CAGE : une librairie de haut niveau dédiée à la composition assistée par ordinateur dans MAX

Andrea Agostini, Eric Daubresse, Daniele Ghisi

CAGE: UNE LIBRAIRIE DE HAUT NIVEAU DÉDIÉE À LA COMPOSITION ASSISTÉE PAR ORDINATEUR DANS MAX

Andrea Agostini
HES-SO, Genève
and.agos@gmail.com

Éric Daubresse
HES-SO, Genève
eric.daubresse@hesge.ch

Daniele Ghisi
HES-SO, Genève
danieleghisi@gmail.com

RÉSUMÉ

Cet article est une introduction à *cage*, une librairie pour l'environnement Max [1] composée d'un certain nombre de modules de haut niveau pouvant être utilisés principalement pour la composition assistée par ordinateur. La librairie, actuellement en version alpha, contient un ensemble d'outils dédiés à plusieurs catégories de problèmes typiquement abordés par cette discipline : génération de notes, génération et traitement de profils mélodiques, processus symboliques inspirés par le traitement du signal audio, interpolations harmoniques et rythmiques, automates et L-systèmes, rendu audio, outils pour la *set theory*, outils pour la gestion de partitions. Ce projet, soutenu par la Haute école de musique de Genève [2], a principalement une vocation pédagogique : en effet, tous les modules de la librairie sont des abstractions, qui se prêtent très facilement à être analysées et modifiées.

1. INTRODUCTION

Dans cet article seront abordés certains des principaux concepts de la librairie *cage* [3] pour Max ; elle contient plusieurs modules de haut niveau pour la composition assistée par ordinateur (CAO). Elle est entièrement basée sur la librairie *bach : automated composer's helper*, développée par deux des auteurs [1, 3]. Comme pour *bach*, *cage* est principalement centrée autour de la notation symbolique dans le domaine du temps réel. Les objets de *cage* communiquent entre eux avec le mécanisme des *Lisp-like linked lists* (*llll*s) [2].

À la différence de *bach*, qui se compose d'un grand nombre d'objets et d'abstractions prenant en charge des opérations de bas niveau sur ces listes (p.e. : rotations, inversions, entrelacements...), ou bien des opérations très avancées mais cependant essentiellement basiques (p.e. : résolution de problèmes par contraintes, quantification rythmique...), les modules de *cage* accomplissent en général des taches de plus haut niveau, ayant une connotation plutôt compositionnelle que strictement technique (p.e. : génération de matériel mélodique ou calcul de modulations de fréquences symboliques).

Deux critères principaux ont motivé la conception et réalisation de la librairie.

Le premier est celui qui a été à la base de la création de *cage* : construire une librairie de modules prêts à l'utilisation pour créer des classes de processus assez universels dans la pratique de CAO. Une partie de cette librairie est donc directement inspirée par d'autres librairies déjà existantes dans quelques logiciels (notamment les librairies *Profile* [9] et *Esquisse* [6, 8] pour *Patchwork*, qui ont ensuite été portées dans *OpenMusic* [4]) ; parallèlement, un autre versant de la librairie trouve sa raison d'être dans des concepts issus du monde du temps réel (p.e. : *cage.granulate*, le moteur de granulation symbolique).

Le second critère est lié à la forte connotation pédagogique du projet : à la difference de *bach*, dont les fonctionnalités principales sont implémentées dans des objets compilés, tous les modules de la *cage* sont des abstractions, qui se prêtent très facilement à être analysées et modifiées. Il n'est donc pas compliqué, pour l'utilisateur qui souhaite apprendre à manipuler des données musicales, de copier, modifier ou ajuster des morceaux de patch pour ses propres besoins. Cette flexibilité des abstractions fait en sorte que, bien que les processus implémentés soient conçus pour fonctionner facilement avec une connaissance moyenne de Max, l'utilisateur plus avancé pourra non seulement partir de ces abstractions et en modifier le comportement selon son projet, mais également les intégrer dans son propre environnement de travail, que ce soit au studio ou dans le cadre d'une performance. Cette vocation pédagogique est complétée par le fait que la librairie sera entièrement documentée, avec des fichiers d'aide, des feuilles de référence et une collection de tutoriaux.

2. UNE APPROCHE TEMPS RÉEL À LA COMPOSITION ASSISTÉE

Le paradigme du temps-réel influence profondément la nature elle-même du processus compositionnel. Par exemple, les compositeurs qui opèrent dans le domaine de la musique électroacoustique ont souvent besoin que la machine réagisse immédiatement à tout changement de paramètres. De la même manière, les compositeurs qui composent avec des donnés symboliques pourraient souhaiter que la machine s'adapte dans les plus brefs délais

1 . *http ://cycling74.com*

2 . La librairie *cage* est développée au sein de la HEM avec le soutien du fonds stratégique de la HES-SO (Projet CPE-MUS12-12 : Développement d'outils destinés à l'enseignement, la composition et l'interprétation)

3 . *www.bachproject.net/cage*

à la nouvelle configuration. Le paradigme sous-jacent à *cage* est donc le même que celui qui a façonné la librairie *bach* : la création et la modification des données musicales n'est pas forcement réduite à une activité hors du temps, mais elle suit, en s'adaptant, le flux temporel du processus compositionnel (voir aussi [3, 5, 11]).

3. COMPOSITION DE LA LIBRAIRIE

La librairie se compose de plusieurs familles de modules qui sont regroupés en catégories.

3.1. Génération de hauteurs

Une première famille de modules s'occupe de la génération des hauteurs selon des critères différents : *cage.scale* et *cage.arpeggio* peuvent générer, respectivement, des échelles et des arpèges, dans un certain ambitus de hauteurs. La typologie des accords ou des échelles peut être donné soit par nom de type symbolique (p.e. **F#m**, **ReM**), soit avec un pattern de valeurs d'intervalles exprimées en midicents (p.e. **100 200 100 200 100 200 200 100**) [4]. Les dénominations des échelles et des accords peuvent aussi bien contenir des quarts et des huitièmes de tons (p.e. **Sol+M**, **Abv7**... [5]). *cage.harmser* génère des séries harmoniques à partir d'une note fondamentale, avec éventuellement un facteur de distortion harmonique.

D'autres modules opèrent en générant des hauteurs une par une : *cage.noterandom* génère des notes au hasard, à partir d'un réservoir donné, selon plusieurs poids de probabilités prédéfinis qui peuvent être soit ignorés, soit définis simplement grâce à *cage.weightbuilder* (voir Fig. 1) ; *cage.notewalk* génère un chemin aléatoire dans un réservoir donné, selon une liste de pas admissibles. Dans les deux cas, le résultat de l'opération est conçu pour être utilisé en combinaison avec *bach.transcribe*, qui va gérer la transcription symbolique en temps réel. Ainsi, dans les deux cas, l'élément choisi au hasard peut être validé *a posteriori* par l'utilisateur via un *lambda loop*. [6]

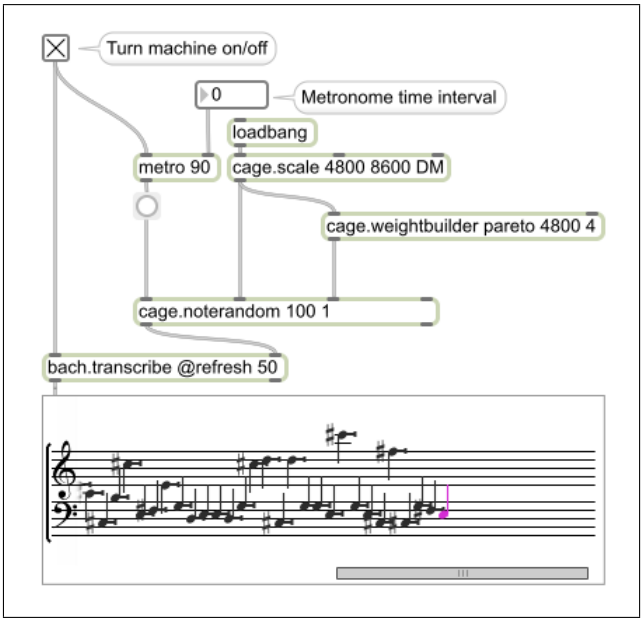

Figure 1. Génération en temps réel de notes tirées d'une échelle de ré majeur, avec des poids de probabilité donnés par une distribution de type 'pareto' avec 4800 midicents comme *pareto wall* et 4 comme valeur d'exposant.

3.2. Génération et traitement de profils mélodiques

Une famille de modules est spécifiquement dédiée à la génération et au traitement des profils mélodiques, d'une manière similaire à la librairie *Profiles* dans Patchwork et OpenMusic [9]. Une *breakpoint function*, par exemple dessinée dans un objet *function* ou un objet *bach.slot* [7] peut être convertie en une séquence de hauteurs (un profil mélodique) grâce à *cage.profile.gen*. Ce profil peut être modifié de plusieurs manières : compressé ou étiré (avec *cage.profile.stretch*), renversé (avec *cage.profile.mirror*), approximé par une grille harmonique ou par une échelle (avec *cage.profile.snap*), forcé dans une région de hauteurs (avec *cage.profile.rectify*), perturbé d'une façon aléatoire (avec *cage.profile.perturb*), ou filtré (avec *cage.profile.filter*). Dans ce dernier cas, le filtrage du profil est réalisé par l'application d'un filtre moyen, médian ou encore 'custom', défini par l'utilisateur via un *lambda loop* (voir aussi Fig. 2).

3.3. Processus d'inspiration électroacoustique

La librairie *cage* contient un groupe de modules dédiés à l'émulation symbolique de processus extraits du domaine de la synthèse sonore et du traitement du signal audio.

cage.freqshift est un outil qui permet de transposer des matériaux de façon linéaire sur l'axe des fréquences, à la

4 . *cage*, comme *bach*, adopte la convention de *Patchwork* et *OpenMusic* [4] d'exprimer les hauteurs et les intervaux en midicents, c'est-à-dire des centièmes de note MIDI.

5 . Par convention, **+** indique une altération de $+1/4$ de ton ; **-** ou **d** indique une altération de $-1/4$ de ton ; ^ et **v** font la même chose pour les huitièmes de ton.

6 . Un *lambda loop* dans *bach*, et donc par extension dans *cage*, est une configuration de feedback symbolique : les objets qui le supportent ont une ou plusieurs sorties dédiées (*lambda outlets*) qui envoient des donnés devant être validées ou modifiées ; ces données sont élaborées dans une section du patch et la 'réponse' est ensuite re-injectée dans une entrée dédiée (*lambda inlet*) de l'objet de départ. Cette configuration est très souvent utilisée dans *bach* pour définir des comportements personnalisés pour certains éléments (p.e. : un critère d'ordonnancement d'une liste, ou un processus qui doit être appliqué à chaque élément d'une *llll*, et ainsi de suite). Le nom 'lambda' est une allusion au fait que cette configuration permet, en quelque sorte, de passer un morceau de patch comme pseudo-argument d'un objet. Ce n'est pas plus qu'une allusion : il ne s'agit en aucun cas de lambda-calcul, ni de fonctions interprétées.

7 . Dans *bach*, un *slot* est en général un conteneur de méta-données associées à une certaine note [1]. L'information contenue dans un *slot* peut avoir des formes différentes, par exemple : *breakpoint functions*, nombres et listes de nombres, filtres, texte, liste de fichiers à parcourir, matrices, trajectoires de spatialisation... *bach.slot* permet l'affichage et l'édition d'un *slot* sans qu'il ne soit associé à aucune note spécifique.

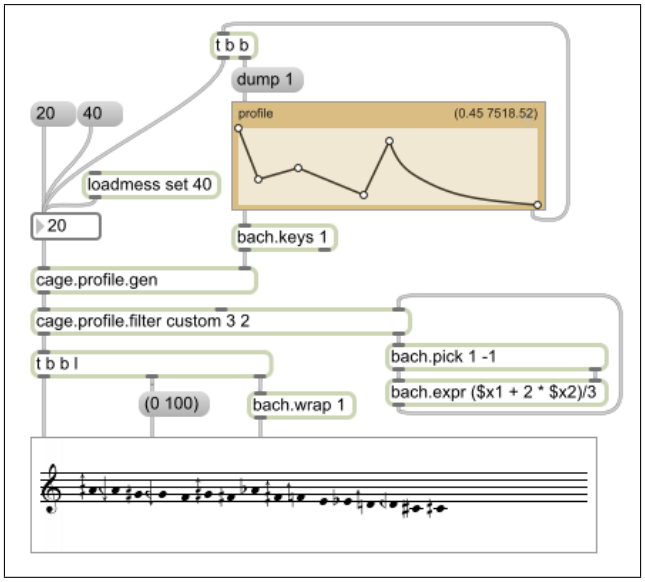

Figure 2. Un profil mélodique est construit à partir d'une fonction définie dans un *bach.slot*. Dans ce cas, la fonction affichée est échantillonnée en 20 points. Ensuite, ce profil est filtré par un processus exprimé à travers un *lambda loop*, qui fonctionne avec des fenêtres de trois notes ; pour chaque fenêtre, on substitue une moyenne du premier et dernier élément de la fenêtre, pondérée avec les poids $(1, 2)$. Ce processus de filtrage est répété deux fois. On remarque que, à cause du fenêtrage, le résultat contient quatre notes de moins que l'échantillonnage original.

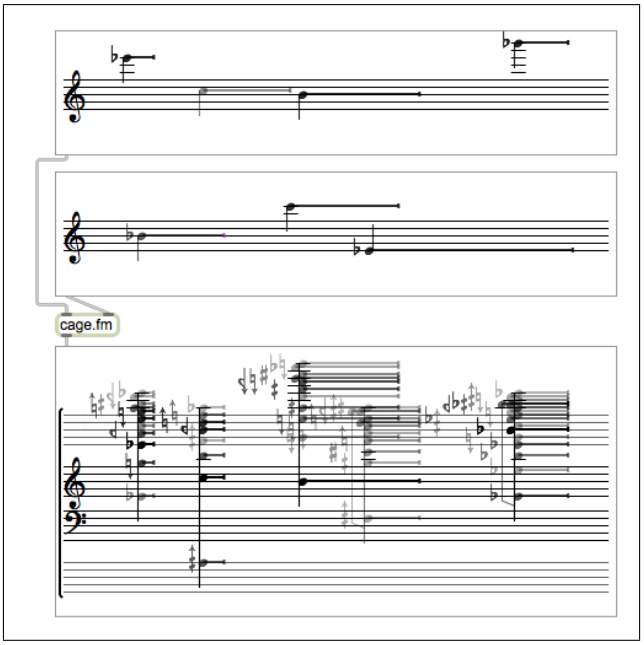

Figure 3. Un exemple de modulation en fréquence entre deux partitions, obtenue à travers l'abstraction *cage.fm*. La 'porteuse' et la 'modulante' sont en haut, le résultat en bas. La vélocité des notes (traitée dans ce cas comme l'amplitude des sinusoïdes correspondantes) est représentée par une échelle de gris.

manière du traitement audio du *frequency shifting*. En raison de la similarité des deux processus, on classifiera aussi *cage.pitchshift* dans cette même catégorie, quoique l'opération de *pitch shifting* sur la notation musicale soit, en fait, une simple transposition.

cage.rm et *cage.fm* sont dédiés respectivement à la modulation en anneaux et en fréquence. L'idée à la base de ces techniques, largement utilisées par les compositeurs associés a l'école spectrale, est la suivante : à partir de deux accords (l'un 'portant' et l'autre 'modulant') dont chaque note est considérée comme une sinusoïde simple, on calcule le spectre qu'on obtiendrait en modulant entre eux ces deux groupes de sinusoïdes. Cette opération nécessite d'effectuer des approximations et des compromis qui peuvent éloigner son résultat de celui obtenu par le même processus appliqué à des signaux audio : cependant, il s'agit d'une approche très efficace pour la génération de familles harmoniques potentiellement très riches, ceci à partir de matériaux simples, d'où leur intérêt compositionnel. Quoique l'inspiration immédiate de *cage.rm* et *cage.fm* provienne de la librairie *Esquisse* [6, 8] pour *OpenMusic*, leur paradigme de fonctionnement et certains détails relatifs aux calculs sont différents. En particulier, ces deux modules sont conçus pour travailler dans le temps, et peuvent ainsi accepter comme données d'entrée des accords simples, mais aussi des suites d'accords, afin de représenter dans le temps les variations des 'porteuses' et des 'modulantes'. Dans ce cas le processus restituera une nouvelle partition, qui tiendra compte de ces variations dans le temps (voir Fig. 3). En ce qui concerne le véritable calcul interne, les deux modules prennent en compte une estimation des oppositions de phase générées par la modulation, et donc permettent l'élimination de certaines fréquences, à la différence de la librairie *Esquisse*. Pour cette raison, les résultats d'un même processus dans les deux environnements peuvent être très différents.

cage.virtfun est un estimateur de la fréquence fondamentale virtuelle d'un accord, comme on la perçoit par exemple à l'issue d'un processus de *waveshaping*. L'implémentation est très simple : on parcourt la série subharmonique de la note la plus grave de l'accord, jusqu'à trouver une fréquence dont les harmoniques approximent toutes les notes du même accord avec un degré de tolérance établi. Il est aussi possible d'utiliser *cage.virtfun* à partir d'une séquence d'accords dans le temps : le résultat sera alors une séquence correspondant à la suite des fondamentales virtuelles de chaque accord.

cage.delay étend le principe de la ligne à retard avec réinjection dans le domaine symbolique. Il s'agit essentiellement d'un outil pour la création de boucles et de structures répétitives, qui permet d'altérer le matériau à chaque pas du processus à travers un *lambda loop*. Le temps de retard lui-même peut être changé d'une répétition à l'autre. Il n'y a pas de limitation à priori dans la richesse des processus auxquels les matériaux sont soumis dans le *lambda loop* : le résultat musical peut donc être beaucoup plus complexe qu'une simple itération.

cage.cascade$\sim$ et *cage.pitchfilter* étendent le principe

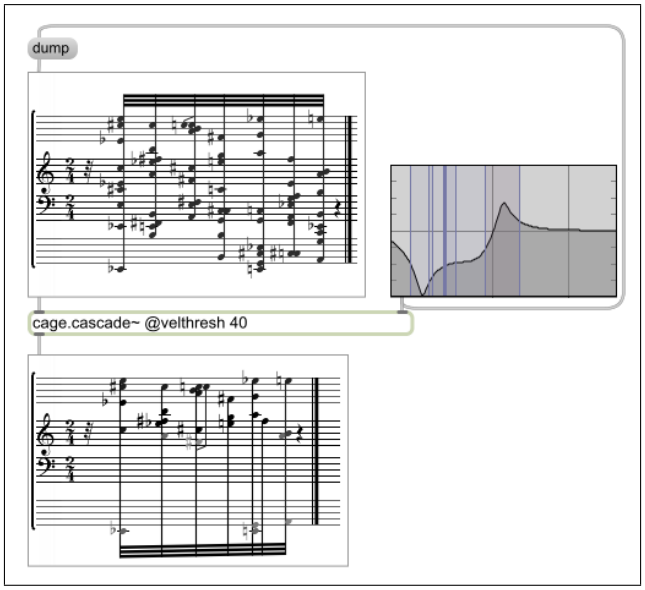

Figure 4. Un exemple de filtrage d'une partition avec l'objet *filtergraph∼* et *cage.cascade∼*. À chaque fois que le filtre est modifié par l'interface, le résultat est automatiquement mis à jour en temps réel.

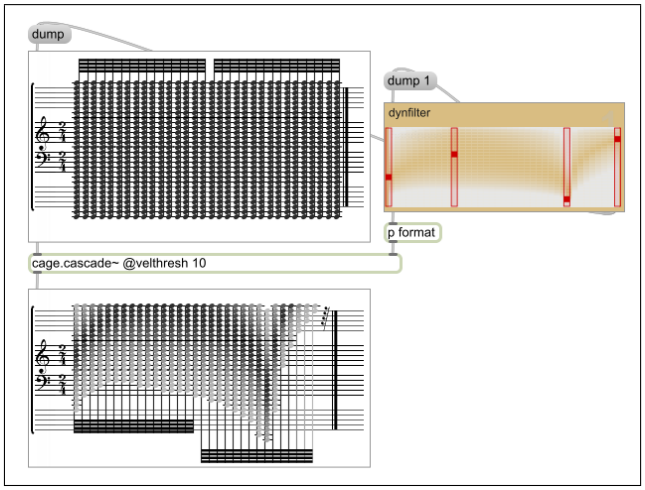

Figure 5. Un exemple de filtrage dynamique d'une partition obtenu avec *cage.cascade∼* piloté par un slot de type *dynfilter* dans l'objet *bach.slot*. À chaque fois que le filtre est modifié par l'interface, le résultat est automatiquement mis à jour en temps réel. Les quatre rectangles de couleur rouge dans le *bach.slot* à droite représentent quatre filtres passe-bande, dont les fréquences centrales sont représentées par les carrés rouges ; à différence de la Fig. 4, ici le résultat est donné par interpolation de ces filtres, et non pas par un unique filtre statique.

du filtrage dans le domaine symbolique. Le premier applique à une partition une séquence de filtres avec deux pôles et deux zéros, de façon similaire aux objets Max *biquad∼* et *cascade∼* (voir fig. 4), en émulant la réponse en fréquence d'un véritable filtre numérique [8] . Le second opère par contre sur le domaine des hauteurs (et non pas des fréquences), en appliquant à une partition un filtre défini tout simplement par une *breakpoint function*, produite par exemple par un objet *function* ou *bach.slot*. Dans les deux cas, la vélocité MIDI de chaque note est modifiée en accord avec la réponse du filtre, et les notes dont la vélocité tombe au dessous d'un certain seuil sont éliminées. Il est aussi possible de définir des changements dans l'interpolation des filtres dans le temps (voir fig. 5).

cage.granulate est un moteur de granulation symbolique. Les paramètres de granulation sont les mêmes que dans les processus correspondant en musique électroacoustique, c'est à dire : l'intervalle de temps entre deux grains, la taille de chaque grain, le début et la fin de la zone de la partition d'où le grain doit être extrait. À partir de ces données, *cage.granulate* se charge des calculs et remplit en temps réel un *bach.roll* relié à sa sortie (voir fig. 6).

3.4. Interpolations harmoniques et rythmiques, formalisation de l'agogique

Le module *cage.chordinterp* opère un calcul d'interpolation harmonique linéaire dans un ensemble d'accords, en fonction de poids attribués pour chacun d'eux par l'utilisateur. De la même manière, on peut obtenir une interpolation rythmique entre un ensemble de figures à travers le module *cage.rhythminterp*.

Un cas particulier d'interpolation rythmique, ou plutôt une extension du même principe, est la formalisation de l'agogique : c'est-a-dire, l'écriture dans un système de notation proportionnelle (qui bien sûr pourra ensuite être quantifié) d'une structure musicale répétée en accelerando ou rallentando. Cette structure musicale peut aussi être une simple impulsion rythmique constante qui représente une grille pour des figures musicales non répétitives : pour cette raison, nous avons choisi de réduire le problème au cas d'une structure itérative. Nous avons donc identifié cinq paramètres qui caractérisent un accelerando ou un rallentando : durée de la première instance, durée de la dernière instance, relation entre deux instances consécutives, nombre d'instances, durée totale de la figuration. Evidemment ces paramètres ne sont pas indépendants : en général, à partir d'un ensemble de paramètres de départ, on peut calculer les autres et formaliser entièrement l'accelerando ou le rallentando. La librairie *cage* comprend donc un groupe de modules dédiés au calcul des paramètres manquants à partir des données disponibles (un module est utilisé pour chaque combinaison de données significatives en entrée), et un module 'central' calcule au final l'accelerando ou le rallentando.

3.5. Automates et L-systèmes

Deux modules se consacrent aux systèmes génératifs et d'automates.

8 . Puisque pour calculer la réponse en fréquence d'un filtre avec deux pôles et deux zéros il est nécessaire de connaître la fréquence d'échantillonnage, l'abstraction *cage.cascade∼* fait partie des abstractions de DSP (*digital signal processing*), comme le ∼ indique, bien qu'elle agisse sur un contenu symbolique.

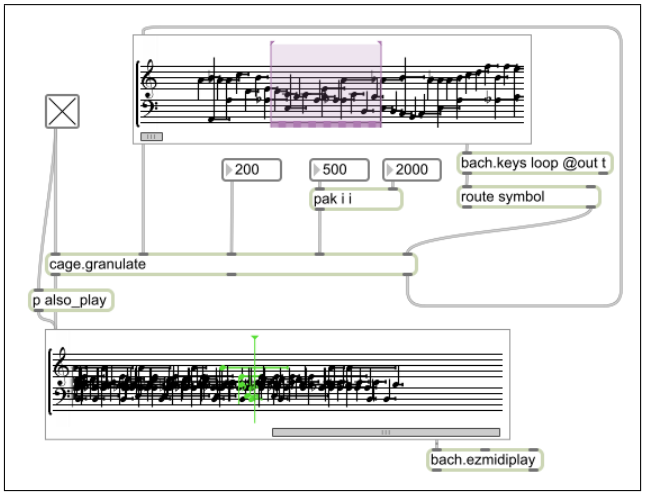

Figure 6. Un exemple de granulation d'une partition en temps réel ; les grains ont une distance de 200ms, et une taille comprise entre 500ms et 2000ms. La région de loop dans le *bach.roll* supérieur est la région d'où les grains sont extraits.

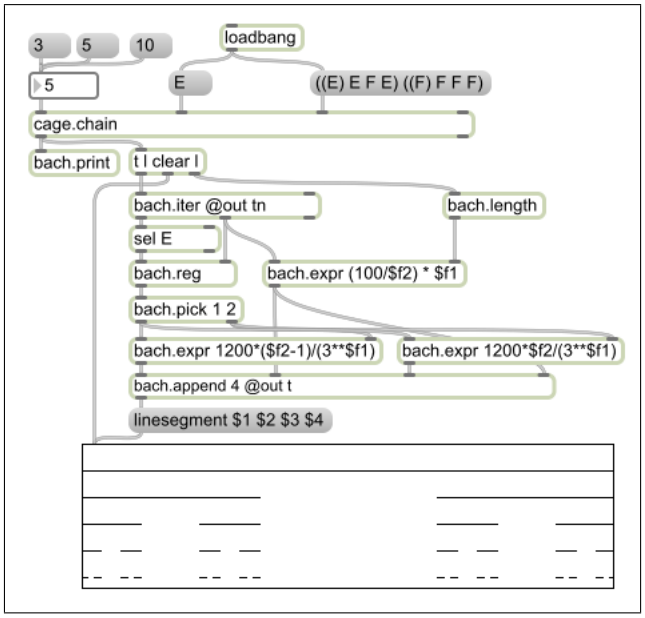

Figure 7. Un exemple de génération et d'affichage d'un comportement fractal, obtenu par une règle de substitution. La liste **E** est la liste initiale ; les règles de substitution sont : à **E** on substitue la séquence **E F E**, à **F** on substitue la séquence **F F F**. La première itération donne **E F E**, la deuxième donne **E F E F F F E F E**, et ainsi de suite. Dans l'affichage, on interprète **E** comme une ligne et **F** comme un espace vide. Ce patch génère cinq itérations qui approximent l'ensemble de Cantor. Toute la partie du patch située en dessous de *cage.chain* est dédiée uniquement à l'affichage.

cage.chain modélise des automates cellulaires en dimension 1, et de L-systèmes. Il opère des ré-écritures d'une liste donnée, selon un certain nombre de règles définies par l'utilisateur soit par message, soit via un *lambda loop*. Les substitutions peuvent intervenir sur des éléments simples (p.e. une certaine lettre ou note est substituée avec une liste de lettres ou notes), ou bien sur des suites d'éléments ayant une longueur fixée (p.e. à chaque couple d'éléments, qu'ils se présentent séparés ou superposés, on substitue un ou plusieurs autres éléments) ; dans ce dernier cas, *cage.chain* gérera le comportement aux limites de la grille en fonction des valeurs de certains attributs (*pad*, *align*). Avec ce module il est ainsi très simple de construire des automates cellulaires, ou des fractales obtenus par substitution (voir Fig. 7).

cage.life s'occupe d'automates cellulaires en dimension 2 (dont le plus fameux exemple est sans doute le 'jeu de la vie' de Conway). Les règles de ces automates sont définies grâce à un *lambda loop*. L'ordre des sous-matrices de substitution peut également être défini par l'utilisateur.

3.6. Rendu audio

Deux modules dans *cage* sont consacrés à la production d'un rendu audio de partitions *bach* : *cage.ezaddsynth*∼ (un moteur de synthèse par ondes sinusoïdales) et *cage.ezseq*∼ (un échantilloneur de fichiers sons). Les deux sont prêts à être utilisés très simplement, à la façon de *bach.ezmidiplay*, en les connectant simplement à la sortie dédiée 'playout' des objets de notation.

Le moteur de synthèse répond à la nécessité d'avoir un rendu audio qui ne soit pas basé sur une sortie MIDI. Cela est par exemple extrêmement utile quand on travaille avec des grilles microtonales non standard, ou quand on a besoin d'enveloppes liées aux notes. *cage.ezaddsynth*∼ peut prendre en compte une enveloppe d'amplitude et une enveloppe de panning (données comme slots). Il suffit ensuite de relier *cage.ezaddsynth*∼ à la sortie 'playout' des objets de notation pour avoir le rendu audio, qui prend en compte les slots pré-cités.

L'échantillonneur répond au besoin d'utiliser les objets *bach.roll* et *bach.score* comme des 'sequencers augmentés' : *cage.ezseq*∼ fournit une palette standard de slots, censés contenir pour chaque note l'information du nom du fichier, l'enveloppe d'amplitude, la vitesse de lecture, l'enveloppe de panning, le filtrage audio, et le point de départ dans la lecture du fichier (en millisecondes). Si ces informations ne sont pas toutes données, des valeurs par défaut sont utilisées. L'intérêt d'utiliser *cage.ezseq*∼ est aussi lié au fait que cet objet prend en charge la mise en mémoire ('preload') des fichiers audio, une fois qu'on lui donne un lien vers un dossier qui les contient. Ce module peut aussi s'occuper d'opérer sur chaque échantillon une transposition automatique sans altération temporelle, à l'aide de l'objet Max *gizmo*∼.

3.7. Outils de *set theory*

Certains modules dans *cage* sont dédiés à des représentations propres de la *set theory* : *cage.chroma2pcset* et *cage.pcset2chroma* opèrent des conversions entre

pitch class sets et vecteurs de chroma (voir [10]) ; *cage.chroma2centroid* et *cage.centroid2chroma* opèrent des conversions entre vecteurs de chroma et centroïdes spectraux. Le centroïde spectral d'un vecteur de chroma est obtenu via la transformée introduite par Harte et Sandler [7]. Dans le passage de chroma au centroïde, une partie de l'information originale est forcément perdue ; par conséquent, la conversion de *cage.centroid2chroma* n'est pas univoque : un seul vecteur de chroma, parmi tous ceux qui ont pour centroïde le vecteur introduit en input, est restitué.

3.8. Partitions

cage contient un groupe de modules pour le traitement global de partitions entières. Dans ce groupe, les outils d'usage le plus courants sont *cage.rollinterp*, qui opère une interpolation entre les partitions contenues dans deux objets *bach.roll*, en utilisant comme paramètre la courbe d'interpolation (ou bien une valeur fixe, dans le cas d'une interpolation statique), et *cage.envelopes*, qui représente une famille de fonctions synchronisées sur la durée totale d'une partition, et qui aident à modifier en temps réel la partition en rapport avec les valeurs des courbes à chaque instant.

4. ROADMAP

Au moment de l'écriture de ce texte, la librairie est encore dans sa phase de développement. Une version alpha publique sera disponible en mai 2014 : il est possible que toutes les fonctionnalités prévues ne soient pas encore implémentées ; la documentation ne sera pas complète. Cependant, la majorité des modules seront déjà utilisables. La première version complète de la librairie sera mise en ligne au mois de octobre 2014, à l'occasion d'une présentation officielle qui se déroulera à Genève. La librairie sera disponible sous le modèle "open source"et son téléchargement libre. À partir de l'année académique 2014-2015, *cage* sera objet d'enseignement dans les cours de composition et de musique électronique à l'Haute École de Musique de Genève et dans un ensemble d'institutions partenaires.

5. REMERCIEMENTS

cage est un projet de recherche mené au sein du centre de musique électroacoustique de la Haute Ecole de Musique de Genève, avec le soutien du domaine musique et arts de la scène de la Haute Ecole Spécialisée de Suisse occidentale. Le nom *cage*, qui dans le parallèle entre lettres et notes représente la fameuse expansion de *bach*, est aussi un acronyme en hommage à ce soutien : *composition assistée Genève*.

6. REFERENCES

[1] A. Agostini and D. Ghisi, "*bach* : an environment for computer-aided composition in Max," in *Proceedings of the International Computer Music Conference (ICMC 2012)*, Ljubljana, Slovenia, 2012, pp. 373–378.

[2] ——, "Gestures, events and symbols in the *bach* environment," in *Proceedings of the Journées d'Informatique Musicale*, Mons, Belgium, 2012, pp. 247–255.

[3] ——, "Real-time computer-aided composition with *bach*," *Contemporary Music Review*, no. 32 (1), pp. 41–48, 2013.

[4] G. Assayag and al., "Computer assisted composition at Ircam : From patchwork to OpenMusic," *Computer Music Journal*, no. 23 (3), pp. 59–72, 1999.

[5] A. Cont, "Modeling Musical Anticipation," Ph.D. dissertation, University of Paris 6 and University of California in San Diego, 2008.

[6] J. Fineberg, "Esquisse - library-reference manual (code de Tristan Murail, J. Duthen and C. Rueda)," Ircam, Paris, 1993.

[7] C. Harte, M. S, and M. Gasser, "Detecting harmonic change in musical audio," in *In Proceedings of Audio and Music Computing for Multimedia Workshop*, 2006.

[8] R. Hirs and B. G. editors, *Contemporary compositional techniques and OpenMusic*. Delatour/Ircam, 2009.

[9] M. Malt and J. B. Schilingi, "Profile - libreria per il controllo del profilo melodico per Patchwork," in *Proceedings of the XI Colloquio di Informatica Musicale (CIM)*, Bologna, Italia, 1995, pp. 237–238.

[10] M. Müller, *Information Retrieval for Music and Motion*. Springer Verlag, 2007.

[11] M. Puckette, "A divide between 'compositional' and 'performative' aspects of Pd," in *Proceedings of the First Internation Pd Convention*, Graz, Austria, 2004.