La librairie abclib : un ensemble de codes Faust rassemblant 20 ans de recherche, enseignement et création en musique mixte

Alain Bonardi

LA LIBRAIRIE *ABCLIB* : UN ENSEMBLE DE CODES FAUST RASSEMBLANT 20 ANS DE RECHERCHE, ENSEIGNEMENT ET CRÉATION EN MUSIQUE MIXTE

Alain Bonardi
CICM/MUSIDANSE
alain.bonardi@univ-paris8.fr

RÉSUMÉ

Cet article présente la librairie de codes Faust *abclib* pour la musique mixte, dont la première version a été publiée en avril 2021 sur GitHub.

Constatant la difficulté à maintenir dans le temps les développements logiciels des centres de recherche en général et de notre équipe CICM / MUSIDANSE en particulier, nous avons lancé ce projet pour avant tout pérenniser les travaux menés autour de la libraire HOA (High Order Ambisonics) développée entre 2012 et 2015. Mais il s'est également agi d'expliciter, formaliser, transcrire en Faust pour transmettre 20 années de développements essentiellement sur Max et Pure Data dans le cadre de nos enseignements, nos recherches et nos créations. Le choix de Faust s'est imposé pour publier, rendre interopérables et pérennes ces codes précédents, avec une précision de rendu sonore remarquable.

Nous exposons les caractéristiques générales de la librairie *abclib* en termes de fonctionnalités musicales et d'organisation du code sur le dépôt GitHub ; nous montrons comment nous avons repris et développé le cadre ambisonique et les traitements spatiaux du projet HOA du CICM ; enfin nous montrons l'approche multicanal et concluons sur les perspectives.

1. INTRODUCTION

La thématique de la préservation des œuvres avec électronique a été largement abordée dans la communauté d'informatique musicale [7] [4] et en particulier lors des éditions précédentes des Journées d'Informatique Musicale [3]. Dans cette perspective, l'AFIM a soutenu un groupe de travail intitulé « Archivage collaboratif et préservation créative » en 2018-2019 qui a poursuivi son travail jusqu'à aujourd'hui pour mettre en place un prototype de système de dépôt collaboratif et pérenne, Antony.

À côté des questions de préservation active des œuvres avec électronique, la fragilité et la menace d'obsolescence technique des logiciels musicaux produits par les centres de recherche et de création se manifeste de plus en plus largement. Le constat est presque toujours le même : après une phase de développement et de première mise en place souvent financée par des projets de recherche (ANR, etc.), il est ensuite quasiment impossible de décrocher des subventions pour assurer au minimum l'adaptation du logiciel aux changements d'environnement technique, et encore moins ses nécessaires évolutions fonctionnelles et d'usage. L'AFIM se penche sur cette question, avec l'organisation à venir en 2021 de journées consacrées aux logiciels musicaux développés dans les centres de recherche et de création, comme enjeu de patrimoine culturel et artistique.

Au CICM / MUSIDANSE de l'Université Paris 8, nos précédents développements informatiques dans le cadre de projets de recherche sont menacés d'obsolescence. Le point de départ du travail présenté ici est le constat que la librairie logicielle HOA[1] (High Order Ambisonics) développée grâce au soutien du Labex Arts-H2H (2012-2015) [9] ne fonctionne plus sur Pure Data, l'implémentation technique de ce logiciel ayant évolué lors des publications des dernières versions (depuis la version 0.47). Il faudrait donc pouvoir financer la mise à jour de l'ensemble du code de la version Pure Data de cette librairie, qui a montré son importance dans le domaine de la mise en espace ambisonique, et a été téléchargée plus de 20 000 fois dans le monde. Malgré ces arguments, il s'avère extrêmement difficile pour un laboratoire public d'obtenir le financement d'une telle opération de mise à niveau d'un environnement logiciel.

Face à ce constat, nous avons entrepris en 2019 une première réflexion sur la question du code de HOA et de sa pérennisation, et plus largement sur la reprise de développements réalisés depuis 20 ans le plus souvent en Max ou Pure Data, dans le cadre de nos enseignements, de nos recherches et de nos créations. Comme nous enseignons[2] et utilisons le langage Faust développé par

[1] http://hoalibrary.mshparisnord.fr/

[2] En Licence 3 de la mineure Composition assistée par ordinateur de la Licence Musicologie du Département

Grame, nous nous sommes tournés vers ce paradigme comme base d'une solution d'expression explicite, interopérable et pérenne de ces développements. Le confinement sanitaire de mars 2020 a donné un véritable élan à ce projet désormais baptisé *abclib*[3], et réalisé pour l'essentiel pendant cette année pandémique. Le projet a également pour vocation d'accueillir de futurs développements logiciels en Faust menés dans le cadre de travaux de recherche-création de notre équipe. Dans le cadre de sa thèse en cours consacrée à l'écriture de la spatialisation, Paul Goutmann a déjà contribué à la bibliothèque *abclib* au niveau des choix de distributions temporelles.

Dans cet article, nous présenterons la librairie *abclib* dans sa version initiale v1.0.1 selon plusieurs perspectives : tout d'abord ses caractéristiques générales au niveau musical et en termes de structuration du code; puis plus spécifiquement ce qui concerne la reprise et parfois l'extension du traitement spatial du son en ambisonie tel que développé dans HOA; ensuite, les traitements multicanaux proposés ; enfin, nous concluons sur les perspectives.

2. PRÉSENTATION DE LA LIBRAIRIE ABCLIB

2.1. Fonctionnalités musicales

La librairie *abclib* comporte une cinquantaine de traitements différents organisés en trois groupes :

- le traitement et la synthèse sonores spatiaux dans un cadre ambisonique 2D :
    - objets généraux : *encoders*, *decoders*, *optimizers*, *scopes* ;
    - objets conçus selon une approche « géométrique » : *maps*, *mirrors*, *rotates*, générateurs de trajectoires planes ;
    - objets pour le traitement spatial du son, qui seront exposés au 3.2.
- le traitement multicanal du son :
    - lignes à retard parallèles ou en série ;
    - *flangers* parallèles ;
    - *frequency shifters* parallèles ;
    - *harmonizers* parallèles ;
    - granulateurs sur ligne à retard parallèles.
- les objets utilitaires pour la pratique de la musique mixte :
    - multi-panners de type 'Chowning' ;
    - matrices ;
    - enveloppes aléatoires de type linéaire ou cosinus ;
    - détecteur d'attaque et de décroissance ;
    - synthétiseurs : synthèse additive, synthèse soustractive, *sound coat*, *sound grain*, cloche de Risset, gouttes d'eau.

Les traitements ambisoniques sont instanciés à des ordres allant de 1 à 7 en 2D (soit un maximum de 15 harmoniques circulaires). Comme il n'est pas possible après compilation du code Faust d'adapter dynamiquement le nombre d'entrées et de sorties d'un objet, nous avons choisi de générer les objets à tous les ordres. Ainsi l'encodeur ambisonique 2D est prévu de l'ordre 1 (*abc_2d_encoder1~)* à l'ordre 7 (*abc_2d_encoder7~).*

Les traitements multicanaux sont instanciés de 1 à 16 fois. Par exemple, les multi-harmonizers sont codés de *abc_harmo1~* (1 seul harmonizer) à *abc_harmo16~* (16 harmonizers en parallèle). L'ensemble des traitements aux différents ordres ou nombres de modules en parallèle dans le cas du multicanal représente à peu près 400 objets. L'avantage de l'interopérabilité du langage Faust apparaît ici clairement, puisque ces presque 400 codes sont écrits une et une seule fois, puis compilés vers quatre cibles correspondant au choix du système d'exploitation – MacOS ou Windows, et du logiciel temps réel d'accueil – Max ou Pure Data.

2.2. Organisation du code de la librairie

La figure 1 présente la structuration du code. L'élément principal est la librairie Faust abc.lib qui regroupe toutes les fonctions Faust nécessaires à la génération des différentes cibles présentées ci-dessus. Ces fonctions comprennent :

- les fonctions élémentaires, utilisées par d'autres fonctions. Ce sont par exemple :
    - un contrôle de l'amplitude en dB : *abc_dbtogain*
    - une ligne à retard double fonctionnant en *overlapping* : *abc_overlappedDoubleDelay*
    - un granulateur sur ligne à retard : *abc_granulator*
    - etc.
- pour chaque famille de traitements est proposée une fonction Faust générique, dépendant d'un indice qui correspond soit à l'ordre d'ambisonie soit au nombre d'instances dans le cas multicanal. Les paramètres de contrôle avec leur interface sont incluses dans le code. Par exemple, les encodeurs ambisoniques sont associés à la fonction *abc_2d_encoder*, comprenant trois contrôles : *speed* (vitesse de rotation en tours par seconde), *angle* (position statique en degrés si la vitesse speed est nulle) et *returntime* (temps d'interpolation en millisecondes entre position statique et rotation)

Dans le répertoire *bashFiles* du dépôt Github sont placés les scripts shell organisés par famille de traitements et permettant de générer tous les codes Faust correspondants. Par exemple le script *encoders2dFaustCodeGeneration.sh* va permettre de générer les 7 codes Faust correspondant aux 7

Musique de l'Université Paris 8. Cours « Langages de programmation en informatique musicale ».

[3] https://github.com/alainbonardi/abclib/

encodeurs ambisoniques 2D *abc_2d_encoder1~*, …, *abc_2d_encoder7~* à partir des fonctions de la librairie Faust abc.lib. Ces codes Faust sont placés dans le répertoire *faustCodes*.

Puis nous compilons ces codes Faust vers les cibles Max (faust2max6) et Pure Data (faust2puredata). Nous adjoignons enfin aux fichiers binaires ainsi générés un ensemble de patchs d'aide et d'abstractions destinées à faciliter la prise en main des fonctionnalités de la librairie dans Max et Pure Data.

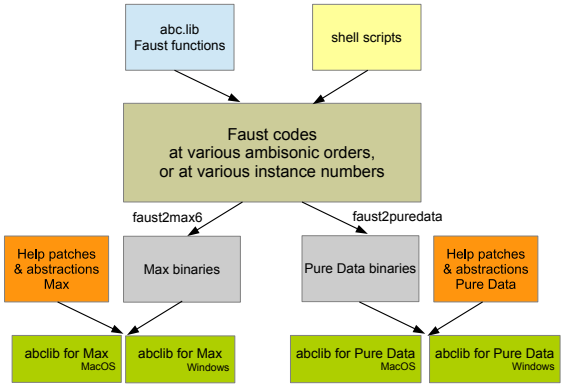

Figure 1. Structuration du code de la librairie *abclib*.

La structuration du code sous forme d'une librairie Faust (abc.lib) permet d'autres cibles de compilation et d'utilisation que Max et Pure Data, comme par exemple les applications Web.

3. LE TRAITEMENT SPATIAL DU SON

3.1. Opérations ambisoniques de base

La librairie *abclib* propose les « classiques » encodeurs et décodeurs ambisoniques 2D. Ils sont écrits à partir du code de la librairie Faust *hoa.lib* mais ont été adaptés à partir des retours d'usage observés dans l'enseignement de la mise en espace[4] aux étudiants de Master du Département de Musique de l'Université Paris 8. *abclib* prolonge l'esprit des travaux de création de HOA, celui d'une conception par et pour les musiciens [5].

Du côté de l'encodeur, si son principe est souvent compris sans difficulté, la mise en œuvre d'un angle mobile de contrôle de la position de la source, qui plus est en radians, pose souvent problème. C'est pourquoi, nous avons choisi de contrôler l'encodeur soit par une vitesse de rotation en tours par seconde – beaucoup plus parlante pour un musicien, soit si cette dernière vaut zéro, par un angle donné en degrés[5]. La figure 2 montre l'implémentation actuelle du contrôle de l'encodeur avec HOA dans Max, et la figure 3 celle réalisée avec *abclib* toujours dans Max.

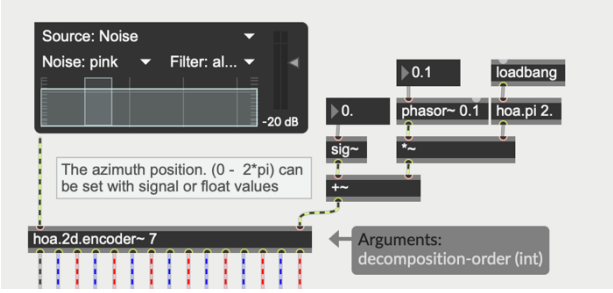

Figure 2. Contrôle de l'encodeur dans HOA (dans Max).

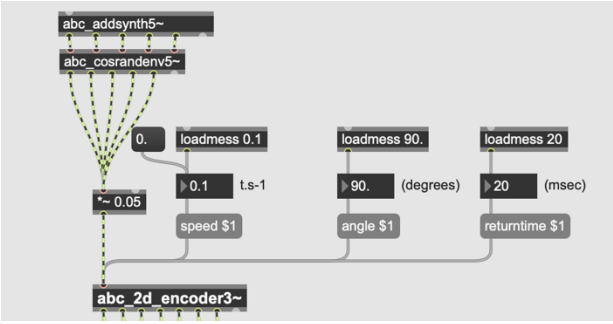

Figure 3. Contrôle de l'encodeur dans *abclib* (dans Max).

Nous y reviendrons à plusieurs reprises, le choix des unités manipulées est très important dans notre propos : en situation de répétition ou de concert, il reste bien plus facile de positionner des sources sonores diffusées ou des haut-parleurs en donnant des angles en degrés plutôt qu'en radians. De ce fait, dans toute notre modélisation spatiale, nous n'utiliserons que des angles en degrés, et pas en radians.

Pour le décodeur, nous reprenons la possibilité d'un décodage irrégulier en indiquant les angles des haut-parleurs en degrés par rapport au repère habituel utilisé en ambisonie. Mais nous ajoutons (cf. figure 4) quatre contrôles supplémentaires permettant d'adapter facilement le patch à des lieux variés, à savoir :

- un angle d'offset en degrés (*angularoffset*) permettant de décaler une situation ambisonique donnée par simple rotation ;
- un booléen (*directangles*) indiquant si l'on utilise des angles directs ou indirects, permettant de basculer instantanément la localisation gauche-droite quel que soit le sens dans lequel les haut-parleurs ont été connectés (sens anti-horaire comme prescrit en ambisonie *versus* sens horaire comme souvent pour l'équipement des salles) ;
- un booléen (*stereo*) permettant de passer instantanément d'une situation de studio de recherche ou de salle de concert avec plusieurs haut-parleurs à celle du *home studio* stéréo du compositeur ou de la compositrice.

[4] Notamment, dans le cadre des séminaires de Master « Création d'espaces sonores », « Musique et outils informatiques 1 », « Musique et outils informatiques 2 ».

[5] Le passage de la situation de rotation à celle de l'arrêt à un angle donné est modulé par une interpolation dont la durée est un des paramètres de contrôle de l'encodeur.

- un réglage du niveau de sortie en dB (*gain*), permettant d'éviter l'ajout d'un objet supplémentaire de gain général multicanal. Soulignons ici que tous les gains de sortie des objets (pour ceux qui en comportent) sont exprimés en dB pour se placer d'emblée dans un contexte professionnel.

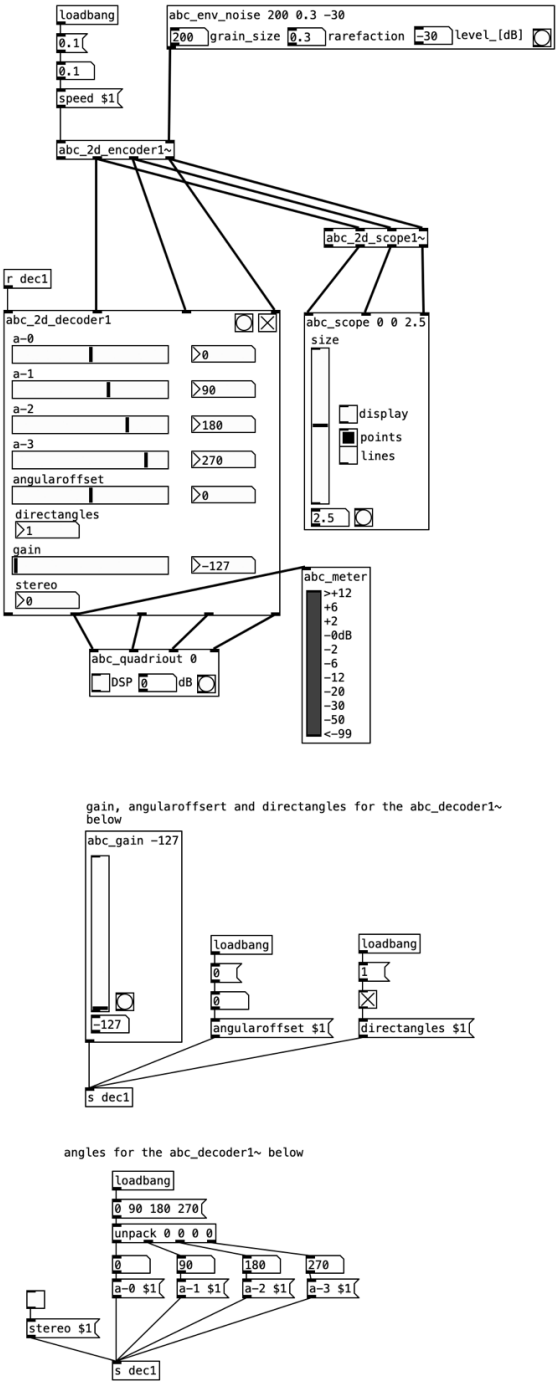

Figure 4. Contrôle du décodeur dans *abclib* (dans Pure Data).

3.2. Implémentation du traitement spatial

La caractéristique principale de la librairie HOA était la notion de traitement spatial, à savoir l'imbrication de traitements dans le modèle ambisonique, incarnée dans les objets *hoa.2d.process~* et *hoa.3d.process~* [2]. Ainsi, dans l'exemple de la décorrélation donné figure 5 [10], à l'ordre 7 d'ambisonie, quinze lignes à retard sont implémentées au niveau de chacune des quinze composantes spatiales ambisoniques en 2D (H_0, H_{-1}, H_1, …, H_{-7}, H_7). Les retards, exprimés en nombre d'échantillons, sont échelonnés linéairement de H_0 à H_{-7} et H_7 (figure 6) grâce à un facteur de décorrélation (compris entre 0 et 1) qui selon des plages de valeur donne une valeur de retard soit à zéro soit fraction d'un retard maximal pour chaque ligne. Ainsi, quand le facteur est proche de zéro, seules les composantes spatiales de degré élevé sont retardées, et quand il se rapproche de 1, progressivement toutes les lignes sont dotées d'un retard non nul.

Ce principe a été proposé dans la librairie HOA pour un certain nombre de traitements, ainsi mis en espace :

- les décorrélations micro-temporelles,
- les lignes à retard,
- les granulateurs sur ligne à retard,
- les modulateurs en anneau,
- les réverbérations,
- les convolutions,
- les dephasers,
- les gains.

Ces traitements sont à chaque fois proposés en deux versions : une dite *syn* partant d'un son mono et créant les composantes spatiales traitées à partir de ce son ; l'autre dite *fx*, traitant un ensemble de sons déjà dans le domaine des harmoniques spatiales.

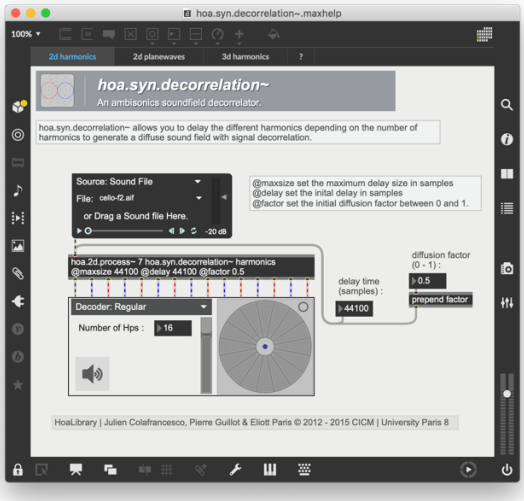

Figure 5. Copie d'écran du patch exemple de la décorrélation à l'ordre 7 dans HOA (dans Max).

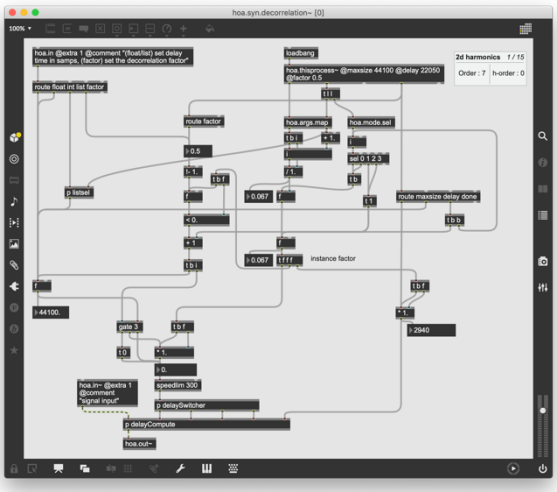

Figure 6. Patch de ligne à retard *hoa.syn.decorrelation~* associé à chacune des quinze composantes spatiales à l'ordre 7 d'ambisonie dans HOA (dans Max).

3.3. Traitements spatiaux en Faust dans *abclib*

Dans la version 1.0.1 de la librairie *abclib*, nous avons choisi de coder en Faust quatre traitements spatiaux, dont trois temporels :

- les décorrélations,
- les lignes à retard,
- les granulateurs sur ligne à retard ;

et un fréquentiel : les modulateurs en anneau. Ces quatre traitements sont proposés dans les versions *syn* et *fx*.

Pour des modules comme les décorrélations ou les modulateurs en anneau, un facteur pilote les valeurs des paramètres de contrôle (durées des retards dans le cas des décorrélations, fréquence de modulation pour les modulateurs en anneau) selon le principe présenté au 3.2. *abclib* reprend le principe initié par HOA mais lui ajoute, grâce aux propositions de Paul Goutmann[6] sur la décorrélation, une panoplie de 22 fonctions de répartition permettant de varier le rendu sonore par rapport au mode linéaire par défaut.

3.4. Synthèses spatiales

Nous avons commencé à intégrer dans *abclib* les travaux que nous avons menés sur la synthèse spatiale du son [1], dans l'idée d'opérer des synthèses de sons complexes dont les partiels seraient répartis dans les différentes composantes spatiales, circulaires ou sphériques. Nous avons montré que les dates des attaques ou *onsets* ainsi que la manière d'ordonner les fréquences jouent un rôle important dans la perception de l'espace ainsi créé. Dans *abclib*, nous avons implémenté la modélisation des sons de cloches effectuée par Risset sous la forme de 11 sons élémentaires [8] dans l'objet *abc_rissetsbell~* comportant 11 sorties séparées plutôt qu'une simple sortie mono. Dans la figure 7 ci-dessous, nous montrons comment nous opérons de légers retards sur les 11 sons avant de les prendre en compte comme harmoniques circulaires à l'ordre 5 d'ambisonie puis de les décoder.

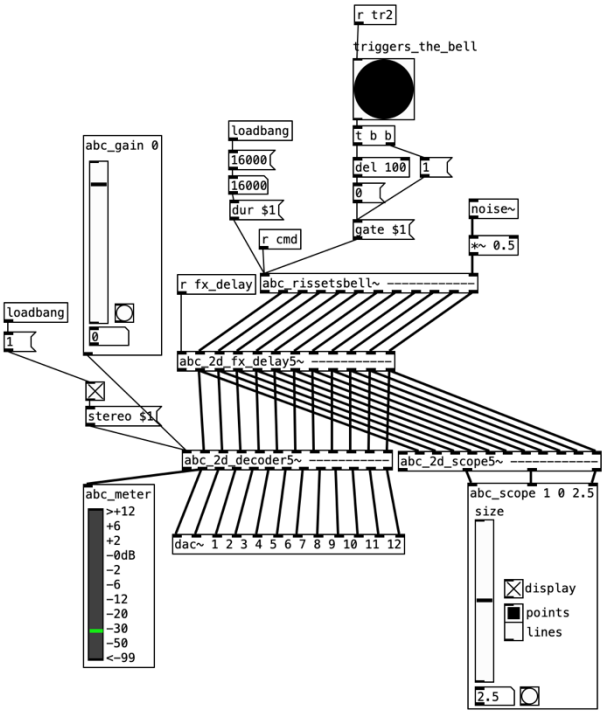

Figure 7. Cloche de Risset considérée comme source d'une synthèse sonore ambisonique à l'ordre 5 (dans Pure Data).

3.5. Représentations graphiques de champs sonores

À côté des traitements sonores, la bibliothèque HOA propose un certain nombre d'objets graphiques pour la manipulation et la visualisation ambisonique. Par exemple, l'objet *hoa.map* permet la manipulation de sources sonores ponctuelles (figure 8) en étant associé à l'objet *hoa.2d.map~*.

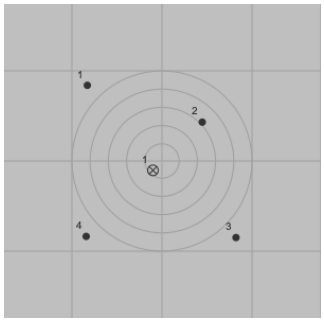

Figure 8. L'objet *hoa.map* dans Max permettant de manipuler des sources ponctuelles.

L'objet *hoa.scope~* permet de visualiser le champ sonore ambisonique comme le montre la figure 9 :

[6] Ces propositions font l'objet d'un papier proposé aux JIM 2021 par Paul Goutmann.

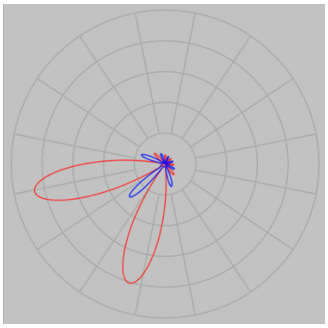

Figure 9. L'objet *hoa.scope~* dans Max permettant de visualiser un champ sonore ambisonique.

Nous avons choisi dans *abclib* de nous concentrer sur la visualisation ambisonique, en utilisant des bibliothèques propres à Max ou Pure Data plutôt qu'en développant des objets graphiques en C++. La première étape est la création d'un traitement Faust permettant de calculer une représentation 2D sous la forme d'un signal XY à partir des harmoniques circulaires : il s'agit des objets *abc_2d_scope1~*, …, *abc_2d_scope7~*.

Puis nous visualisons ce signal XY grâce à des fonctions OpenGL, en nous appuyant sur des objets de la bibliothèque *Jitter* dans Max, et sur des objets de la bibliothèque *Gem* dans Pure Data (cf. figure 10) encapsulées dans une abstraction *abc_scope*. On obtient ainsi un tracé des composantes spatiales avec les lobes positifs en rouge et les lobes négatifs en bleu.

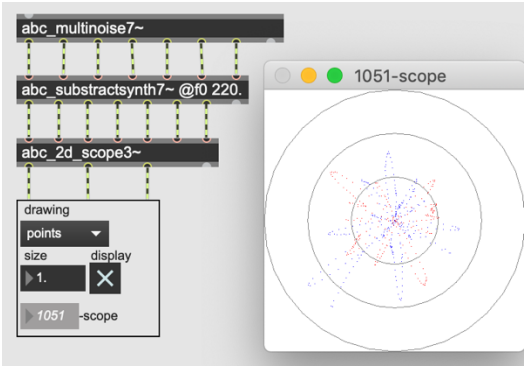

Figure 10. L'objet *abc_2d_scope3~* et l'abstraction de visualisation *abc_scope* dans Max.

4. TRAITEMENTS MULTICANAUX

Les traitements multicanaux permettent d'offrir en un seul et même objet un ensemble de n modules du même type. Il y a alors deux possibilités :

- soit les n modules sont totalement indépendants, que ce soit par rapport aux signaux en entrée ou aux paramètres de contrôle de ces modules ;
- soit les n modules proposent des corrélations musicalement intéressantes entre leurs entrées et/ou leurs contrôles.

Les *frequency_shifters* *abc_freqshift1~* jusqu'à *abc_freqshift16~* entrent dans le premier cas (cf. figure 11) : ce sont des modules parallèles implémentés dans le même objet mais indépendants en signal et en contrôle. Chaque *frequency shifter* possède sa fréquence de modulation et son gain.

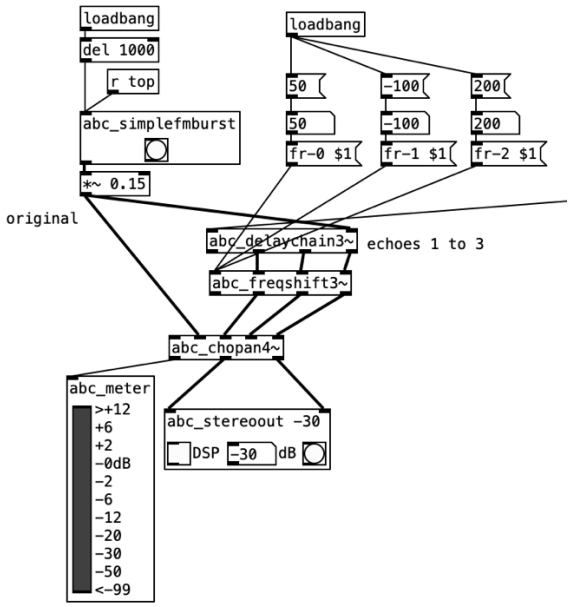

Figure 11. Exemple de *abc_freqshift3~* dans Pure Data.

À l'inverse, les lignes à retard chaînées *abc_delaychain2~*, …, *abc_delaychain16~* ne sont pas indépendantes mais placées en série pour permettre de générer des motifs rythmiques, chaque ligne ayant une sortie. La figure 12 montre une implémentation de l'objet *abc_delaychain3~* dans Max, avec trois retards donnés sous la forme de durées musicales (dans l'exemple croche, noire, double croche) converties en durées physiques grâce au tempo exprimé en bpm.

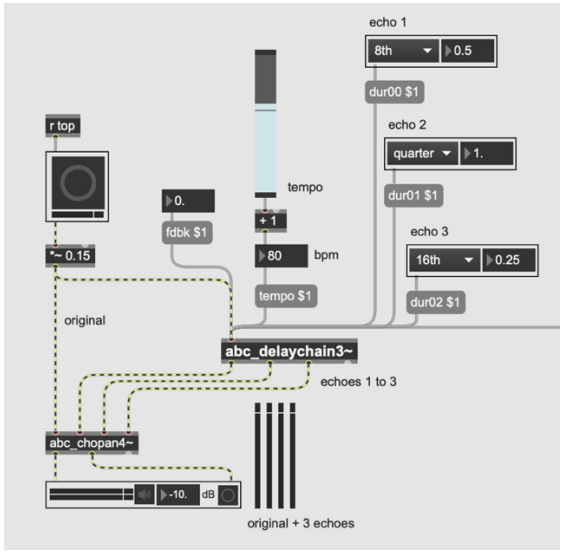

Figure 12. L'objet *abc_delaychain3~* dans Max.

Dans certains cas, ce sont à la fois l'organisation des entrées et les paramètres de contrôle qui lient les modules parallélisés. Par exemple, les *flangers* parallèles sont tous reliés au même signal en entrée (cf. *abc_flanger7~* sur la figure 13). Les contrôles du retard variable sinusoïdal sont identiques pour toutes les instances (*rate* pour la fréquence de la sinusoïde, *depth* pour son amplitude, *offset* pour le décalage en millisecondes, *fdbk* pour la part de son retardé ajoutée au son d'origine).

Le paramètre *spread* a été conçu pour donner la sensation que l'on « ouvre » ou que l'on « ferme » le champ sonore. Quand il vaut 0, les sinusoïdes des retards sont toutes en phase, produisant un même son dans toutes les sorties. Quand il augmente et se rapproche de 1, le déphasage des sinusoïdes augmente, ce qui produit une décorrélation entre les instances des *flangers*.

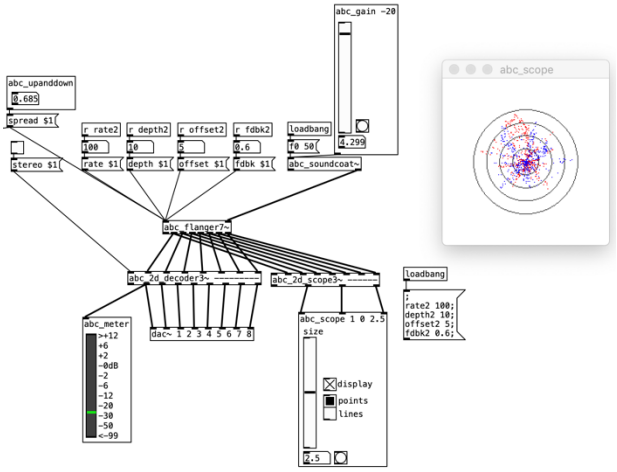

Figure 13. L'objet *abc_flanger7~* dans Pure Data.

5. CONCLUSION

Dans cet article, nous avons donné un aperçu des grandes lignes de la librairie Faust *abclib* pour la musique mixte. Partis d'une nécessité de pérennisation de développements antérieurs, notamment la librairie HOA, nous avons intégré de nombreux travaux précédents, que ce soit dans le cadre de nos enseignements, nos recherches et nos créations. L'expérience de la pédagogie de la mise en espace du son a inspiré certains développements et approfondissements d'implémentations précédentes, notamment dans le domaine du traitement spatial du son. L'utilisation du langage Faust garantit l'interopérabilité et la pérennité des développements réalisés.

À l'heure où nous rédigeons cet article, la librairie est proposée sur GitHub dans sa version 1.0.1. Elle est évidemment amenée à s'enrichir de futures contributions en Faust liées à des travaux de recherche-création menés au CICM / MUSIDANSE. Mais aussi de fonctionnalités qui nous semblent importantes rapidement, pour ajouter la 3D aussi bien au niveau de l'encodage, du décodage [6] que des traitements spatiaux, par exemple la réverbération fondée sur des réseaux de lignes à retard [11]. Nous approfondirons également nos recherches et contributions dans le domaine de la synthèse spatiale.

4. REFERENCES

1. Alain Bonardi, « Composer l'espace sonore », *Revue Francophone d'Informatique et Musique* [En ligne], Numéros, n° 7-8 - Culture du code, mis à jour le : 04/01/2021, URL : https://revues.mshparisnord.fr:443/rfim/index.php?id=624.
2. Alain Bonardi, Pierre Guillot, « Concevoir des traitements sonores ambisoniques en 2D et en 3D - l'exemple de Pianotronics 2 », *Actes des Journées d'Informatique Musicale 2015*, Université de Montréal, Canada, 2015.
3. Alain Bonardi, « Approches pratiques de la préservation/virtualisation des œuvres interactives mixtes : *En Echo* de Manoury », *Actes des 17èmes Journées d'Informatique Musicale (JIM 2011)*, Université Jean-Monnet Saint-Etienne, 25-27 mai 2011.
4. Bonardi, A. and Barthelemy, J. 2008. The preservation, emulation, migration, and virtualization of live electronics for performing arts: An overview of musical and technical issues. *ACM J. Comput. Cultur. Heritage* 1, 1, Article 6 (June 2008), 16 pages.
5. Julien Colafrancesco, *Spatialisation de sources auditives étendues*, Thèse de doctorat de l'Université Paris 8, 2015.
6. Pierre Lecomte, AMBITOOLS : TOOLS FOR SOUND FIELD SYNTHESIS WITH HIGHER ORDER AMBISONICS -V1.0. *Proceedings of the 1st International Faust Conference (IFC-18)*, Jul 2018, Mainz, Germany.
7. Lemouton, S., Bonardi, A., Pottier, L., Warnier, J. "On The Documentation of Electronic Music", *Computer Music Journal, 42:4*, p. 1–18, Winter 2018.
8. Risset Jean-Claude (1969), *An Introductory Catalogue of Computer Synthesized Sounds*, Murray Hill (NJ), Bell Telephone Laboratories, GRC-11874.
9. Sèdes, A., Guillot, P., Paris, E., "The HOA library, review and prospects", Proceedings of the ICMC/SMC Conference, Athènes, Grèce, 2014.
10. Anne Sèdes, « Approche musicale de la décorrélation microtemporelle dans la bibliothèque HOA », *Actes des Journées d'Informatique Musicale 2015*, Université de Montréal, Canada, 2015.
11. Stautner, J., Puckette, M., "Multi-Channel Reverberators", Computer Music Journal, 6:1, p. 52–65, Spring 1982.