Numéros / n° 3 - automne 2013

« La bibliothèque de spatialisation HOA pour Max/MSP, Pure Data, VST, FAUST... »

Pierre Guillot, Eliott Paris et Manuel Deneu

Résumé

Dans cet article, nous présentons la bibliothèque HOA, un ensemble de classes C++ dédié à la spatialisation du son et fondé sur des techniques ambisoniques d'ordre supérieur. A travers les mises en oeuvre pour les logiciels Max/MSP et Pure Data, sous forme d'un *plugin* VST ainsi que le portage en langage FAUST, nous mettons en avant les difficultés rencontrées, les améliorations réalisées ainsi que les avantages de chaque plateforme logicielle pour notre approche particulière de l'espace sonore.

1. Introduction

La bibliothèque HOA est un ensemble de classes C++ destiné à l'ambisonie d'ordre supérieur, résultat issu de deux projets du Labex Arts-H2H de l'université Paris 8, « La spatialisation du son pour le musicien, par le musicien » et « Des interfaces pour la mise en espace du son » qui se sont déroulés au CICM [(1)]. Ces recherches proposent une approche musicale des techniques ambisoniques en s'appropriant les fondements scientifiques de l'ambisonie. La bibliothèque vise à offrir à la fois des traitements de l'espace et du son pertinents sur un plan artistique et un ensemble d'interfaces graphiques de représentation et d'interaction, facilitant pour le musicien la compréhension des principes sous-jacents à l'ambisonie ainsi que le contrôle et la manipulation de champs sonores. L'enjeu était de proposer une première mise en oeuvre utilisable par les musiciens et nous permettant d'expérimenter et d'approfondir nos recherches (Colafrancesco *et al.*, 2013) qui, suite à nos pratiques et à la demande en flexibilité de la plateforme de déploiement, a été réalisée pour le logiciel Max/MSP. Après ce premier livrable concluant sur bien des aspects, il était nécessaire d'élargir nos propositions logicielles afin d'offrir ces outils à une communauté plus large de musiciens et de développeurs, et de voir émerger de nouveaux usages et de nouvelles pratiques. Avec la nouvelle version de la bibliothèque HOA pour Max/MSP [(2)], nous proposons donc, à présent, une mise en oeuvre pour le logiciel Pure Data [(3)], des *plugins* VST [(4)] ainsi qu'un portage de la bibliothèque en langage FAUST [(5)].

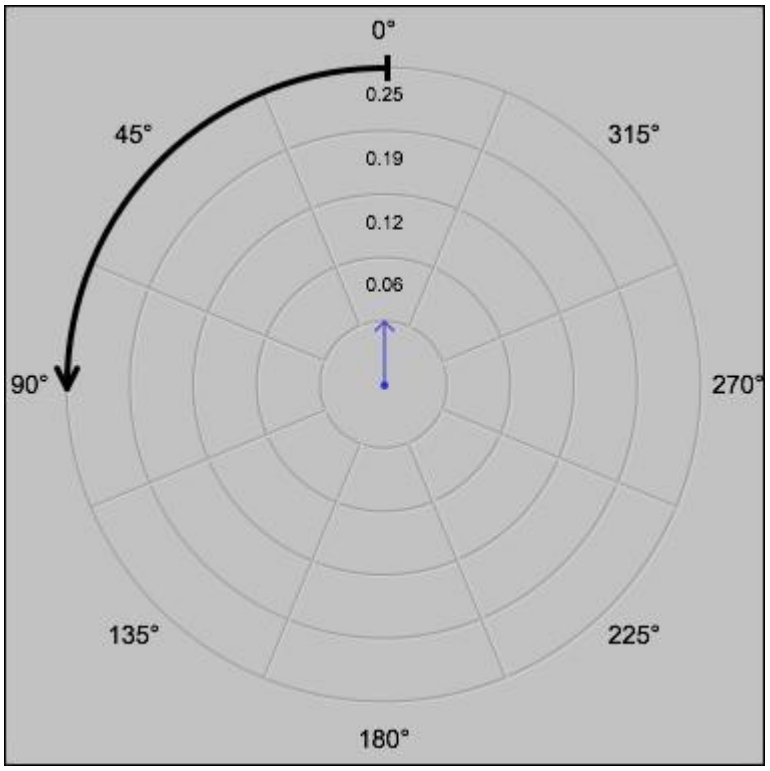
Nous présentons dans un premier temps l'architecture du code C++ de la bibliothèque HOA afin de préciser les choix que nous avons faits et les contraintes auxquelles nous avons dû faire face pour créer des classes facilement adaptables sur différentes plateformes logicielles. Par la suite, nous revenons sur les différentes mises en oeuvre réalisées afin de présenter les fonctionnalités disponibles dans chacune d'elles et les avantages qu'elles offrent, mais aussi de mettre en avant les difficultés liées à ces plateformes et les changements que nous avons dû opérer sur les fondements de la bibliothèque. Enfin, nous soumettons les améliorations que nous souhaitons réaliser ainsi que les futurs développements que nous envisageons.

2. Architecture de la bibliothèque HOA

Offrir un ensemble de classes facilement portable sur plusieurs systèmes d'exploitation pose certains problèmes d'ordre technique pour la bibliothèque HOA. En effet, étant donné que l'ambisonie d'ordre supérieur nécessite un grand nombre de calculs parallèles, dans de nombreuses situations les opérations deviennent peu adaptées aux ressources des ordinateurs personnels et il devient vite nécessaire

d'optimiser les traitements. Notre approche s'est portée sur l'utilisation des bibliothèques extérieures proposées dans le framework *Accelerate* d'Apple (6) ou dans la suite d'Intel *Composer XE* (7) qui sont particulièrement adaptées au traitement du signal. L'exemple le plus probant de cet usage est la mise en oeuvre du décodage binaural, impossible à réaliser pour des ordinateurs personnels sans des optimisations de calcul matriciel (Colafrancesco *et al.*, 2013). Ces bibliothèques possèdent néanmoins certaines contraintes. Le framework *Accelerate*, bien que gratuit, est développé par Apple et ne fonctionne que sur le système d'exploitation des ordinateurs Macintosh. Les bibliothèques d'Intel, quant à elles, fonctionnent sur la majorité des systèmes d'exploitation mais présentent un conflit avec notre approche par le fait qu'elles sont payantes. Afin de remédier à cela, nous avons commencé à implémenter des fonctions de bibliothèques de calcul multiplateformes, libres et gratuites, telles BLAS (8) ou FFTW (9). Cette approche nous a ainsi amené à réaliser un ensemble de *macros* pour des opérations de bases (10) qui font appel aux différentes bibliothèques afin de répondre à un ensemble de contextes dépendants des systèmes d'exploitation, du type de processeur, des optimisations que l'on souhaite réaliser ou du choix des licences, libres et gratuites ou propriétaires.

Figure 1.



Représentation de l'espace dans la bibliothèque HOA. Le sens de rotation est antihoraire (flèche noire) et le 0° est déphasé de 90° par rapport à une représentation mathématique d'un cercle. L'auditeur est généralement orienté vers 0° (flèche bleue).

De manière plus générale, permettre à une bibliothèque réalisant à la fois du traitement du signal et des interfaces graphiques d'interaction et de représentation d'être facilement portée pour servir de greffon à des plateformes logicielles, nécessitent de prendre en compte les particularités des kits de développement associés à ces logiciels. Étant donné que les architectures des processeurs et les architectures de traitement du signal varient, l'ensemble de nos fonctions se doivent d'offrir la possibilité d'effectuer les traitements avec une précision de *double* ou de *float*, de réaliser les opérations échantillon par échantillon ou sur un bloc d'échantillons mais aussi de donner la possibilité de contrôler certaines variables par du signal (11). Plus qu'une réelle contrainte, adapter les fonctions à ces caractéristiques permet de réaliser de nombreuses optimisations sur les traitements en tirant parti des bibliothèques de calcul. Cependant, il faut noter que certaines opérations perdent en clarté et deviennent complexes à envisager au regard des buts escomptés. De plus, il reste toujours certaines spécificités à chaque logiciel qui amènent à réaliser certaines opérations complémentaires lors des mises en oeuvre comme la recopie du signal, lorsque les chaînes de traitement utilisent les mêmes signaux en entrée qu'en sortie ; en effet nos algorithmes sont en général *out-of-place*. Les outils permettant de créer des représentations graphiques et des interfaces utilisateurs varient, eux aussi, selon les logiciels. Ils ont, en outre, des échelles de représentation de

l'espace très différentes où les ordonnées sont incrémentées du haut vers le bas ou inversement, où le 0 radian est à droite ou en haut, etc. Ainsi, nous avons fait le choix de fixer un modèle qui nous semble clair et pertinent pour une approche musicale (12) (figure 1) auquel doivent s'adapter les mises en oeuvre.

3. Mises en oeuvre et portage de la bibliothèque HOA

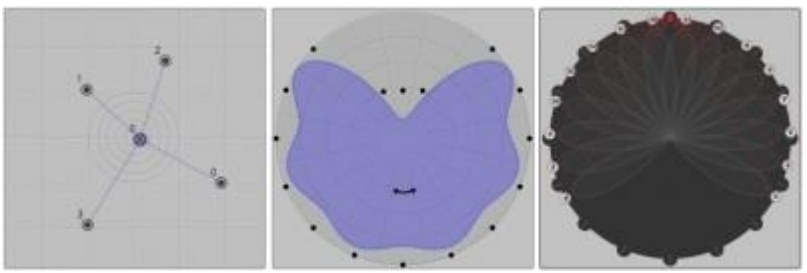
La version actuelle de la bibliothèque HOA offre donc l'avantage de répondre simplement à de multiples conditions de mise en oeuvre. Nous revenons donc à présent sur celles que nous avons réalisées pour les logiciels Max/MSP, Pure Data et sous forme de VST afin de présenter les résultats de ces applications, les aspects essentiels de ces élaborations qui ont permis de faire évoluer cette bibliothèque et les contraintes auxquelles nous avons dû faire face. Nous présenterons, par la suite, la création de la bibliothèque HOA en langage FAUST, qui offre de nombreux avantages, tant par les optimisations que par sa capacité à répondre à de nombreuses conditions de mise en oeuvre.

3. 1. Max/MSP

La première mise en oeuvre de la bibliothèque HOA s'est faite pour le logiciel de programmation graphique Max/MSP. Outre le fait, non-négligeable, que cet environnement logiciel soit utilisé par une large communauté de musiciens et de compositeurs, le choix de ce logiciel a surtout été motivé par l'approche modulaire qui le caractérise et par une API (13) assez complète et bien pensée. L'architecture de la chaîne de traitement du signal, offrant des possibilités d'opérations *in-place* ou *out-of-place*, est pleinement compatible avec la nôtre. Au niveau des fonctions graphiques, le système de calques et de fonctions de transformations matricielles a permis de faciliter le développement et d'optimiser le rendu graphique tout en augmentant la lisibilité du code. Pour ces raisons, ce logiciel s'est révélé être pour nous un terrain idéal au prototypage, à l'expérimentation et à la création de nouveaux traitements au cours de l'élaboration de la bibliothèque HOA. Plusieurs adaptations ont dû néanmoins être faites pour faciliter la mise en place de ces traitements et améliorer l'ergonomie de cette mise en oeuvre logicielle.

En effet, le nombre d'entrées et sorties à connecter entre chaque objet est en fonction de l'ordre de décomposition ambisonique, plus celui-ci est élevé, plus le nombre de cordes à connecter augmente. De plus, effectuer un traitement dans le domaine des harmoniques circulaires se résume le plus souvent à appliquer ce même traitement en parallèle sur différents canaux. Or, les spécificités d'un environnement de programmation graphique tel que Max/MSP ne facilitent pas ce type de mise en place. Nous nous retrouvons à devoir effectuer un grand nombre de manipulations répétitives telles que l'instanciation et la connexion d'objets. Afin de faciliter ces interventions, nous avons développé les objets *hoa.plug~* et *hoa.connect*. Un même traitement, créé sous forme d'un patch, peut être appliqué à l'ensemble des canaux ambisoniques et variabilisé selon l'ordre et l'indice des harmoniques circulaires au sein de l'objet *hoa.plug~*. L'objet *hoa.connect* permet quant à lui de faciliter les connections entre les objets en automatisant cette tâche. Ces outils, créés spécifiquement pour l'environnement de programmation Max/MSP, nous ont permis de tester et de faire émerger facilement des traitements sonores pertinents dans le domaine des harmoniques circulaires tels que la granulation, la modulation d'amplitude ou encore la décorrélation microtemporelle, qui auraient été beaucoup plus laborieux à mettre en place sans cela.

Figure 2.



Trois interfaces de contrôle de la bibliothèque HOA pour Max/MSP, de gauche à droite : les objets

hoa.map, *hoa.space*, *hoa.recomposer*.

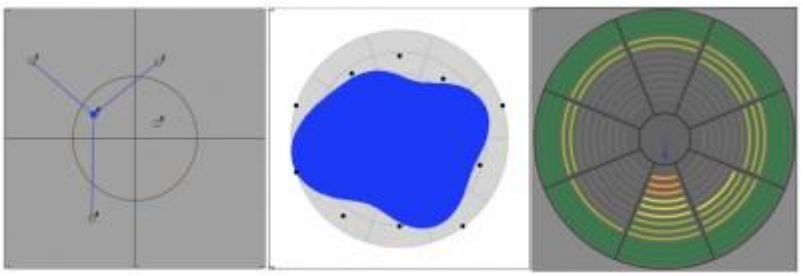
La dernière version de la bibliothèque HOA pour Max/MSP intègre plusieurs nouveautés. Une nouvelle interface d'édition de trajectoire associée à un moteur sonore permet désormais une gestion temporelle des positions spatiales d'un ensemble de sources ponctuelles sur un espace à deux dimensions (figure 2). La réverbération dans le domaine des harmoniques circulaires a été repensée grâce à l'amélioration de la réverbération par convolution et la mise en oeuvre d'une réverbération algorithmique de type Freeverb adaptée à l'approche ambisonique. De nouveaux traitements sont de même disponibles dans le domaine des ondes planes comme le filtrage spatial (Colafrancesco *et al.*, 2013) ou la déformation d'un champs sonore par déplacement de microphones virtuels, facilités par des interfaces interactives de contrôle qui aident à leur manipulation (figure 2). Nous proposons, à présent, une version améliorée du décodeur adaptée à des configurations particulières des systèmes de restitution (ambisonique, binaural, stéréophonique, 5.1, 7.1, etc.). En tirant parti des avancées présentes dans les dernières versions du logiciel Max/MSP (notamment la création d'entrées et de sorties dynamiques) nous avons pu aussi améliorer l'ergonomie de notre bibliothèque et notamment offrir des outils plus flexibles aux changements de configuration de haut-parleurs.

3. 2. Pure Data

La mise en oeuvre de la bibliothèque HOA pour Pure Data ouvre le champ d'une utilisation plus large, alternative, différente. En plus du fait qu'il soit libre et gratuit, l'immense intérêt de Pure Data réside dans sa compatibilité avec d'autres systèmes d'exploitation que Windows ou Mac, et en particulier des environnements Unix libres. Il devient alors possible d'utiliser les objets de la bibliothèque HOA dans un contexte de systèmes embarqués tels que le récent *Raspberry-Pi* (14), qui propose à prix très bas un ordinateur de la taille d'une carte de crédit que l'on peut très facilement intégrer dans une installation (15). La mise en oeuvre pour le logiciel Pure Data a nécessité certains aménagements, qui se sont révélés bénéfiques pour la bibliothèque HOA.

Les logiciels Pure Data et Max/MSP possèdent des architectures et des fonctionnements extrêmement similaires pour le traitement du signal. Ainsi, hormis, la nécessité de procéder à une recopie des signaux due à des vecteurs d'entrée et de sortie identiques, la mise en oeuvre des objets de traitement du signal a pu être réalisée facilement. La réelle difficulté réside dans le portage des traitements réalisés sous forme de patchs pour l'objet *hoa.plug~* dans Max/MSP, tels que la granulation ou la décorrélation microtemporelle. En effet, le code de l'objet *hoa.plug~* est totalement dépendant de cette plateforme logicielle Max/MSP. Or dans Pure Data, il existe des différences du kit de développement, notamment pour créer des sous-patchs et des objets via un code C. La création d'un tel objet dans Pure Data devra encore faire l'objet d'une étude de faisabilité. La solution proposée pour cette mise en oeuvre a consisté à recréer les traitements réalisés par les patchs Max/MSP en langage C++. Nous proposons ainsi des classes permettant la synthèse granulaire quasi-synchrone, la décorrélation microtemporelle variable sans effet de *flanger* et la m

Figure 3.



Trois interfaces de contrôle de la bibliothèque HOA pour Pure Data, de gauche à droite : les objets *hoa.map*, *hoa.space* et *hoa.meter*.

La partie interface de la bibliothèque, et particulièrement les objets graphiques ont nécessité certains aménagements. Pour gérer l'affichage d'un patch et de ses composants (objets, cordes, etc.), Pure Data

utilise *Tcl/Tk* [17], la combinaison d'un langage de script multiplateforme, *Tcl* pour *Tool Command Language*, et d'une bibliothèque de création et de gestion d'interfaces graphiques, *Tk* (Puckette, 1996). Afin de définir la manière dont l'objet doit être affiché dans la fenêtre d'un patch, Pure Data communique avec un autre programme gérant l'affichage, X11 [18], via un système lignes de commandes. Ainsi, il est possible de créer des formes, de détecter une action de l'utilisateur, etc. Cependant, l'ensemble des objets créés n'est pas contrôlé par Pure Data, il appartient alors au développeur de ne pas dessiner en dehors de l'objet, ou encore de faire attention de bien effacer l'ensemble de son interface lors de la destruction de l'objet. De plus, la manière de dessiner dans l'objet est différente : là où des logiciels comme Max/MSP permettent au développeur de très rapidement redessiner un calque à la volée, *Tcl/Tk* n'offre pas les mêmes performances et doit donc garder la trace de chaque entité dessinée, pour ensuite aller la déplacer, la transformer, etc.

En ce qui concerne les interactions avec l'utilisateur, Pure Data diffère aussi de Max/MSP par le fait qu'il ne donne accès qu'à très peu de commandes du clavier et de la souris et qu'il n'est pas possible par exemple de récupérer le clic droit pour faire apparaître un autre menu contextuel autre que celui proposé par défaut par Pure Data. Ainsi certaines commandes ont dû être repensées par rapport à Max/MSP, afin de les adapter au mieux à l'environnement de Pure Data. D'une manière générale, les fonctions décrivant la manière de présenter et de gérer les données à l'écran sont présentes dans le code C++ de la bibliothèque HOA, et le portage ne nécessite qu'une adaptation aux fonctions de dessin de *Tk*.

Dans l'état actuel de la mise en oeuvre, nous offrons la majorité des traitements de l'espace et du son, ainsi que les interfaces graphiques de spatialisation de sources ponctuelles, de filtrage spatial et le VU-mètre circulaire (figure 3).

3. 3. *Plugin* VST

Nous réalisons actuellement le portage d'une série de nos traitements et outils de mise en espace du son créés au sein de la bibliothèque HOA sous la forme d'un *plugin* audio. Ce choix a été fait pour s'adapter aux pratiques des musiciens habitués à utiliser des logiciels de type *time-line* plutôt que des environnements de programmation de type Max/MSP ou Pure Data pour leurs créations. Le principal problème que pose une entreprise comme celle-ci réside dans le fait qu'un *plugin* doive fonctionner et répondre au niveau du rendu sonore, des automations de paramètres ou de son comportement, de la même manière au sein d'un maximum de logiciels hôtes.

Pour s'adapter aux spécificités de chacun d'eux et à toutes les plateformes tout en gardant une certaine intelligibilité du code, nous avons fait le choix d'utiliser la bibliothèque de classes C++ JUCE [19], qui a l'avantage d'être multiplateforme et distribuée sous licence GPL. Elle permet de compiler des *plugins* au format *VST*, *AudioUnit* [20] ou *RTAS* [21] pour les systèmes d'exploitation Windows, Mac OS et Linux, grâce à un seul et même code source, ce qui représente pour nous un gain de temps de développement considérable. Cette bibliothèque, bien conçue et puissante, met en effet à la disposition du programmeur une série de classes et de fonctions lui facilitant le développement. Des fonctions utilitaires permettant la création d'interfaces graphiques, la communication entre celles-ci, le moteur audio et l'hôte ou encore la sauvegarde des paramètres ont été élaborées afin de s'adapter à l'ensemble des formats de *plugin*. Des modules viennent de plus compléter cette bibliothèque, notamment l'extension *pluginParameter* [22] qui permet de créer des paramètres de contrôle qui s'ajustent automatiquement aux nécessités des différents formats.

Nous avons fait le choix de travailler avec le logiciel *Reaper* [23] pour élaborer et tester nos *plugins*. Celui-ci s'est révélé assez bien conçu pour le traitement multicanal, nous laissant une grande flexibilité dans le choix de la configuration audio grâce à une matrice de *routing* présente à la fois de manière générale mais aussi spécifiquement au sein de chaque *plugin*. Il a été intéressant de constater que d'autres DAW [24], tels que *Live* [25], *Audacity* [26], n'étaient pas assez bien pensés pour le multicanal, et encore moins pour la spatialisation ambisonique ? le logiciel *Adobe Audition* [27] ne proposent par exemple pas de configuration libre des canaux, mais seulement des configurations de sortie pour des formats précis (ex. stéréo, 5.1, 7.1.).

Figure 4.

Le *plugin HoaMeter* et *HoaMap* au sein du logiciel *Reaper*.

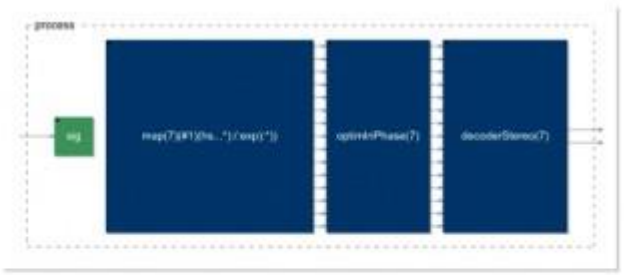
La bibliothèque HOA dans sa version *plugin* (figure 4) dispose d'un VU-mètre circulaire qui aide à la visualisation de la contribution de chacun des haut-parleurs, et d'un module d'encodage multiple de sources sonores avec compensation de la distance et décodage ambisonique intégré. Nous envisageons comme perspective de développement future de porter un maximum d'autres traitements de la bibliothèque HOA (granulation, delay, filtrage spatial de champs sonore...) sous forme de *plugins* indépendants ou au sein d'un seul et même *plugin* complet.

3. 4. FAUST

Suite à la première version de la bibliothèque HOA, l'équipe de recherche du GRAME [(28)] et le CICM ont entrepris d'étudier la possibilité d'un portage de la bibliothèque du langage C++ vers le langage FAUST [(29)]. Cette entreprise a été concluante et a permis une première ébauche de bibliothèque ambisonique d'ordre supérieur pour FAUST, déjà disponible dans la distribution. Ainsi, il est possible de réaliser l'ensemble des traitements de base tels que l'encodage, le décodage, l'ordre factionnaire, la compensation de la distance et les optimisations du champ sonore [(30)] dans ce langage qui offre de nombreux avantages.

Ce portage complète en effet les lacunes actuelles de la bibliothèque HOA liées à ses dépendances aux bibliothèques de calcul qui rendent impossible la compilation de certains traitements sous une licence gratuite pour Windows ou Linux. Le langage FAUST et son compilateur sont libres et gratuits et permettent de compiler les codes pour un panel de systèmes d'exploitation variés tels que Windows, Mac OS, Linux ou encore Android (Letz *et al.*, 2013). Il offre, de plus, la possibilité de compiler les traitements pour un grand nombre de plateformes logicielles telles que SuperCollider, Csound, Max/MSP, Pure Data ou sous forme d'applications autonomes de type *JackAudio* [(31)] ou *Qt* [(32)]. L'utilisateur désireux de réaliser un programme Pure Data fonctionnant sur Linux peut ainsi, par exemple, utiliser FAUST afin de créer certains traitements, comme les optimisations qui ne sont pas encore implémentés en C++ avec des fonctions utilisant une bibliothèque libre et gratuite.

Figure 5.




```
import("filter.lib");
import("hoa.lib") ;

radius = hslider("Source Radius", 1., 0, 10, 0.001) : smooth(tau2pole(0.02));
angle = hslider("Source Angle", 0., -2*PI, 2*PI, 0.001) : smooth(tau2pole(0.02));

process(sig) = map(7, sig, x1, y1) : optimInPhase(7) : decoderStereo(7);
```

Création d'un traitement ambisonique réalisant un encodage avec compensation de la distance, application d'une optimisation dite *inPhase* et d'un décodage stéréophonique en FAUST pour un ordre de décomposition 7.

La syntaxe fonctionnelle de FAUST permet de concevoir un code concis et clair mettant en valeur les opérations mathématiques sous-jacentes à l'ambisonie. Grâce à cette syntaxe particulière, l'enchaînement des traitements est facilité, quelques mots suffisent à créer une chaîne de traitements complète (figure 5). Le compilateur génère, en outre, des diagrammes (Orlarey *et al.*, 2002) ainsi qu'une documentation mathématique des opérations réalisées assurant ainsi une certaine pérennité pour la compréhension des traitements mis en oeuvre (Barkati *et al.*, 2012).

De plus, grâce au compilateur et à l'interface de compilation, que ce soit FaustWorks ou la plateforme internet, les traitements deviennent très facilement portables pour de multiples logiciels et systèmes d'exploitation. De par la simplicité de compilation, nous pouvons tirer profit d'applications où la chaîne de traitement et l'ordre de décomposition sont fixes. Par cette caractéristique, le compilateur associé au langage FAUST optimise le code et offre des applications bien plus performantes dans de nombreuses mises en oeuvre (Orlarey *et al.*, 2009).

À présent, la bibliothèque HOA en FAUST offre en plus la possibilité de réaliser un décodage stéréophonique et une rotation du champ sonore. Nous pouvons envisager de nouvelles mises en oeuvre en tirant parti notamment des nombreux traitements déjà disponibles dans ce langage, comme la Freeverb ou les synthèses de la bibliothèque STK pour FAUST (Michon *et al.*, 2011), en les adaptant au domaine ambisonique. Notons néanmoins que le langage FAUST n'est pas adapté à certaines opérations comme le chargement de fichier ou les transformées de Fourier, empêchant certaines mises en oeuvre comme la réverbération par convolution ou encore le décodage binaural.

4. Conclusion

La présentation des différentes mises en oeuvre de la bibliothèque HOA pour les logiciels Max/MSP, Pure Data, ou sous forme d'un *plugin* VST a permis de mettre en valeur la capacité d'adaptation de cet ensemble de classes C++ à de nombreuses plateformes logicielles. À présent, nous proposons aux musiciens un ensemble d'outils de mise en espace du son pouvant répondre à des situations variées d'utilisation et auxquelles le portage de la bibliothèque en langage FAUST vient aussi contribuer. Par ces mises en oeuvre, nous espérons diversifier et agrandir la communauté d'utilisateurs afin de voir émerger de nouvelles pratiques et utilisations de nos outils, dont les retours d'usage influencent énormément notre approche et nos propositions. Suite à ce travail, nous souhaitons poursuivre l'amélioration de l'architecture même de la bibliothèque C++ en continuant, bien sûr, l'implémentation des bibliothèques de calcul libres et gratuites mais aussi en créant un ensemble de fonctions permettant, de manière analogue à JUCE, de concevoir facilement les interfaces graphiques pour différentes plateformes logicielles comme Max/MSP et Pure Data. Enfin, nous envisageons encore de futurs développements que ce soit sur les plateformes logicielles existantes, comme la mise en oeuvre de l'ensemble des interfaces graphiques pour Pure Data, ou pour de nouvelles plateformes, comme proposer une version openFrameworks [(33)] de la bibliothèque

ou encore réaliser des mises en oeuvre des interfaces pour tablettes tactiles.

---

1. Centre de recherche Informatique et Création musicale, EA 1572, université Paris 8, http://cicm.mshparisnord.org/

2. http://www.cycling74.com/

3. http://puredata.info/

4. http://www.steinberg.fr/no/produits/vst.html

5. Les mises en oeuvre sont téléchargeables sur le site du projet HOA : http://www.mshparisnord.fr/hoalibrary/ et le portage FAUST est disponible dans la distribution de la bibliothèque FAUST http://faust.grame.fr/

6. https://developer.apple.com/performance/accelerateframework.html

7. http://software.intel.com/en-us/intel-composer-xe

8. http://www.netlib.org/blas/

9. http://www.fftw.org/

10. Fonctions permettant des calculs comme le produit scalaire, le produit matrice ? vecteur, les opérations trigonométriques, etc.

11. Le lecteur peut se référer à l'encodage ambisonique où l'angle d'encodage peut être défini par du signal ou comme paramètre de contrôle asynchrone.

12. Pour de plus amples informations, le lecteur peut se référer aux tutoriels de la mise en oeuvre Max/MSP de la bibliothèque HOA.

13. http://cycling74.com/downloads/sdk/

14. http://www.raspberrypi.org/

15. Page d'information sur le site de Pure Data à propos de la compatibilité avec le Raspberry-Pi http://puredata.info/docs/raspberry-pi

16. Cette bibliothèque est disponible sur le site GitHub dans le répertoire CicmLibrary du CICM : https://github.com/CICM/CicmLibrary

17. http://www.tcl.tk/

18. http://www.x.org

19. http://www.juce.com/

20. http://developer.apple.com/library/ios/#documentation/AudioUnit/Reference/AudioUnit_Framework/_index.html

21. http://www.avid.com/us/partners/audio-plugin-dev-program

22. https://github.com/4drX/pluginparameters

23. http://www.cockos.com/reaper/

24. Digital Audio Workstation

25. https://www.ableton.com/

26. http://audacity.sourceforge.net/

27. http://www.adobe.com/fr/products/audition.html

28. Le centre national de création musicale de Lyon http://www.grame.fr/.

29. FAUST est un langage de programmation fonctionnel destiné à la synthèse et au traitement du signal en temps réel disponible sur le site http://faust.grame.fr/.

30. Une présentation de ces traitements de l'espace sonore est disponible dans le mémoire de Pierre Guillot (2013).

31. http://jackaudio.org/

32. http://qt.digia.com/

33. http://www.openframeworks.cc/

Pour citer ce document:
Pierre Guillot, « La bibliothèque de spatialisation HOA pour Max/MSP, Pure Data, VST, FAUST... »,
RFIM [En ligne], Numéros, n° 3 - automne 2013, Mis à jour le 12/12/2013
URL: http://revues.mshparisnord.org/rfim/index.php?id=245