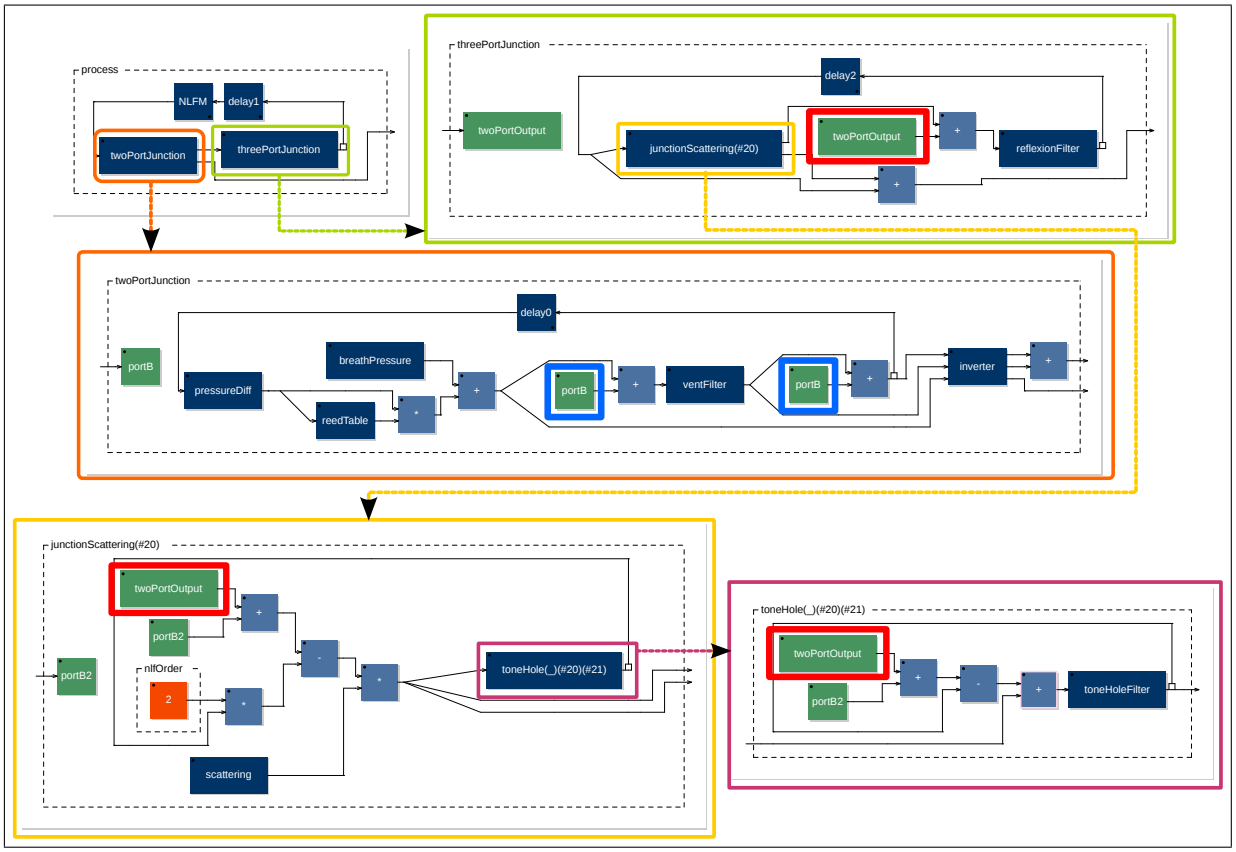

FIGURE 3.16 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `blowHole.dsp` et détail de certain de ses composants. Le rectangle orange contient le modèle de clef d'octave utilisant une jonction de dispersion à deux ports. Les deux ports sont indiqués par des carrés bleu clair. Le rectangle vert contient le modèle de clef utilisant une jonction de dispersion à trois ports. Leur position dans l'algorithme est délimitée par des rectangles rouges. Le rectangle jaune affiche le contenu de la boîte `junctionScattering` de la fonction `threePortJunction` et le rectangle violet le contenu la boîte `toneHole` de la fonction `junctionScattering`. Le diagramme `process` placé en haut à gauche de la figure est le diagramme de haut niveau dans lequel les différents éléments de l'algorithme sont assemblés.

du SYNTHESIS TOOLKIT implémenté par Gary SCAVONE [199].

Le corps de l'instrument est composé d'un guide d'ondes bi-dimensionnel qui est excité avec la fonction modélisant un bec de clarinette utilisée dans les instruments `clarinette.dsp` et `blowHole.dsp` (cf. partie 3.4.2). L'utilisation d'un guide d'ondes bi-dimensionnel est très importante dans le cas présent dans la mesure où celui-ci donne la possibilité de contrôler la position du point d'excitation en allongeant ou en réduisant de manière proportionelle les longueurs des deux lignes de retards qui le constituent. En effet, il est évidemment nécessaire que la longueur combinée des deux lignes de retards reste la même afin que la fréquence de la note produite ne soit pas changée (cf. partie 1.2).

Pour que le point d'excitation soit placé au centre du guide d'ondes, la longueur des deux lignes de retards doit donc être égale. Si la longueur d'une des lignes de retards est augmentée au détriment de l'autre, le point d'excitation sera déplacé vers la gauche ou vers la droite le long du guide d'ondes en fonction de leur position respective.

Changer la position du point d'excitation du guide d'ondes a un impact déterminant sur la nature du son produit. Le tableau 3.1 classe un ensemble d'exemples sonores obtenus à l'aide du patch PUREDATA `voi-che-saxophony.pd` [200] en fonction de la position du point d'excitation utilisé lors de leur production. Ce dernier est défini par un chiffre dont la valeur est comprise entre 0 et 1 ou 0,5 correspond au centre du guide d'ondes et 0 et 1 ses extrémités. Dans le cas des exemples sonores présentés dans le tableau 3.1, la valeur du point d'excitation varie entre 0 et 0,5 dans la mesure où des résultats similaires seraient obtenus pour des valeurs comprises entre 0,5 et 1.

Des sons de violon « sul ponticello » sont obtenus lorsque le point d'excitation est placé aux extrémités du guide d'ondes. En effet, cela reviendrait sur un instrument « réel » à jouer plus près des extrémités d'une des cordes de l'instrument, c'est à dire, soit au niveau du chevalet, soit au niveau des chevilles.

Un diagramme du modèle de l'instrument `saxophony.dsp` est donné dans la figure 3.17. Plus d'informations sur ce dernier peuvent êtres trouvées dans l'annexe F.

199. SCAVONE, Gary, *Time-domain Synthesis of Conical Bore Instrument Sounds : actes de International Computer Music Conference, Göteborg (Suède), 2002*, Göteborg : Computer Music Association, 2002, p. 9-15.

200. Disponible sur le CD dans le dossier `/saxophone/`.

Nom du fichier	Position du point d'excitation	Type de son produit
voi-che-saxophony-050.wav	0.5	Clarinette
voi-che-saxophony-040.wav	0.4	Saxophone
voi-che-saxophony-025.wav	0.25	Violon
voi-che-saxophony-015.wav	0.15	Violon « sul ponticello »
voi-che-saxophony-010.wav	0.1	Violon « sul sul ponticello »

TABLE 3.1 – Exemples sonores de l'instrument `saxophony.dsp` classés en fonction de la position du point d'excitation utilisé sur le guide d'ondes et du type de son obtenu. La positon du point d'excitation est exprimée par un chiffre compris entre 0 et 1 où 0,5 correspond au centre du guide d'ondes et 0 et 1 ses extrémités.

3.4.4 Cuivres

A la différence d'autres familles d'instruments, les cuivres ont tous une forme assez commune d'un point de vue physique. En effet, ils sont constitués d'un tube en cuivre qui se termine par un pavillon et le son est produit par une embouchure dans laquelle les lèvres de l'instrumentiste sont placées et sont mises en mouvement en augmentant la pression de l'air dans la bouche. Bien qu'il existe un nombre important de différences permettant de distinguer chaque instrument, il est possible d'implémenter un modèle générique pour la famille des cuivres qui peut aussi bien être utilisé pour produire des sons de trompette que des sons de trombone à coulisse par exemple. Ainsi, le modèle de cuivre implémenté dans le SYNTHESIS TOOLKIT peut être considéré comme un modèle générique pour l'ensemble de ces instruments. Il a fait l'objet d'une ré-implémentation complète dans le FAUST-STK et peut donc être trouvé dans le fichier `brass.dsp` [201].

Dans un premier temps, il est important de préciser qu'il s'agit d'un modèle très « basique » et qu'un nombre important de simplifications ont été effectuées et l'éloignent fortement d'un modèle physique complet de ce type d'instrument. Julius SMITH donne un bon panorama de toutes les techniques perfectionnées existantes pour la modélisation par guides d'ondes d'instruments de la famille des cuivres [202].

Le modèle d'embouchure utilisé dans `brass.dsp`, bien que très rudimentaire, permet d'obtenir des résultats tout à fait étonnants à condition de contrôler de manière très précise et d'adapter les paramètres physiques de l'instrument en fonction de la note et du caractère souhaité. En effet, à la différence de tous les modèles physiques d'instruments

201. Fichier disponible sur le CD dans le dossier `/cuivre/` et décrit dans l'annexe M.
202. SMITH, Julius, « Brasses », *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, Palo Alto (USA) : Stanford University, 2010, vol. 3, p. 436-439.

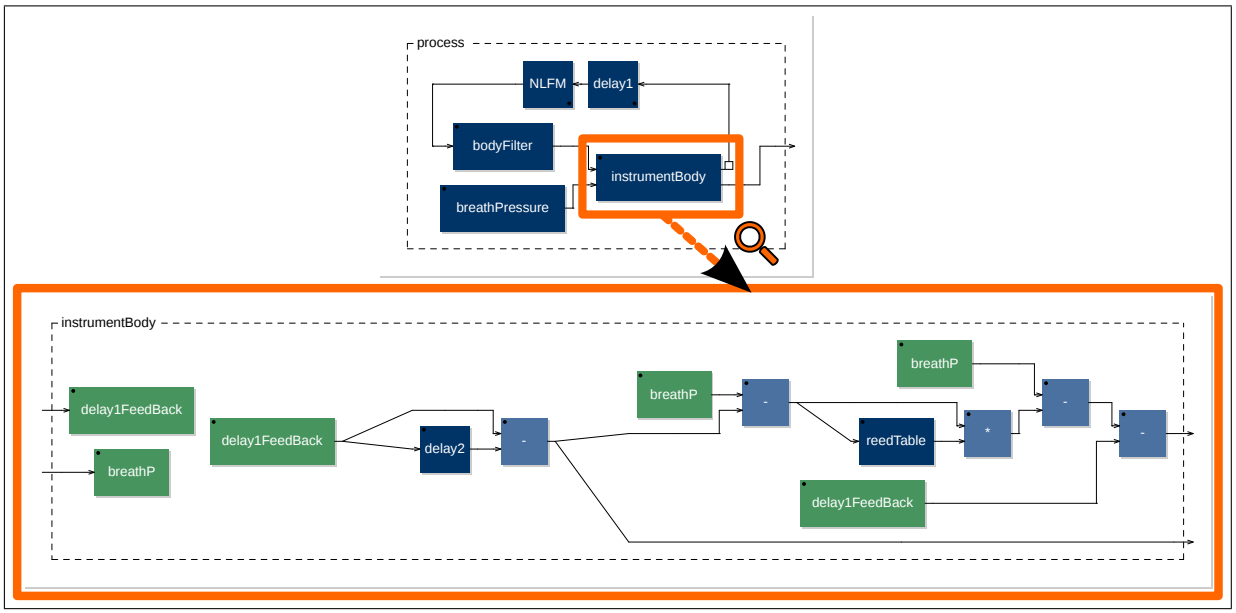

FIGURE 3.17 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `saxophony.dsp`. Le détail du contenu de la boîte `instrumentBody` est donné dans la partie inférieure de la figure. Plus d'informations sur le rôle de chacune des fonctions utilisées peuvent êtres trouvées dans l'annexe F.

présentés jusqu'ici, `brass.dsp` implique l'utilisation de valeurs cohérentes et différentes pour chaque paramètre physique (tension des lèvres, pression, etc.) en fonction de la note jouée et de la pression de l'air au niveau de l'embouchure. Si de mauvaises valeurs sont fournies au modèle, le son risquera au mieux de sonner peu naturel, et au pire de ne pas sonner du tout. Les fichiers audios `voi-che-brass-low.wav` et `voi-che-brass-high.wav` [203] mettent en valeur ce problème.

Le coefficient de tension des lèvres utilisé dans le premier fichier est plus bas que dans le deuxième. La pression est constante et est la même dans les deux cas et la tension des lèvres ne varie pas au cours du temps. La première observation qui peut être faite est que la tension des lèvres a un impact très important sur la fréquence de la note jouée (ce qui est naturellement aussi le cas dans la réalité). On entend également très bien que les notes graves de la phrase musicale ont un meilleur rendu avec une tension faible des lèvres et que à l'inverse, les notes aiguës ont un rendu plus réaliste avec une tension élevée. Des travaux futurs pourraient porter sur l'implémentation d'un algorithme qui permettrait d'adapter automatiquement les paramètres physiques du modèle en fonction de la fréquence et de l'amplitude souhaitées de la note afin d'obtenir le meilleur résultat

203. Fichiers générés avec le patch PUREDATA `voi-che-brass.pd`, disponibles sur le CD dans le dossier `/cuivre/`.

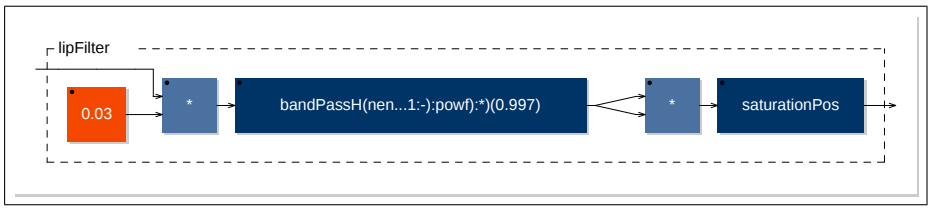

FIGURE 3.18 – Diagramme (produit avec l'outil `faust -svg`) du modèle d'embouchure de l'instrument `brass.dsp`. `bandPassH` est un filtre résonant et `saturationPos` permet de faire saturer le signal à une valeur définie (cf. annexe M).

possible pour chaque configuration.

Le modèle d'embouchure utilisé correspond à celui implémenté par Perry COOK dans son instrument *Tbone*[204]. Il est basé sur un filtre résonant dont la fréquence centrale varie en fonction de la hauteur de la note à jouer. Le signal issu de ce filtre est mis au carré et est saturé de manière très radicale afin de créer les non-linéarités qui permettront de faire résonner le guide d'ondes. Le diagramme de cet algorithme peut être visualisé dans la figure 3.18.

Le tube acoustique de `brass.dsp` est modélisé avec un guide d'ondes unidimensionnel de forme très simple. Un coefficient de réflexion de valeur assez basse est placé dans la boucle du guide d'ondes afin de simuler les déperditions d'énergie. Aucun filtre de réflexion n'est utilisé. Un bloqueur de composante continue permet de rendre le système plus robuste. Un diagramme de haut niveau de l'algorithme du modèle de `brass.dsp` peut être visualisé dans la figure 3.19.

Il faut insister sur le fait que le modèle physique de cuivre présenté ici reste très basique. En effet, en plus des améliorations présentées brièvement précédemment pouvant être apportées au niveau de l'embouchure, un certain nombre d'éléments pourrait venir compléter le modèle du tube acoustique de cet instrument. Un ensemble de filtres pourrait par exemple être placé à la fin du guide d'ondes pour simuler les effets du pavillon sur le son produit et sur les ondes de réflexions. La forme du guide d'ondes utilisé pourrait également être modifiée afin de lui conférer une spécificité plus grande en fonction de l'instrument souhaité (trompette, trombone, etc.). Dans le cas d'une trompette, il serait par exemple nécessaire de prendre en compte l'effet des différents points de torsion du tube acoustique ainsi que celui de l'utilisation de pistons.

204. COOK, Perry, *Tbone : an Interactive Waveguide Brass Instrument Synthesis Workbench for the NeXT Machine : actes de the Interational Computer Music Conference, Montreal, 1991*, Montreal : Computer Music Association, 1991, p. 297-299.

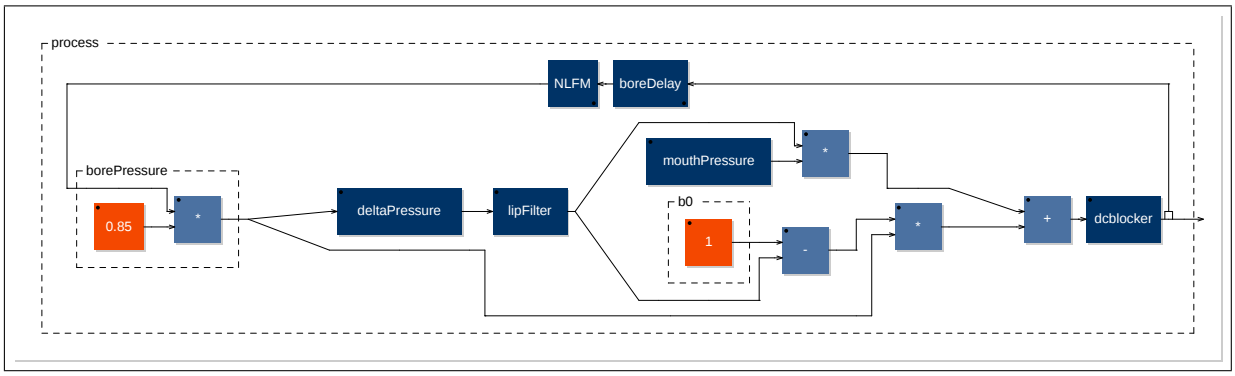

FIGURE 3.19 – Diagramme (produit avec l'outil `faust -svg`) de l'instrument `brass.dsp`. Plus d'informations sur le fonction des différentes boîtes de ce diagramme sont données dans l'annexe M.

3.5 Percussions

L'implémentation d'instruments de musique de la famille des percussions avec la technique des guides d'ondes diffère quelque peu de celle des instruments présentés dans les parties précédentes. En effet, il est tout d'abord bon de mentionner que les techniques de « synthèse » par échantillonnage ont très tôt permis d'obtenir d'excellents résultats pour la reproduction de sons de percussions. Cela a dans une certaine mesure freiné la recherche (notamment à but commercial) dans le domaine de la modélisation physique de ce type d'instruments. De plus, comme cela est montré par la suite, modéliser un instrument de cette famille avec la technique des guides d'ondes n'est pas aussi aisé que dans le cas des instruments à vent ou à cordes. Toutefois, il est important de noter un regain d'intérêt dans ce domaine depuis une dizaine d'années, avec la volonté d'offrir des modèles de percussions « exotiques » à la forme parfois très variable et qui demandent donc un contrôle accru de leurs paramètres :

> « Tandis que la synthèse par échantillonnage est particulièrement efficace pour les instruments à percussion qui sont supposés jouer en arrière plan, une certaine forme de paramétrisation est nécessaire pour des utilisations plus expressives, ou dans le cas d'instruments à la forme très variable tel que le tabla Indien. » [205]

205. SMITH, Julius, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, Palo Alto (USA) : Stanford University, 2010, vol. 3, p. 441, citation traduite de l'anglais : « While sample-playback synthesis is especially effective for percussion instruments that are supposed to play "in the background", some form of parametrization is needed for the more expressive performances, or for highly variable percussion instruments such as the Indian tabla. ».

Pour conclure cette introduction, il est bon de préciser que la technique de modélisation physique par synthèse modale imaginée par Jean-Marie ADRIEN [206] et implémentée dans le programme *Modalys* [207] [208] permet d'obtenir des résultats très impressionnants dans ce domaine. Celle-ci peut aussi être apparentée à la technique des guides d'ondes par bandes de fréquences [209] qui offre la possibilité d'effectuer de la synthèse modale en utilisant des banques de guides d'ondes. Cette dernière n'est pas décrite dans ce mémoire dans la mesure où elle s'éloigne trop de son sujet. Elle est toutefois implémentée dans les instruments `tibetanBowl.dsp`, `tunedBar.dsp`, `uniBar.dsp` et `glassHarmonica.dsp` [210] du FAUST-STK.

Dans un premier temps, il est important de remettre les travaux présentés dans cette partie dans leur contexte. En effet, les modèles physiques décrits par la suite ont été implémentés dans le cadre de mon séjour au CCRMA de l'université Stanford au début de l'année 2011 en collaboration avec Julius SMITH, mon directeur de recherche sur place, spécialiste et inventeur de la technique de modélisation physique d'instruments de musique par guides d'ondes numériques.

A cette époque, Julius SMITH travaillait sur l'utilisation de filtres passifs non-linéaires dans des réseaux multi-dimensionnels de guides d'ondes afin de pouvoir modéliser des instruments de la famille des percussions au comportement hautement non-linéaire tels que des cymbales ou des gongs. Notre travail commun a consisté à implémenter ce type de modèle, purement expérimental dans le langage FAUST. Julius SMITH m'a fourni les résultats de ses travaux et je me suis chargé avec son aide de leur implémentation.

Deux approches différentes ont été adoptées pour mettre en place ces modèles : une dans laquelle un réseau de guides d'ondes « classique » est utilisé (cf. sous partie 3.5.1) et une autre basée sur un réverbérateur *FDN* (cf. sous partie 3.5.2).

206. ADRIEN, Jean-Marie, « The Missing Link : Modal Synthesis », *Representations of Musical Signals*, éd. sous la direction de Curtis Roads, Cambridge : MIT Press, 1991, p. 269-297.
207. ECKEL, Gerhard ; IOVINO, Fransisco ; CAUSSÉ, René, *Sound Synthesis by Physical Modelling with Modalys*, Paris : IRCAM, 1995.
208. MORISON, Joseph ; WAXMAN, David, *Modalys Introduction*, Paris : IRCAM, 1997 (troisième édition).
209. ESSL, Georg ; COOK, Perry, *Banded Waveguides : Towards Physical Modeling of Bowed Bar Percussion Instruments : actes de International Computer Music Conference (ICMC), Pékin, 10/1999*, Pékin : International Computer Music Association, 1999.
210. Disponible sur le CD dans le dossier `/autres/`.

Le travail effectué dans le cadre de ce projet a fait l'objet d'une publication lors de la *International Conference on Digital Audio Effects* (DAFx) [211].

3.5.1 Réseaux multi-dimensionnels de guides d'ondes : « filets » de guides d'ondes

Le modèle physique présenté dans cette partie n'a pas encore été intégré au FAUST-STK dans la mesure où il reste très expérimental. Il implémente une plaque carrée dont la nature est définie par les paramètres qui lui sont fournis et dont la taille, variable, est établie avant la compilation. Cette plaque est modélisée à l'aide d'un réseau de guides d'ondes à deux dimensions de la même forme que celui décrit par Scott VAN DUYNE [212] et utilise le filtre passe-tout passif non-linéaire présenté dans la partie 2.2.

Implémentation d'un réseau de guides d'ondes dans FAUST

Bien que la théorie autour des réseaux de guides d'ondes n'est pas donnée ici puisque cela a déjà été fait de manière très précise par Julius SMITH [213], il est bon de rappeler quelques notions à ce sujet afin de comprendre la suite de cette partie.

Un réseau de guide d'ondes est un ensemble de jonctions de dispersion à quatre ports, interconnectées à la manière des mailles d'un filet (*mesh* en anglais) comme le montre la figure 3.20. La vélocité à chaque intersection ou « nœuds » peut être calculée de manière simple à l'aide de la formule suivante :

$$v = \frac{in_1 + in_2 + in_3 + in_4}{2}$$

où in_k sont les signaux des différentes ondes progressives.

Puisque la vélocité de ce type de connexion est continue, on peut calculer la valeur du signal issu de cette jonction de dispersion avec :

$$out_k = v - in_k$$

211. SMITH, Julius ; MICHON, Romain, *Nonlinear Allpass Ladder Filters in FAUST : actes de International Conference on Digital Audio Effects (DAFx-11), Paris, 19-25/09/2011*, Paris : IRCAM, 2011.
212. VAN DUYNE, Scott ; SMITH, Julius, *Physical Modeling with the 2-D Digital Waveguide Mesh : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993.
213. SMITH, Julius, « Digital Waveguide Mesh », *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, Palo Alto (USA) : Stanford University, 2010, vol. 3, p. 605-616.

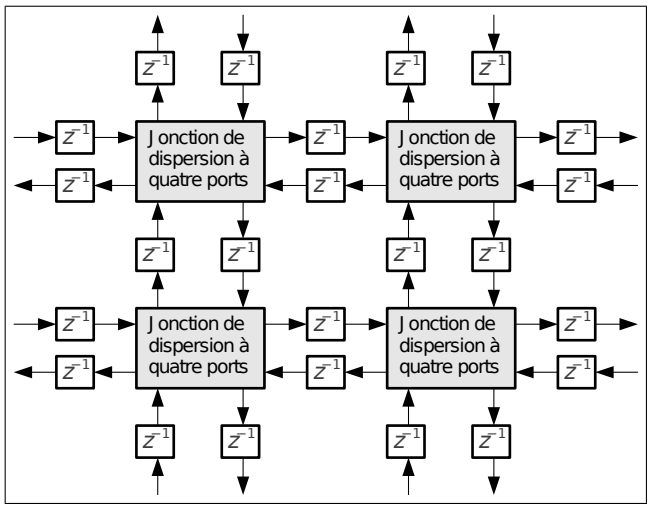

FIGURE 3.20 – Réseau de guides d'ondes à deux dimensions de taille 2x2.

où $k = 1, 2, 3, 4$.

Ainsi, ce type de jonction de dispersion peut être implémenté dans le langage FAUST de la manière suivante :

```
process = bus(4) <: par(i,4,*(-1)),(bus(4) :> *(.5) <: bus(4)) :>
   bus(4);
```

Le diagramme de cette expression peut être visualisé sur la figure 3.21 de la page 99 dans le rectangle en pointillés oranges.

L'implémentation de réseaux de guides de d'ondes faite ici donne la possibilité de choisir avant la compilation la taille N de ce dernier qui s'exprime sous la forme d'un entier positif dont la valeur doit être, dans l'état actuel, un ou un multiple de deux compris entre deux et seize. Ce chiffre correspond au nombre de jonctions de dispersion placées sur un coté du carré. Autrement dit, si par exemple $N = 8$, la taille du réseau sera 8x8.

Cela est rendu possible grâce au système de filtrage par motif de FAUST [214] combiné avec des boucles. Cette opération est détaillée dans la description du fichier `mesh.dsp` [215] dans l'annexe P.

Difficultés liées à l'utilisation de filtrage par motif

L'utilisation de cette technique a mis à jour un certain nombre de défaillances dans le fonctionnement de FAUST pour l'implémentation d'algorithmes impli-

214. Terme défini dans le glossaire à la page 120.
215. Fichier disponible sur le CD dans le dossier `/mesh/`.

quant beaucoup de signaux récursifs. En effet, pour une raison qui reste encore à déterminer, l'implémentation de réseaux de guides d'ondes d'une taille supérieure à 16x16, bien que possible théoriquement, engendre un blocage du compilateur de FAUST. La machine Linux utilisée [216] pour faire les tests de compilation met par ailleurs environ 24 secondes pour compiler le code FAUST d'un réseau de taille 16x16, ce qui est anormalement long.

Un autre problème concerne l'optimisation du code C++ généré par FAUST. Pour illustrer ce propos, un réseau de guides d'ondes de taille 16x16 a été compilé sous forme d'une application FAUST *jack* [217] avec une interface utilisateur *Qt* [218]. Il a été utilisé sur la même machine que celle mentionnée précédemment avec les paramètres suivants pour *jack* :

- frames par périodes : 512 ;
- fréquence d'échantillonnage : 48000 Hz ;
- périodes par buffer : 2.

Le taux moyen d'utilisation processeur alors donné par *jack* était d'environ cinquante pour cent, ce qui est très élevé pour une exécution en temps réel d'un algorithme de ce type. Il est d'ailleurs fort probable qu'une machine moins puissante ne soit pas capable de faire fonctionner cet instrument.

Tout comme pour les instruments de la famille des percussions dans la réalité, la taille d'un réseau de guides d'ondes a un impact très important sur la richesse et la complexité des sons qu'il produit. Par exemple, le timbre du son généré par une timbale est beaucoup plus riche que celui d'un tambour dont la membrane ne ferait pas plus de deux centimètres de diamètre. Cela est dû au fait que la taille d'une membrane ou de toutes plaques rigides est proportionnellement liée à son nombre de modes de vibrations.

Un réseau de guides d'ondes de taille 16x16 ne permet de modéliser une surface rigide que de très petite taille. Par conséquent, le type de sons productible reste très limité. Le but de nos travaux était d'implémenter un réseau de guides d'ondes non-linéaires qui aurait permis de produire des sons de cymbales ou de gongs. Pour obtenir des résultats convaincants, il nous aurait fallu être en mesure d'implémenter un réseau de guide d'ondes d'une taille supérieure à 32x32 ce qui à l'heure actuelle n'est pas possible dans FAUST.

216. Plus d'informations sur cette machine peuvent être trouvées dans l'annexe A à la page 124.
217. Plus de détails sur ce programme sont donnés dans le glossaire à la page 120.
218. Plus d'informations sur ce framework peuvent être trouvées dans le glossaire à la page 120.

Une solution temporaire à ce problème pourrait consister à implémenter un réseau de guides d'ondes de taille finie ce qui permettrait de se passer de l'utilisation de filtrage par motif et solutionnerait donc le problème du temps de compilation. Toutefois, un tel modèle, en plus d'être peu évolutif, serait particulièrement long à implémenter à cause de la redondance de certaines zones du code qui serait inévitable.

Un modèle de plaque carrée : `mesh.dsp`

Bien qu'il soit possible de mettre en place des réseaux de guides d'ondes de formes diverses (rectangles, triangles, cercles, etc.), nous nous sommes contentés avec Julius SMITH d'implémenter dans le cadre de nos recherches sur l'utilisation de filtres passe-touts passifs non-linéaires (cf. partie 2.2) une simple plaque de forme carrée dans FAUST : `mesh.dsp` [219]. Cela est assez facilement réalisable en « fermant » un réseau de guides d'ondes de forme carrée en connectant les signaux sortants de deux de ses côtés sur les entrées des deux côtés restants. Cette opération revient à ajouter des terminaisons rigides à chacun des côtés de la plaque (ou membrane) modélisée, ce qui lui permet d'entrer en vibration lorsqu'elle est excitée. Tout comme dans le cas d'une corde, il est nécessaire de modéliser l'effet de dispersion engendré par l'air qui entoure la plaque et par les terminaisons dont la rigidité ne peut être parfaite (cf. partie 1.1.3). Celui-ci a pour effet d'empêcher que le réseau de guides d'ondes ne vibrent de manière infinie et a donc un effet direct sur le temps de vibration.

Dans le cas d'un modèle complexe de plaque, ces dispersions pourraient êtres simulées par un filtre. Comme le modèle implémenté ici est très expérimental, un simple coefficient est appliqué sur le signal renvoyé aux deux côtés du réseau de guides d'ondes (cf. figure 3.21).

A cette même position dans l'algorithme, le filtre passe-tout passif non-linéaire décrit dans la partie 2.2 est appliqué. Comme cela a été expliqué précédemment, ce dernier est censé permettre de générer avec un réseau de guides d'ondes des sons d'instruments au comportement fortement non-linéaire tels que ceux produits par une cymbale ou un gong. Hélas, la taille des réseaux de guides d'ondes, fortement limitée par les problèmes liés à l'utilisation de filtrage par motif dans FAUST ne nous a pas permis d'obtenir de tels sons. Dans la sous partie suivante, un ensemble d'exemples sonores produits avec ce modèle est présenté.

219. Disponible sur le CD dans le dossier `/mesh/`.

La plaque carrée modélisée dans `mesh.dsp` est excitée en introduisant une simple impulsion au niveau d'un des angles du réseau de guides d'ondes.

L'ensemble des opérations décrites précédemment est effectué à l'aide du code FAUST suivant :

```
squared_mesh_test(N,x) = squared_mesh(N)~(busi(4*N,x))
with{
  busi(N,x) = bus(N) : par(i,N,*(-1*(0.99+resonance)) : nonLin) :
     par(i,N-1,_), +(x);
};
process = excitation : squared_mesh_test(meshSize);
```

`nonLin` est le filtre passe-tout passif non-linéaire, `*(0.99+resonance)` correspond au calcul du coefficient de résonance où `resonance` est un paramètre contrôlé par l'utilisateur. Enfin, `squared_mesh` est le réseau de guides d'ondes (cf. figure 3.21).

n°	Nom du fichier	Taille du réseau	Taux de non-linéarités	Résonance
1	`mesh8_1.wav`	8x8	0	0.3
2	`mesh8_2.wav`	8x8	0	0.85
3	`mesh8_3.wav`	8x8	0	0.95
4	`mesh8_4.wav`	8x8	0.1	0.95
5	`mesh8_5.wav`	8x8	0.3	0.95
6	`mesh8_6.wav`	8x8	0.8	0.95
7	`mesh16_1.wav`	16x16	0	0.3
8	`mesh16_2.wav`	16x16	0	0.85
9	`mesh16_3.wav`	16x16	0	0.95
10	`mesh16_4.wav`	16x16	0.01	0.95
11	`mesh16_5.wav`	16x16	0.1	0.95
12	`mesh16_6.wav`	16x16	0.3	0.95
13	`mesh16_7.wav`	16x16	0.8	0.95
14	`mesh16_8.wav`	16x16	0.95	0.95

TABLE 3.2 – Caractéristiques des sons enregistrés avec le patch PUREDATA `jouer_sineAllpasnn.pd` (disponible sur le CD dans le dossier `/allpassnn/utilisation/sineAllpassnn/`). Chacun d'entre-eux ont une fréquence principale de 440Hz.

3.5.2 Réverbérateur non-linéaire basé sur un réseau de chaînes de retards récursives

Comme cela a été montré dans la partie précédente, il n'a pas été possible d'implémenter dans FAUST un modèle de plaque non-linéaire permettant de produire des sons de cymbales ou de gongs. Pour cette raison, Julius SMITH a suggéré d'exploiter le potentiel extrêmement non-linéaire des filtres passe-tout passifs non-linéaires d'ordre

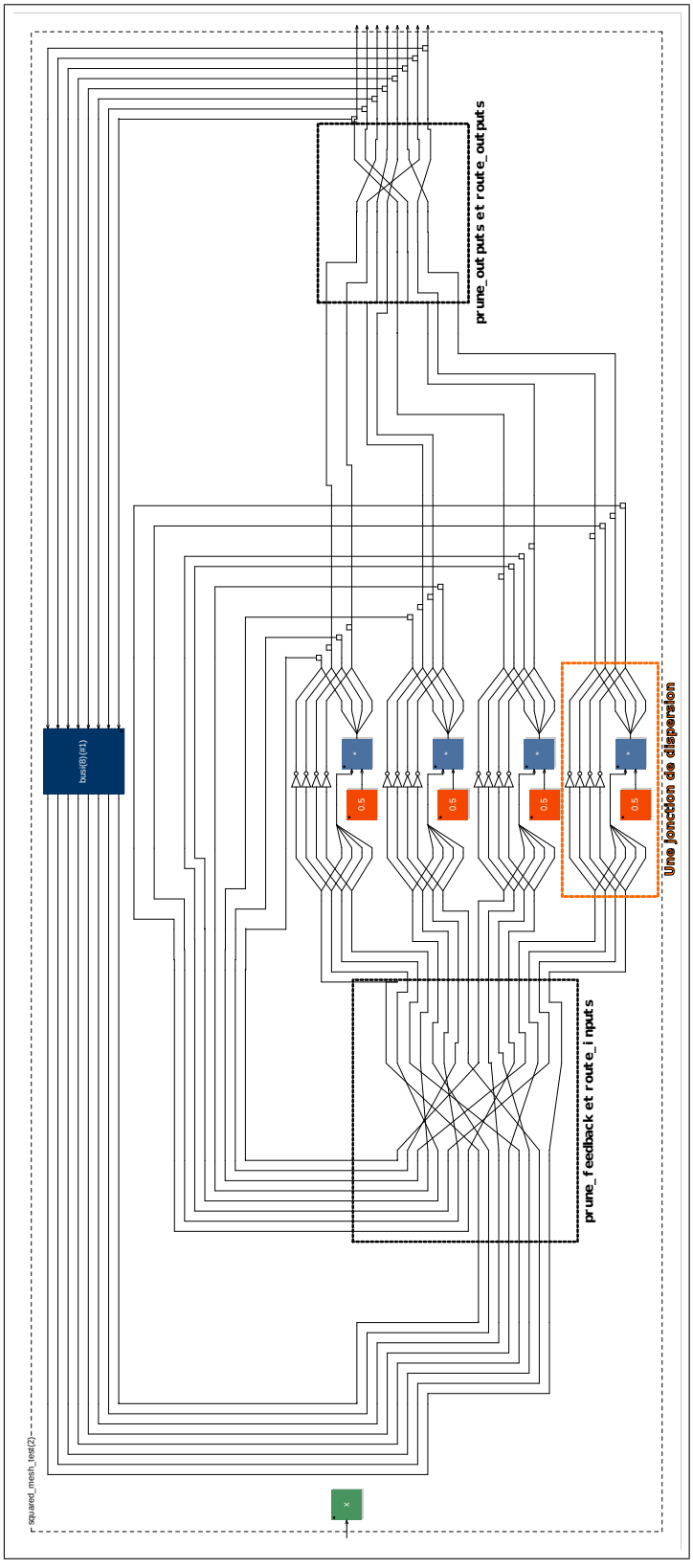

FIGURE 3.21 – Diagramme (produit avec l'outil `faust -svg`) du réseau de guides d'ondes de taille 2x2 de l'instrument `mesh.dsp`. Le rectangle en pointillés oranges délimite l'emplacement d'une des quatre jonctions de dispersion qui le constitue. Le rectangle noir *prune_feedback et route_inputs* contient le résultat produit par la combinaison des fonctions `prune_feedback` et `route_inputs` dont le fonctionnement est décrit dans l'annexe P. La boîte `busi` utilise le filtre passe-tout passif non-linéaire sur chacun des signaux qu'elle prend en argument et applique le coefficient de dispersion du son.

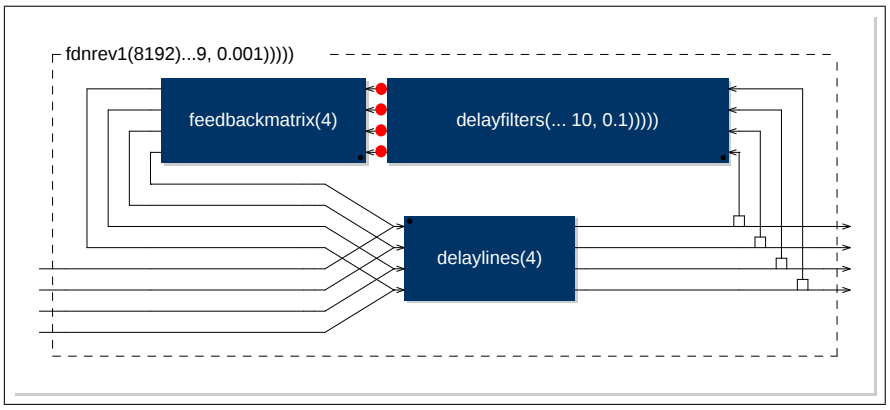

FIGURE 3.22 – Diagramme (produit avec l'outil `faust -svg`) du réverbérateur FDN de l'instrument `nlfFdnRev.dsp`. Les points rouges symbolisent l'emplacement au niveau duquel les filtres passe-touts passifs non-linéaires sont appliqués (à l'entrée de la boîte `feedbackmatrix`).

élevé présentés dans la partie 2.2 pour pouvoir les utiliser dans un réverbérateur à réseau de lignes de retards récursives [220] [221] « FDN » [222]. Selon lui, des filtres de ce type, avec un ordre élevé, utilisés dans ce contexte, avaient peut être le potentiel de générer des sons semblables à ceux d'une structure plus complexe tel qu'un réseau de guides d'ondes de taille importante mais utilisant des filtres passe-touts passifs non-linéaires d'ordre peu élevé.

Implémentation

L'implémentation d'un tel instrument dans FAUST a été particulièrement aisée dans la mesure où Julius SMITH avait déjà travaillé à la mise en place d'un algorithme de réverbérateur FDN dans la bibliothèque `effect.lib` de la distribution de FAUST au travers de la fonction `fdnrev0`. Cette dernière a donc simplement été légèrement modifiée pour pouvoir intégrer le filtre passe-tout passif non-linéaire au niveau de chaque signal récursif du réverbérateur comme le montre la figure 3.22.

Un instrument utilisant la fonction `fdnrev0` en l'excitant avec une simple impulsion à différents endroits de l'algorithme avait fait l'objet d'une implémentation dans le cadre de l'article que Julius SMITH et moi-même avions présenté à la *International Conference on Digital Audio Effects* (DAFx) [223]. Cet instrument a été intégré à

220. JOT, Jean-Marc, *Etude et Réalisation d'un Spatialisateur de Sons par Modèles Physiques et Perceptifs*, Thèse de doctorat inédite, Télécom Paris, 1992.
221. JOT, Jean-Marc ; CHAIGNE, Antoine, « Digital Delay Networks for Designing Artificial Reverberators », *Audio Engineering Society Convention*, 1991, n° 90.
222. Feedback Delay Network.
223. SMITH, Julius ; MICHON, Romain, *Nonlinear Allpass Ladder Filters in FAUST : actes de Inter-*

FIGURE 3.23 – Capture d'écran de l'instrument `nlfFdnRev.dsp` compilé sous la forme d'une application *Jack* avec une interface *Qt*.

la bibliothèque `effect.lib` de la distribution de FAUST sous la forme d'une fonction : `fdnrev0_demo` qui a donc également été réutilisée pour implémenter l'instrument `nlfFdnRev.dsp`[224] dont il est question dans cette partie. Le fonctionnement de `fdnrev0` et de `fdnrev0_demo` n'est pas décrit dans ce mémoire dans la mesure où cela est déjà fait sur le site internet de Julius SMITH[225] et que ce sujet est très différent de celui dont il est ici question.

L'interface graphique de `fdnrev0_demo` a légèrement été modifiée afin de pouvoir contrôler le taux de non-linéarités de l'ensemble des filtres passe-touts passifs non-linéaires utilisé dans l'algorithme. Une capture d'écran de cette dernière (application *Jack* avec interface *Qt*) peut être visualisée sur la figure 3.23.

Exemples sonores

Les sons produits par le modèle décrit précédemment s'avèrent être très encourageants et montrent que cette approche possède un véritable potentiel et peut dans une certaine mesure se substituer à un réseau de guides d'ondes beaucoup plus complexe.

national Conference on Digital Audio Effects (DAFx-11), Paris, 19-25/09/2011, Paris : IRCAM, 2011.

224. Disponible sur le CD dans le dossier `/nlfFdnRev/`.

225. `https://ccrma.stanford.edu/~jos/pasp/FDN_Reverberators_Faust.html`.

Un ensemble d'exemples sonores a ainsi été généré. Ils ont été classés dans le tableau 3.3 en fonction des paramètres utilisés pour leur production.

Le réverbérateur FDN employé a une taille de 16x16 et l'ordre des filtres passe-touts, passifs non-linéaires utilisés est de 4. D'un point de vue computationnel, le modèle, bien qu'assez lourd, est beaucoup moins onéreux qu'un réseau de guides d'ondes de la même taille qui permettrait d'obtenir des résultats beaucoup moins convaincants. En effet, lorsqu'il est compilé sous la forme d'une application *Jack* avec une interface *Qt*, son pourcentage moyen d'occupation du processeur est d'environ dix-huit pourcents sur la machine Linux utilisée pour les tests [226], ce qui est tout à fait raisonnable pour un tel algorithme.

Des sons de membranes (exemples numéros douze et onze), de plaques métalliques (exemples numéros dix, huit, sept, trois et deux) et de cymbales (exemple numéro neuf) ont été obtenus.

n°	Nom du fichier	P1	P2	P3	P4	P5	P6	P7	P8
1	`nlfFdnRev1.wav`	2123	5245	2,6	2,6	2,6	0,1	0,1	0
2	`nlfFdnRev2.wav`	2123	5245	2,6	2,6	2,6	0,1	0,1	0,999
3	`nlfFdnRev3.wav`	2123	5245	2,6	2,6	2,6	0,1	0,1	-0,999
4	`nlfFdnRev4.wav`	2123	5245	2,6	2,6	2,6	0,1	9,5	0,999
5	`nlfFdnRev5.wav`	2123	5245	2,6	2,6	2,6	6,9	9,5	0,999
6	`nlfFdnRev6.wav`	2123	5245	2,6	2,6	2,6	1,4	1	0,999
7	`nlfFdnRev7.wav`	2123	5245	2,6	2,6	2,6	0,5	1	0,999
8	`nlfFdnRev8.wav`	2123	5245	2,6	2,6	2,6	2,2	0,1	0,999
9	`nlfFdnRev9.wav`	2123	5245	2,6	2,6	2,6	0,1	5,8	-0,999
10	`nlfFdnRev10.wav`	2123	5245	1,6	4,2	6,8	1	0,1	0,999
11	`nlfFdnRev11.wav`	2123	5245	7,8	4,2	0,1	1	0,1	0,999
12	`nlfFdnRev12.wav`	526	3152	7,8	4,2	0,1	1	0,1	0,999

TABLE 3.3 – Exemples sonores de l'instrument `nlfFdnRev.dsp` classés en fonction des paramètres utilisés avec P1 : fréquence de croisement de la bande n° 0 ; P2 : fréquence de croisement de la bande n° 1 ; P3, P4 et P5 : temps de résonance des lignes de retards récursives pour trois bandes de fréquences (leur nombre est défini dans `nlfFdnRev.dsp`) ; P6 : taille minimum de l'aire acoustique (taille minimum de la salle modélisée) ; P7 : taille maximum de l'aire acoustique (taille maximum de la salle modélisée) ; P8 : taux de non-linéarités.

226. Plus d'informations sur cette machine peuvent être trouvées dans l'annexe A à la page 124.

3.6 Optimisation et performances des instruments du Faust-STK

3.6.1 Comparaison de la taille du code C++ du Synthesis Tool-Kit 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 effectuée à titre indicatif en prenant en compte en Faust tout comme en C++ les éléments concernant 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 le tableau 3.4. En moyenne, le code Faust est quatre fois plus court que le code C++.

3.6.2 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 l'université Stanford par Michael GUREVICH et Chris CHAFE [227]. Les codes Faust ont de leur côté aussi été transformés en plug-ins pour PureData en utilisant l'architecture

227. GUREVICH, Michael ; CHAFE, Chris, *stk2pd*, Stanford University : CCRMA, 2007, disponible en ligne à `https://ccrma.stanford.edu/wiki/Stk2pd`.

Nom du fichier FAUST	A	B	C	D	E	F
blowBottle.dsp	74	30	**59.5%**	237	54	**77.2%**
blowHole.dsp	131	66	**49.6%**	373	104	**72.1 %**
bowed.dsp	92	45	**51.1%**	274	69	**74.8%**
brass.dsp	90	36	**60%**	272	63	**76.8%**
clarinet.dsp	78	35	**55.1%**	255	60	**76.5%**
flutestk.dsp	109	43	**60.6%**	309	70	**77.3%**
modalBar.dsp *	63	37	**42.3%**	217	78	**64%**
saxophony.dsp	98	42	**57.1%**	308	69	**77.6%**
sitar.dsp	57	25	**56.1%**	193	42	**78.2%**
bars *	164	35	**78.7%**	396	70	**82.3%**
voiceForm.dsp *	121	65	**46.3%**	325	109	**66.5%**
piano.dsp *	292	158	**45.9%**	750	246	**67.2%**

TABLE 3.4 – Tableau comparant le nombre de lignes du code C++ du *STK* avec le code FAUST du FAUST-STK avec **A**, le nombre de déclarations dans le code C++ ; **B**, le nombre de déclarations dans le code FAUST ; **C**, le gain de taille en terme de nombre de déclarations ; **D**, le nombre de lignes dans le code C++ ; **E**, le nombre de lignes dans le code FAUST et **F**, le gain de taille en terme de nombre de lignes. Dans les deux cas, le nombre de déclarations a été calculé en comptant le nombre de points virgules présents dans le code. Pour les 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.

`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 [228]. Les résultats présentés dans le tableau 3.5 démontrent que les plug-ins pour PUREDATA générés par FAUST sont en moyenne environ quarante-et-un pour cent plus efficace que ceux du *STK*.

Il aurait été intéressant de pouvoir mener à bien une comparaison entre la quantité de mémoire occupée par un instrument du FAUST-STK et celle du même instrument dans le SYNTHESIS TOOLKIT mais ce type de test est beaucoup plus complexe à mettre en œuvre dans PUREDATA, il n'a donc pas été effectué.

Les tests computationnels des instruments du FAUST-STK ne faisant pas partie du SYNTHESIS TOOLKIT sont présentés directement dans leurs parties correspondantes puisqu'il n'est pas question de les comparer.

228. Plus d'informations sur cette machine peuvent être trouvées dans l'annexe A à la page 124.

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 3.5 – Tableau comparant les performances des plug-ins pour PUREDATA générés à partir du *STK* avec ceux créés dans FAUST. Les valeurs contenues dans les colonnes *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.

Chapitre 4

Utilisation de la synthèse par guides d'ondes numériques dans le répertoire musical depuis sa création

4.1 Contexte

L'activité artistique autour de la synthèse par guides d'ondes depuis sa création peut être qualifiée de « monophasée » dans la mesure où elle a été fortement limitée par les licences qui régissaient son utilisation et aussi par sa complexité d'implémentation par rapport à d'autres techniques comme la synthèses FM. En effet, il a été expliqué précédemment dans le chapitre un que les deux seules institutions à détenir les droits de développement de la synthèse par guides d'ondes étaient le CCRMA de l'université Stanford et la firme Yamaha. Ainsi, durant les années quatre-vingt-dix, en dehors de quelques rares exceptions dans le cadre de partenariats, la majorité des applications mis en place autour de cette technique a été fait par ces deux protagonistes ce qui a fortement orienté son utilisation artistique.

D'un côté, le CCRMA a utilisé la synthèse par guides d'ondes dans un but très expérimental pour la mise en place de modèles d'instruments de musiques parfois très détaillés qui étaient donc la plupart du temps des tentatives de reproductions virtuelles d'instruments du monde réel. Peu de compositeurs de cette institution se sont intéressés à cette technique très probablement car sa manipulation demandait des compétences dans le domaine du traitement du signal et de la physique, la rendant plus difficile à utiliser dans un contexte de création musicale que d'autres techniques plus ludiques et intuitives.

Aussi, les modèles, parfois très complexes, pouvaient difficilement être utilisés à d'autres fins que la reproduction de sons d'instruments déjà existants, les connaissances de l'époque ne permettant pas ou peu aux compositeurs de mettre en place des modèles d'instruments inédits qui auraient eu un intérêt musical de plus grande valeur.

De son côté, Yamaha a mis au point le synthétiseur *VL1* (présenté dans la partie 1.5.2) qui avait pour but d'offrir une alternative plus expressive aux synthétiseurs par échantillonnage, plus répandus à l'époque. Il mettait à disposition de son utilisateur un ensemble de modèles physiques d'instruments du monde réel qu'il était difficile d'utiliser dans un contexte différent de celui pour lequel ils avaient été créés limitant ainsi grandement l'intérêt que les compositeurs de musique contemporaine ont pu lui porter.

Il faut enfin encore souligner la complexité induite par l'implémentation de modèles physiques en utilisant la techniques des guides d'ondes par rapport à d'autres techniques de synthèse beaucoup plus intuitives comme la synthèse FM qui a eu un immense succès dans les années quatre-vingt.

Néanmoins, un certain nombre d'œuvres utilisant cette technique ont vues le jour. La plus célèbre d'entre elles est probablement *Silicon Valley Breakdown* de David JAFEE. Elle est présentée dans la partie suivante.

4.2 L'œuvre pionnière : *Silicon Valley Breakdown*

Silicon Valley Breakdown a été composée en 1982 par David JAFFE au CCRMA de l'université Stanford avec la collaboration de Julius SMITH. Cette pièce a été la première de l'histoire de la musique électronique à utiliser un instrument utilisant la technique de synthèse par guides d'ondes numériques qui était alors en développement au CCRMA. Elle est entièrement basée sur l'utilisation d'un algorithme de Karplus-Strong améliorés du même type que celui présenté dans la partie 1.3. Il s'agit donc d'une pièce pour un « ensemble de cordes pincées électroniques ».

Cette œuvre pour bande était diffusée sur quatre canaux dans sa version originale. La synthèse était effectuée sur le *Systems Concept Digital Synthesizer*[229] du CCRMA de Stanford et était contrôlée par une machine FOONLY F-4. Son exécution se faisait en temps réel et une partie du contrôle de la synthèse était faite directement par David JAFFE pendant la représentation. Un enregistrement stéréo a été commercialisé et

229. Plus d'informations sur cette machine sont données dans le glossaire à la page 120.

est encore disponible chez Well-Tempered Productions [230]. Un extrait de ce dernier peut être entendu dans le fichier `Silicon-Valley-Breakdown.mp2` [231].

Sur la plan musical, *Silicon Valley Breakdown* allie deux styles très différents : le bluegrass d'un côté et la musique chromatique de l'autre. Au début de la pièce, les deux parties musicales basées sur ces deux styles évoluent de manière parallèle pour s'unir à la fin de l'œuvre. David JAFFE, qui était d'origine juive et joueur de mandoline, explique également sur son site internet [232] qu'il a intégré à sa pièce des éléments de musique traditionnelle.

Dans une interview vidéo de George OLCZAK [233], David JAFFE explique qu'il était en mesure de contrôler la force avec laquelle la corde « virtuelle » était mise en vibration ainsi que son temps de résonance (concepts très nouveaux à l'époque). Dans *Silicon Valley Breakdown*, il joue donc beaucoup sur ces deux paramètres en effectuant un travail sur le timbre du son produit.

La première exécution publique de *Silicon Valley Breakdown* s'est faite à la biennale de Venise de 1982. L'accueil de l'audience et des critiques a été très positif comme en témoigne cet extrait d'un article du journal *le Monde* du 5 octobre 1982 :

> « Heureusement, une lumiere dansait dans *Silicon Valley Breakdown*, de David Jaffe... Travaillant dans un secteur étroit mais déjà prodigieusement complexe, il prouve que l'on peut maîtriser l'ordinateur et le forcer à entrer dans la musique. Toute la pièce est fondée sur la synthèse d'un son de guitare, mais avec toutes ses qualités d'attaques et de résonances, qui va donner lieu à une vaste étude pleine de fantaisie et parfois assez saisissante, uniquement réalisée par la machine et témoignant d'un sens de la continuité et du développement, d'une intention rythmique et spaciale très rares à ce niveau de souplesse... Elle ouvre sans doute une brèche dans le despotisme de l'ordinateur, un peu comme les premières œuvres de Pierre Henry affirmaient la possibilité de jouer d'une musique nouvelle. Enfin quelqu'un qui joue ! » [234]

Dans le même article, Jacques LONGCHAMPT a également annoncé que *Silicon Valley Breakdown* était en bonne voie pour devenir une « œuvre de référence » dans le répertoire de la musique électronique.

---

230. JAFFE, David, *XXIst Century Mandolin*, JAFFE David, 1 CD Well-Tempered Productions WTP5164 (enreg. : 1982).
231. Fichier disponible sur le CD dans le dossier `/audio/`.
232. `http://www.jaffe.com/svb.html`.
233. OLCZAK, George, *David Jaffe – Silicon Valley Breakdown*, interview de David Jaffe, 1984, disponible sur le CD dans le fichier `JAFFE-Interview.mp4` contenu dans le dossier `/videos/`.
234. LONGCHAMPT, Jacques, « Jouer de l'ordinateur à la Biennale de Venise », *le Monde*, journal du 05/10/1982, Paris : France, 1982.

4.3 Utilisation dans le répertoire de musique contemporaine

D'autres pièces de musique contemporaine utilisant des modèles physiques d'instruments de musique par guides d'ondes ont vu le jour depuis la création de cette technique. Il est intéressant de noter que la plupart d'entre elles sont de compositeurs américains, probablement parce que cette technique a été inventée aux Etats-Unis. Il est d'ailleurs possible de mettre en parallèle cette observation avec la modélisation physique utilisant la synthèse modale [235] qui, à l'inverse, a été conçue en France à l'IRCAM et donc utilisée par un grand nombre de compositeurs Européens.

La majorité des œuvres de ce répertoire utilise les modèles implémentés dans le Synthesis ToolKit dans la mesure où ceux-ci étaient facilement accessibles et connus de tous. Néanmoins, il faut noter que certains travaux, beaucoup plus rares, utilisent des modèles plus spécifiques, conçus parfois sur mesure.

Dans cette partie, quelques œuvres du répertoire de musique contemporaine utilisant des modèles d'instruments de musique par guides d'ondes sont présentées.

4.3.1 Pièces utilisant des instruments du Synthesis ToolKit

Comme cela a été expliqué précédemment, la complexité relative à la conception d'algorithmes de modèles physiques par guides d'ondes a dans une certaine mesure imposé aux compositeurs désireux d'utiliser cette technique l'emploi de modèle préconçu. Le Synthesis ToolKit est rapidement devenu un outil de référence dans le domaine et a par conséquent été à la base d'un nombre important de projets. Toutefois, il est montré dans cette partie qu'en dépit de la volonté des compositeurs de détourner les modèles de leur utilisation principale : la reproduction de sons d'instruments « réels », peu d'entre eux ont été en mesure d'atteindre pleinement cet objectif, encore une fois, très probablement à cause de la complexité des modèles utilisés [236].

235. ADRIEN, Jean-Marie, « The Missing Link : Modal Synthesis », *Representations of Musical Signals*, éd. sous la direction de Curtis Roads, Cambridge : MIT Press, 1991, p. 269-297.

236. CHAFFE, Chris, *Case Studies of Physical Models in Music Composition : actes de 18th International Congress of Acoustics (ICA), Kyoto, 4-9/04/2004*, Kyoto : Kyoto International Conference Hall, 2004.

Things she carried (1996) de Paul Lansky

Things she carried est une courte pièce pour ordinateur et lectrice d'environ sept minutes composée en 1997 par Paul LANSKY. Comme dans beaucoup de ses œuvres, ce dernier utilise la machine comme un medium permettant de faire des « tableaux » de scènes de la vie courante. Ainsi, dans *Things she carried*, une femme énumère l'ensemble des objets en sa possession lors d'un de ses déplacements quotidien [237].

Un modèle de flûte « géante », difficilement réalisable dans la réalité est utilisé :

> « Je suis intéressé par la possibilité d'émuler des instruments réels mais seulement si cela me permet de leur donner des proportions irréelles. Par exemple, j'utilise le modèle physique de flûte de Perry Cook depuis quelques années, mais ma façon préférée de l'employer est en construisant un modèle de flute d'environ six mètres de long et de quatre-vingt-dix centimètres de diamètre. Ce dernier produit des sons géniaux. » [238]

Ce dernier est basé sur le modèle de flûte du SYNTHESIS TOOLKIT et est donc le même que celui présenté dans la partie 3.4.1. Seules ses proportions ont été modifiées. Il est possible de l'entendre dès le début de l'œuvre qui est disponible dans le fichier `things-she-carried.mp3` [239] [240].

S-Morphe-S (2002) de Matthew BURTNER

S-Morphe-S est une pièce pour saxophone soprano et ordinateur composée en 2002 par Matthew BURTNER. Elle a été créée dans le cadre des travaux de ce dernier sur la mise en place d'un *Métasaxophone* [241], saxophone équipé d'un ensemble de capteurs permettant à l'instrumentiste, en plus de jouer de l'instrument, de contrôler une machine traitant son son.

Dans *S-Morphe-S*, le son du saxophone est envoyé dans un modèle physique

237. LANSKY, Paul, *About Things She Carried*, 1997, article en ligne : `http://silvertone.princeton.edu/~paul/liner_notes/tsc.html`.
238. CLARK, Paul, *Paul Lansky Interview*, 1997, article en ligne : `http://www.electronicmusic.com/features/interview/paullansky.html`, citation traduite de l'anglais : « I am interested in being able to emulate real instruments but mainly in the sense that I can get them to assume unreal proportions. For example, I've been using Perry Cook's flute physical model for a few years, but my favorite way of using it is to construct a model of a flute which is about 20 feet long and has a diameter of 3 feet. It produces great sounds. ».
239. Fichier disponible sur le CD dans le dossier `/audio/`.
240. LANSKY, Paul, *A Computer Opera*, lectrice : MACKAY Hanna, 1CD Bridge Records BR 9076 (enreg : 1995).
241. BURTNER, Matthew, « The Metasxophone : Concept, Implementation, and Mapping Strategies for a New Computer Music Instrument », *Organised Sound*, VII (2002), n° 2, p. 201-213.

de bol tibétain qui agit comme une sorte de *réverbérateur* : « L'œuvre de Burtner *S-Morph-S*, décrite comme un « instrument hybride bol chantant/saxophone soprano » a été inspirée par un bol tibétain de prière » [242]. Ce dernier est issu d'un algorithme du SYNTHESIS TOOLKIT utilisant la technique de synthèse des guides d'ondes par bandes qui a brièvement été mentionnée dans la partie 3.5. Ainsi, le modèle est utilisé sans qu'aucune modification ne lui aient été apportée, seul son mode d'excitation est différent.

Un extrait de cette pièce peut être trouvé dans le fichier `SmorpheS.mp3` [243]. L'œuvre complète est disponible chez Innova Record [244]. Enfin, il est intéressant de mentionner la pièce *S-Trance-S* disponible sur le même CD dans laquelle le Métasaxophone est utilisé pour contrôler un modèle physique d'instrument à cordes frottées (le même que celui présenté dans la partie 3.3.1).

Pipe Dream (2003) et *Air Study I* (2002) de Gary SCAVONE

Air Study I et *Pipe Dream* constituent un ensemble de courtes pièces expérimentales composées par Gary SCAVONE dans le cadre des ses travaux sur la modélisation physique du saxophone avec la technique des guides d'ondes numériques.

La première est un trio pour deux modèles de saxophone et un saxophone ténor « réel » de nature très particulière puisqu'il ne possédait aucun trou et ne pouvait donc produire qu'un *si bémol* grave. Dans cette pièce, un travail important sur le timbre est effectué en modulant de manière lente et progressive les paramètres de pressions contrôlant le modèle. Ce dernier a été intégré au SYNTHESIS TOOLKIT en 2003 (cf. partie 3.4.3 sur l'implémentation d'un modèle physique de saxophone dans FAUST).

Pipe Dream est une pièce pour modèle physique de saxophone (le même que dans *Air Study I*). Celui-ci y est contrôlé à l'aide de l'interface *The Pipe* [245] mise au point par Gary SCAVONE. Celle-ci à la forme d'un tube sur lequel ont été placés des touches électroniques, des potentiomètres, etc. et intègre un contrôleur de souffle ainsi que des accéléromètres.

242. HART, Ruth, *When Sax and Computer Collide – It's Metasaxophone Colossus*, Charlottesville (USA) : University of Virginia, 2004, article disponible en ligne : `http://aands.virginia.edu/x2262.xml`, citation traduite de l'anglais : « Burtner's composition, "S-Morph-S", described as a "singing bowl soprano saxophone hybrid computer instrument", was inspired by a Tibetan prayer bowl. ».

243. Fichier disponible sur le CD dans le dossier `/audio/`.

244. BURTNER, Matthew, *Metasaxophone Colossus*, Matthew BURTNER, 1 CD Innova Records 620 (enreg : 2004).

245. SCAVONE, Gary, *The Pipe : Explorations with Breath Control : actes de Conference on New Interfaces for Musical Expression (NIME-03), Montreal, 2003*, Montreal : Université McGill, 2003.

Aucune de ces deux pièces, encore une fois, très expérimentales n'a fait l'objet d'un CD. Un extrait de *Air Study I* est toutefois disponible sur le site internet de Gary SCAVONE [246] et dans le fichier `AirStudyOne.mp3` [247]

4.3.2 Pièces utilisant des modèles sur mesure

Lorsqu'un nouveau modèle physique d'instrument de musique est mis en place et si ce dernier décrit un instrument dans son ensemble, il est bien évidemment intéressant de le tester en l'utilisant dans un contexte musical. Quelques pièces ont vu le jour dans ce type de cadre et ont été créées dans des laboratoires d'informatique musicale dans lesquels compositeurs et chercheurs se côtoient régulièrement. Les pièces *Pipe Dream* et *Air Study I* de Gary SCAVONE peuvent être classées dans cette catégorie puisqu'elles ont été composées au moment où ce dernier travaillait sur un modèle physique de saxophone à l'université MCGILL de Montréal. Deux autres travaux de ce genre sont présentés dans cette partie.

Garden of the Dragon (2003) de Juraj KOJS

Garden of the Dragon est une courte pièce d'environ neuf minutes pour papier cellophane, tubes ondulés et ordinateur, composée par Juraj KOJS avec l'assistance de Stefania SERAFIN en 2003 à l'université de Virginie. Elle a été inspirée par un jouet Japonais très populaire dans les années soixante-dix appelé *hummer*. Il était constitué d'un tube en plastique ondulé qui, lorsqu'il était agité de manière vive dans l'air produisait un son très pur familièrement appelé « voix du dragon ». Stefania SERAFIN a mis en place un modèle physique de cet objet [248] qui est donc utilisée dans *Garden of the Dragon*.

Comme l'explique le compositeur, le but recherché dans son œuvre est d'établir un dialogue entre les objets réels : le cellophane et les tubes et leur équivalent informatique :

> « Dans *Garden of the Dragon*, un monde sonore entre deux objets de la vie courante (cellophane et tube ondulé) a été exploré. Considéré ces objets comme des instruments de musique et les modifier a fourni la base pour mettre en place un nouvel écosystème acoustique. A travers une communication interactive

246. `http://www.music.mcgill.ca/~gary/composing.html`.
247. Fichier disponible sur le CD dans le dossier `/audio/`.
248. SERAFIN, Stefania ; KOJS, Juraj, *The Voice of the Dragon : a Physical Model of a Rotating Corrugated Tube : actes de Conference on Digital Audio Effects (DAFx-03), Londres, 8-11/09/2003*, Londres : Queen Mary University, 2003.

> entre l'instrument réel et son équivalent informatique (ex. le modèle physique de tube et le son de cellophane traité), cet environnement a été transformé en une sculpture sonore virtuelle. » [249]

Cette pièce n'a jamais fait l'objet d'un disque. Néanmoins, deux extraits sonores de cette dernière sont mis à disposition sur le site de son compositeur [250] et peuvent être entendus dans les fichiers `GardenDragon1.mp3` et `GardenDragon2.mp3` [251]

Virtual String (1997) Achim BORNHÖFT

Virtual String est une pièce pour ordinateur composée par Achim BORNHÖFT. Elle est basée sur l'utilisation d'un modèle physique de corde par guides d'ondes certainement assez proche d'un algorithme KARPLUS-STRONG avancé tel que celui présenté dans la partie 1.3. Ce dernier est implémenté dans un programme informatique conçu par le compositeur nommé *vstring*. Une interface graphique permet de contrôler en temps réel les différents paramètres du modèle physique.

Un important travail sur le timbre a été effectué dans cette pièce en testant des paramètres physiques improbables pour la corde qui prend parfois des dimensions gigantesques. Ainsi dans les extraits sonores [252] présents dans les fichiers `VirtualString1.mp3` et `VirtualString2.mp3` [253] il est possible d'entendre le son d'un câble de plusieurs kilomètres de long et d'environ un mètre de diamètre mis en vibration par un archet [254].

4.4 Utilisation dans le répertoire des musiques actuelles

Comme cela a été expliqué précédemment, le « grand public » a eu accès à la synthèse par guides d'ondes numériques principalement via le synthétiseur *VL1*. Bien que ce dernier ait eu un succès très limité à cause de son prix élevé et de sa complexité

249. KOJS, Juraj, *About Garden of the Dragon*, 2003, article en ligne : `http://www.kojs.net/Dragon.html`, citation traduite de l'anglais : « In *Garden of the Dragon*, a sonic world of two objects that may be found in daily life (cellophane and corrugated tubes) was explored. Viewing these objects as music instruments and amplifying them provided the base for designing a new acoustic ecosystem. Through an interactive communication between the real instruments and their computer counterparts (i.e. the tube physical model and processed cellophane), this environment was transformed into a virtual sound sculpture. ».
250. *Ibid.*
251. Fichiers disponibles sur le CD dans le dossier `/audio/`.
252. EDWARDS, Michael, *Stryngbite*, 1 CD Sumtone stcd1 (enreg : 1997 ; R/2003).
253. Fichiers disponibles sur le CD dans le dossier `/audio/`.
254. EDWARDS, Michael, *op. cit*, livret du CD.

d'utilisation, il a été employé par des centaines de musiciens pour se substituer à des instruments du monde réel dans le répertoire des musiques actuelles. Les exemples d'utilisation de cette technique sont donc très nombreux, mais impossibles à répertorier et à quantifier.

Le groupe Tesseract dont Julius SMITH faisait partie dans les années quatre-vingt-dix constitue une bonne illustration (et aussi un clin d'oeil à ce dernier) des propos tenus précédemment. En effet, aussi surprenant que cela puisse paraître, ce groupe dont les répétitions se faisaient dans le garage de la maison du chanteur a donné naissance à une « composition » ayant une véritable valeur historique aujourd'hui : *Vantage Point* (1992). Julius SMITH possédait à l'époque un des prototypes du synthétiseur *VL1* encore non commercialisé et l'a utilisé dans ce morceau pour jouer les parties de flûte et de trompette. Le fichier `vantagepoint.mp3` [255] contient un enregistrement [256] de *Vantage Point* datant de 1997. Il est possible d'y entendre les modèles physiques d'instruments par guides d'ondes du synthétiseur *VL1* :

- de 1'05 à 2'27 : section de cuivres ;
- de 2'39 à 2'55 : flute ;
- de 5'33 à la fin : trompette.

Avec le temps, les sons produits par le *VL1* sont devenus assez connotés et ont par conséquent été de moins en moins utilisés.

255. Fichier disponible sur le CD dans le dossier `/audio/`.
256. TILLMAN, Don, *Vantage Point*, Tesseract, 1 CD Don TILLMAN (enreg : 1997).

Conclusion

La synthèse par guides d'ondes numériques permet de créer de manière simple des modèles physiques concis de la plupart des instruments de musique. Les modèles implémentés sont généralement très efficaces sur le plan computationnel dans la mesure où des formes complexes peuvent être grandement simplifiées et représentées avec des éléments simples de traitement du signal comme des lignes de retards récursives, des filtres et des opérations numériques basiques. Si les modèles par guides d'ondes sont contrôlés de manière précise et cohérente d'un point de vue du jeux de l'instrumentiste, ils sont en mesure de générer des sons très réalistes et proches de leurs équivalents dans le monde réel.

L'implémentation libre et ouverte la plus connue de cette technique est le Synthesis ToolKit qui a été utilisé par des dizaines de développeurs d'applications audios libres depuis sa mise en place au milieu des années quatre-vingt-dix. Toutefois, seule une minorité d'entre d'entre-eux a su apprécier la richesse des modèles physiques qui y sont implémentés, ce qui dans une certaine mesure a amené à un nombre important d'utilisations trop « basiques » de ces instruments dont le son est progressivement devenu assez connoté, voir même désuet par rapport aux résultats obtenus avec des techniques d'échantillonnage qui ont souvent été préférées aux modèles physiques durant les vingt dernières années. Ce phénomène a certainement été en partie dû à l'échec commercial du *VL1*, très coûteux pour Yamaha, qui dans une certaine mesure a orienté les choix des fabricants de synthétiseurs depuis cet évènement.

Un autre élément a fortement joué sur les développements engagés sur cette technique de synthèse depuis sa mise en place. En effet, la technologie autour de la synthèse par guides d'ondes a été régie pendant plus de trente ans par une licence détenue par le CCRMA de l'université Stanford et la firme Yamaha. Ces deux institutions étant donc les seules à pouvoir travailler au développement de cette technologie, les seuls travaux extérieurs dans ce domaine se sont faits sous le couvert de partenariats qui n'ont été que

très peu nombreux. Cette licence arrivant à terme cette année, il est possible d'observer depuis quelques temps l'apparition de nouveaux programmes utilisant la synthèse par guides d'ondes. Il est évident qu'une partie importante du potentiel de cette technique n'a pas encore été explorée. Les travaux présentés ici se sont donc attachés à prouver ce postulat de plusieurs façons.

Il a tout d'abord été question de mettre à disposition le plus grand nombre possible de modèles physiques par guides d'ondes pour la plupart des plateformes pour la synthèse audio et le traitement du signal dans une bibliothèque d'objet appelée Faust-STK. Beaucoup des treize algorithmes implémentés ont été inspirés de ceux disponibles dans le Synthesis ToolKit et le programme SynthBuilder. Le langage de programmation Faust a permis de mener à bien cette opération. Sa syntaxe simple, ludique et imagée s'est révélée être particulièrement adaptée pour l'implémentation de modèles utilisant ce type de technique. Une dimension pédagogique a également été donnée à ce travail en commentant de la manière la plus détaillée possible chaque étape du code.

Une étude autour des différentes possibilités existantes pour introduire des non-linéarités dans le son d'un algorithme de guides d'ondes a été menée. Celle-ci a également permis de présenter un nouveau type de filtre non-linéaire d'ordre arbitraire dont les coefficients peuvent être modulés à chaque échantillon de manière passive. Ce filtre a été intégré à l'ensemble des modèles physiques implémenté dans le Faust-STK dans le but de rendre leurs sons plus réalistes ou de pouvoir les détourner de leur utilisation d'origine. En effet, il a été démontré que lorsque ces filtres sont utilisés de manière ciblée et que le taux de non-linéarités introduites dans le son produit est contrôlé avec des enveloppes, il est possible d'améliorer de manière significative la caractère « vivant » du modèle physique. D'autre part, une modulation extrême des coefficients de ce filtre non-linéaire a permis la production de sons totalement inédits au potentiel musical important.

Un modèle de violon amélioré basé sur celui du Synthesis ToolKit a été mis en place. Les réponses impulsionnelles d'une caisse de résonance et d'un chevalet de violon lui sont appliquées à l'aide de banques de filtres biquadratiques dont les coefficients ont été calculés dans le programme MatLab. Ce modèle a également été adapté pour pouvoir utiliser des données de suivi de gestes d'instrumentistes pour le contrôle de la synthèse. Les résultats obtenus lors de cette expérience ont été assez convaincants. Ils ont permis de montrer que la qualité du contrôle des différents paramètres physiques joue un rôle décisif dans la reproduction du réalisme du son du modèle.

Deux modèles inédits d'instruments de la famille des percussions ont été implémentés. Le but de cette expérience a été de générer des sons d'instruments caractérisés par des comportements hautement non-linéaires tels que des cymbales, des gongs, etc. Deux approches ont été adoptées pour mener à bien cette opération. La première a consisté à utiliser un réseau de guides d'ondes traditionnel dans lequel des filtres non-linéaires ont été placés à chacune de ses extrémités. Bien que d'un point de vue théorique un tel modèle aurait permis l'acquisition du type de résultat souhaité, cela n'a hélas pas été possible à cause d'un problème dans le compilateur de FAUST qui n'a pas été en mesure de générer des réseaux de guides d'ondes de taille importante, nécessaires à la production de ce type de sons. L'autre approche adoptée était basée sur l'utilisation d'un réverbérateur à lignes de retards récursives dans lequel des filtres non-linéaires ont été placés à chacune de ses extrémités. Celui-ci a permis de générer des sons très variés de plaques métalliques, de membranes et de cymbales.

Un état des lieux des créations musicales qui ont utilisé la modélisation physique par guides d'ondes dans un but créatif a été fait. Ce dernier a permis de mettre en valeur le caractère assez tumultueux du passé de cette technique. En effet, il a été indiqué que la synthèse par guides d'ondes a souvent été délaissée par les compositeurs principalement car elle ne leur permettait pas de concrétiser pleinement leurs souhaits créatifs à cause de sa complexité de compréhension et d'implémentation. Parallèlement, il a été indiqué que sa diffusion auprès du « grand public » via la gamme de synthétiseur *VL* de Yamaha lui a permis d'être utilisée sous la forme de modèles physiques d'instruments du « monde réel » dans des centaines d'*œuvres* du répertoire des musiques actuelles.

Les travaux ici présentés se sont attachés à répondre aux différentes problématiques exposées dans l'introduction. Néanmoins, ils amènent une foule de nouvelles questions auxquelles seul l'avenir permettra de répondre comme cela a été souligné en 2002 par Thomas ROSSING, Richard MOORE et Paul WHEELER dans leur ouvrage *The Science of Sound* :

> « L'informatique moderne a grandement soulagé le besoin en simplifications excessives dans le domaine de la modélisation physique. Néanmoins, ce domaine n'en est encore qu'à ses balbutiements en tant que technique de synthèse musicale aux applications pratiques. » [257]

257. ROSSING, Thomas ; MOORE, Richard ; WHEELER, Paul, *The Science of Sound*, San Fransisco : Pearson Education (USA), 2002, p. 672, citation traduite de l'anglais : « Modern computing power has greatly alleviated the need of oversimplification in physical models. Nevertheless, physical modeling is

Pendant plus de trente ans, les chercheurs dans le domaine de la synthèse par guides d'ondes se sont attachés à reproduire virtuellement ce qui était déjà disponible dans le réel. Le principal objectif scientifique de cette démarche était de posséder une bonne compréhension de ce qui est déjà en notre possession avant de pouvoir orienter la recherche dans ce domaine vers l'inouï.

Il est par ailleurs intéressant de mettre en parallèle ces affirmations avec une autre technique de modélisation physique qui a connu un destin tout à fait opposé : la modélisation physique particulaire de Claude CADOZ. En effet, cette dernière, qui est d'un point de vue conceptuel beaucoup plus intuitive et simple à utiliser dans un but créatif devient en revanche bien plus complexe à manipuler lorsqu'il s'agit de modéliser des objets existants.

La question induite par ces diverses observations consiste donc à se demander dans quelle mesure la modélisation physique par guides d'ondes pourra un jour être abordée par les compositeurs de la même manière que l'est la modélisation physique particulaire dans le programme GENESIS.

L'expiration de la licence placée sur la technique de synthèse par guides d'ondes cette année et l'engouement des musiciens depuis quelques temps pour la modélisation physique permettront peut-être d'atteindre de tels objectifs. Il semble que les nouveaux outils à but commercial, aux résultats parfois bluffants, qui ont fleuri ces dernières années ont pour ambition de réussir là où le synthétiseur *VL1* de Yamaha a échoué il y a un peu moins de vingt ans : progressivement remplacer les techniques d'échantillonnage par des modèles physiques. Peut-être que les prédictions faite par Martin RUSS à l'époque de la sortie du *VL1* seraient donc sur le point de se concrétiser : « Le futur ne réside pas dans des banques d'échantillons encore plus allégoriques et impliquant plus de mémoire [...], mais plutôt dans la synthèse, et je place le Yamaha *VL1* en première position. » [258].

Néanmoins, le domaine du logiciel libre, par l'intermédiaire de puissants outils comme FAUST, a assurément son rôle à jouer dans la « redécouverte » de cette technique en offrant aux compositeurs et aux créateurs de nouvelles voies d'exploration dans le sens de la vision que Caude CADOZ et Nicolas CASTAGNE ont de la modélisation

still in its infancy as a practical music synthesis technique. ».

258. RUSS, Martin, « Yamaha VL1, Virtual Acoustic Synthesizer », *Sound Of Sound*, Juillet 1994, citation traduite de l'anglais : « The future does not lie with larger ROM samples and ever more cliched sample sets [...] ; it lies with synthesis, and I put the Yamaha *VL1* in pole position. ».

physique :

> « Il semblerait que pratiquer de la modélisation physique n'est pas – ou n'est pas seulement – pratiquer la synthèse sonore. La principale approche en modélisation physique pour la musique n'a pas pour seul objectif de mettre au point de nouveaux sons, mais plutôt de proposer de nouveaux systèmes pour la création musicale et sonore et à encourager la mise en place de nouveaux processus créatifs utilisant ces systèmes. Par conséquent, la modélisation physique appelle à un *changement de paradigme* dans l'approche faite pour la synthèse sonore et plus généralement pour la création musicale. » [259]

Ces travaux ont donné lieux à la publication de deux articles : un lors des Journées de l'Informatique Musicale (JIM-2011) [260] et un autre lors de International Conference on Digital Audio Effects (DAFx-11) [261].

259. CASTAGNE, Nicolas ; CADOZ, Claude, *op. cit*, citation traduite de l'anglais : « It appears that practicing PM is not – or not only – practicing sound synthesis. The major approaches in PM for music do not aim only at developing new sounds, but rather at proposing new systems for sound and music creation, and at encouraging new creative processes by using these systems. PM thus calls for a *paradigm shift* in the approaches to sounds synthesis and more generally to musical creation. ».

260. MICHON, Romain, *Faust-STK : une bibliothèque de modèles physiques pour le langage Faust : actes des Journées de l'informatique musicale (JIM-2011), Saint-Etienne, 25-27/05/2011*, Saint-Etienne : Université Jean Monnet, 2011.

261. SMITH, Julius ; MICHON, Romain, *Nonlinear Allpass Ladder Filters in FAUST : actes de International Conference on Digital Audio Effects (DAFx-11), Paris, 19-25/09/2011*, Paris : IRCAM, 2011.

Glossaire

C++ : `C++` est un langage de programmation impérative orienté objet. Plus d'informations sur ce langage peuvent être trouvées sur son site officiel : `http://www.cplusplus.com`.

CCRMA : Le *Center for Computer Research in Music and Acoustics* est un centre de recherche sur les technologies pour la musique et l'acoustique basé à l'université Stanford. C'est un laboratoire pionnier dans son domaine qui a fortement contribué au développement de l'informatique musicale grâce notamment à Max MATHEWS et à John CHOWNING.

Csound : Csound désigne un langage de programmation pour la création sonore, ainsi que son compilateur sonore. Le nom Csound provient du langage C, avec lequel il fut écrit au MIT par Barry Vercoe. Ce langage est inspiré de MUSIC, une série de programmes plus anciens développés par Max Mathews (d'après la définition de *Wikipedia*). Plus d'informations à ce sujet peuvent être trouvées sur le site de Csound : `http://www.csounds.com/`.

Faust : Functional AUdio STream est un langage de programmation fonctionnel pour le traitement du signal et la synthèse de sons en temps réel. Grâce à un système de fichiers d'architectures, un seul et unique programme Faust peut être utilisé pour générer du code pour un ensemble de types d'applications et de plug-ins. Plus d'informations à ce sujet peuvent être trouvées sur le site Web suivant : `http://faust.grame.fr`.

faust2plot : `faust2plot` est un script bash utilisant l'architecture `matlabplot.cpp` de Faust pour afficher la valeur des n premiers échantillons retournés par un programme Faust sur la sortie par défaut, en l'occurrence, le terminal dans lequel le script est exécuté.

Filtrage par motif : Le filtrage par motif, ou *pattern matching* en anglais, est la vérification de la présence de constituants d'un motif par un programme informatique, ou parfois même par un matériel spécialisé (définition *Wikipedia*). Dans le cas du langage de programmation FAUST, cette technique permet de créer et de manipuler de manière algorithmique des expressions pour créer des boucles intelligentes par exemple.

JACK : Jack Audio Connection Kit est un serveur son professionnel pour système POSIX tel GNU/Linux et Mac OSX. Il a été conçu pour obtenir des connexions temps réels à faibles latences et une exécution synchrone de ses clients audio et MIDI (définition *Wikipedia*). Plus d'informations sur ce système peuvent êtres trouvées sur le site internet `http://jackaudio.org/`.

MATLAB : MATLAB est un langage de programmation combiné à un environnement de développement utilisé à des fins de calcul numérique. Il est maintenu et développé par la société *The MathWorks* et permet la manipulation de matrice, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs et peu s'interfacer avec d'autres langages comme C++, Java, etc. (d'après la définition donnée sur le site de *The MathWorks* : `http://www.mathworks.fr/`.)

NeXT : NeXT était une entreprise californienne spécialisée dans la fabrication d'ordinateurs entre 1985 et 1996. Fondée et dirigée par Steve JOBS, elle avait pour but de fabriquer des machines entièrement équipées et orientées pour une utilisation dans un cadre universitaire ou d'entreprise. Ces machines utilisaient le système d'exploitation NeXTSTEP dont beaucoup d'éléments ont servi de base à la construction de MacOSX, l'actuel système d'exploitation des machines Apple. Du point de vue des capacités audio, NeXTSTEP intégrait le NeXT MusicKit qui a lui aussi servi de base à la constitution des pilotes audios actuels utilisés sur les machines de la firme Apple. Plus d'informations peuvent être trouvées à ce sujet sur le site « historique » de NeXT : `http://simson.net/ref/NeXT/`.

NeXT MusicKit : Le NeXT MusicKit était système informatique de gestion de ressources audios pour le système d'exploitation NeXTSTEP développé à partir de 1987. Il permettait de construire facilement des applications pour l'informatique musicale et le traitement numérique du signal et était compatible avec le standard MIDI. Le NeXT

MusicKit est encore développé et utilisé à l'heure actuelle pour certaines applications. Plus d'informations à ce sujet peuvent être trouvée sur son site de développement : `http://musickit.sourceforge.net/`.

PureData : PureData (ou *pd*) est un logiciel de programmation graphique pour la création musicale et multimédia en temps réel. Il permet également de gérer des signaux entrants dans l'ordinateur (signaux de capteurs ou événements réseau par exemple) et de gérer des signaux sortants (par des protocoles de réseau ou protocoles électroniques pour le pilotage de matériels divers) (définition *Wikipedia*). Plus d'informations à ce sujet peuvent être trouvées sur le site de développement de ce programme : `http://puredata.info/`.

Qt : *Qt* est un framework orienté objet et développé en C++ par Qt Development Frameworks, filiale de Nokia. Il offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc. *Qt* est par certains aspects un framework lorsqu'on l'utilise pour concevoir des interfaces graphiques ou que l'on architecture son application en utilisant les mécanismes des signaux et slots par exemple. (définition *Wikipedia*). Plus d'information sur *Qt* peuvent être trouvées sur le site `http://qt.nokia.com/`.

Synthèse par échantillonage : Technique de synthèse de son consistant à enregistrer le son d'un instrument « réel » et à le rejouer à différentes hauteurs (soit en constituant une base de données d'échantillon, soit en le transposant) dans le but d'imiter le son du modèle.

Systems Concept Digital Synthesizer : Le *Systems Concept Digital Synthesizer*, plus familièrement appelé *Samson Box*, du nom de son créateur, était un puissant ordinateur au CCRMA de l'université Stanford utilisé pour la synthèse de sons en temps réel entre la fin des années soixante-dix et le début des années quatre-vingt. Il permettait de contrôler de manière combinée 256 générateurs d'ondes et 128 processeurs de traitement du signal qui pouvaient prendre la forme de filtres du second ordre, de générateurs de bruit, etc. Il a été utilisé de manière intensive à l'époque dans un grand nombre de pièces de musique électronique par des compositeurs et des chercheurs tel que John CHOWNING, Xavier RODET, David JAFFE, Julius SMITH, etc. Plus d'informations sur cette machine peuvent être trouvées à l'adresse suivante dans le chapitre *Samson Box* :

https://ccrma.stanford.edu/guides/planetccrma/Some.html.

Annexe A

Note technique

Machine Linux utilisée pour les tests

Lenovo ThinkPad T420S

processeur : Intel Core i7 2620M (quatre cœurs, 2.70 GHz, L3 cache 8 MB)

RAM : DDR3 8GO

ROM : 320GO, 7200rpm

système d'exploitation : Linux Ubuntu 3.0.0-16-generic 64bits

Machine Macintosh utilisée pour les tests

MacBook Pro 3,1

processeur : Intel Core 2 Duo (deux cœurs, 2.2 GHz, L2 cache 4 MB)

RAM : DDR2 2GO

ROM : 500GO, 7200rpm

système d'exploitation : Mac OS X 10.6.8 (64bits)

Fichiers MIDI utilisés pour générer les exemples

L'ensemble des fichiers MIDI utilisés pour générer les exemples peuvent être trouvés dans le dossier `/util/` du CD. Chacun d'entre eux a été écrit dans le programme *Cubase*. `funk.mid` est utilisé pour mener à bien les tests du modèle de guitare basse. `muneira.mid` est une retranscription du suivi de fréquence fournie avec les données de suivie de gestes d'instrumentistes utilisées pour contrôler la synthèse dans la partie 3.3.3. `take5.mid` est une retranscription du thème de *Take 5* de Paul DESMOND. `voi-che.mid` est une re-

transcription de l'air *Voi che sapete* des *Noces de Figaro* de Wolfang A. MOZART. Ces deux derniers fichiers ont été utilisés pour générer les exemples sonores des instruments monophoniques. `turkish-march.mid` est une retranscription de *La marche turque* de Wolfang A. MOZART et a été utilisé pour produire les exemples sonores des instruments polyphoniques.

Compilation et utilisation des exemples du CD

Dans la mesure où le système d'exploitation peut varier d'un lecteur à un autre, aucun fichier binaire de modèle physique du FAUST-STK n'a été intégré au CD. Il est possible d'obtenir ces derniers de deux manières. La plus simple est certainement de compiler les fichiers `.dsp` du CD sur le compilateur en ligne de FAUST, disponible à l'adresse suivante : `http://faust.grame.fr/index.php/online-examples`. Il est également possible de générer les fichiers binaires en local à condition que FAUST et les dépendances de l'architectures que vous souhaitez utiliser soit correctement installés sur le système. Pour cela, les *MakeFiles* contenues dans le dossier `/util/` du CD doivent être placées dans le dossier de l'exemple à compiler. Pour plus d'informations sur leur utilisation, tapez `make help`. Pour être sur de la compatibilité de ces *MakeFiles* avec la version de FAUST installée sur votre système, il est recommandé d'utiliser les *MakeFiles* fournies avec la distribution de FAUST dans le dossier `/examples/`. Il sera toutefois très probablement nécessaire de légèrement les adapter pour que les différentes dépendances de certain fichier `.dsp` du FAUST-STK soient correctement reconnues. Enfin, il est important de préciser que les fichiers générés par *faust2pd* ont été intégrés au CD, il suffit donc simplement de placer le fichier binaire correspondant dans le dossier de l'exemple à utiliser.

Sauf dans les cas où les paramètres sont spécifiés, les exemples sonores de modèles physiques présentés dans ce mémoire ont été produits en utilisant les valeurs par défaut des paramètres des objets FAUST.

Les fichiers `instrument.lib` et `instrument.h` se trouvent dans le dossier `/util/` du CD.

Annexe B

Description du fichier `ks.dsp`

L'instrument FAUST `ks.dsp`[262] implémente le très célèbre *digitar* : un simple instrument à corde pincée basé sur l'algorithme de KARPLUS-STRONG dont le principe de fonctionnement est décrit dans la partie 1.2.
Dans un premier temps, l'interface utilisateur est créée. Cette dernière est compatible avec le programme `faust2pd` (cf. partie 1.2).

```
import("music.lib");

freq = nentry("freq", 440, 20, 20000, 1);
gain = nentry("gain", 1, 0, 10, 0.01);
gate = button("gate");
```

L'excitation est produite par un générateur de bruit blanc dont l'amplitude est contrôlée par une enveloppe. Cette dernière est composée de deux phases : une attaque et une chute. La durée de l'attaque est constante et très courte (un échantillon). La durée de la chute est déterminée par le paramètre `P` qui correspond à la longeur de la ligne de retards du guide d'onde calculée en fonction de la fréquence de la note à jouer.

```
diffgtz(x) = (x-x') > 0;
decay(n,x) = x - (x>0)/n;
release(n) = + ~ decay(n);
trigger(n) = diffgtz : release(n) : > (0.0);
P = SR/freq;
```

La fonction `average` fait la moyenne de la valeur d'une échantillon avec celle du précédent. Elle permet de simuler de façon assez grossière l'effet de dispersion de l'énergie dans la corde.

```
average(x) = (x+x')/2;
```

262. Fichier disponible dans le dossier `/ks/` du CD.

L'algorithme de KARPLUS-STRONG visible dans la figure 1.3 à la page 15 est implémenté.

```
resonator = (+ : delay(4096, P)) ~ (average);
process = noise : *(gain) : *(gate : trigger(P)) : resonator;
```

Annexe C

Description du fichier `eks.dsp`

L'instrument FAUST `eks.dsp`[263] implémente une version améliorée (cf. partie 1.3) par Julius SMITH de l'algorithme de KARPLUS-STRONG.
Dans un premier temps, l'interface utilisateur est créée. Cette dernière est compatible avec le programme `faust2pd` (cf. partie 1.2). La bibliothèque `filter.lib` contient les fonctions `smooth`, `ffcombfilter` et `fdelay4` qui seront utilisées par la suite. La bibliothèque `effect.lib` contient la fonction `stereopanner`.

```
import("music.lib");
import("filter.lib");
import("effect.lib");

freq = nentry("freq", 440, 20, 7040, 1);
gain = nentry("gain", 1, 0, 10, 0.01);
gate = button("gate");
```

Le paramètre `pickangle` correspond à l'angle de pincement de la corde.

```
pickangle = 0.9 * nentry("pick_angle",0,0,0.9,0.1);
```

`beta` est la position du pincement sur la corde.

```
beta = nentry("pick_position", 0.13, 0.02, 0.5, 0.01);
```

`t60` correspond au temps de vibration de la corde en secondes (temps pour atteindre une amplitude de -60dB).

```
t60 = nentry("decaytime_T60", 4, 0, 10, 0.01);
```

`B` est la brillance du son.

```
B = nentry("brightness", 0.5, 0, 1, 0.01);
```

263. Fichier disonible dans le dossier `/eks/` du CD.

`L` indique la dynamique du son en décibels à la limite de Nyquist.

```
L = nentry("dynamic_level", -10, -60, 0, 1) : db2linear;
```

`W` et `A` sont des paramètres pour la spatialisation du signal de sortie qui est ici en stéréo.

```
W = nentry("center-panned spatial width", 0.5, 0, 1, 0.01);
A = nentry("pan angle", 0.5, 0, 1, 0.01);
```

L'excitation est fournie par un générateur de bruit blanc sur lequel est appliqué une enveloppe d'amplitude constituée d'une attaque d'une durée de un échantillon et d'une chute d'une durée `SR/freq`.

```
P = SR/freq; //periode fondamentale en nombre d'echantillons

noiseburst(gate,P) = noise : *(gate : trigger(P))
with {
  diffgtz(x) = (x-x') > 0;
  decay(n,x) = x - (x>0)/n;
  release(n) = + ~ decay(n);
  trigger(n) = diffgtz : release(n) : > (0.0);
};

excitation = noiseburst(gate,P) : *(gain);
```

La position du pincement sur la corde est simulée par un filtre en peigne contenu dans la bibliothèque `filter.lib`. Il est utilisé par la suite pour filtrer le son de l'excitation produit lors de l'étape précédente.

```
ppdel = beta*P;
pickposfilter = ffcombfilter(4096,ppdel,-1);
```

Un filtre passe-bas : `dampingfilter`, est utilisé pour simuler l'amortissement de l'énergie présente dans la corde ce qui permet d'entrainer la diminution des partiels hauts du spectre avant les partiels bas (cf. partie 1.3) comme c'est le cas lors de la vibration d'une corde « réelle ». Ici, le temps de vibration de la corde est calculé en fonction du paramètre `t60` défini par l'utilisateur et de la fréquence de la note jouée.

```
rho = pow(0.001,1.0/(freq*t60));

dampingfilter(x) = rho * ((b0 * x) + (b1 * x'))
      with{
    b1 = 0.5*B; b0 = 1.0-b1;
      };
loopfilter = dampingfilter1;
```

L'excitation est traitée par le filtre permettant de modéliser l'effet de la position du pincement sur la corde ainsi que par la fonction `smooth` qui simule l'angle du pincement

en lissant le signal de l'excitation.

```
filtered_excitation = excitation : smooth(pickangle)
            : pickposfilter : levelfilter(L,freq);
```

L'algorithme de KARPLUS-STRONG tel qu'il est décrit dans 1.3 est implémenté.

```
stringloop = (+ : fdelay4(Pmax, P-2)) ~ (loopfilter);
```

La fonction `widthdelay` permet de créer un effet de stéréo à partir d'un signal mono en le partageant en deux canaux et en introduisant entre eux un léger retard.

```
widthdelay = _,delay(Pmax,W*P/2);
stereopanner(A) = (_,_) : *(1.0-A), *(A);
```

Pour terminer, les différents éléments de l'algorithmes sont assemblés entre-eux.

```
process = hgroup("EKS",
    vgroup("[1]Excitation",filtered_excitation) :
    vgroup("[2]Corde",stringloop) <:
    vgroup("[3]Sortie",widthdelay : stereopanner(A)));
```

La figure 1.6 à la page 18 contient un diagramme de cet algorithme.

Annexe D

Description du fichier `clarinette.dsp`

L'instrument FAUST `clarinette.dsp`[264] implémente une version légèrement simplifiée de l'instrument `clarinet.dsp` du FAUST-STK (cf. partie 1.4). Ce modèle physique est intéressant à étudier dans la mesure où sa forme assez générique donne une très bonne vue d'ensemble de l'utilisation de la technique des guides d'ondes pour la modélisation d'instruments à vent.

La bibliothèque `instrument.lib`[265] du FAUST-STK contient un ensemble de fonctions utiles à l'implémentation de modèles physiques par guides d'ondes (filtres, enveloppes, etc.).

```
import("music.lib");
import("instrument.lib");
```

Dans un soucis de gain d'espace, la déclaration de l'interface utilisateur de cet instrument n'est pas donnée ici. Il est toutefois nécessaire d'indiquer la fonction de chaque variable utilisée afin de comprendre la suite du code :

- `freq` : fréquence du son produit (compatible `faust2pd`) ;
- `gain` : amplitude du son produit (compatible `faust2pd`) ;
- `gate` : variable binaire pour les note-in et les note-off (compatible `faust2pd`) ;
- `reedStiffness` : dureté de l'anche ;
- `noiseGain` : amplitude du bruit du souffle ;
- `pressure` : pression du souffle ;
- `vibratoFreq` : fréquence du vibrato ;
- `vibratoGain` : amplitude du vibrato ;

264. Fichier disponible dans le dossier `/clarinette/` du CD.
265. Fichier disponible dans le dossier `/util/` du CD.

– `vibratoAttack` : durée de l'attaque du vibrato en secondes (s) ;
– `vibratoRelease` : durée de la chute du vibrato (s) ;
– `envelopeAttack` : durée de l'attaque de l'amplitude (s) ;
– `envelopeDecay` : durée du relâchement de l'amplitude (s) ;
– `envelopeRelease` : durée de la chute de l'amplitude (s).

La fonction `reed`, qui est décrite dans l'annexe R à la page 175, génère un signal non-linéaire semblable à celui produit par les interactions entre le son d'une anche en vibration et les ondes de réflexions renvoyées par la fin du tube acoustique de l'instrument. Le paramètre `reedStiffness` a un effet direct sur la pente de la courbe générée par la fonction implémentée dans `reed`.

```
reedTableOffset = 0.7;
reedTableSlope = -0.44 + (0.26*reedStiffness);
reedTable = reed(reedTableOffset,reedTableSlope);
```

La taille de la ligne de retards est calculée en fonction de la fréquence de la note jouée et de la fréquence d'échantillonnage. Une ligne de retards avec interpolation est utilisée afin de palier aux problèmes de justesse pour les notes aiguës (cf. partie 1.3).

```
delayLength = SR/freq*0.5 - 2;
delayLine = fdelay(4096,delayLength);
```

Un filtre passe-tout (déclaré dans la bibliothèque `instrument.lib`) est utilisé pour simuler l'effet du pavillon de la clarinette sur les ondes réfléchies.

```
filter = oneZero0(0.5,0.5);
```

La pression du souffle alimentant le bec de la clarinette est contrôlée par une enveloppe de type $ADSR$[266]. Sa valeur est légèrement modulée par un signal sinusoidal afin de créer un effet de vibrato d'amplitude. Enfin, du bruit blanc est ajouté dans le but de reproduire le son du souffle de l'instrumentiste.

```
envelope = adsr(envelopeAttack,envelopeDecay,100,envelopeRelease,
   gate)*pressure*0.9;
vibrato = osc(vibratoFreq)*vibratoGain*envVibrato(0.1*2*
   vibratoAttack,0.9*2*vibratoAttack,100,vibratoRelease,gate);
breath = envelope + envelope*noise*noiseGain;
breathPressure = breath + breath*vibrato;
```

Les différents éléments de l'algorithme sont assemblés. Le corps de la clarinette est composé d'un seul guide d'ondes à une dimension. Les ondes réfléchies à son extrémité sont

266. Attack – Decay – Sustain – Release : Attaque – Relâchement – Maintien – Chute

traitées avec le filtre passe-tout et leur phase est inversée avec une légère diminution de l'amplitude du signal en les multipliants par -0.95. L'excitation est produite par la fonction non-linéaire `reedTable` qui est contrôlée par la variable `breathPressure` additionnée aux ondes réfléchies par le pavillon.

```
process = (_,(breathPressure <: _,_) : (filter*-0.95 - _ <: *(
   reedTable)) + _) ~ delayLine : *(gain);
```

Le diagramme de cette dernière opération peut être visualisé dans la figure 1.10 à la page 24.

Le fichier `take5-clarinette.wav`[267] qui a été produit à l'aide du patch PureData `jouer_clarinette.pd`[268] contient un exemple sonore dans lequel le fichier `take5.mid`[269] est joué par le modèle de clarinette.

267. Fichier disponible sur le CD dans le dossier `/clarinette/`.
268. *Ibid.*
269. Fichier disponible sur le CD dans le dossier `/util/`. Plus d'informations sur ce fichier sont données dans l'annexe A.

Annexe E

Description du fichier `blowHole.dsp`

L'instrument FAUST `blowHole.dsp`[270] implémente un modèle avancé de clarinette (cf. partie 3.4.2) basé sur celui présent dans le SYNTHESIS TOOLKIT. Il intègre deux modèles de clefs utilisés respectivement pour modéliser la clef d'octave de l'instrument et une clef pour changer la hauteur des sons produits. Il est important de préciser que celles-ci sont intégrées au modèle à titre purement expérimental et ne peuvent pas dans le cas présent être utilisées en pratique pour jouer de l'instrument. Ainsi, la fréquence du son produit dans le fichier d'exemple `voi-che-blowHole.wav`[271] est contrôlée en modifiant la longueur du guide d'ondes principal du modèle, comme cela est le cas dans tous les autres instruments du FAUST-STK.

Dans un premier temps, il est important de s'intéresser à l'interface utilisateur. En plus des paramètres communs à l'ensemble des instruments à vent du FAUST-STK : fréquence, amplitude, note-on/off, etc. (cf. annexe D) et des paramètres relatifs à l'utilisation de la fonction `nonLinearModulator` (cf. annexe Q), `blowHole.dsp` donne la possibilité de contrôler les paramètres physiques suivants :

– `reedStiffness` : la dureté de l'anche ;
– `toneHoleOpenness` : le coefficient d'ouverture de la touche ;
– `ventOpenness` : le coefficient d'ouverture de la clé d'octave ;
– `pressure` : la pression de l'air au niveau de l'embouchure.

La déclaration du filtre passe-tout passif non-linéaire de second ordre n'est pas donnée ici dans la mesure où celle-ci fait déjà l'objet d'une description dans l'annexe K.

La fonction `reed` de la bibliothèque `instrument.lib` dont le fonctionnement est décrit

270. Fichier disponible dans le dossier `/clarinette2/` du CD.
271. *Ibid.*

dans l'annexe R à la page 175 génère un signal non-linéaire semblable à celui produit par les interactions entre le son d'une anche en vibration et les ondes de réflexions renvoyées par la fin du tube acoustique de l'instrument. Le paramètre `reedStiffness` a un effet direct sur la pente de la courbe générée par la fonction implémentée dans `reed`. Elle est aussi utilisée avec l'instrument `clarinette.dsp` décrit l'annexe D.

```
reedTableOffset = 0.7;
reedTableSlope = -0.44 + (0.26*reedStiffness);
reedTable = reed(reedTableOffset,reedTableSlope);
```

Le coefficient de dispersion au niveau du trou permettant de définir la hauteur de la note jouée est défini. La variable `rth` désigne le rayon du trou et `rb` correspond au rayon du tube du corps de l'instrument.

```
rb = 0.0075;
rth = 0.003;
scattering = pow(rth,2)*-1 / (pow(rth,2) + 2*pow(rb,2));
```

Le filtre récursif à un pôle qui constitue un des éléments du modèle de la clef d'octave est implémenté. `r_rh` est le rayon du trou dans le corps de l'instrument au niveau de la clef d'octave et `teVent` est la distance entre le trou et le tampon lorsque ce dernier est en position totalement levée. Le paramètre `ventOpenness` déclaré dans l'interface utilisateur permet de contrôler le gain du signal envoyé au filtre. Les calculs effectués ici ne sont pas commentés puisque cela l'est déjà fait dans l'implémentation originale de cet instrument dans le Synthesis ToolKit.

```
r_rh = 0.0015;
teVent = 1.4*r_rh;
xi = 0 ;
zeta = 347.23 + 2*PI*pow(rb,2)*xi/1.1769;
psi = 2*PI*pow(rb,2)*teVent/(PI*pow(r_rh,2));
rhCoeff = (zeta - 2*SR*psi)/(zeta + 2*SR*psi);
rhGain = -347.23/(zeta + 2*SR*psi);
ventFilterGain = rhGain*ventOpenness;

ventFilter = *(ventFilterGain) : poleZero(1,1,rhCoeff);
```

Le filtre utilisé dans le modèle de clef permettant de déterminer la hauteur du son produit est implémenté. Tout comme dans le cas précédent, il s'agit d'un filtre récursif à un pôle. La distance entre le trou et la clef est définie par la variable `teHole` dont la valeur est calculée en fonction de `rth`, le rayon du trou.

```
teHole = 1.4*rth;
coeff = (teHole*2*SR - 347.23)/(teHole*2*SR + 347.23);
```

```
scaledCoeff = (toneHoleOpenness*(coeff - 0.9995)) + 0.9995;

toneHoleFilter = *(1) : poleZero(b0,-1,a1)
with{
  b0 = scaledCoeff;
  a1 = -scaledCoeff;
};
```

Le filtre de réflexion est un filtre passe-bas dont le coefficient à une valeur constante.

```
reflexionFilter = oneZero0(0.5,0.5)*-0.95;
```

Le modèle utilise trois lignes de retards dont deux possèdent une valeur fixe. Ces dernières sont utilisées pour l'implémentation des guides d'ondes des deux clefs (octave et note jouée) permettant de changer la hauteur de la note jouée. `delay1` est la ligne de retards du guide d'ondes principal utilisé en pratique pour contrôler la fréquence de la note.
Il est intéressant de noter que la longueur de cette dernière est ajustée en fonction de l'ordre du filtre passe-tout non-linéaire et de son coefficient de non-linéarité. Si cette étape n'est pas implémentée, la fréquence des notes produites ne correspondrait pas à celle donnée par l'utilisateur (cf. partie 3.4.2). Il est primordial dans le cas présent d'utiliser une ligne de retards avec interpolation pour éviter tout problème de justesse.

```
delay0Length = 5*SR/22050;
delay2Length = 4*SR/22050;
delay1Length = (SR/freq*0.5 - 3.5) - (delay0Length + delay2Length)
   - (nlfOrder*nonLinearity)*(typeModulation < 2);

delay0 = fdelay(4096,delay0Length);
delay1 = fdelay(4096,delay1Length);
delay2 = fdelay(4096,delay2Length);
```

La fonction `stereoizer` dont le fonctionnement est décrit dans l'annexe R à la page 175 permet d'appliquer un effet de stéréo sur un signal mono.

```
stereo = stereoizer(SR/freq);
```

Le signal d'excitation introduit dans le corps du modèle est très similaire à celui de l'instrument `clarinette.dsp` et a par conséquent déjà fait l'objet d'une description dans l'annexe D. Son implémentation n'est donc pas à nouveau commentée ici.

```
envelope = (0.55 + pressure*0.3)*asr(pressure*envelopeAttack,100,
   pressure*envelopeRelease,gate);
vibratoEnvelope = envVibrato(0.1*2*vibratoAttack,0.9*2*
   vibratoAttack,100,vibratoRelease,gate);
vibrato = vibratoGain*osc(vibratoFreq)*vibratoEnvelope;
breath = envelope + envelope*noiseGain*noise;
breathPressure = breath + (breath*vibrato);
```

Le guide d'ondes implémentant la clef d'octave et les différentes jonctions de dispersion le connectant au reste de l'instrument est mis en place. Le bec de la clarinette (`reedTable`) est également intégré à l'algorithme. Le diagramme de cette zone du modèle peut être visualisé dans le figure 3.16 à la page 87.

```
twoPortJunction(portB) = (pressureDiff : ((_ <: breathPressure + *(
   reedTable)) <: (+(portB) : ventFilter <: +(portB),_),_))~delay0
   : inverter : + ,_
with{
  pressureDiff = -(breathPressure);
  inverter(a,b,c) = b,c,a;
};
```

Le guide d'ondes implémentant le modèle de clef permettant de changer la hauteur du son produit ainsi que ses différentes jonctions de dispersion le reliant au reste de l'instrument est mis en place. Notons que cette partie de l'algorithme contient également le filtre de réflexion.

```
threePortJunction(twoPortOutput) =  (_ <: junctionScattering(
   twoPortOutput),_ : +(twoPortOutput), + : reflexionFilter,_)~
   delay2 : !,_
with{
  toneHole(temp,portA2,portB2) = (portA2 + portB2-_ + temp :
     toneHoleFilter)~_;
  junctionScattering(portA2,portB2) = (((portA2+portB2-2*_)*
     scattering) <: toneHole(_,portA2,portB2),_,_)~_ : !,_,_;
};
```

Le guide d'ondes principal est mis en place et les différents éléments du modèle sont assemblés.

```
process = (twoPortJunction : threePortJunction,_) ~ (delay1 : NLFM)
    : !,*(gain)*1.5 : stereo : instrReverb;
```

La figure 3.16 de la page 87 présente un diagramme de cet instrument.

Annexe F

Description du fichier `saxophony.dsp`

L'instrument FAUST `saxophony.dsp`[272] implémente un modèle physique d'instrument à vent à anche simple pouvant être utilisé pour produire une variété de types de sons allant de la clarinette au saxophone en passant par le violon (cf. partie 3.4.3). Son corps est basé sur un guide d'ondes bi-dimensionnel et le signal d'excitation utilisé est le même que celui employé pour les instruments `clarinette.dsp` (cf. annexe D) et `blowHole.dsp` (cf. annexe E).

En plus des paramètres communs à l'ensemble des instruments à vent du FAUST-STK : fréquence, amplitude, note on/off, etc. (cf. annexe D) et à ceux impliqués par l'utilisation de la fonction `nonLinearModulator` (cf. annexe Q), `saxophony.dsp` donne la possibilité de contrôler les paramètres physiques suivants :

- `pressure` : la pression de l'air au niveau du bec et de l'anche de l'instrument ;
- `reedStiffness` : le coefficient de rigidité de l'anche ;
- `blowPosition` : la position du bec et donc du point d'excitation sur le corps de l'instrument.

La déclaration du filtre passe-tout passif non-linéaire n'est pas redonnée puisqu'il s'agit de la même que celle de `flutestk.dsp` (cf. annexe K).

Une fois ces deux premières étapes mis en place, la fonction `reed`, décrite dans l'annexe R à la page 175 est appelée. Elle génère le signal d'excitation (anche en vibration) et modélise les interactions entre ce dernier et les ondes de réflexion qui donnent lieu à des non-linéarités dans le son produit. Le paramètre `reedStiffness` a un effet direct sur la pente de l'onde d'excitation produite.

```
reedTableOffset = 0.7;
```

272. Fichier disponible sur le CD dans le dossier `/saxophone/`

```
reedTableSlope = 0.1 + (0.4*reedStiffness);

reedTable = reed(reedTableOffset,reedTableSlope);
```

Les deux lignes de retards avec interpolation du guide d'ondes sont déclarées. Le paramètre `blowPosition` permet de réduire et d'augmenter de manière équilibrée la longueur de chaque ligne de retards afin de déplacer le point d'excitation le long du guide d'ondes. Ainsi, si `blowPosition` est égal à zéro, le point d'excitation se trouvera à une extrémité du guide d'ondes. S'il est égal à un, le signal d'excitation sera introduit à l'autre extrémité. Enfin, si `blowPosition` est égal à 0.5, le point d'excitation sera placé au centre.

```
fdel1 = (1-blowPosition) * (SR/freq - 3);
fdel2 = (SR/freq - 3)*blowPosition +1 ;

delay1 = fdelay(4096,fdel1);
delay2 = fdelay(4096,fdel2);
```

Le signal d'excitation introduit dans le corps du modèle est très similaire à celui de l'instrument `clarinette.dsp` et a par conséquent déjà fait l'objet d'une description dans l'annexe D. Son implémentation n'est donc pas commentée à nouveau ici.

```
envelope = (0.55+pressure*0.3)*asr(pressure*envelopeAttack,100,
   pressure*envelopeRelease,gate);
breath = envelope + envelope*noiseGain*noise;
vibrato = vibratoGain*envVibrato(vibratoBegin,vibratoAttack,100,
   vibratoRelease,gate)*osc(vibratoFreq);
breathPressure = breath + breath*vibratoGain*osc(vibratoFreq);
```

Le filtre de réflexion est un simple filtre passe-bas du premier ordre.

```
bodyFilter = *(gain) : oneZero1(b0,b1)
with{
  gain = -0.95;
  b0 = 0.5;
  b1 = 0.5;
};
```

La fonction `instrumentBody` implémente la partie inférieure du guide d'ondes dans laquelle se trouve le point d'excitation et une des deux lignes de retards.

```
instrumentBody(delay1FeedBack,breathP) = delay1FeedBack <: -(delay2
   ) <: ((breathP - _ <: breathP - _*reedTable) - delay1FeedBack),_
   ;
```

La partie supérieure du guide d'ondes est construite et est combinée avec la partie inférieure. Le filtre de réflexion est placé dans la boucle.

```
process = (bodyFilter,breathPressure : instrumentBody) ~ (delay1 :
   NLFM) : !,*(gain) : stereo : instrReverb;
```

Un diagramme détaillé du modèle de cet instrument peut être visualisé sur la figure 3.17 à la page 90.

Annexe G

Description du fichier `sitar.dsp`

L'instrument FAUST `sitar.dsp`[273] implémente un algorithme de KARPLUS-STRONG dont la longueur de la ligne de retards est modulée légèrement de façon aléatoire dans le but de simuler le comportement non-linéaire de certains instruments à cordes tel que le sitar ou le tampura (cf. partie 2.1.2).

Dans un premier temps, l'interface utilisateur est créée :

```
freq = nentry("h:Basic Parameters/freq [1][unit:Hz] [tooltip:Tone
   frequency]",440,20,20000,1);
gain = nentry("h:Basic Parameters/gain [1][tooltip:Gain (value
   between 0 and 1)]",1,0,1,0.01);
gate = button("h:Basic Parameters/gate [1][tooltip:noteOn = 1,
   noteOff = 0]");

resonance = hslider("v:Physical Parameters/Resonance
[2][tooltip:A value between 0 and 1]",0.7,0,1,0.01)*0.1;
nonLinearity = hslider("v:Physical Parameters/nonLinearity
[2][tooltip:A value between 0 and 1]",0,0,1,0.01);
```

L'attaque et le relâchement d'une enveloppe *ADSR*[274] sont utilisés pour créer une courte impulsion dans le but d'exciter la corde de l'instrument.

```
envelope = adsr(0.001,0.04,0,0.5,gate);
```

La longueur de la ligne de retards dont la longueur est calculée en fonction de la fréquence de la note à jouer (cf. partie 1.2) est modulée de façon aléatoire à l'aide d'un générateur de bruit blanc (fonction `noise`). La fonction `smooth` permet d'effectuer une interpolation linéaire entre les différentes valeurs de la longueur de la ligne de retards afin d'éviter toute discontinuité. Le diagramme de cette opération peut être visualisé sur la figure 2.5 à la

273. Fichier disponible dans le dossier `/sitar/` du CD.
274. Attack – Decay – Sustain – Release : Attaque – Relâchement – Maintien – Chute.

page 41.

```
targetDelay = SR/freq;
delayLength = targetDelay*((1+(0.5*noise*nonLinearity)) : smooth
   (0.9992));
delayLine = delay(4096,delayLength);
```

La variable `loopGain` permet de contrôler le temps de résonance de la corde en fonction de la fréquence de la note jouée et du paramètre `resonance`.

```
loopGain = 0.895 + resonance + (freq*0.0000005);
```

Le signal de retour de la boucle de la corde est filtré par un filtre de type passe-bas.

```
filter = oneZero1(b0,b1)
    with{
        zero = 0.01; b0 = 1/(1 + zero); b1 = -zero*b0;
    };
```

Les différents éléments de l'algorithme sont assemblés.

```
process = (*(gain) : filter + (envelope*noise*amGain))~delayLine;
```

Annexe H

Description du fichier `harpsi.dsp`

L'instrument FAUST `harpsi.dsp`[275] implémente un modèle physique de clavecin par guides d'ondes commutées (cf. partie 3.2.1) basé sur celui du SYNTHESIS TOOLKIT. Le filtre passe-tout passif non-linéaire d'ordre arbitraire décrit dans la partie 2.2 est utilisé pour rendre plus naturel le son produit.

Dans un premier temps, une version du sixième ordre de ce filtre est créée grâce à la fonction `nonLinearModulator` dont le fonctionnement est détaillé dans l'annexe Q.

```
nlfOrder = 6;
NLFM =  nonLinearModulator((nonLinearity : smooth(0.999)),1,freq,
   typeModulation,(frequencyMod : smooth(0.999)),nlfOrder);
```

La déclaration de l'interface utilisateur de `harpsi.dsp` n'est pas détaillée ici dans la mesure ou très peu de paramètres physiques peuvent être contrôlés par l'utilisateur pour ce modèle.

Les coefficients du filtre de réflexion des cordes, la durée de la chute de l'enveloppe d'amplitude de l'excitation et la durée de la vibration de la corde ont des valeurs qui varient en fonction de la fréquence de la note jouée (cf. partie 3.2.1). Ces valeurs sont contenues dans des fonctions C++ déclarées dans le fichier `harpsichord.h`[276] qui sont invoquées en utilisant le système de fonction étrangère de FAUST. L'argument pris par ces fonctions étant le numéro MIDI de la note jouée, il est nécessaire d'implémenter dans un premier temps une fonction pour convertir une fréquence exprimée en Hertz en un numéro de note MIDI.

```
freqToNoteNumber = (log-log(440))/log(2)*12+69+0.5 : int;
freqn = freq : freqToNoteNumber;
```

275. Fichier disponible dans le dossier `/harpsi/` du CD.
276. *Ibid.*

L'excitation de la corde est modélisée par une impulsion complexe créée avec un générateur de bruit blanc sur lequel une enveloppe d'amplitude exponentielle est appliquée avec la fonction `dryTapAmpT60` de la bibliothèque `instrument.lib`. Cette dernière génère un signal dont la valeur décroît sur une durée définie en fonction de la fréquence de la note jouée (les valeurs sont extraites de `harpsichord.h`) pour atteindre de manière asymptotique zéro.

```
soundBoard = dryTapAmp*noise
with{
  dryTapAmpT60 = ffunction(float getValueDryTapAmpT60(float), <
     harpsichord.h>,"");
  noteCutOffTime = freqn : dryTapAmpT60*gain;
  dryTapAmp = asympT60(0.15,0,noteCutOffTime,gate);
};
```

Le filtre de réflexion des cordes est un filtre biquadratique. Ces coefficients évoluent en fonction de la fréquence de la note jouée et sont aussi extraits du fichier `harpsichord.h`.

```
loopFilter = TF2(b0,b1,b2,a1,a2)
with{
  loopFilterb0 = ffunction(float getValueLoopFilterb0(float), <
     harpsichord.h>,"");
  loopFilterb1 = ffunction(float getValueLoopFilterb1(float), <
     harpsichord.h>,"");
  loopFilterb2 = ffunction(float getValueLoopFilterb2(float), <
     harpsichord.h>,"");
  loopFiltera1 = ffunction(float getValueLoopFiltera1(float), <
     harpsichord.h>,"");
  loopFiltera2 = ffunction(float getValueLoopFiltera2(float), <
     harpsichord.h>,"");
  b0 = loopFilterb0(freqn);
  b1 = loopFilterb1(freqn);
  b2 = loopFilterb2(freqn);
  a1 = loopFiltera1(freqn);
  a2 = loopFiltera2(freqn);
};
```

La longueur de la ligne de retards du guide d'ondes modélisant une corde du clavecin est calculée. De plus, la fonction `stereoizer` (cf. annexe R à la page 175) est invoquée.

```
delayLength = SR/freq;
stereo = stereoizer(delayLength);
```

La corde du clavecin est implémentée. Sa durée de vibration lorsque la touche est maintenue enfoncée est calculée en fonction de la fréquence de la note jouée. Un diagramme de cette corde peut être vu sur la figure 3.3 à la page 65.

```
stringLoopGainT = gate*0.9996 + (gate<1)*releaseLoopGain(freqn)*0.9
    : smooth(0.999)
with{
  releaseLoopGain = ffunction(float getValueReleaseLoopGain(float),
      <harpsichord.h>,"");
};
string = (*(stringLoopGainT)+_ : delay(4096,delayLength) :
   loopFilter)~NLFM;
```

Les différents éléments constitutifs de l'instrument sont assemblés.

```
process = soundBoard : string : stereo : instrReverb;
```

Annexe I

Description du fichier `piano.dsp`

L'instrument FAUST `piano.dsp`[277] implémente un modèle physique complexe monophonique de piano alliant la technique des guides d'ondes commutés pour les notes graves et médiums et la synthèse modale pour les notes aiguës. Il est basé sur l'algorithme de piano du programme SYNTHBUILDER. L'implémentation d'un instrument polyphonique basé sur ce modèle est possible avec certaines architectures de FAUST comme ceci l'est expliqué dans la partie 3.2.1.

En plus des paramètres classiques communs à l'ensemble des instruments du FAUST-STK compatibles avec *faust2pd* (fréquence, amplitude et note-on/off du son joué), le modèle ici présenté donne la possibilité à son utilisateur de régler la valeur des paramètres suivants :

- `brightnessFactor` : la brillance du son exprimée sous la forme d'une valeur entre zéro et un ;
- `detuningFactor` : facteur de « désaccordage » des cordes couplées pour une même note (valeur entre zéro et un) ;
- `stiffnessFactor` : facteur de rigidité des cordes (valeur entre zéro et un) ;
- `hammerHardness` : facteur de dureté des marteaux (valeur entre zéro et un).

Des variables globales sont mis en place. `DCB2_TURNOFF_KEYNUM` définit la plus haute MIDI jouable par l'instrument, `FIRST_HIGH_NOTE` donne le numéro de la note MIDI à partir de laquelle le modèle modal est utilisé au lieu de celui par guides d'ondes (cf. 3.2.2).

```
DCB2_TURNOFF_KEYNUM = 92;
FIRST_HIGH_NOTE = 88;
PEDAL_ENVELOPE_T60 = 7;
```

Des fonctions pour la conversion de paramètres sont mis en place. `dbinv` convertit une am-

277. Fichier disponible dans le dossier `/piano/` du CD.

plitude de type $0dbfs = 1$ en une amplitude de type dB_{FS}, `freqToNoteNumber` convertit une fréquence en Hz en un numéro de note MIDI et `cntSample` est un compteur d'échantillons remis à zéro à chaque évènement « note-on ».

```
dbinv(x) = pow(10,0.05*x);
freqToNoteNumber = (log-log(440))/log(2)*12+69+0.5 : int;
freqn = freq : freqToNoteNumber;
cntSample = *(gate)+1~_ : -(1);
```

L'excitation agrégée pour la synthèse commutée (cf. partie 3.2.1) est créée. Elle est semblable à celle utilisée dans le cas de `harpsi.dsp` (cf. annexe H) bien qu'elle soit beaucoup plus complexe. Elle consiste en l'addition de deux générateurs de bruit blanc sur lesquels une enveloppe d'amplitude exponentielle, différente pour chacun d'entre-eux est appliquée. Comme il s'agit de synthèse par guides d'ondes commutées, l'un des deux générateurs produit le son qui correspond au choc du marteau sur la corde puis à la résonance de la table d'harmonie qui en résulte. L'autre générateur modélise l'effet de la résonance des cordes sur la table d'harmonie lorsque la pédale de maintien est enfoncée. La durée des chutes de chacune des enveloppes d'amplitude est calculée en fonction de la fréquence de la note jouée à partir des informations contenues dans le fichier C++ `piano.h`[278]. Un diagramme résumant cette opération est visible sur la figure 3.5 à la page 67.

```
asympT60pedal(value,T60) = (*(factor) + constant)~_
with{
  attDur = hammerHardness*float(SR);
  target = value*((cntSample < attDur) & (gate > 0));
  factorAtt = exp (-1/(attDur));
  factorG = exp(-1/(2*float(SR)));
  factorT60 = exp(-7/(T60*float(SR)));
  factor = factorAtt*gate*(cntSample < attDur) + (cntSample >=
     attDur)*gate*factorG + ((gate-1)*-1)*factorT60;
  constant = (1 - factor)*target;
};

soundBoard = dryTapAmp*noise + pedalEnv*noise : *(0.5)
with{
  dryTapAmpT60 = ffunction(float getValueDryTapAmpT60(float), <
     piano.h>,"");
  sustainPedalLevel = ffunction(float getValueSustainPedalLevel(
     float), <piano.h>,"");
  pedalEnvCutOffTime = 1.4;
  noteCutOffTime = freqn : dryTapAmpT60*gain;
  pedalEnvValue = freqn : sustainPedalLevel*0.2;
  noteEnvValue = 0.15;
  dryTapAmp = asympT60(noteEnvValue,0,noteCutOffTime,gate);
```

278. *Ibid.*

```
  pedalEnv = asympT60pedal(pedalEnvValue,pedalEnvCutOffTime);
};
```

Une série de quatre filtres biquadratiques connectés en série est utilisée pour appliquer la réponse impulsionnelle de la table d'harmonie du piano sur le son produit lors de l'étape précédente. La réponse impulsionnelle ici utilisée est très simple et a été extraite du modèle de piano implémenté dans le programme SynthBuilder. Les valeurs de gains et des coefficients pour chaque filtre sont contenues dans le fichier C++ `piano.h` et varie en fonction de la fréquence de la note jouée.

```
calcHammer = onePole((1-hammerPole)*hammerGain,-hammerPole)
with{
  loudPole = ffunction(float getValueLoudPole(float), <piano.h>,"")
     ;
  softPole = ffunction(float getValuePoleValue(float), <piano.h>,""
     );
  loudGain = ffunction(float getValueLoudGain(float), <piano.h>,"")
     ;
  softGain = ffunction(float getValueSoftGain(float), <piano.h>,"")
     ;
  loudPoleValue = loudPole(freqn) + (brightnessFactor*-0.25) +
     0.02;
  softPoleValue = softPole(freqn);
  normalizedVelocityValue = 1;
  loudGainValue = loudGain(freqn);
  softGainValue = softGain(freqn);
  overallGain = 1;
  hammerPole = softPoleValue + (loudPoleValue - softPoleValue)*
     normalizedVelocityValue;
  hammerGain = overallGain*(softGainValue + (loudGainValue -
     softGainValue)*normalizedVelocityValue);
};

hammer = seq(i,4,calcHammer);
```

Les différents bloqueurs de composantes continues permettant de réguler le signal à différentes étapes de l'algorithme sont déclarés. Ces derniers concernent aussi bien la partie modal que la partie guides d'ondes du modèle. Le pôle du bloqueur de composante continue utilisé pour cette dernière varie en fonction de la fréquence de la note jouée.

```
DCBa1 = ffunction(float getValueDCBa1(float), <piano.h>,"");
dCBa1Value = freqn : DCBa1;
dCBb0Value = 1 - dCBa1Value;

dcBlock1 = poleZero((dCBb0Value*0.5),(dCBb0Value*-0.5),dCBa1Value);
dcBlock2a = oneZero1(0.5,-0.5);
dcBlock2b = onePole(dCBb0Value,dCBa1Value);
```

Afin de pallier aux problèmes de justesse liés à la quantification de la longueur de la ligne de retards des guides d'ondes pour les notes aiguës, un algorithme simple de synthèse modale est utilisé pour produire le son des notes dont la hauteur dépasse la note MIDI n° 88. Quatre filtres biquadratiques sont ainsi connectés en série et mis en résonance en fonction de la fréquence de la note jouée. Le système de filtrage par motif de FAUST est utilisé pour assigner les bonnes valeurs de coefficients et de gains à chacun d'entre-eux.

```
r1_1 = ffunction(float getValuer1_1db(float), <piano.h>,"");
r1_2 = ffunction(float getValuer1_2db(float), <piano.h>,"");
r2 = ffunction(float getValuer2db(float), <piano.h>,"");
r3 = ffunction(float getValuer3db(float), <piano.h>,"");
e = ffunction(float getValueSecondStageAmpRatio(float), <piano.h>,"
   ");
second_partial_factor = ffunction(float getValueSecondPartialFactor
   (float), <piano.h>,"");
third_partial_factor = ffunction(float getValueThirdPartialFactor(
   float), <piano.h>,"");
bq4_gEarBalled = ffunction(float getValueBq4_gEarBalled(float), <
   piano.h>,"");

r1_1Value = r1_1(freqn)/SR : dbinv;
r1_2Value = r1_2(freqn)/SR : dbinv;
r2Value = r2(freqn)/SR : dbinv;
r3Value = r3(freqn)/SR : dbinv;
eValue = e(freqn) : dbinv;
second_partial_factorValue = second_partial_factor(freqn);
third_partial_factorValue = third_partial_factor(freqn);

gainHighBq(0) = bq4_gEarBalled(freqn)/0.5;
gainHighBq(1) = bq4_gEarBalled(freqn)/0.5;
gainHighBq(2) = 1;
gainHighBq(3) = 1;

b0HighBq(0) = 1;
b0HighBq(1) = 1;
b0HighBq(2) = 1;
b0HighBq(3) = 1;

b1HighBq(0) = 0;
b1HighBq(1) = 0;
b1HighBq(2) = -2*(eValue*r1_1Value+(1-eValue)*r1_2Value)*cos(2*PI*
   freq/SR);
b1HighBq(3) = 0;

b2HighBq(0) = 0;
b2HighBq(1) = 0;
b2HighBq(2) = eValue*r1_1Value*r1_1Value+(1-eValue)*r1_2Value*
   r1_2Value;
b2HighBq(3) = 0;
```

```
a1HighBq(0) = -2*r3Value*cos(2*PI*freq*third_partial_factorValue/SR
    );
a1HighBq(1) = -2*r2Value*cos(2*PI*freq*second_partial_factorValue/
    SR);
a1HighBq(2) = -2*r1_1Value*cos(2*PI*freq/SR);
a1HighBq(3) = -2*r1_2Value*cos(2*PI*freq/SR);

a2HighBq(0) = r3Value*r3Value;
a2HighBq(1) = r2Value*r2Value;
a2HighBq(2) = r1_1Value*r1_1Value;
a2HighBq(3) = r1_2Value*r1_2Value;

highBqs = seq(i,4,*(gainHighBq(i)) : TF2(b0HighBq(i),b1HighBq(i),
    b2HighBq(i),a1HighBq(i),a2HighBq(i)));
```

Un filtre passe-haut simple est utilisé pour supprimer les composantes graves du son de l'excitation avant que la réponse impulsionnelle de la table d'harmonie du piano ne soit appliquée sur celle-ci dans le cas des notes aiguës (cf. figure 3.4 page 66).

```
hiPass = oneZero1(b0,b1)
with{
  b0 = -0.5;
  b1 = -0.5;
};
```

Le filtre suivant permet de modéliser l'effet de la position du point d'impact du marteau sur la corde en fonction de la fréquence de la note jouée.

```
eq = _*filterGain : TF2(b0,b1,b2,a1,a2)
with{
  strikePosition = ffunction(float getValueStrikePosition(float), <
     piano.h>,"");
  bandwidthFactors = ffunction(float getValueEQBandWidthFactor(
     float), <piano.h>,"");
  eq_gain = ffunction(float getValueEQGain(float), <piano.h>,"");
  eq_tuning = freq/strikePosition(freqn);
  eq_bandwidth = bandwidthFactors(freqn)*freq;
  filterGain = eq_gain(freqn);
  a2 = (eq_bandwidth / SR) * (eq_bandwidth / SR);
  a1 = -2*(eq_bandwidth / SR)*cos(2*PI*eq_tuning/SR);
  b0 = 0.5 - 0.5 * a2;
  b1 = 0;
  b2 = -b0;
};
```

Comme cela a été expliqué dans la partie 3.2.2, deux cordes placées côte à côte sont utilisées pour générer la même note. Ce couple de corde est modélisé dans les étapes suivantes.

Dans un premier les différents paramètres dépendant de la fréquence de la note jouée sont

extraits des fonctions C++ contenues dans `piano.h`.

```
singleStringDecRate = ffunction(float getValueSingleStringDecayRate
   (float), <piano.h>,"");
singleStringZero = ffunction(float getValueSingleStringZero(float),
    <piano.h>,"");
singleStringPole = ffunction(float getValueSingleStringPole(float),
    <piano.h>,"");
stiffnessCoefficient = ffunction(float getValueStiffnessCoefficient
   (float), <piano.h>,"");
```

Les paramètres du filtre de couplage sont calculés.

```
g = pow(10,((singleStringDecRate(freqn)/freq)/20));
b = singleStringZero(freqn);
a = singleStringPole(freqn);
tempd = 3*(1-b)-g*(1-a);
b0Coupling = 2*(g*(1-a)-(1-b))/tempd;
b1Coupling = 2*(a*(1-b)-g*(1-a)*b)/tempd;
a1Coupling = (g*(1-a)*b - 3*a*(1-b))/tempd;
```

Le filtre modélisant la rigidité de la corde est implémenté. La valeur de ses paramètres est influencée par le coefficient de rigidité contrôlable par l'utilisateur.

```
stiffness = stiffnessFactor*stiffnessCoefficient(freqn);
stiffnessAP = poleZero(b0s,b1s,a1s)
with{
  b0s = stiffness;
  b1s = 1;
  a1s = stiffness;
};
```

Le modèle générique de ligne de retards qui sera utilisé dans le guide d'ondes de chaque corde est déclaré.

```
delayG(frequency,stiffnessCoefficient) = fdelay(4096,delayLength)
with{
  allPassPhase(a1,WT) = atan2((a1*a1-1.0)*sin(WT),(2.0*a1+(a1*a1
     +1.0)*cos(WT)));
  poleZeroPhase(b0,b1,a1,WT) = atan2(-b1*sin(WT)*(1 + a1*cos(WT)) +
      a1*sin(WT)*(b0 + b1*cos(WT)), (b0 + b1*cos(WT))*(1 + a1*cos(
     WT)) + b1*sin(WT)*a1*sin(WT));
  wT = frequency*2*PI/SR;
  delayLength = (2*PI + 3*allPassPhase(stiffnessCoefficient, wT) +
     poleZeroPhase((1+2*b0Coupling), a1Coupling + 2*b1Coupling,
     a1Coupling, wT)) / wT;
};
```

L'algorithme pour les cordes couplées est implémenté.

```
coupledStrings = (parallelStrings <: (_,(_+_ <: _,_),_ : _,_,(_ :
   couplingFilter),_ : adder))~(_,_) : !,!,_
```

```
with{
  releaseLoopGain = ffunction(float getValueReleaseLoopGain(float),
      <piano.h>,"");
  hz = ffunction(float getValueDetuningHz(float), <piano.h>,"");
  coupledStringLoopGain = gate*0.9996 + ((gate-1)*-1)*
     releaseLoopGain(freqn)*0.9 : smooth(0.999);
  couplingFilter = poleZero(b0Coupling,b1Coupling,a1Coupling);
  hzValue = hz(freqn);
  freq1 = freq + 0.5*hzValue*detuningFactor;
  freq2 = freq - 0.5*hzValue*detuningFactor;
  delay1 = delayG(freq1,stiffness);
  delay2 = delayG(freq2,stiffness
  parallelStrings(x,y) = _ <: (+(x)*coupledStringLoopGain : seq(i
     ,3,stiffnessAP) : delay1), (_+y*coupledStringLoopGain : seq(i
     ,3,stiffnessAP) : delay2);
  adder(w,x,y,z) = (y <: +(w),+(z)),x ;
};
```

Les différents éléments du modèle sont assemblés et le système permettant de sélectionner l'algorithme utilisé en fonction de la fréquence de la note jouée est implémenté.

```
conditionLowNote = freqn < FIRST_HIGH_NOTE;
conditionHighNote = freqn >= FIRST_HIGH_NOTE;

process = soundBoard <: (*(conditionLowNote)*6 : hammer : dcBlock1
   : coupledStrings <: +(eq)),(*(conditionHighNote) : hiPass :
   dcBlock1 : hammer : dcBlock2a : highBqs : dcBlock2b) :> + ;
```

Annexe J

Description du fichier `bass.dsp`

L'instrument FAUST `bass.dsp`[279] implémente un modèle physique de guitare basse par guides d'ondes basé sur celui disponible dans le programme SYNTHBUILDER bien que quelques modifications lui aient été apportées (cf. partie 3.2.3). Il utilise par exemple dans le cas présent le filtre passe-tout passif non-linéaire décrit dans la partie 2.2 qui permet d'introduire des non-linéarités dans le son produit.

Tout d'abord, les éléments de l'interface graphique sont déclarés. En plus des paramètres MIDI commun à l'ensemble des instruments du FAUST-STK (fréquence, amplitude, note-on/note-off), l'interface utilisateur de cet instrument permet de contrôler la durée de l'excitation avec le paramètre `touchLength`. Les éléments de contrôle habituels pour le filtre passe-tout passif non-linéaire sont également présents. La fonction `nonLinearModulator` (cf. annexe Q) permet de déclarer un filtre du sixième ordre de ce type.

```
nlfOrder = 6;
NLFM =  nonLinearModulator((nonLinearity : smooth(0.999)),1,freq,
   typeModulation,(frequencyMod : smooth(0.999)),nlfOrder);
```

La longueur de la ligne de retards est calculée en fonction de la note jouée et la fréquence d'échantillonnage. Le fonction `stereoizer` (cf. annexe R à la page 175) est utilisée pour créer un effet de stéréo sur le son produit par l'instrument.

```
delayLength = float(SR)/freq;
stereo = stereoizer(delayLength);
```

L'excitation est modélisée par un générateur de bruit blanc sur lequel une enveloppe d'amplitude exponentielle est appliquée. Son signal est filtré par deux types de filtres récursifs du premier ordre. Le premier a un pôle variable dont la valeur « glisse » de

279. Fichier disponible sur le CD dans le dossier `/basse/`.

manière exponentielle entre deux valeurs. Le second a deux pôles de valeurs fixes et est utilisé deux fois de manière séquentielle.

```
excitation = asympT60(-0.5,-0.985,0.02,gate),noise*asympT60(gain,0,
   touchLength,gate) : onePoleSwep : excitationFilter :
   excitationFilter
with{
  excitationFilter = onePole(0.035,-0.965);
};
```

Un filtre passe bande modélise grossièrement la réponse impulsionnelle du corps de la guitare basse.

```
bodyFilter = bandPass(108,0.997);
```

Les coefficients du filtre de réflexion varient en fonction de la fréquence de la note jouée et son donnés par comparaison avec des valeurs contenues dans différents tableaux C++ déclarés dans `bass.h`[280] en utilisant la tecthnique décrite dans la partie 3.2.1.

```
reflexionFilter = poleZero(b0,b1,a1)
with{
  loopFilterb0 = ffunction(float getValueBassLoopFilterb0(float), <
     bass.h>,"");
  loopFilterb1 = ffunction(float getValueBassLoopFilterb1(float), <
     bass.h>,"");
  loopFiltera1 = ffunction(float getValueBassLoopFiltera1(float), <
     bass.h>,"");
  freqToNoteNumber = (log - log(440))/log(2)*12 + 69 + 0.5 : int;
  freqn = freq : freqToNoteNumber;
  b0 = loopFilterb0(freqn);
  b1 = loopFilterb1(freqn);
  a1 = loopFiltera1(freqn);
};
```

La ligne de retards du guide d'ondes est déclarée. Le coefficient multiplicateur permettant de contrôler la durée de résonance est calculé et les différents éléments de l'algorithme sont assemblés. Le diagramme de ce dernier peut être visualisé sur la figure 3.8 à la page 71.

```
delayLine = asympT60(0,delayLength,0.01,gate),_ : fdelay(4096);
resonanceGain = gate + (gate < 1 <: *(asympT60(1,0.9,0.05)));
process = excitation : (+)~(delayLine : NLFM : reflexionFilter*
   resonanceGain) <: bodyFilter*1.5 + *(0.5) : *(4) : stereo :
   instrReverb;
```

280. *Ibid.*

Annexe K

Description du fichier `flutestk.dsp`

L'instrument FAUST `flutestk.dsp`[281] implémente un modèle physique très simple de flute traversière avec la technique des guides d'ondes numériques. Il est basé sur le modèle du SYNTHESIS TOOLKIT et son fonctionnement est expliqué dans la partie 3.4.1.
Dans un soucis de gain d'espace, la déclaration de l'interface utilisateur de cet instrument n'est pas donnée ici. De plus, la fonction des paramètres `freq`, `gain`, `gate`, `noiseGain`, `pressure`, `vibratoFreq`, `vibratoGain`, `vibratoBegin`, `vibratoAttack`, `vibratoRelease`, `envelopeAttack`, `envelopeDecay` et `envelopeRelease` qui sont ici utilisés est la même que celle de `clarinette.dsp` dont le fonctionnement est décrit dans l'annexe D. Le seul paramètre unique à cet instrument est `embouchureAjust` qui permet de contrôler la position de la bouche au niveau de l'embouchure.
Le filtre passe-tout passif non-linéaire décrit dans la partie 2.2 est utilisé dans le guide d'ondes pour introduire des non-linéarités dans le son produit. La fonction `nonLinearModulator` qui est décrite dans l'annexe Q est utilisée pour implémenter ce filtre. Une enveloppe de type *ASR*[282] est utilisée pour contrôler le taux de non-linéarités introduites dans le son. L'ordre du filtre déclaré est de six.

```
nlfOrder = 6;
envelopeMod = asr(nonLinAttack,100,envelopeRelease,gate);
NLFM =  nonLinearModulator((nonLinearity : smooth(0.999)),
   envelopeMod,freq,typeModulation,(frequencyMod : smooth(0.999)),
   nlfOrder);
```

Les différentes variables utilisées dans l'algorithme du modèle sont déclarées. `jetReflexion` et `endReflexion` sont les taux de réflexion du son à chaque extrémité du guide

281. Fichier disponible dans le dossier `/fluteSTK/` du CD.
282. Attack – Sustain – Release : Attaque – Maintien – Chute.

d'ondes. `jetRatio` est calculé en fonction de la position de la bouche au niveau de l'embouchure et a un effet direct sur la longueur d'une des deux lignes de retards utilisées dans le guide d'ondes modélisant le corps de l'instrument : `jetDelayLength`. La longueur de l'autre ligne de retards est indiquée par la valeur contenue dans `boreDelayLength`.
Il est intéressant de noter que en conséquence de l'utilisation d'un guide d'ondes bi-dimensionnel, la fréquence utilisée pour calculer la longueur de chaque ligne de retards est multipliée par deux (cf. partie 1.2).

```
jetReflexion = 0.5;
jetRatio = 1+(0.5-embouchureAjust);
endReflexion = 0.5;nonLinearModulator

jetDelayLength = (SR/(freq*2)-2)*jetRatio;
boreDelayLength = SR/(freq*2)-2;
filterPole = 0.7 - (0.1*22050/SR);
```

Le filtre de réflexion du guide d'ondes est composé d'un filtre passe-bas à un pôle connecté à un simple bloqueur de composante continue. La fonction `stereoizer` permet d'appliquer un effet de stéréo sur le signal d'un instrument mono (cf. annexe R à la page 175).

```
onePoleFilter = _*gain : onePole(b0,a1)
with{
  gain = -1;
  pole = 0.7 - (0.1*22050/SR);
  b0 = 1 - pole;
  a1 = -pole;
};
reflexionFilters = onePoleFilter : dcblocker;
stereo = stereoizer(SR/freq);
```

Le signal d'excitation est constitué d'une enveloppe de type *ADSR*[283] additionnée au signal d'un générateur de bruit blanc. L'amplitude globale de ces deux éléments est modulée avec une onde sinusoïdale de basse fréquence afin d'obtenir un effet de vibrato. L'enveloppe de l'amplitude du vibrato est elle aussi contrôlée par un générateur d'enveloppe. Le bruit blanc permet de modéliser de manière assez grossière le son du souffle produit par l'instrumentiste. L'effet qu'il crée reste néanmoins très important et augmente de manière significative l'aspect naturel du son généré.

```
vibrato = vibratoGain*envVibrato(vibratoBegin,vibratoAttack,100,
   vibratoRelease,gate)*osc(vibratoFreq);
envelopeBreath = pressure*adsr(pressure*envelopeAttack,
   envelopeDecay,80,envelopeRelease,gate);
```

283. Attack – Decay – Sustain – Release : Attaque – Relâchement – Maintien – Chute.

```
breathPressure = envelopeBreath + envelopeBreath*(noiseGain*noise +
    vibrato) + 10.0^(-15.0);
```

Les deux lignes de retards constituant le guide d'ondes sont déclarées. Des lignes de retards à interpolation sont utilisées afin d'éviter les problèmes de justesse pour les notes aiguës.

```
jetDelay = fdelay(4096,jetDelayLength);
boreDelay = fdelay(4096,boreDelayLength);
```

La fonction `jetDelay` (cf. annexe R) est utilisée pour modéliser les interactions entre l'énergie renvoyée par la fin du tube acoustique de la flûte et la pression introduite par l'instrumentiste.

Enfin, les différents éléments de l'algorithme sont assemblés. Le diagramme de ce dernier est visible dans la figure 3.13 à la page 82.

```
process = (reflexionFilters <: ((breathPressure - _*jetReflexion) :
    jetDelay : jetTable) + (_*endReflexion)) ~ (boreDelay : NLFM) :
    *(0.3*gain) : stereo : instrReverb;
```

Annexe L

Description du fichier `flute.dsp`

L'instrument FAUST `flute.dsp`[284] implémente un modèle physique de flûte traversière (cf. partie 3.4.1) avec la technique des guides d'ondes numériques. Il est très similaire à celui décrit précédemment dans l'annexe K mais possède quelques simplifications et modifications lui permettant de rendre plus naturel le son qu'il génère.

Les paramètres de l'interface utilisateur sont les mêmes que ceux de `flutestk.dsp` bien que leurs noms varient légèrement. La seule exception vient du paramètre `embouchureAjust` qui n'existes pas dans le cas de `flute.dsp`.

Le filtre passe-tout passif non-linéaire décrit dans la partie 2.2 est utilisé de la même manière que dans `flutestk.dsp`, cette étape n'est donc pas détaillée ici.

Dans un premier temps, les coefficients de réflexion placés à chaque extrémité du guide d'ondes sont déclarés. Les deux lignes de retards sont implémentées et leur longueur est calculée en fonction de la fréquence de la note jouée.

```
feedBack1 = 0.4;
feedBack2 = 0.4;

embouchureDelayLength = (SR/freq)/2-2;
boreDelayLength = SR/freq-2;
embouchureDelay = fdelay(4096,embouchureDelayLength);
boreDelay = fdelay(4096,boreDelayLength);
```

La fonction polynomiale permettant de modéliser les interactions entre l'énergie renvoyée par le fin du tube acoustique et la pression introduite par l'instrumentiste est implémentée.

Le filtre de réflexion utilisé est un filtre passe-bas du premier ordre avec une fréquence de coupure de deux mille Hertz.

La fonction `stereoizer` (cf. annexe R à la page 174) est invoquée.

284. Fichier disponible dans le dossier `/flute/` du CD.

```
poly = _ <: _ - _*_*_;
reflexionFilter = lowpass(1,2000);
stereo = stereoizer(SR/freq);
```

L'excitation est très similaire à celle utilisée dans le cas de `flutestk.dsp`, son fonctionnement n'est donc pas détaillé ici.

```
env1 = adsr(env1Attack,env1Decay,90,env1Release,(gate |
   pressureEnvelope))*pressure*1.1;
env2 = asr(env2Attack,100,env2Release,gate)*0.5;
vibratoEnvelope = envVibrato(vibratoBegin,vibratoAttack,100,
   vibratoRelease,gate)*vibratoGain;
vibrato = osc(vibratoFreq)*vibratoEnvelope;
breath = noise*env1;
flow = env1 + breath*breathAmp + vibrato;
```

Pour terminer, les différents éléments de l'algorithme sont assemblés. Un diagramme de ce dernier peut être visualisé sur la figure 3.14 de la page 82.

```
process = (_ <: (flow + *(feedBack1) : embouchureDelay : poly) + *(
   feedBack2) : reflexionFilter)~(boreDelay : NLFM) : *(env2)*gain
   : stereo : instrReverb;
```

Annexe M

Description du fichier `brass.dsp`

L'instrument FAUST `brass.dsp` [285] implémente un modèle physique de cuivre (cf. partie 3.4.4) avec la technique des guides d'ondes numériques. Il peut être utilisé pour imiter le son de tout instruments dont le corps est en cuivre et dont l'embouchure est de la même forme que celle d'une trompette, d'un cor, d'un trombone, d'un tuba, etc. Il est basé sur le modèle implémenté dans le SYNTHESIS TOOLKIT. La seule différence vient de l'utilisation du filtre passe-tout passif non-linéaire présenté dans la partie 2.2 afin d'ajouter des non-linéarités dans le son produit.

En plus des paramètres compatibles avec *faust2pd* et communs à l'ensemble des instruments du FAUST-STK : fréquence, amplitude, note-on/off (cf. annexe G), paramètres relatifs à l'utilisation de la fonction `nonLinearModilator`, etc., `brass.dsp` donne la possibilité de contrôler les paramètres physiques suivant :

- `pressure` : la pression de l'air permettant de produire le signal d'excitation ;
- `lipTension` : la tension des lèvres de l'instrumentiste ;
- `slideLength` : la longueur de la zone mobile du tube de l'instrument dans le cas d'un trompette ou d'un trombone.

La déclaration du filtre passe-tout passif non-linéaire étant exactement la même que celle de l'instrument `harpsi.dsp` (cf. annexe H), cette dernière n'est pas redonnée ici.

L'algorithme modélisant l'embouchure de l'instrument est implémenté. Il s'agit d'un filtre résonant dont la fréquence centrale est déterminée en fonction de la hauteur de la note jouée et du coefficient de tension des lèvres. Le signal produit par ce filtre est mis au carré et est saturé. La fonction `bandPassH` est déclarée dans la bibliothèque `filter.lib` et le fonctionnement de `saturationPos` est détaillé dans l'annexe R. Un diagramme de

285. Fichier disponible dans le dossier `/cuivre/` du CD.

cet algorithme peut être visualisé sur la figure 3.18 à la page 91.

```
lipFilterFrequency = freq*pow(4,(2*lipTension)-1);
lipFilter = *(0.03) : bandPassH(lipFilterFrequency,0.997) <: * :
   saturationPos;
```

La fonction `stereoiszer` est utilisée pour produire un signal stéréo (cf. annexe R).
La longueur de la ligne de retards est calculée en fonction de la fréquence de la note jouée et de la longueur de la zone mobile du tube de l'instrument. Une ligne de retards avec interpolation est utilisée afin de pallier aux problèmes de justesse lorsque des notes aiguës sont jouées (cf. partie 1.2).

```
stereo = stereoizer(SR/freq);

slideTarget = ((SR/freq)*2 + 3)*(0.5 + slideLength);
boreDelay = fdelay(4096,slideTarget);
```

Le signal d'excitation est produit avec un générateur d'enveloppe d'amplitude de type *ADSR*[286]. Il est légèrement modulé avec un oscillateur de basse fréquence afin de créer un effet de vibrato. Du bruit est ajouté au signal produit dans le but d'imiter le son du souffle de l'instrumentiste.

```
vibrato = vibratoGain*osc(vibratoFreq)*envVibrato(vibratoBegin,
   vibratoAttack,100,vibratoRelease,gate);
breathPressure = pressure*adsr(envelopeAttack,envelopeDecay,100,
   envelopeRelease,gate) + vibrato;
mouthPressure = 0.3*breathPressure;
```

La variable `borePressure` est le coefficient de réflexion utilisé à la fin du tube de l'instrument pour contrôler la quantité d'énergie renvoyée au début du guide d'ondes. `deltaPressure` modélise de manière très simplifiée les interactions entre les ondes de réflexions et le signal produit au niveau de l'embouchure.

```
borePressure = *(0.85);
deltaPressure = mouthPressure - _;
```

Les différents éléments du modèle sont assemblés. On peut noter l'utilisation d'un bloqueur de composante continue à la fin de l'algorithme qui permet de rendre plus stable le modèle.

```
process = (borePressure <: deltaPressure,_ : (lipFilter <: *(
   mouthPressure),(1-_)),_ : _, * :> + : dcblocker) ~ (boreDelay :
   NLFM) ;
```

286. Attack – Decay – Sustain – Release : Attaque – Relâchement – Maintien – Chute.

Annexe N

Description du fichier `bowed.dsp`

L'instrument FAUST `bowed.dsp`[287] implémente un modèle physique générique d'instrument à cordes frottées (cf. partie 3.3.1) avec la technique des guides d'ondes numériques. Il est peut être utilisé pour imiter le son de la plupart des instruments de cette famille (violon, violoncelle, alto, etc.).

Son corps est basé sur deux guides d'ondes unidimensionnels qui sont excités avec une fonction non-linéaire prenant en compte les interactions possibles entre les ondes se déplaçant sur les cordes et le signal généré par la friction de l'archet sur l'une d'entre elles.

La déclaration de l'interface utilisateur n'est pas donnée ici dans la mesure où elle est identique à celle des autres instruments du FAUST-STK et notamment à celle de `clarinette.dsp` (cf. annexe D). Toutefois, un certain nombre de paramètres physiques propres à ce modèle peuvent être contrôlés :

– `bowPosition` : la position de l'archet sur la corde ;
– `bowPressure` : la pression appliquée par l'archet sur la corde.

`bowed.dsp` utilise le filtre passe-tout passif non-linéaire décrit dans la partie 2.2. Sa déclaration n'est pas redonnée ici puisqu'il s'agit de la même que celle décrite dans l'instrument `flutestk.dsp` (cf. annexe K).

La fonction `bow`, qui est décrite dans l'annexe R à la page 174 est appelée. Elle permet de simuler la friction de l'archet sur une corde. Le paramètre `bowPressure` a un effet direct sur la raideur de la pente (`tableSlope`) de la table de fonction utilisée.

```
tableOffset =  0;
tableSlope = 5 - (4*bowPressure);
bowTable = bow(tableOffset,tableSlope);
```

287. Fichier disponible sur le CD dans le dossier `/violon/`.

Une enveloppe de type *ADSR*[288] est utilisée pour contrôler la vélocité de l'archet qui permettra de produire le signal d'excitation.

```
envelope = adsr(gain*envelopeAttack,envelopeDecay,90, (1-gain)*
   envelopeRelease,gate);
```

Les deux lignes de retards à interpolation utilisées dans les guides d'ondes modélisant le corps de l'instrument sont déclarées. Leurs longueurs respectives sont calculées en fonction de la fréquence de la note souhaitée, de la fréquence d'échantillonnage et du paramètre `bowPosition`. En effet, il est possible de contrôler la position du point d'excitation utilisé sur le modèle en réduisant et en augmentant de manière proportionelle la longueur des deux lignes retards des guides d'ondes. La même technique est employée avec l'instrument `saxophony.dsp` et est détaillée dans l'annexe F et dans la partie 3.4.3.
Un oscillateur de basse fréquence est utilisé pour moduler la longueur des deux lignes de retards dans le but de créer un effet de vibrato. Notons qu'il est indispensable d'utiliser des lignes de retards fractionnelles pour pratiquer une telle action.

```
maxVelocity = 0.03 + (0.2 * gain);
betaRatio = 0.027236 + (0.2*bowPosition);
fdelneck = (SR/freq-4)*(1 - betaRatio);
vibratoEnvelope = envVibrato(vibratoBegin,vibratoAttack,100,
   vibratoRelease,gate);
vibrato = fdelneck + ((SR/freq - 4)*vibratoGain*vibratoEnvelope*osc
   (vibratoFreq));
neckDelay = fdelay(4096,vibrato);
fdelbridge = (SR/freq - 4)*betaRatio;
bridgeDelay = delay(4096,fdelbridge);
```

Un filtre passe-bande dont la fréquence centrale est placée à cinq cent hertz est utilisé pour modéliser de manière très grossière l'effet de l'application de la réponse impulsionnelle de la caisse de résonance de l'instrument sur le son produit par les cordes.

```
bodyFilter = *(0.2) : bandPass(500,0.85);
```

Le filtre de réflexion ici nommé `stringFilter` est un simple filtre passe-bas du premier ordre.

```
stringFilter = *(0.95) : -onePole(b0,a1)
with{
  pole = 0.6 - (0.1*22050/SR);
  gain = 0.95;
  b0 = 1-pole;
  a1 = -pole;
};
```

288. Attack – Decay – Sustain – Release : Attaque – Relâchement – Maintien – Chute.

La fonction `instrumentBody` implémente le guide d'ondes inférieur du modèle ainsi que les différentes jonctions de dispersions. Il intègre également la fonction non-linéaire d'excitation (`bow`) à l'algorithme. Le diagramme de cette fonction peut être visualisé sur la figure 3.9 à la page 73.

```
bowVelocity = envelope*maxVelocity;
instrumentBody(feedBckBridge) = (*(-1) <: +(feedBckBridge),_ : (
   bowVelocity-_ <: *(bowTable) <: _,_),_ : _, + : +(feedBckBridge)
   ,_) ~ (neckDelay) : !,_;
```

Le guide d'ondes supérieur du modèle est mis en place et les différents éléments de l'algorithme sont assemblés. Le diagramme de ce dernier est lui aussi visualisable sur la figure 3.9.

```
process = (stringFilter : instrumentBody) ~ (bridgeDelay : NLFM) :
   bodyFilter : *(gain) : stereo : instrReverb;
```

Annexe O

Description du fichier `violin.dsp`

Une rapide description d'assez haut niveau du fichier `violin.dsp`[289] est donnée dans cette annexe. Plus d'informations sur la mise en place du modèle d'instrument à corde sur lequel l'instrument ici présenté est basé sont données dans l'annexe N.
Dans un premier temps, les différents paramètres contrôlables via l'interface utilisateur sont déclarés. La fonction de cette dernière est différente de celle qui lui est habituellement attribuée. En effet, elle sert simplement à envoyer dans le cas présent des flux de données gestuelles lorsque l'objet FAUST est utilisé sous la forme d'un plug-in PUREDATA (cf. 3.3.3).
`freqIn` est la fréquence de la note à jouer, `stringNumber` correspond au numéro de la corde à utiliser, `bowPositionIn` est la position du point de frottement de l'archet sur la corde, `forceIn` correspond à la force appliquée sur la corde par l'archet et `bowVelIn` est la vélocité de l'archet.

```
freqIn = nentry("h:Basic Parameters/freq", 1, 20, 20000, 1);
stringNumber = nentry("stringNumber",0,0,4,1) : int;
bowPositionIn = hslider("v:Physical Parameters/bowPosition"
   ,0.01,0.01,1,0.01);
forceIn = hslider("v:Physical Parameters/force",0,0,7,0.01)/7;
bowVelIn = hslider("bowVel",0,0,130,0.01)/127;
```

Les quatre filtres de réflexion utilisés pour chaque corde du violon sont implémentés. Il s'agit de séries de filtres biquadratiques dont les coefficients sont extraits de fonctions C++ (cf. 3.3.2) implémentées dans le fichier `instrument.h`[290]. La déclaration de chaque filtre est faite à l'aide du système de filtrage par motifs de FAUST qui permet par la suite d'implémenter chaque corde du violon avec une seule et même déclaration.

289. Fichier disponible sur le CD dans le dossier `/violin/`.
290. Fichier disponible sur le CD dans le dossier `/util/`.

```
stringFilter(3) = stringFilterOP : seq(i,5,tf2(fCString(i,0),
   fCString(i,1),fCString(i,2),fCString(i,3),fCString(i,4)))
   *0.857261640575012
with{
  fCString = ffunction(float violinFDBFiltS4(int,int), <instrument.
     h>,"");
  filterPole = 0.6;
  filterGain = 0.95;
  pole = filterPole - (0.1*22050/SR);
  b0 = 1-pole;
  a1 = -pole;
  stringFilterOP = _*filterGain : -onePole(b0,a1);
};

stringFilter(2) = stringFilterOP : seq(i,5,tf2(fCString(i,0),
   fCString(i,1),fCString(i,2),fCString(i,3),fCString(i,4)))
   *0.896576688334173
with{
  fCString = ffunction(float violinFDBFiltS3(int,int), <instrument.
     h>,"");
  filterPole = 0.7;
  filterGain = 1;
  pole = filterPole - (0.1*22050/SR);
  b0 = 1-pole;
  a1 = -pole;
  stringFilterOP = _*filterGain : -onePole(b0,a1);
};

stringFilter(1) = stringFilterOP : seq(i,5,tf2(fCString(i,0),
   fCString(i,1),fCString(i,2),fCString(i,3),fCString(i,4)))
   *0.857261640575012
with{
  fCString = ffunction(float violinFDBFiltS4(int,int), <instrument.
     h>,"");
  filterPole = 0.6;
  filterGain = 0.95;
  pole = filterPole - (0.1*22050/SR);
  b0 = 1-pole;
  a1 = -pole;
  stringFilterOP = _*filterGain : -onePole(b0,a1);
};

stringFilter(0) = stringFilterOP : seq(i,5,tf2(fCString(i,0),
   fCString(i,1),fCString(i,2),fCString(i,3),fCString(i,4)))
   *0.857261640575012
with{
  fCString = ffunction(float violinFDBFiltS4(int,int), <instrument.
     h>,"");
  filterPole = 0.6;
  filterGain = 0.95;
  pole = filterPole - (0.1*22050/SR);
  b0 = 1-pole;
```

```
  a1 = -pole;
  stringFilterOP = _*filterGain : -onePole(b0,a1);
};
```

Le corps de l'instrument est déclaré. Cette section du code est très similaire à celle de `bowed.dsp`, elle n'est donc pas détaillée à nouveau. Il est toutefois important de noter que les paramètres `force` et `bowVelIn`, déclarés dans l'interface utilisateur sont utilisés directement dans l'algorithme (cf. partie 3.3.3). Aussi, les valeurs du paramètre `force` sont interpolées à l'aide de la fonction `smooth` afin d'éviter toutes discontinuités dans le son. La valeur 0,993 permet d'obtenir une interpolation très rapide de qualité convenable.

```
instrumentBody(force,bowVel,bowPosition,freq,trig,feedBckBridge) =
   (_*-1 <: _ + feedBckBridge,_ : (bowVelocity-_ <: bowTable*_ : _
   *(forceCondition*trig : smooth(0.993)) <: _,_),_ : _,_+_ : _ +
   feedBckBridge,_) ~ neckDelay : !,fdelbridge,_
with{
  forceCondition = force > 0;
  tableOffset =  0;
  tableSlope = 5 - (4.5*force);
  bowTable = bow(tableOffset,tableSlope);
  maxVelocity = 0.03 + 0.2;
  bowVelocity = maxVelocity*bowVel;
  betaRatio = bowPosition;
  fdelneck = (SR/freq-4) * (1 - betaRatio);
  neckDelay = fdelay(4096,fdelneck);
  fdelbridge = (SR/freq-4) * betaRatio;
  bridgeDelay = delay(4096,fdelbridge);
};
```

La série de filtres biquadratiques modélisant la caisse de résonance du violon est déclarée (cf. 3.3.2). Tout comme dans le cas des filtres de réflexions, les coefficients sont extraits d'une fonction C++ contenue dans le fichier `instrument.h`.

```
bodyFilter = seq(i,6,tf2(fC(i,0),fC(i,1),fC(i,2),fC(i,3),fC(i,4)))
   : *(0.0637)
with{
  fC = ffunction(float violinImpRes2(int,int), <instrument.h>,"");
};
```

Les valeurs des paramètres `freqIn` et `bowPosition` sont maintenues lorsque qu'une corde n'est plus utilisée afin d'éviter toutes discontinuités lors de futurs attaques et pour conserver la hauteur de la dernière note jouée quand la corde vibre librement après avoir été excitée par l'archet (cf. 3.3.3).

```
SH(trig,x) = (*(1 - trig) + x * trig) ~ _;
cnt = (_+1)~_ : _-1;
conditionHold(n) = ((stringNumber == (n + 1)) | (cnt<1));
```

```
freqHold(n) = SH(conditionHold(n),freqIn);
bowPositionHold(n) = SH(conditionHold(n),bowPositionIn);
```

Les différents éléments de l'algorithme sont assemblés et les quatre instances de corde sont déclarées. Le système de condition permettant de sélectionner la corde à utiliser est également implémenté. Il permet de choisir la corde à laquelle le flux de données reçu au niveau du paramètre `forceIn` est envoyé.

```
bridgeDelay = fdelay(4096);
process = par(i,4,(stringFilter(i) : instrumentBody(forceIn*(
   stringNumber == (i+1)),bowVelIn,bowPositionHold(i),freqHold(i),(
   stringNumber == (i+1))))~fdelay(4096) : !,_*2) :> + : *(2) :
   bodyFilter <: _,_ : instrReverb;
```

Un diagramme de haut niveau de cet instrument peut être visualisé sur la figure 3.10 à la page 78.

Annexe P

Description du fichier `mesh.dsp`

L'instrument FAUST `mesh.dsp`[291] implémente un modèle physique de plaque carrée avec la technique des guides d'ondes numériques. Il utilise un réseau de guides d'ondes carré dont la taille peut être définie par l'utilisateur avant la compilation. Les terminaisons rigides autour du réseau sont simulées en connectant les deux côtés adjacents de la plaque modélisée avec les deux autres côtés opposés. Des filtres passe-touts passifs non-linéaires tels que ceux présentés dans la partie 2.2 sont utilisés au niveau de chacune de ses extrémités dans le but de générer des comportements non-linéaires. En fonction de sa taille et du temps de résonance qui lui est attribué, il est sensé (cf. partie 3.5.1) pouvoir produire des sons allant de la plaque métallique à la cymbale en passant par le gong.
Dans un premier temps, les éléments de l'interface utilisateur sont déclarés. Il est possible de contrôler en temps réel le temps de résonance de la plaque (une valeur entre 0 et 0,99) ainsi que le taux de non-linéarités introduites aux niveaux des extrémités du réseau de guides d'ondes. Il est également possible de définir avant la compilation l'ordre des filtres non-linéaires ainsi que la taille du réseau (un entier positif correspondant à la taille d'un côté de la plaque carrée).

```
resonance = hslider("resonance",0,0,0.99,0.01)*0.01;
nonlinearity = hslider("nonlinearity",0,0,1,0.01);

nlfOrder = 2;
meshSize = 16;
```

Dans le cas hypothétique où un réseau de taille 1x1 serait utilisé, une simple jonction de dispersion est déclarée.

291. Fichier disponible sur le CD dans dossier `/mesh/`.

```
squared_mesh(1) = bus(4) <: par(i,4,*(-1)), (bus(4) :> (*(.5)) <:
   bus(4)) :> bus(4);
```

Un réseau de guides d'ondes carré de taille NxN est mis en place. Il s'agit simplement d'un ensemble de jonctions de dispersions à quatre ports interconnectées à la manière d'un « filet » (cf. figure 3.20 à la page 95). Le système de filtrage par motif de FAUST est utilisé pour mener à bien cette opération et pour donner la possibilité à l'utilisateur de définir la taille du réseau.

La plupart des sous-fonctions définies ici permettent de diriger le signal à différents endroits de l'algorithme. En effet, la difficulté dans l'implémentation d'un réseau de guides d'ondes réside principalement dans le « routage » des différents signaux qui le traverse.

```
squared_mesh(N) = bus(4*N) : (route_inputs(N/2) : par(i,4,
   mesh_square(N/2))) ~ (prune_feedback(N/2)) : prune_outputs(N/2)
   : route_outputs(N/2) : bus(4*N)
with {
  block(N) = par(i,N,!);
  s(i,N,M) = par(j, M*N, Sv(i, j))
  with {
    Sv(i,i) = bus(N); Sv(i,j) = block(N);
  };
  prune_outputs(N) = bus(16*N) : block(N), bus(N), block(N), bus(N)
     , block(N), bus(N), bus(N), block(N), bus(N), block(N), block(
     N), bus(N), bus(N), block(N), bus(N),   block(N) : bus(8*N);
  route_outputs(N) = bus(8*N) <: s(4,N,8), s(5,N,8), s(0,N,8), s(2,
     N,8), s(3,N,8), s(7,N,8), s(1,N,8), s(6,N,8) : bus(8*N);
  prune_feedback(N) = bus(16*N) : bus(N), block(N), bus(N), block(N
     ), bus(N), block(N), block(N), bus(N), block(N), bus(N), bus(N
     ), block(N), block(N), bus(N), block(N), bus(N) : bus(8*N);
  route_inputs(N) = bus(8*N), bus(8*N) <: s(8,N,16), s(4,N,16), s
     (12,N,16), s(3,N,16), s(9,N,16), s(6,N,16), s(1,N,16), s(14,N
     ,16), s(0,N,16), s(10,N,16), s(13,N,16), s(7,N,16), s(2,N,16),
      s(11,N,16), s(5,N,16), s(15,N,16) : bus(16*N);
};
```

Le filtre passe-tout passif non-linéaire est déclaré (la fonction `nonLinearModulator` de la bibliothèque `instrument.lib` est utilisée). Ce dernier est instancié au niveau de chaque signal renvoyé à l'autre extrémité du réseau de guides d'ondes.

Cette dernière opération est menée à bien dans la fonction `squared_mesh_test`. Celle-ci permet également d'appliquer le coefficient multiplicateur simulant les effets de dispersion de l'énergie engendrés par la résistance avec l'air et le fait que les terminaisons de la plaque ne puissent être parfaitement rigides.

```
nonLin = nonLinearModulator(nonlinearity,1,0,0,0,nlfOrder);
squared_mesh_test(N,x) = squared_mesh(N)~(busi(4*N,x))
```

```
with{
  busi(N,x) = bus(N) : par(i,N,*(-1*(0.99+resonance)) : nonLin) :
     par(i,N-1,_), +(x);
};
```

Une fonction très simple permettant de générer une impulsion d'un échantillon lorsqu'un bouton placé dans l'interface graphique est pressé est implémentée. Elle est utilisée pour exciter le réseau de guides d'ondes au niveau d'un de ses angles.

```
impulsify = _ <: _-_' : _>0;
excitation = button("play") : impulsify;

process = excitation : squared_mesh_test(meshSize) :> _,_;
```

Annexe Q

Description de la fonction `nonLinearModulator` de `instrument.lib`

La fonction FAUST `nonLinearModulator` contenue dans la bibliothèque `instrument.lib` [292] du FAUST-STK permet d'utiliser le filtre décrit dans la partie 2.2. Elle prend en arguments six paramètres ainsi que le signal à traiter. Le paramètre `nonlinearity` a une valeur comprise entre zéro et un qui permet de définir l'importance de la modulation des coefficients du filtre et donc la quantité de « non-linéarités » introduites dans le son. L'argument `env` permet de connecter une fonction pour le contrôle de l'enveloppe des non-linéarités appliquées sur le son. Le paramètre `typeMod` permet de définir le type de modulation des coefficients du filtre. Il peut prendre pour valeur :

- 0 : les coefficients sont modulés par le signal traité par le filtre ;
- 1 : les coefficients sont modulés par la moyenne entre l'échantillon traité et celle du précédent ($coefficient = \frac{x(n)+x(n-1)}{2}$) ;
- 2 : les coefficients sont modulés par le carré de l'échantillon traité ;
- 3 : les coefficients sont modulés par un signal sinusoïdal de fréquence `freqMod` ;
- 4 : les coefficients sont modulés par un signal sinusoïdal ayant la même fréquence que la note jouée.

```
nonLinearModulator(nonlinearity,env,freq,typeMod,freqMod,order) =
```

Le signal est envoyé vers les différentes instances du filtre non-linéaire en fonction de `typeMod`. Dans le cas où les coefficients du filtre sont modulés par le signal traité, un mixage est effectué entre le signal entrant et le signal traité en fonction de la valeur de

292. Fichier disponible sur le CD dans le dossier `/util/`.

`nonLinearity` (cf. partie 2.2).

```
_ <: nonLinearFilterOsc*(typeMod >= 3), (_ <: nonLinearFilterSig*
   nonlinearity,_*(1 - nonlinearity) :> +)*(typeMod < 3)
:> +
with{
```

La fréquence de la sinusoïde est choisie en fonction de la valeur de `typeMod`.

```
freqOscMod = (typeMod == 4)*freq + (typeMod != 4)*freqMod;
```

Les signaux modulants sont calculés et sélectionnés en fonction de la valeur de `typeMod`. L'amplitude des modulations est controlée par les valeurs de `nonlinearity` et de `env`. Dans la mesure où les jonctions de dispersion du filtre sont normalisées (cf. partie 2.2), il est nécessaire de mettre à l'échelle le signal modulant les coefficients du filtre ($\theta \in [-\pi, \pi]$).

```
tsignorm(x) = nonlinearity*PI*x*env;
tsigsquared(x) = nonlinearity*PI*x*x*env;
tsigav(x) = nonlinearity*PI*((x + x')/2)*env;
tosc = nonlinearity*PI*osc(freqOscMod)*env;

tsig(x) = tsignorm(x)*(typeMod == 0) + tsigav(x)*(typeMod == 1) +
   tsigsquared(x)*(typeMod == 2);
```

Les différentes instances du filtre non-linéaire sont créées pour être utilisées dans `nonLinearModulator`.

```
nonLinearFilterSig(x) = x <: allpassnn(order,(par(i,order,tsig(x)))
   );
nonLinearFilterOsc = _ <: allpassnn(order,(par(i,order,tosc)));
};
```

Annexe R

Description des fonctions importantes de `instrument.lib`

La bibliothèque `instrument.lib`[293] contient un ensemble de fonctions aux buts divers et variés, utilisé par les différents instruments du FAUST-STK. Le fonctionnement de certaines d'entres-elles est ici détaillé.

`bow`

La fonction `bow` modélise le signal généré lorsqu'un archet est frotté sur une corde. Elle prend en compte les interactions possibles entre les ondes se déplaçant sur la corde et celles produites par la friction de l'archet qui peuvent donner naissance à des comportements non-linéaires. Elle est basée sur une version simplifiée de la fonction donnée par Micheal MCINTYRE et Jim WOODHOUSE[294] :

$$y = (|(x + offset).slope| + 0.75)^{-4}$$

où *slope* est la raideur de la pente de chaque côté de le courbe et $offset$ le décalage du point de départ de la fonction.
La déclaration de la fonction `saturationPos` est donnée dans la description de la fonction `jetTable` dans ce même annexe à la page 175.

```
bow(offset,slope) = pow(abs(sample) + 0.75, -4) : saturationPos
with{
```

293. Fichier disponible sur le CD dans le dossier `/util/`.
294. MCINTYRE, Michael ; WOODHOUSE, Jim, « On the Fundamentals of Bowed String Dynamics », *Acustica*, XLIII (1979), p. 93-108.

```
  sample(y) = (y + offset)*slope;
};
```

jetTable

La fonction `jetTable` est utilisée avec les instruments dont l'excitation est générée par une embouchure. En effet, lorsqu'elle est placée dans le guide d'ondes d'un instrument de ce type, elle permet de modéliser les interactions entre l'énergie renvoyée par la fin du tube acoustique de ce dernier et la pression introduite par l'instrumentiste en créant un signal non-linéaire comme cela est expliqué dans la partie 3.4.1.
Les fonctions `saturationPos` et `saturationNeg` permettent réspectivement de faire saturer positivement et négativement le signal qui leur ait fourni.

```
saturationPos(x) = x <: (_>1),(_<=1 : *(x)) :> +;
saturationNeg(x) = x <: (_<-1),(_>=-1 : *(x)) :> *(-1) + _;
jetTable(x) = x <: _*(_*_-1) : saturationPos : saturationNeg;
```

reed

La fonction `reed` produit une excitation non-linéaire de la forme : $y(x) = \phi + \theta \times x$ où ϕ est le décalage (`offset`), θ la pente de la courbe (`slope`) et x l'index de consultation de la table [295]. Les fonctions `saturationPos` et `saturationNeg` décrites précédemment permettent d'accentuer le caractère non-linéaire du signal produit lors de l'étape précédente et d'assurer une certaine stabilité au sein du guide d'ondes dans lequel la fonction `reed` est placée.

```
reed(offset,slope) = reedTable : saturationPos : saturationNeg
with{
  reedTable = offset + (slope*_);
};
```

setereoizer

La fonction `stereoizer` permet de transformer artificiellement un signal mono en un signal stéréo. Pour cela, dans un premier temps, le signal mono est scindé en deux signaux distincts. Un retard est alors introduit sur l'un d'entre-eux ce qui permet d'obtenir l'effet

295. MCINTYRE, Michael ; SHUMACHER, Robert ; WOODHOUSE, James, « On the Oscillations of Musical Instruments », *Journal of the Acoustical Society of America*, LXXIV (1983), n° 5, p. 1325-1345.

« artificiel » de stéréo. La durée du retard varie en fonction du paramètre `spatial width` défini par l'utilisateur et le paramètre `periodDuration` qui doit correspondre à la période de la fréquence de la note jouée exprimée en nombre d'échantillons. Il est également possible de contrôler la panoramique gauche/droite du son produit.

La plupart des instruments du FAUST-STK utilise cette fonction.

```
stereoizer(periodDuration) = _ <: _,widthdelay : stereopanner
with{
  W = hslider("v:Spat/spatial width", 0.5, 0, 1, 0.01);
  A = hslider("v:Spat/pan angle", 0.6, 0, 1, 0.01);
  widthdelay = delay(4096,W*periodDuration/2);
  stereopanner = _,_ : *(1.0-A), *(A);
};
```

Bibliographie

Monographies – Articles

ADRIEN, Jean-Marie, « The Missing Link : Modal Synthesis », *Representations of Musical Signals*, éd. sous la direction de Curtis Roads, Cambridge : MIT Press, 1991, p. 269-297.

ALEMBERT, Jean le Rond (d'), *Recherche sur les cordes vibrantes*, 1747.

BANK, Balázs ; ZAMBON, Stefano ; FONTANA, Frederico, « A Modal-Based Real-Time Piano Synthesizer », *Transactions on Audio, Speech, and Language Processing*, XVIII (2010), n° 4, p. 809-821.

BONADA, Jordi ; SERRA, Xavier, « Synthesis of the Singing Voice by Performance Sampling and Spectral Models », *Signal Processing Magazine*, XXIV (2007), n° 2, p. 67-79.

BRAN-RICCI, Josiane, « Instruments de musique », *Dictionnaire des Musiques*, Paris : Universalis, 2009, p. 554.

BURTNER, Matthew, « The Metasxophone : Concept, Implementation, and Mapping Strategies for a New Computer Music Instrument », *Organised Sound*, VII (2002), n° 2, p. 201-213.

CADOZ, Claude, *Synthèse sonore par simulation de mécanismes vibratoires, application aux sons musicaux*, Thèse de doctorat inédite, Institut Polytechnique de Grenoble, 1979.

CADOZ, Claude ; LUCIANI, Annie ; FLORENS, Jean-Loup, « CORDIS-ANIMA : A Modeling and Simulation System for Sound and Image Synthesis - the General Formalism », *Computer Music Journal*, XVII (1993), n° 4, 1993, p. 19-29.

CASTAGNE, Nicolas ; CADOZ, Claude, *10 Criteria for Evaluating Physical Modelling Schemes for Musical Creation : actes de Conference on Digital Audio Effects (DAFx-03), Londres, 8-11/09/2003*, Londres : Queen Mary University, 2003.

CASTAGNE, Nicolas ; CADOZ, Claude, *GENESIS : A Friendly Musician-Oriented Environment for Mass-Interaction Physical Modeling : actes de International Computer Music Conference (ICMC), Gothenburg (Suède), 2002*, Gothenburg : Computer Music Association, 2002.

CHAFFE, Chris, *Case Studies of Physical Models in Music Composition : actes de 18th*

International Congress of Acoustics (ICA), Kyoto, 4-9/04/2004, Kyoto : Kyoto International Conference Hall, 2004.

CHAIGNE, Antoine, « Numerical Simulations of Stringed Instruments - Today's Situation and Trends for the Future », *Catgut Acoustical Society Journal*, IV (2002), n° 5, p. 12-20.

CHAIGNE, Antoine ; ASKENFELT, Anders, « Numerical Simulations of Piano Strings, a Physical Model for a Struck String Using Finite Difference Methods », *Journal of Acoustical of America*, II (1994), n° 95, p. 1112-1118.

CHOWNING, John ; BISTROW, David, *FM Thoery and Applications*, Tokyo : Yamaha Corporation, 1986.

CLARK, Paul, *Paul Lansky Interview*, 1997, article en ligne : `http://www.electronicmusic.com/features/interview/paullansky.html`.

COAKLEY, William, « Amazing Instruments : Yamaha VL1, VL7 and VL70m », septembre 2001, article en ligne à l'adresse suivante : `http://williamcoakley.com/articles.php?article=yamaha.php`

COOK, Perry, *A Meta-wind-instrument Physical Model, and a Meta-controller for Real Time Performance Control : actes de the International Computer Music Conference, San Jose (US), 1992*, San Jose : Computer Music Association, 1992.

COOK, Perry, « SPASM, a Real-Time Vocal Tract Physical Model Controller and Singer », *Computer Music Journal*, XVII (1993), n° 1, p. 30-44.

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

COOK, Perry, *Tbone : an Interactive Waveguide Brass Instrument Synthesis Workbench for the NeXT Machine : actes de the Interational Computer Music Conference, Montreal, 1991*, Montreal : Computer Music Association, 1991, p. 297-299.

ECKEL, Gerhard ; IOVINO, Fransisco ; CAUSSÉ, René, *Sound Synthesis by Physical Modelling with Modalys*, Paris : IRCAM, 1995.

ESSL, Georg ; COOK, Perry, *Banded Waveguides : Towards Physical Modeling of Bowed Bar Percussion Instruments : actes de International Computer Music Conference (ICMC), Pékin, 10/1999*, Pékin : International Computer Music Association, 1999.

FLETCHER, Neville ; ROSSING, Thomas, *The Physics of Musical Instruments*, New-York : Springer-Verlag, 1991.

GRAY, Augustine ; MARKEL, John, « A Normalized Digital Filter Structure », *IEE Transaction on Acoustics, Speech and Signal Processing*, ASSP-23 (1975), n° 3, p. 268-277.

GUREVICH, Michael ; CHAFE, Chris, « Stk2pd », Stanford University : CCRMA, 2007, article disponible en ligne à : `https://ccrma.stanford.edu/wiki/Stk2pd`.

HART, Ruth, *When Sax and Computer Collide – It's Metasaxophone Colossus*, Charlottesville (USA) : University of Virginia, 2004, article disponible en ligne : `http://aands.virginia.edu/x2262.xml`

HELMHOTZ, Hermann (Von), *On the Sensations of Tone as a Physiological Basis for the Thoery of Music*, New York : Dover, 1863, R/1954.

HILLER, Lejaren ; RUIZ, Pierre-Michel, « Synthesizing Sounds by Solving the Wave Equation for Vibrating Objects », *Journal of the Audio Engineering Society*, XIX (1971), n° 7, p. 542-551.

HUFSCHMITT, Aline, *La synthèse par modèle physique*, Mémoire de master, Université de Paris Sorbonne (Paris IV), 2000, disponible en ligne à l'adresse suivante : `http://alinehuf3.free.fr/`.

JAFFE, David ; SMITH, Julius, « Extensions of the Karplus-Strong Plucked String Algorithm », *Computer Music Journal*, VII (1983), n° 2, p. 56-69.

JOT, Jean-Marc, *Etude et Réalisation d'un Spatialisateur de Sons par Modèles Physiques et Perceptifs*, Thèse de doctorat inédite, Télécom Paris, 1992.

JOT, Jean-Marc ; CHAIGNE, Antoine, « Digital Delay Networks for Designing Artificial Reverberators », *Audio Engineering Society Convention*, 1991, n° 90.

KARJALAINEN, Matti ; LAINE, Unto ; LAAKSO, Timo ; VÄLIMÄKI, Vesa, *Transmission-Line Modeling and Real-Time Synthesis of String and Wind Instruments : actes de International Computer Music Conference, Montreal, 1991*, Computer Music Association, 1991, p. 293-296.

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

KELLY, John ; LOCHBAUM, Carol, *Speech Synthesis : actes de Fourth International Congress on Acoustics, Copenhague, septembre 1962*, Copenhague, 1962.

KOJS, Juraj, *About Garden of the Dragon*, 2003, article en ligne : `http://www.kojs.net/Dragon.html`.

LAAKSO, Timo ; VÄLIMÄKI, Vesa ; KARJALAINEN, Matti ; LAINE, Unto, « Splitting the Unit Delay – Tools for Franctional Delay Filter Design », *Signal Processing Magazine*, XIII (1996), p. 30-60.

LANSKY, Paul, *About Things she Carried*, 1997, article en ligne : `http://silvertone.princeton.edu/~paul/liner_notes/tsc.html`.

LEGGE, Karl ; FLETCHER, Neville, « Nonlinear Generation of Missing Modes on a Vi-

brating String », *Journal of the Acoustical Society of America*, II (1984), n° 76, p. 5-12.

LONGCHAMPT, Jacques, « Jouer de l'ordinateur à la Biennale de Venise », *le Monde*, journal du 05/10/1982, Paris : France, 1982.

MAESTRE, Esteban, *Modeling Instrumental Gestures : an Analysis/Synthesis Framework for Violin Bowing*, Thèse de doctorat inédite, Université Pompeu Fabra (Barcelone), 2009.

MALHAM, David, « Toward Reality Equivalence in Spatial Sound Diffusion », *Computer Music Journal*, XXV (2001), n° 4, p. 31-38.

MARKEL, John ; GRAY, Augustine, *Linear Prediction of Speech*, New York : Springer Verlag, 1976.

MASSIE, Dana, « An Engineering Study of the Four-Multiply Normalized Ladder Filter », *Journal of Audio Engineering Society*, XLI (1993), n° 7/8, p. 564-582.

MCINTYRE, Michael ; SHUMACHER, Robert ; WOODHOUSE, James, « On the Oscillations of Musical Instruments », *Journal of the Acoustical Society of America*, LXXIV (1983), n° 5, p. 1325-1345.

MCINTYRE, Michael ; WOODHOUSE, Jim, « On the Fundamentals of Bowed String Dynamics », *Acustica*, XLIII (1979), p. 93-108.

MICHON, Romain, *Faust-STK : une bibliothèque de modèles physiques pour le langage Faust : actes des Journées de l'informatique musicale (JIM-2011), Saint-Etienne, 25-27/05/2011*, Saint-Etienne : Université Jean Monnet, 2011.

MILLER, Dayton, *Anecdotal History of the Science of Sound*, New-York : MacMillan, 1935.

MORISON, Joseph ; ADRIEN, Jean-Marie, « MOSAIC : a Framework for Modal Synthesis », *Computer Music Journal*, XVII (1993), n° 1, p. 45-56.

MORISON, Joseph ; WAXMAN, David, *Modalys Introduction*, Paris : IRCAM, 1997 (troisième édition).

NEBOT, Josep, « Csound Implementation of Physical Models of Woodwind Instruments », *The Csound Magazine*, 1999, article de journal en ligne disponible à l'adresse suivante : `http://www.csounds.com/ezine/autumn1999/woodwinds/`.

NEWTON, Isaac, *Philosophiae Naturalis Principia Mathematica*, Londres : Royal Society, 1687.

OLSON, Harry, *Music, Physics, and Engineering*, New York : Dover, 1967.

ORLAREY, Yann ; FOBER, Dominique ; LETZ, Stéphane, *An Algebra for Block-diagram Languages : actes de International Computer Music Conference, Göteborg (Suède), 2002*, Göteborg : International Computer Music Association, 2002.

ORLAREY, Yann ; FOBER, Dominique ; LETZ, Stéphane, *FAUST Quick Reference*, Lyon : GRAME, 2012.

PIERCE, John ; VAN DUYNE, Scott, « A Passive Non-linear Filter Design Which Facilitates Physics-based Sound Synthesis of Highly Nonlinear Musical Instruments », *Journal of Acoustical Society of America*, II (1997), n° 101, p. 1120-1126.

PORCARO, Nick ; JAFFE, David ; SCANDALIS, Gregory ; SMITH, Julius ; STILSON, Tim ; VAN DUYNE, Scott, « SynthBuilder : A Graphical Rapid-Prototyping Tool for Development of Music Synthesis and Effect Patches on Multiple Platforms », *Computer Music Journal*, XXII (1998), n° 2, p. 35-43.

POYNTING, John ; THOMSON, Joseph, *Sound*, Londres : Charles Griffin, 1900.

RAYLEIGH, John William Strutt, *The Theory of Sound*, Londres : Macmillan, 1894.

REID, Gordon, « Korg OASYS – Workstation Synth », *Sound on Sound*, Novembre 2005, article disponible en ligne : `http://www.soundonsound.com/sos/nov05/articles/korgoasys.htm`.

ROADS, Curtis, *The Computer Music Tutorial*, Cambridge (USA) : MIT, 1996.

RODET, Xavier ; DEPALLE, Philippe ; FLEURY, Gilles ; LAZARUS, Francis, *Modèles de signaux et modèles physiques d'instruments, études et comparaisons : actes du colloque sur les modèles physiques, la création musicale et les ordinateurs, 1990, Grenoble*, Paris : Edition de la maison des sciences de l'homme, 1994.

RODET, Xavier ; VERGEZ, Christophe, *Physical Models of Trumpet-like Instruments Detailed Behavior and Model Improvements : actes de International Computer Music Conference (ICMC), Hong-Kong, 1996*, Hong-Kong : University of science and technology, 1996.

ROSSING, Thomas ; MOORE, Richard ; WHEELER, Paul, *The Science of Sound*, San Fransisco : Pearson Education (USA), 2002.

ROWLAND, Nicholas, « Alesis Fusion – Synth Workstation », Sound on Sound, Mai 2006, article disponible en ligne : `http://www.soundonsound.com/sos/may06/articles/alesisfusion.htm`

RUIZ, Pierre-Michel, *A Technique for Simulating the Vibrations of Strings With a Digital Computer*, Mémoire de Master inedit, University of Illinois School of Music, 1970.

RUSS, Martin, « Technics SX-WSA1 – Acoustic Modeling Synthesizer », *Sound on Sound*, Décembre 1995, article disponible en ligne : `http://www.soundonsound.com/sos/1995_articles/dec95/technicswsa1.html`.

RUSS, Martin, « Yamaha VL1, Virtual Acoustic Synthesizer », *Sound Of Sound*, Juillet 1994, article disponible en ligne : `http://www.soundonsound.com/sos/1994_articles/jul94/yamahavl1.htm`

SCANDALIS, Gregory, « Music Technology », 2004, article disponible sur le site internet de l'auteur : `http://www.scandalis.com/Jarrah/`.

SCAVONE, Gary, *The Pipe : Explorations with Breath Control : actes de Conference on New Interfaces for Musical Expression (NIME-03), Montreal, 2003*, Montreal : Université McGill, 2003.

SCAVONE, Gary, *Time-domain Synthesis of Conical Bore Instrument Sounds : actes de International Computer Music Conference, Göteborg (Suède), 2002*, Göteborg : Computer Music Association, 2002, p. 9-15.

SERAFIN, Stefania ; KOJS, Juraj, *The Voice of the Dragon : a Physical Model of a Rotating Corrugated Tube : actes de Conference on Digital Audio Effects (DAFx-03), Londres, 8-11/09/2003*, Londres : Queen Mary University, 2003.

SMITH, Julius, *Efficient Synthesis of Stringed Musical Instruments : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993.

SMITH, Julius, *Efficient Simulation of the Reed-bore and Bow-string Mechanisms : actes de International Computer Music Conference, La Haye, 1986*, La Haye : Computer Music Association, 1986, p. 275-280.

SMITH, Julius, *Making Virtual Electric Guitars and Associated Effects Using Faust*, Stanford University : CCRMA, 2005, article disponible en ligne à l'adresse suivante : `https://ccrma.stanford.edu/~jos/`.

SMITH, Julius, *Rapport de recherche n° STAN-M-39 : Music Applications of Digital Waveguides*, Stanford University : Center for Computer Music and Acoustics, 1987.

SMITH, Julius, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, Palo Alto (USA) : Stanford University, 2010, vol. 3.

SMITH, Julius, *Waveguide Simulation of Non-cylindrical Acoustic Tubes : actes de International Computer Music Conference, Montreal, 1991*, Montreal : Computer Music Association, 1991, p. 303-307.

SMITH, Julius ; MICHON, Romain, *Nonlinear Allpass Ladder Filters in FAUST : actes de International Conference on Digital Audio Effects (DAFx-11), Paris, 19-25/09/2011*, Paris : IRCAM, 2011.

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

STEVENS, Ken ; FANT, Gunnar, « An Electrical Analog of the Vocal Tract », *Journal of the Acoustical Society of America*, XXV (1953), p. 734-742.

STEWARD, John, « An Electrical Analogue of the Vocal Organs », *Nature*, 1922, n° 110,

p. 311-312.

STRUBE, Hanz, « Linear Prediction on Warped Freqeuncy Scale », *Journal of the Acoustical Society of America*, LXVIII (1980), n° 4, p. 1071-1076.

TOLONEN, Tero ; VALIMAKI, Vesa ; KARJALAINEN, Matti, « Modeling of Tension Modulation Nonlinearity in Plucked Strings », *IEE Transaction on Speech and Audio Processing*, SAP-8 (2000), p. 300-310.

VALIMAKI, Vesa ; KARJALAINEN, Matti ; LAAKSO, Timo, *Modeling of Woodwind Bores with Finger Holes : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993, p. 32-29.

VAN DUYNE, Scott ; PIERCE, John ; SMITH, Julius, *A Passive Nonlinear Mode-coupling Circuit and the Wave Digital Hammer : actes de la International Computer Music Conference, Århus (Danemark), 1994*, Århus : Danish Institute of Electroacoustic Music, 1994.

VAN DUYNE, Scott ; SMITH, Julius, *Physical Modeling with the 2-D Digital Waveguide Mesh : actes de International Computer Music Conference, Tokyo, 1993*, Tokyo : Computer Music Association, 1993.

VAN WALSTIJN, Maarten ; SCAVONE, Gary, *The Wave Digital Tonehole : actes de Computer Music Conference, Berlin, 2000*, Berlin : Computer Music Association, 200, p. 465-468.

VERCOE, Barry, « Waveguide Physical Modeling », *The canonical CSOUND Reference Manual, Version 5.09, éd. sous la direction de Barry Vercoe*, Cambridge : MIT, 2005, p. 92.

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

VERGEZ, Christophe ; RODET, Xavier, « New Algorithm for Nonlinear Propagation of a sound Wave, Aplication to a Physical Model of a Trumpet », *Journal of Signal Processing*, IV (2000), n° 2, p. 79-87.

WEINREICH, Gabriel, « Coupled Piano Strings », *Journal of the Acoustical Society of America*, LXII (1977), p. 1474-1484.

Sites Internet

Les liens vers des sites Internet présentés dans ce mémoire ont été vérifiés le 09/05/2012.

`http://www.arturia.com/evolution/en/products/brass/intro.html` Site commercial du plug-in *Brass* de la firme Arturia.

`https://ccrma.stanford.edu` Site internet du CCRMA.

`https://ccrma.stanford.edu/~esteban` Site professionnel d'Esteban MAESTRE.

`https://ccrma.stanford.edu/~jos/pasp` Version en ligne et donc toujours à jour de l'ouvrage de Julius SMITH : *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*.

`https://ccrma.stanford.edu/software/stk` Site internet du SYNTHESIS TOOLKIT.

`https://ccrma.stanford.edu/wiki/Stk2pd/` Site du programme `stk2pd`.

`http://www.csounds.com` Site du programme CSOUND.

`http://faust.grame.fr` Site du langage de programmation FAUST.

`http://jackaudio.org` Site du système JACK Audio Connection Kit.

`http://www.jaffe.com/svb.html` Page web dans laquelle David JAFFE commente son œuvre *Silicon Valley Breakdown*.

`http://www.korg.com/oasys` Site commercial de la gamme de clavier *Oasys* de la firme Korg.

`http://www.music.mcgill.ca/~gary` Site internet de Gary SCAVONE.

`http://www.phy6.org/stargaze/Fnewt2.htm` Page web présentant la théorie autour de la deuxième lois de NEWTON.

`http://www.pianoteq.com` Site commercial du plug-in *Pianoteq* de la firme Modart.

`http://puredata.info` Site de développement du programme PUREDATA.

`http://pure-lang.googlecode.com/svn/docs/faust2pd.html` Documentation de *faust2pd*.

`http://qt.nokia.com` Site du framework *Qt*.

`http://www.scandalis.com/Jarrah` Site internet de Gregory Scandalis.

`http://simson.net/ref/NeXT` Site « historique » de la firme NeXT.

`http://www.sondiusxg.com` Site internet officiel du projet *Sondius*.

Enregistrements

BURTNER, Matthew, *Metasaxophone Colossus*, Matthew BURTNER, 1 CD Innova Records 620 (enreg : 2004).

EDWARDS, Michael, *Stryngbite*, 1 CD Sumtone stcd1 (enreg : 1997 ; R/2003).

JAFFE, David, *XXIst Century Mandolin*, JAFFE David, 1 CD Well-Tempered Productions WTP5164 (enreg. : 1982).

LANSKY, Paul, *A Computer Opera*, lectrice : MACKAY Hanna, 1CD Bridge Records BR 9076 (enreg : 1995).

MATHEWS, Max, *Music from Mathematics*, 1CD Decca DL 9103 (enreg. : 1960 ; R/1962).

TILLMAN, Don, *Vantage Point*, Tesseract, 1 CD Don TILLMAN (enreg : 1997).

Vidéos

HERNANDEZ, Edgar, *Yamaha VL1 Demostracion*, disponible sur YouTube à l'adresse suivante : `http://youtu.be/lX9l1LSw8gY` et sur le CD dans le fichier `VL1-comp.mp4` contenu dans le dossier `/VL1/`.

NORDIN, Hud, *HighTech Heroes n° 6 : Julius Smith and David Jaffe*, Montain View (US), 1983, disponible sur YoutTube à l'adresse suivante : `http://youtu.be/15jG1zfx-IM` et sur le CD dans le fichier `High-Tech-Hereos-6.mp4` contenu dans le dossier `/videos/`.

OLCZAK, George, *David Jaffe – Silicon Valley Breakdown*, interview de David Jaffe, 1984, disponible sur YouTube à l'adresse suivante : `http://youtu.be/_p4DGE5t3x0` et sur le CD dans le fichier `JAFFE-Interview.mp4` contenu dans le dossier `/videos/`.

Index des fichiers du CD-ROM

`/allpassnn` :

Ce dossier contient l'ensemble des éléments relatifs à l'objet FAUST `allpassnn.dsp` : tests de stabilités et exemples d'utilisations avec un modèle de clarinette, un modèle de corde et un simple signal sinusoïdal.

`/audio` :

L'ensemble des fichiers audios non liés à des modèles physiques d'instruments de musique peuvent être trouvés dans ce dossier.

`/autres` :

Certains des instruments du FAUST-STK dont les algorithmes ne sont pas basés sur la modélisation physique par guides d'ondes sont disponibles dans ce dossier.

`/basse` :

Les éléments relatifs à l'instrument `bass.dsp` peuvent être trouvés dans ce dossier.

`/bouteille` :

Les éléments relatifs à l'instrument `blowBottle.dsp` peuvent être trouvés dans ce dossier.

`/clarinette` :

Les éléments relatifs à l'instrument `clarinette.dsp` peuvent être trouvés dans ce dossier.

`/clarinette2` :

Les éléments relatifs à l'instrument `blowHole.dsp` peuvent être trouvés dans ce dossier.

`/csound` :

Ce dossier contient un programme CSOUND implémentant plusieurs modèles physiques d'instruments de musique ainsi que le fichier audio qu'il permet de générer.

`/brass` :

Les éléments relatifs à l'instrument `brass.dsp` peuvent être trouvés dans ce dossier.

`/eks` :

Les éléments relatifs à l'instrument `eks.dsp` peuvent être trouvés dans ce dossier.

`/flute` :

Les éléments relatifs à l'instrument `flute.dsp` peuvent être trouvés dans ce dossier.

`/fluteSTK` :

Les éléments relatifs à l'instrument `fluteSTK.dsp` peuvent être trouvés dans ce dossier.

`/harpsi` :

Les éléments relatifs à l'instrument `harpsi.dsp` peuvent être trouvés dans ce dossier.

`/ks` :

Les éléments relatifs à l'instrument `ks.dsp` peuvent être trouvés dans ce dossier.

`/mesh` :

Les éléments relatifs à l'instrument `mesh.dsp` peuvent être trouvés dans ce dossier.

`/muneira` :

Ce dossier contient les éléments relatifs à la mise en place du modèle de violon `violin.dsp` ainsi que le matériel utilisé pour mener à bien les expériences de contrôle de cet instrument avec des données de suivis de gestes.

`/nlfFdnRev` :

Les éléments relatifs à l'instrument `nlfFdnRev.dsp` peuvent être trouvés dans ce dossier.

`/nlfk1k2` :

Les éléments relatifs à l'instrument `ksk1k2.dsp` peuvent être trouvés dans ce dossier.

`/piano` :

Les éléments relatifs à l'instrument `piano.dsp` peuvent être trouvés dans ce dossier.

`/saxophone` :

Les éléments relatifs à l'instrument `saxophony.dsp` peuvent être trouvés dans ce dossier.

`/sitar` :

Les éléments relatifs à l'instrument `sitar.dsp` peuvent être trouvés dans ce dossier.

`/synthbuilder` :

Des exemples sonores produits par des algorithmes de modèles physiques implémentés dans le programme SynthBuilder sont stockés dans ce fichier.

`/util` :

Ce dossier contient les éléments nécessaires à la compilation et à l'utilisation des exemples proposés dans certains dossiers du CD tel que les fichiers MIDIs utilisés pour produire les exemples audios, les bibliothèques Faust et C++ relatives au Faust-STK, les *MakeFiles* pour compiler les objets Faust et certains sous-patchs PureData relatifs à l'utilisation de *faust2pd*.

`/videos` :

Certaines des vidéos citées dans ce mémoire sont disponibles dans ce dossier.

`/violon` :

Les éléments relatifs à l'instrument `sitar.dsp` peuvent être trouvés dans ce dossier.

`/vl1` :

Des documents audios et vidéos sur le synthétiseur *VL1* de Yamaha sont disponible dans ce dossier.

Index

A

Abel, Jonathan 74
Adrien, Jean-Marie 6, 69, 93, 109
Alembert, Jean le Rond (d') 10
Askenfelt, Anders 35

B

Bank, Balázs 66
Bistrow, David 55
Bonada, Jordi 81
Bornhöft, Achim 113
Bran-Ricci, Josiane 1
Burtner, Matthew 110, 111

C

Cadoz, Claude 5, 6, 119
Castagne, Nicolas 5, 6, 119
Caussé, René 93
Chaffe, Chris 109
Chaigne, Antoine 5, 35, 100
Chowning, John 55
Clark, Paul 110
Coakley, William 25
Cook, Perry 22, 28, 81, 91, 93

D

Depalle, Philippe 6

E

Eckel, Gerhard 93
Edwards, Michael 113
Essl, Georg 93

F

Fant, Gunnar 4
Fletcher, Neville 35, 36, 41, 72
Fleury, Gilles 6
Florens, Jean-Loup 6
Fober, Dominique 45, 64
Fontana, Frederico 66

G

Gray, Augustine 43, 47
Gurevich, Michael 29, 103

H

Hart, Ruth 111
Helmotz, Hermann (Von) 3
Hiller, Lejaren 5
Hufshmitt, Aline 22

I

Iovino, Fransisco 93

J

Jaffe, David 18, 26, 108
Jot, Jean-Marc 100

K

Karjalainen, Matti 20, 41, 63, 72, 86
Karplus, Kevin 15
Kelly, John 4, 22, 46
Kojs, Juraj 112, 113

L

Laakso, Timo 20, 63, 72, 86
Laine, Unto 20, 63
Lansky, Paul 110
Lazarus, Francis 6
Legge, Karl 36
Letz, Stéphane 45, 64
Lochbaum, Carol 4, 22, 46
Longchamp, Jacques 108
Luciani, Annie 6

M

Maestre, Esteban 74, 75, 78, 81
Malham, David 3
Markel, John 43, 47
Massie, Dana 47
Matthews, Max 4
Mcintyre, Michael 5, 174, 175
Michon, Romain 94, 100, 119
Miller, Dayton 3
Moore, Richard 117
Morison, Joseph 6, 93

N

Nebot, Josep 29

Newton, Isaac 3

O

Olczak, George 108
Olson, Harry 4
Orlarey, Yann 45, 64

P

Pierce, John 35, 36
Porcaro, Nick 26
Poynting, John 3

R

Rayleigh, John William Strutt 3
Reid, Gordon 33
Roads, Curtis 2, 3
Rodet, Xavier 6
Rossing, Thomas 35, 41, 72, 117
Rowland, Nicholas 33
Ruiz, Pierre-Michel 5
Russ, Martin 31, 33, 118

S

Scandalis, Gregory 26
Scavone, Garry 85, 88, 111, 112
Scavone, Gary 85, 88, 111
Serafin, Stefania 112
Shumacher, Robert 5, 175
Smith, Julius . 1–3, 6, 11, 12, 14, 15, 18, 21, 24, 26, 28, 35, 44–47, 63, 66, 71, 74, 85, 89, 92–94, 97, 98, 100, 101, 107, 114, 119, 128
Stevens, Ken 4
Steward, John 4
Stilson, Tim 26
Strong, Alexander 15
Strube, Hanz 75

T

Thomson, Joseph 3
Tillman, Don 114
Tolonen, Tero 41

V

Valimaki, Vesa 20, 63, 72, 86
Van Duyne, Scott 26, 35, 36, 94
Van Walstijn, Maarten 85
Vercoe, Barry 29
Verge, Marc-Pierre 83
Vergez, Christophe 6

W

Waxman, David 6, 93
Weinreich, Gabriel 68
Wheeler, Paul 117
Woodhouse, James 5, 175

Z

Zambon, Stefano 66

Table des matières

Introduction **1**

1 Principe de fonctionnement et outils existants **10**
1.1 Modèle théorique d'une corde en vibration avec la technique des guides d'ondes numériques . . . 10
1.1.1 Equation d'onde d'une corde idéale de longueur infinie . . . 11
1.1.2 Corde avec des terminaisons rigides . . . 12
1.1.3 Amortissement de l'énergie contenue dans la corde . . . 14
1.2 L'algorithme de KARPLUS-STRONG . . . 14
1.3 Améliorations apportées à l'algorithme de KARPLUS-STRONG dans le cadre des travaux sur la synthèse par guides d'ondes . . . 18
1.4 Application à d'autres types d'instruments . . . 21
1.5 Outils existants . . . 24
1.5.1 De SYNTHBUILDER au SYNTHESIS TOOLKIT . . . 26
1.5.2 Applications commerciales . . . 30

2 Implémentation de guides d'ondes non-linéaires **35**
2.1 Techniques existantes . . . 36
2.1.1 Filtre passe-tout du premier ordre avec modulation binaire du coefficient . . . 36
2.1.2 Modulation aléatoire de la longueur de la ligne de retards . . . 40
2.2 Filtre passe-tout passif non-linéaire d'ordre arbitraire . . . 42
2.2.1 Définition . . . 43
2.2.2 Stabilité énergétique . . . 48
2.3 Utilisation du filtre passe-tout passif non-linéaire d'ordre arbitraire . . . 49
2.3.1 La fonction `nonLinearModulator` du FAUST-STK . . . 51
2.3.2 Test de base sur un signal sinusoidal . . . 52
2.3.3 Utilisation sur des modèles physiques par guides d'ondes . . . 54

3 Implémentation de modèles physiques d'instruments de musique **61**
3.1 Gérer les modèles d'instruments polyphoniques dans FAUST . . . 61
3.2 Instruments à cordes pincées et frappées . . . 62
3.2.1 Clavecin . . . 63
3.2.2 Piano . . . 65
3.2.3 Guitare basse . . . 69
3.3 Instruments à cordes frottées . . . 71
3.3.1 Modèle physique de base d'instruments à cordes frottées . . . 71
3.3.2 Modèle physique avancé de violon . . . 74
3.3.3 Utilisation de données de suivi de gestes . . . 77
3.4 Instruments à vent . . . 81

3.4.1 Flûtes traversières . . . 81
3.4.2 Clarinettes . . . 84
3.4.3 Saxophone . . . 86
3.4.4 Cuivres . . . 89
3.5 Percussions . . . 92
3.5.1 Réseaux multi-dimensionnels de guides d'ondes : « filets » de guides d'ondes . . . 94
3.5.2 Réverbérateur non-linéaire basé sur un réseau de chaînes de retards récursives . . . 98
3.6 Optimisation et performances des instruments du FAUST-STK . . . 103
3.6.1 Comparaison de la taille du code C++ du SYNTHESIS TOOLKIT avec le code FAUST du FAUST-STK . . . 103
3.6.2 Comparaison des performances . . . 103

4 Utilisation de la synthèse par guides d'ondes numériques dans le répertoire musical depuis sa création 106
4.1 Contexte . . . 106
4.2 L'œuvre pionnière : *Silicon Valley Breakdown* . . . 107
4.3 Utilisation dans le répertoire de musique contemporaine . . . 109
4.3.1 Pièces utilisant des instruments du SYNTHESIS TOOLKIT . . . 109
4.3.2 Pièces utilisant des modèles sur mesure . . . 112
4.4 Utilisation dans le répertoire des musiques actuelles . . . 113

Conclusion 115

A Note technique 124

B Description du fichier `ks.dsp` 126

C Description du fichier `eks.dsp` 128

D Description du fichier `clarinette.dsp` 131

E Description du fichier `blowHole.dsp` 134

F Description du fichier `saxophony.dsp` 138

G Description du fichier `sitar.dsp` 141

H Description du fichier `harpsi.dsp` 143

I Description du fichier `piano.dsp` 146

J Description du fichier `bass.dsp` 153

K Description du fichier `flutestk.dsp` 155

L Description du fichier `flute.dsp` 158

M Description du fichier `brass.dsp` 160

N Description du fichier `bowed.dsp` 162

O Description du fichier `violin.dsp` 165

P Description du fichier `mesh.dsp` 169

Q Description de la fonction `nonLinearModulator` de `instrument.lib` 172

R Description des fonctions importantes de `instrument.lib` 174

Bibliographie 177
Monographies – Articles 177
Sites Internet 183
Enregistrements 184
Vidéos 185