Une nouvelle implémentation du Spatialisateur dans Max

Thibaut Carpentier

UNE NOUVELLE IMPLÉMENTATION DU SPATIALISATEUR DANS MAX

Thibaut Carpentier
CNRS – Ircam – Sorbonne Université – Ministère de la Culture
Sciences et Technologies de la Musique et du Son (UMR 9912 STMS)
1, place Igor Stravinsky, 75004 Paris
thibaut.carpentier@ircam.fr

RÉSUMÉ

Le Spatialisateur (`spat~`) est une bibliothèque d'outils dédiés à la spatialisation sonore, la réverbération artificielle, et la diffusion multicanale. Cet article présente une révision majeure de l'environnement (version 5) et son intégration dans Max.

1. INTRODUCTION

Le Spatialisateur [20, 22], communément appelé `spat~`, est un outil temps réel dédié au traitement de spatialisation sonore, à la réverbération artificielle, et à la diffusion multicanale. Il est développé à l'Ircam depuis le début des années 1990, et s'incarne principalement dans l'environnement Max [27]. Il se présente sous la forme d'une bibliothèque de processeurs articulés autour d'un moteur de réverbération à réseau de retards rebouclés [21] et de modules de panoramique. Ces processeurs sont pilotés par le biais d'une interface de haut-niveau qui permet de moduler la qualité acoustique de l'effet de salle synthétisé selon plusieurs attributs perceptivement pertinents [23, 24].

Visant des domaines d'application variés (concerts, mixage, post-production, réalité virtuelle, installations interactives, etc.), le logiciel est développé selon des méthodologies agiles [26] et il agrège en continu les principaux résultats de recherche de l'équipe Espaces Acoustiques et Cognitifs (anciennement Acoustique des Salles). L'outil a donc connu au fil des ans diverses évolutions, et la dernière révision majeure, `spat~` 4, fut diffusée en 2009 et présentée dans [13, 4].

Cet article présente `spat~` 5, une nouvelle mouture de l'environnement. Comme tout logiciel exploité au long cours, son cycle de développement s'oriente selon différentes directions : l'amélioration des fonctionnalités existantes (incluant maintenance corrective, adaptative et évolutive), l'ajout de nouvelles fonctionnalités, et le réusinage (*refactoring*) de l'infrastructure logicielle (afin d'assainir l'architecture et assurer sa pérennité). Nous présentons dans les sections 2 et 3 une profonde refonte architecturale, et des nouvelles fonctionnalités sont décrites dans la section 4.

Le modèle interne du `spat~` se fonde historiquement sur une représentation orientée objet de l'effet de salle, constitué de quatre sections temporelles filtrées et pannées indépendamment : son direct, réflexions précoces, réflexions tardives et queue de réverbération (voir par exemple la section 3 dans [4]). Cette architecture est perpétuée dans `spat~` 5 ; en revanche, l'une des principales évolutions introduites dans cette nouvelle version concerne l'interfaçage de l'outil avec son (ou ses) environnement(s) hôte(s).

2. INTERFAÇAGE OSC

2.1. Motivations

Sorti en 2008, Max 5 introduisit pour la première fois la notion "d'attributs". Par analogie avec les paradigmes de programmation orientée objet, on peut dire que chaque objet (*external*) dans Max est une instance de classe, et les attributs sont des variables membres de cette classe. Grâce aux fonctionnalités offertes dans l'API, il est possible d'exposer publiquement (i.e. à l'utilisateur du patch Max) les attributs et les manipuler via des accesseurs (*getter*) et mutateurs (*setter*). `spat~` 4 utilise intensivement ce concept, puisque les paramètres utilisateur des modules du `spat~` sont exposés majoritairement via leurs attributs. Ce choix de conception présente des avantages et, rétrospectivement, un certain nombre d'inconvénients. Les avantages sont clairs : le mécanisme d'attributs est très bien intégré dans Max et il est aisé de manipuler (`attrui`, `getattr`, *inspector*) ou sauvegarder (`pattr`, `pattrstorage`) les attributs d'objets ; leur syntaxe est explicite (par opposition aux "arguments"), et leur mise en œuvre (pour le développeur) via l'API est aisée. Dans le contexte spécifique du `spat~`, il peut cependant s'avérer que les attributs fussent inadéquats pour plusieurs raisons :

— Les objets `spat~`, multicanaux par essence, présentent généralement un grand nombre d'attributs (souvent proportionnel au nombre de canaux). Dans les menus contextuels de Max, ceci peut conduire à une navigation mal commode et lente. En outre, les attributs dans Max sont stockés sous forme de listes (*array*) ce qui s'avère assez inapproprié aux paramètres du `spat~` ; une structure hiérarchique serait plus pertinente pour refléter l'arborescence des données [1]. En outre, stocker

1 . Il est en théorie possible de créer des attributs d'attributs, et donc d'élaborer une forme d'arborescence ; toutefois cela ne résout pas vraiment la question et ne va pas dans le sens de la simplicité.

Thibaut Carpentier, *Une nouvelle implémentation du Spatialisateur dans Max*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 45–52, 2018.

les attributs sous forme de listes peut nuire aux performances : par exemple `pattr` (mécanisme de sauvegarde des attributs) aura besoin de copier l'ensemble de la liste dès lors qu'un des éléments constitutif est modifié.

— Les objets `spat~` sont souvent polymorphiques : le nombre et la nature des paramètres exposés peuvent évoluer au cours du temps. Un exemple est l'objet `spat.pan~` qui présente des propriétés différentes selon qu'il opère en stéréo, binaural, Ambisonics, etc. Les attributs de l'API Max sont mal adaptés pour refléter ce polymorphisme [2].

— Les attributs sont spécifiques à l'API de Max ; or la bibliothèque logicielle `spat~` est *bindée* vers de nombreux autres environnements (Matlab [3], Spat Revolution [4], plugins Ircam Tools [5], Open Music [17], Pure Data [6], etc.). Chaque nouveau binding requiert une couche de code de liaison (*glue code*) pour s'interfacer avec l'environnement hôte. Côté développeur, il est important de minimiser l'effort induit par ce *glue code* ; côté utilisateur, il est pertinent d'avoir un interfaçage similaire (e.g. même syntaxe) dans les différents hôtes.

Ces considérations nous ont amené à "changer notre fusil d'épaule" et adopter dans `spat~` 5 un interfaçage selon le protocole et la syntaxe Open Sound Control (OSC [31]). Les attributs sont désormais abandonnés.

Le choix de l'OSC s'est imposé de façon somme toute évidente : ce protocole est largement répandu et maîtrisé dans la communauté audio, il encourage et facilite la communication (e.g. via UDP/IP) inter-applications ou avec des interfaces de contrôle compatibles ; son implémentation est très simple (de nombreuses bibliothèques sont par ailleurs disponibles dans plusieurs langages). L'espace d'adressage hiérarchique se prête bien à la plupart des applications d'informatique musicale et notamment du `spat~`. Outre les messages conventionnels, le protocole permet également d'encapsuler plusieurs messages au sein d'un *bundle*, simplifiant la transmission synchrone de grandes quantités d'événements. Enfin, il offre un puissant mécanisme de distribution (*dispatching*) de messages grâce à sa sémantique de *pattern matching*.

2.2. Intégration dans Max

2.2.1. Syntaxe

Les *externals* utilisent donc une syntaxe OSC. À l'interface avec Max, les messages sont convertis de/vers le format natif de Max, les `atoms`. Cette conversion est triviale puisque le typage des arguments OSC est proche de celui des `atoms` (int, float, symbole, etc.). Les *bundles* OSC sont quant à eux transmis sous forme de `FullPacket` véhiculant uniquement un pointeur vers la mémoire du *bundle* (à l'instar des `dict` de Max). Ceci peut donc permettre de transmettre d'importantes quantités de données de façon efficace (le scheduler de Max n'est sollicité qu'une fois par bundle, et non pour chaque message).

La syntaxe adoptée s'inspire du style REST [29] (Representational State Transfer), et s'avère très proche de la syntaxe de `spat~` 4 (en y ajoutant le séparateur `/`). Par exemple pour spécifier la position cartésienne d'une source dans `spat~` 5, on pourra utiliser le message :

```
/source/1/xy [float][float]
```

Les *externals* supportent les sémantiques habituelles de *pattern matching*, favorisant le *grouping* d'éléments :

```
/source/*/mute [boolean]
/source/[2-5]/mute [boolean]
/source/{3,6,7}/mute [boolean]
```

Le routage et la distribution de *patterns* d'adresses OSC dans Max peut nécessiter la manipulation d'expressions régulières (`regexp`) ce qui est généralement mal commode et peu performant ; aussi avons-nous développé une bibliothèque d'outils utilitaires (environ 25 *externals*) pour simplifier les opérations usuelles (voir Figure 1).

D'un point de vue du développement logiciel, les adresses OSC les plus couramment utilisées dans `spat~` sont stockées (lors de la compilation) dans une table de hash. Ceci évite toute manipulation (coûteuse) de *strings* lors de l'exécution et garantit ainsi une résolution dynamique aussi performante que les tables de symboles traditionnelles de Max.

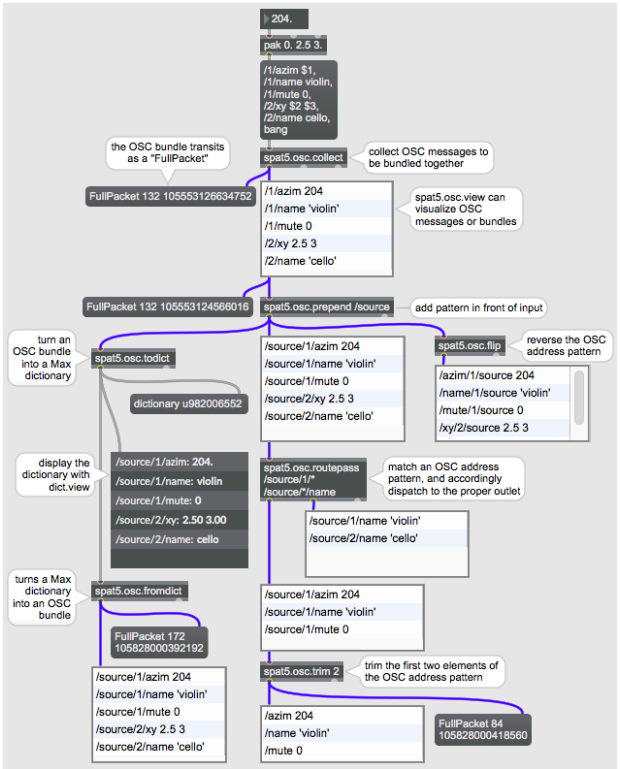

Figure 1. Exemple de manipulations OSC usuelles. Les *patchcords* représentées en bleu véhiculent des bundles OSC (`FullPacket`).

2 . Certes, il existe des attributs d'objet – par opposition aux attributs de classe – qui peuvent être ajoutés/supprimés dynamiquement, mais cela contrevient également à la simplicité de mise en œuvre.

3 . Non diffusé publiquement au moment de la rédaction de cet article.

4 . `www.spatrevolution.com`

5 . `www.ircamtools.com`

6 . Non diffusé publiquement au moment de la rédaction de cet article.

2.2.2. *Inter-opérabilité et compatibilité*

Un avantage potentiel de l'interfaçage OSC est que les utilisateurs peuvent également bénéficier des bibliothèques et outils OSC existants, notamment :

— le *package* `odot` [16] qui offre un langage de manipulation d'expressions (parseur et interpréteur) opérant sur des bundles OSC. Typiquement, `odot` pourra être utilisé pour la génération et la transformation algorithmiques de données de spatialisation (trajectoires).
— ToscA [6, 5], un plugin insérable dans les stations de travail et qui permet la transmission via OSC de données d'automation. Bien que générique, ce plugin a, dès sa conception, été pensé pour un usage couplé avec `spat~`.

En revanche, il convient de noter que la syntaxe OSC ne préserve pas la compatibilité descendante avec `spat~` 4. Cet aspect n'est évidemment pas négligeable puisque plusieurs centaines d'œuvres s'appuient sur `spat~` 4. Il n'existe à l'heure actuelle pas de mécanisme de "portage automatique" de `spat~` 4 vers `spat~` 5 ; toutefois nous estimons que, dans la majorité des cas, la transition pourra se faire de façon indolore (il s'agit essentiellement d'adaptation syntaxique mineure). Des exemples de portage de patchers canoniques sont également fournis dans la distribution.
Notons enfin que `spat~` 4 et `spat~` 5 peuvent cohabiter sans conflit (notamment pour permettre le portage progressif des pièces) puisque les *externals* `spat~` 5 s'inscrivent dans un espace d'adressage (*namespace*) dédié (préfixe `spat5.*`).

3. AUTRES REFONTES DE L'ENVIRONNEMENT

La refonte de l'environnement engagée dans `spat~` 5 ne se restreint pas uniquement à la syntaxe des objets, mais a une incidence à plusieurs niveaux : ergonomie générale, processeurs audio, et interfaces graphiques de contrôle.

3.1. Ergonomie

Comme signalé dans la section 2, les *externals* de `spat~` 5 n'utilisent plus d'attributs. En conséquence, ils ne bénéficient plus de certaines fonctionnalités de Max telles que l'inspecteur ou les mécanismes de documentation automatique (*hints*). Pour pallier cela, des mécanismes "de substitution" ont été introduits : chaque objet dispose d'une fenêtre de *status* et de *help* (voir Figure 2). La fenêtre de *status* affiche l'état courant de tous les paramètres de l'objet ; à l'instar de l'inspecteur Max, il est possible de filtrer les requêtes, et sélectionner et copier des messages depuis cette fenêtre vers le patcher. La fenêtre *help* affiche une documentation textuelle de tous les messages OSC supportés. Des pages de référence sont également générées automatiquement et sont accessibles via le navigateur standard de documentation de Max.

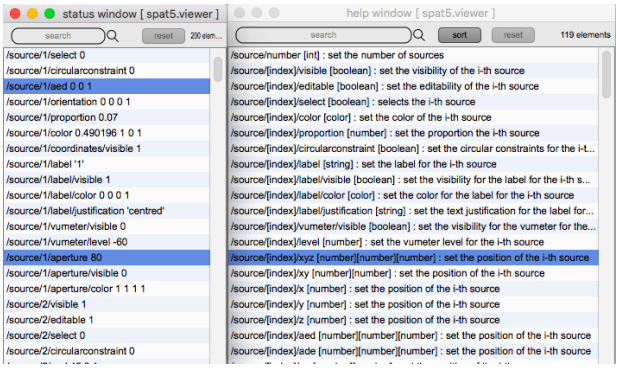

Figure 2. Fenêtres de statut (à gauche) et de documentation (à droite) de l'objet `spat5.viewer`.

Si l'on compare avec l'inspecteur d'attributs dans `spat~` 4 (Figure 3), on peut comprendre certains bénéfices de la nouvelle architecture : la fenêtre de *status* propose une vision hiérarchique des paramètres qui est plus claire que les listes (*array*) de valeurs dans l'inspecteur ; en outre ceci permet une granularité beaucoup plus fine : par exemple l'attribut `sourceseditable [boolean]` de `spat~` 4 (qui autorise ou non l'édition des sources) agit globalement sur l'ensemble des sources, tandis que `spat~` 5 offre un contrôle indépendant pour chaque élément :
`/source/i/editable [boolean]`

L'actualisation de paramètres indépendants plutôt que de (longues) listes de données peut également améliorer les performances générales de l'outil.

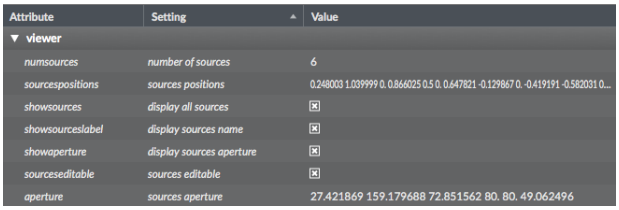

Figure 3. Inspecteur du `spat.viewer` dans `spat~` 4.

3.2. Ordonnancement et *thread-safety*

Par rapport aux attributs, l'encapsulation des événements dans des messages OSC simplifie considérablement le développement d'une queue *thread-safe* pour la synchronisation de données partagées entre les différents processus impliqués dans Max (*thread* audio, *thread* message de l'application, *thread* d'événements à haute priorité, etc.), épurant ainsi l'hygiène du code et limitant significativement le risque de bogues : les événements entrants (messages ou bundles) sont empilés dans une queue FIFO puis traités au moment et dans le *thread* opportuns. L'accès à cette queue est garanti *thread-safe* et non-bloquant [7]. A contra-

7. Ce mécanisme est décrit en détail dans [11].

rio, seuls quelques objets dans `spat~` 4 étaient garantis *thread-safe*.

Pour les objets DSP, le comportement par défaut est que la queue est dépilée dans le *thread* audio, au début du *callback* (voir Figure 4). Ce comportement correspond peu ou prou au mécanisme dit *scheduler in audio interrupt* dans Max. Il contribue également à une meilleure sécurité (*thread-safety*) du code. Notons que tous les objets `spat~` 5 fonctionnent selon ce mécanisme, indépendamment des réglages *overdrive* ou *interrupt* de l'environnement hôte.

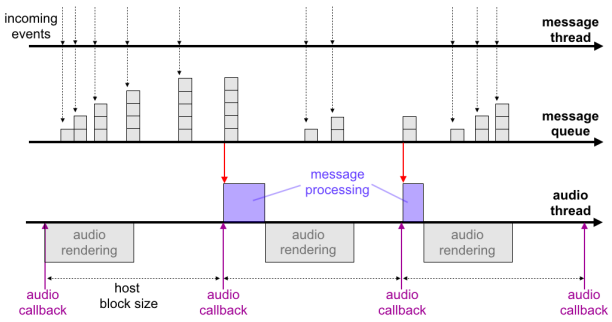

Figure 4. Ordonnancement des événements selon un mécanisme *scheduler in audio interrupt*.

3.3. Interfaces de contrôle

Le *package* `spat~` contient une vingtaine d'*externals* d'interfaces graphiques de contrôle (voir par exemple Figure 5). Ces interfaces ont été "rafraîchies" pour une meilleure lisibilité et ergonomie, et de nombreuses options de personnalisation y ont été ajoutées. Ont également été ajoutés un grand nombre de raccourcis clavier pour un accès immédiat aux fonctionnalités usuelles ; ces raccourcis sont par ailleurs configurables par l'utilisateur (Figure 6).

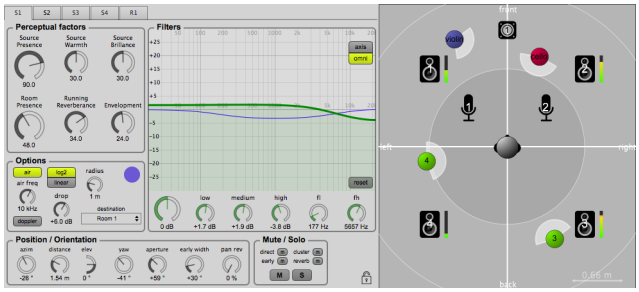

Figure 5. Interface du `spat5.oper` (opérateur perceptif de contrôle haut-niveau du `spat~`). Facteurs perceptifs de contrôle de l'effet de salle (gauche) ; filtrages (centre) ; représentation schématique 2D de la scène sonore (droite).

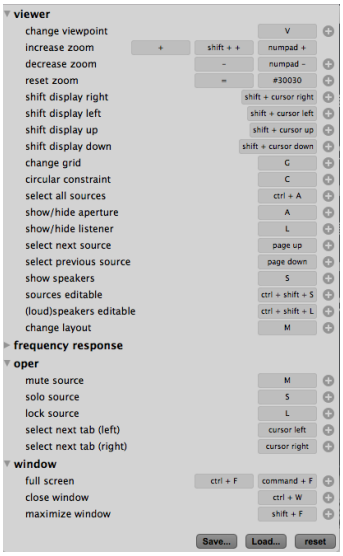

Figure 6. Fenêtre de personnalisation des raccourcis clavier pour `spat5.oper`.

Les *externals* portant ces interfaces graphiques bénéficient par ailleurs du mécanisme de queue *thread-safe* présenté au paragraphe précédent. Ceci a permis de les implémenter sans recourir à *deferlow* (renvoi dans le *thread* à basse priorité), offrant ainsi une réactivité bien supérieure à `spat~` 4 (dans lequel la plupart des GUIs utilisaient *deferlow*, ce qui conduit, en cas de charge importante, à un empilement des messages et un ralentissement global de l'interface et du scheduler).

L'état global (i.e. l'ensemble des paramètres) d'un *external* GUI est représenté sous forme d'un bundle OSC (voir par exemple Figure 2). Ce bundle peut facilement être exporté sous forme de fichier texte (donc éditable), puis ré-importé. Ceci peut donc être utilisé comme mécanisme simple de sauvegarde/chargement de presets. Le bundle peut également être stocké directement dans le patcher (on dira qu'il est *embedded*) ou via les *snapshots* de Max (en activant le "Parameter Enable Mode") ; dans ces cas, le bundle OSC est préalablement converti sous forme de *blob* binaire (non éditable) sauvé avec le patcher.

3.4. Aspects de génie logiciel

Le code `C++` des bibliothèques sous-jacentes a été considérablement modernisé, en utilisant les nouveaux idiomes de programmation offerts par les standards `C++11` à `C++17` [8] : expressions constantes à la compilation (`constexpr`), déduction de typage automatique (`auto`), *lambda* fonctions, *range-based for loop*, etc. Ces nouvelles fonctionnalités permettent d'assainir l'hygiène du code, de diminuer les risques de bogues, et incidemment de réduire la taille du code source ; par exemple, le *glue code* requis pour l'interfaçage avec l'API de Max (voir section 2) a pu

8 . ISO International Standard ISO/IEC 14882 :2017(E) – Programming Language `C++`

être réduit de 80%.

Les algorithmes de traitement du signal, déjà vectorisés et optimisés grâce au *framework* `Accelerate` [9], ont encore été améliorés par l'utilisation des primitives hautes performances Intel® Integrated Performance Primitives (IPP) [10].

4. NOUVELLES FONCTIONNALITÉS

Cette section recense, sans souci d'exhaustivité, plusieurs nouveautés marquantes du *package* `spat~` 5.

4.1. Higher Order Ambisonics

Concernant les processeurs audio pour l'analyse/synthèse de scènes sonores spatialisées, de nombreuses améliorations et nouvelles fonctionnalités ont été introduites dans la bibliothèque `spat~` 5. Il serait fastidieux d'en tenir une liste complète. Toutefois les recherches des dernières années ont mis un accent particulier sur la technique Higher Order Ambisonics (HOA [14]), et nous énumérons ici quelques nouveautés afférentes :

— Les différents schémas de normalisation des composantes HOA en vigueur (FuMa, MaxN, SN3D, N3D, etc.) sont une source fréquente de confusion pour les utilisateurs, et ils nuisent à l'inter-opérabilité des outils. Un travail de formalisation [10] et de documentation a été mené afin de clarifier leur impact dans une chaîne de production Ambisonics, et de limiter les risques d'erreur.

— Plusieurs stratégies de décodage HOA sont offertes dans `spat~` (voir [4]). Dans `spat~` 5, nous y avons ajouté les décodeurs dits "all-rad" (All-Round Ambisonic Panning and Decoding [32]) et "constant-spread" (Constant Angular Spread Ambisonic Decoding [15]) qui s'appuient sur un décodage HOA régulier, sur une sphère virtuelle de type *t-design*, projeté via VBAP ou MDIP sur le dispositif de haut-parleurs physiques.

— Les différents décodeurs HOA disponibles (*sampling decoder*, *mode-matching* [14], *energy-preserving* [33], *all-rad* [32], *constant-spread* [15]) peuvent conduire à des champs sonores de puissances significativement différentes (jusqu'à plusieurs dizaines de dB d'écart selon les configurations). Ceci empêche toute étude/écoute comparative des différents décodages, aussi avons nous développé une technique de compensation énergétique qui permet d'aligner les décodeurs entre eux (ou sur une référence arbitraire). La méthode s'appuie sur une estimation de la puissance rendue en condition de champ diffus (voir par exemple la section 4.4 dans [33]).

— `spat5.hoa.blur~` est un nouvel outil permettant de manipuler la "résolution spatiale" d'un champ encodé HOA. Il permet de modifier continûment l'ordre du flux HOA (simulant de la sorte des ordres fractionnaires), tout en maintenant la puissance globale constante [9]. Il peut être utilisé pour adapter l'ordre de contenus existants, ou comme effet créatif (typiquement en faisant varier dynamiquement le facteur de "flou").

— `spat5.hoa.focus~` est un autre effet opérant dans le domaine HOA. Inspiré de [25], il permet de synthétiser des diagrammes de directivité virtuels qui "filtrent" un flux HOA. L'orientation et la sélectivité des diagrammes virtuels peuvent être éditées via une interface graphique dédiée (Figure 7). L'outil trouve un intérêt aussi bien dans le domaine de la post-production ("zoom" dans une scène sonore enregistrée) que de la création.

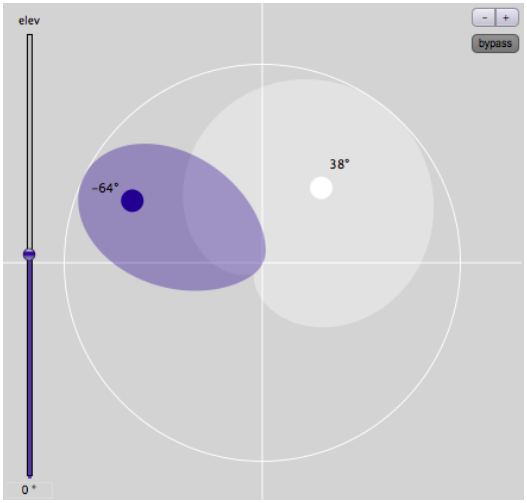

Figure 7. Module `spat5.hoa.focus` pour la synthèse de directivités virtuelles dans le domaine HOA.

4.2. Audio Definition Model

Ces dernières années ont vu un regain d'intérêt pour les modèles orientés objet pour la production et la diffusion de contenus multicanaux. Plusieurs formats d'échange ont ainsi été proposés ; en particulier Audio Definition Model (ADM [19]) est un standard ouvert publié par ITU et EBU [11] pour la description de médias orientés objet et encapsulés dans un conteneur Broadcast Wave Format (BWF). ADM spécifie un ensemble de métadonnées (notamment positions et gains d'objets sonores évoluant dans le temps et l'espace) codées selon une représentation XML. `spat~` est l'une des premières bibliothèques proposant une solution complète pour la création et le rendu de fichiers BWF-ADM : `spat5.adm.record~` permet l'écriture de fichier BWF embarquant des métadonnées de spatialisation, et `spat5.adm.renderer~` assure le rendu temps réel de médias ADM sur différents dispositifs de restitution (casque ou haut-parleurs) ; d'autres *externals* permettent

9. `http://developer.apple.com`
10. `http://software.intel.com`
11. International Telecommunication Union et European Broadcasting Union

également de gérer l'interactivité des objets. Ces outils sont décrits plus en détail dans [18]. Notons toutefois que seul un sous-ensemble des spécifications ADM est actuellement supporté (couvrant cependant la plupart des cas d'usage classiques), et qu'une intégration plus étroite du format au sein de l'architecture du spat∼ reste à faire (e.g. import/export de fichiers ADM directement depuis les processeurs tels que spat5.spat∼). Ceci fait l'objet de travaux en cours.

4.3. Panoramix

panoramix est une station de travail pour la spatialisation et la réverbération en contexte de mixage et de post-production 3D. L'outil a été présenté dans plusieurs publications [7, 12, 8]. Il s'appuie de façon sous-jacente sur la bibliothèque C++ spat∼, et son développement rapide n'a été possible que grâce à la profonde refonte logicielle exposée dans les sections 2 et 3. panoramix peut être vu comme une déclinaison spécialisée des outils spat5.oper et spat5.spat∼ (qui sont plus génériques), dotée d'un *frontend* ad-hoc pour les situations de mixage multicanal. Bien que distribué séparément sous forme d'exécutable standalone, panoramix est également inclus dans le *package* spat∼ 5, sous forme d'*externals* pouvant être utilisés pour des applications moins conventionnelles. Le lecteur est invité à consulter [7, 12, 8] pour plus de détails sur la conception et l'exploitation de panoramix.

4.4. Quaternions

Avec la démocratisation des équipements de réalité virtuelle, spat∼ est de plus en plus employé pour assurer le rendu audio d'applications multimédia immersives, typiquement présentées selon une modalité binaurale. Ces environnements requièrent des manipulations géométriques 3D plus ou moins complexes. En particulier, la gestion de l'orientation des entités (auditeur ou objets sonores/visuels) dans l'espace est une source fréquente de confusion pour les utilisateurs, en raison des nombreuses conventions en vigueur (et incompatibles entre elles). Pour pallier cette difficulté, nous avons développé une bibliothèque d'*externals* permettant de manipuler quaternions, angles d'Euler et matrices de rotation 3D. Ils assurent notamment les conversions entre ces différentes représentations. Les processeurs audio de spat∼ (par exemple spat5.binaural∼) peuvent être pilotés indifféremment par des quaternions ou des angles d'Euler, simplifiant l'inter-opérabilité avec les différents SDK de réalité virtuelle.

4.5. Time Code

spat∼ est fréquemment utilisé dans des contextes audiovisuels ; lors du couplage avec des environnements vidéo/graphiques, il est nécessaire de mettre en place des mécanismes de synchronisation entre les flux audio et vidéo. Une des techniques les plus populaires consiste alors à utiliser un *Linear Timecode* (LTC [28]) encodant des *frames* SMPTE et transmis sous forme de signal audio longitudinal. Ce standard n'est malheureusement pas supporté nativement par Max, aussi avons nous développé des outils permettant la réception (spat5.ltc.decode∼), et la génération (spat5.ltc.encode∼) de tels *timecodes*. En outre, l'objet spat5.ltc.trigger∼ peut être utilisé comme gestionnaire d'événements (*cue manager*) déclenchant des actions en fonction du timecode courant ; la granularité temporelle est certes assez faible (typiquement 30 fps $\approx$ 33 millisecondes), mais suffisante pour la plupart des applications en spatialisation sonore.

4.6. Intégration dans Open Music

La spatialisation sonore n'est pas seulement déployée lors de l'exécution des œuvres, mais elle doit également être intégrée au processus compositionnel. Aussi est-il pertinent de proposer des bindings de spat∼ dans des environnements de composition assistée par ordinateur. Comme discuté au paragraphe 2, une des motivations pour l'adoption d'une interface OSC est de s'abstraire des spécificités de Max, et d'accélérer l'intégration de spat∼ dans différentes applications hôtes, en offrant une syntaxe unifiée. Dans le cadre du projet EFFICACe [3], nous avons tiré profit de cette nouvelle architecture OSC pour insérer plusieurs modules de la bibliothèque spat∼ dans les environnements Open Music et o7. Les premiers fruits de cette intégration ont été présentés dans [17, 1, 2], et ils ouvrent la porte à de nouveaux champs d'expérimentation faisant l'objet de travaux et de résidences artistiques en cours.

5. CONCLUSION

Cet article présente spat∼ 5, une nouvelle mouture de l'environnement de spatialisation et réverbération spat∼ implémentée dans Max. La bibliothèque contient près de 200 *externals* couvrant toutes les activités liées à la diffusion multicanale. Par rapport aux versions précédentes, cette implémentation propose un nouvel interfaçage avec l'application hôte, basé sur le protocole OSC. Hormis les aspects syntaxiques, cette nouvelle structure OSC s'accompagne d'une refonte plus profonde de la bibliothèque, visant à améliorer son ergonomie, sa stabilité, ses performances et son inter-opérabilité. La distribution spat∼ 5 inclut par ailleurs nombre de nouveaux objets (contrôle, DSP, et GUI) qui élargissent le spectre des traitements réalisables. Signalons encore que plusieurs "sous-ensembles" du *package* (e.g. les modules OSC, LTC, quaternions, etc.) sont indépendants du modèle du spatialisateur à proprement parler, et peuvent être d'un intérêt plus général pour la communauté d'informatique musicale.

Impulsé tant par les innovations technologiques que par les productions artistiques, spat∼ demeure un outil en constante évolution ; les principaux travaux en cours et les

perspectives à court terme concernent :

— l'intégration de formats orientés objet, tel que discuté au paragraphe 4.2 ;
— la compatibilité avec les protocoles d'interrogation du *namespace* OSC ; de nombreuses propositions ont été faites à cet égard (OSC Query[30], Minuit [12] , libmapper [13] , OSNIP [14] , OSCQueryProposal [15]), et il n'existe pour l'heure pas de consensus dans la communauté. L'intégration de l'un ou l'autre de ces protocoles dans `spat~` 5 est en cours d'évaluation ;
— l'extension des formalismes d'analyse/synthèse de scènes sonores spatiales ;
— et l'exploitation dans des environnements de composition assistée par ordinateur.

6. REFERENCES

[1] Agger, S., Bresson, J., Carpentier, T., "*Landschaften* – Visualization, Control and Processing of Sounds in 3D Spaces", *Proc. of the International Computer Music Conference (ICMC)*, Shanghai, China, Oct 2017.

[2] Bresson, J., Bouche, D., Carpentier, T., Schwarz, D., Garcia, J., "Next-generation Computer-aided Composition Environment : A New Implementation of OpenMusic", *Proc. of the International Computer Music Conference (ICMC)*, Shanghai, China, Oct 2017.

[3] Bresson, J., Bouche, D., Garcia, J., Carpentier, T., Jacquemard, F., MacCallum, J., Schwarz, D., "Projet EFFICACe : Développements et perspectives en composition assistée par ordinateur", *Journées d'Informatique Musicale (JIM)*, Montreal, Canada, May 2015.

[4] Carpentier, T., "Récents développements du Spatialisateur", *Journées d'Informatique Musicale (JIM)*, Montreal, Canada, May 2015.

[5] Carpentier, T., "ToscA : Un plugin de communication OSC pour le mixage spatialisé orienté objet", *Journées d'Informatique Musicale (JIM)*, Montreal, Canada, May 2015.

[6] Carpentier, T., "ToscA : An OSC Communication Plugin for Object-Oriented Spatialization Authoring", *Proc. of the 41st International Computer Music Conference (ICMC)*, pages 368 – 371, Denton, TX, USA, Sept. 2015.

[7] Carpentier, T., "Panoramix : 3D mixing and post-production workstation", *Proc. 42nd International Computer Music Conference (ICMC)*, pages 122 – 127, Utrecht, Netherlands, Sept 2016.

[8] Carpentier, T., "A versatile workstation for the diffusion, mixing, and post-production of spatial audio", *Proc. of the Linux Audio Conference (LAC)*, Saint-Etienne, France, May 2017.

[9] Carpentier, T., "Ambisonic spatial blur", *Proc of the 142nd Convention of the Audio Engineering Society (AES)*, Berlin, Germany, May 2017.

[10] Carpentier, T., "Normalization schemes in Ambisonic : does it matter ?", *Proc of the 142nd Convention of the Audio Engineering Society (AES)*, Berlin, Germany, May 2017.

[11] Carpentier, T., "Synchronisation de données interprocessus dans les applications audio temps réel : qu'est-ce qui débloque ?", *Journées d'Informatique Musicale*, Amiens, France, May 2018.

[12] Carpentier, T., Cornuau, C., "panoramix : station de mixage et post-production 3D", *Journées d'Informatique Musicale*, pages 162 – 169, Albi, France, April 2016.

[13] Carpentier, T., Noisternig, M., Warusfel, O., "Twenty Years of Ircam Spat : Looking Back, Looking Forward", *Proc. of the 41st International Computer Music Conference (ICMC)*, pages 270 – 277, Denton, TX, USA, Sept. 2015.

[14] Daniel, J., *Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia*, Ph.D. thesis, Université de Paris VI, 2001.

[15] Epain, N., Jin, C., Zotter, F., "Ambisonic Decoding With Constant Angular Spread", *Acta Acustica united with Acustica*, volume 100, pages 928 — 936, 2014.

[16] Freed, A., MacCallum, J., Schmeder, A., "Dynamic, instance-based, object-oriented programming in Max/MSP using Open Sound Control message delegation", *Proc. of the 37th International Computer Music Conference (ICMC)*, pages 491 – 498, Huddersfield, Aug. 2011.

[17] Garcia, J., Carpentier, T., Bresson, J., "Interactive-compositional authoring of sound spatialization", *Journal of New Music Research – Special Issue on Interactive Composition*, volume 46(1), pages 74 – 86, 2017.

[18] Geier, M., Carpentier, T., Noisternig, M., Warusfel, O., "Software tools for object-based audio production using the Audio Definition Model", *Proc. of the 4th International Conference on Spatial Audio (ICSA)*, Graz, Austria, Sept 2017.

[19] ITU, "ITU-R BS.2076 (ADM Audio Definition Model)", Technical report, www.itu.int/rec/R-REC-BS.2076, 2015.

[20] Jot, J.M., "Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces", *ACM Multimedia Systems Journal (Special issue on Audio and Multimedia)*, volume 7(1), pages 55 – 69, 1999.

[21] Jot, J.M., Chaigne, A., "Digital Delay Networks for Designing Artificial Reverberators", *Proc. of the 90th Convention of the Audio Engineering Society (AES)*, Paris, France, Feb 1991.

12 . `https://github.com/Minuit`
13 . `http://libmapper.github.io`
14 . `https://github.com/jamoma/osnip/wiki`
15 . `https://github.com/mrRay/OSCQueryProposal`

[22] Jot, J.M., Warusfel, O., “A Real-Time Spatial Sound Processor for Music and Virtual Reality Applications”, *Proc. of the International Computer Music Conference (ICMC)*, pages 294 – 295, Banff, Canada, 1995.

[23] Jullien, J.P., “Structured model for the representation and the control of room acoustic quality”, *Proc. of the 15th International Congress on Acoustics (ICA)*, pages 517 — 520, Trondheim, Norway, June 1995.

[24] Kahle, E., Jullien, J.P., “Subjective Listening Tests in Concert Halls : Methodology and Results”, *Proc. of the 15th International Congress on Acoustics (ICA)*, pages 521 – 524, Trondheim, Norway, June 1995.

[25] Kronlachner, M., Zotter, F., “Spatial transformations for the enhancement of Ambisonic recordings”, *2nd International Conference on Spatial Audio (ICSA)*, Erlangen, Germany, February 2014.

[26] Larman, C., *Agile and Iterative Development : A Manager’s Guide*, Addison Wesley, 2003.

[27] Puckette, M., “The Patcher”, *Proc. of the International Computer Music Conference (ICMC)*, pages 420 – 429, San Francisco, CA, USA, 1988.

[28] Ratcliff, J., *Timecode : A user’s guide, 3rd Edition*, Focal Press, 1999.

[29] Schmeder, A., Freed, A., Wessel, D., “Best Practices for Open Sound Control”, *Proc. of the Linux Audio Conference (LAC)*, Utrecht, Netherlands, May 2010.

[30] Schmeder, A.W., Wright, M., “A Query System for Open Sound Control (Draft Proposal)”, Technical report, Center for New Music and Audio Technology (CNMAT), UC Berkeley, 2004.

[31] Wright, M., “Open Sound Control : an enabling technology for musical networking”, *Organised Sound*, volume 10(3), pages 193 – 200, Dec 2005.

[32] Zotter, F., Frank, M., “All-Round Ambisonic Panning and Decoding”, *Journal of the Audio Engineering Society*, volume 60(10), pages 807 – 820, 2012.

[33] Zotter, F., Pomberger, H., Noisternig, M., “Energy-Preserving Ambisonic Decoding”, *Acta Acustica united with Acustica*, volume 98, pages 37 – 47, 2012.