ARCHITECTURE LOGICIELLE DE LA SORTIE DE RESIDENCE D'ALEAS

Thibaud Keller
SCRIME
thibaud.keller@u-bordeaux.fr

Camille Rocailleux
E.V.E.R.
compagnie.ever@gmail.com

Bernard P. Serpette
INRIA
Bernard.Serpette@inria.fr

RÉSUMÉ

Le but du projet Aléas est de faire éclore un dialogue artistique entre l'ordinateur et l'humain, de concevoir un cadre propice à une improvisation homme-machine, de capter des mouvements et des sons pour en faire quelque chose. À mi-chemin du projet nous avons effectué une résidence de 12 jours dans le MÉCAstudio qui est un espace de l'OARA [1] à l'intérieur de la MÉCA [2] de Bordeaux. Quatre représentations sont venues clore cette résidence le 21 novembre, dans le MÉCAstudio, de 20 minutes chacune. Nous donnons ici les éléments techniques ayant contribué à la réalisation de ces événements.

1. PRESENTATION

Aléas [3] est un projet Arts&Sciences FACTS (Festival Arts Créativité Technologies Sciences) de l'Université de Bordeaux reliant un artiste de la compagnie EVER [4] et un scientifique d'Inria. Pour cette sortie de résidence à l' OARA, nous avons eu le soutien du SCRIME et la prestation de 4 artistes. Les intervenants furent :

— La compagnie EVER : Camille Rocailleux (directeur artistique et porteur du projet) et Aurélie Favre (production/diffusion).

— Inria [5] : Bernard Serpette (chercheur en informatique et porteur du projet).

— Le SCRIME [6] : Thibaud Keller (réalisateur en informatique musicale) et Gaël Jaton (régisseur).

— Artistes : Clara Pertuy (chanteuse), Karoline Rose (chanteuse), Laurie Batista (chanteuse) et Jean-Sébastien Jacques (danseur).

— L'OARA : Vanessa Lechat, (régie générale) et toute l'équipe technique.

Après un rapide récapitulatif des éléments matériels mis en jeu, nous rentrerons dans les détails de l'architecture logiciel.

2. MATERIEL

Les représentations nécessitaient la captation des voix des chanteuses ainsi que la réception des mouvements du danseur. La restitution sonore a été faite sur un ensemble de points de diffusion permettant la spatialisation de sources sonores. La salle a été configurée de telle sorte que les chanteuses étaient dans trois coins de la pièce, le public au milieu, pour apprécier les effets de spatialisation, et le danseur évoluait autour du public. Nous donnons ici les composantes matériels utilisées :

— un bracelet accéléromètre-gyroscope MetaMotionR [7] de *MbientLab* acheté pour les besoins du projet ;

— 8 points de diffusion et un caisson de grave fournis par le SCRIME ;

— 4 points de diffusion au plafond du MÉCAstudio (couplés par deux, ce qui techniquement rajoute deux points de diffusion pour la spatialisation) ;

— 3 microphones fournis par le SCRIME (*Sennheiser e 965* et *Shure SM58*) ;

— une carte son fournie par le SCRIME (*Motu Ultralight mk3*) ;

— un ordinateur portable Linux prêté par Inria [8] ;

— un ordinateur Macintosh fourni par Inria ;

— un boitier Tp-Link pour relier les deux ordinateurs par une connexion ethernet filaire ;

— enfin, une des chanteuses avait apporté sa propre table d'effets.

Techniquement, l'introduction de deux ordinateurs a été motivée par les raisons suivantes : la première vient du fait que nous n'avons pu faire marcher les outils permettant de capturer les données de l'accéléromètre que sur le système d'exploitation Linux, alors que les outils Java sont développés sur Mac. La seconde raison est que cela permet d'avoir deux postes de travail pour les ajustements, donc deux personnes peuvent travailler simultanément. Nous n'avons pas observé des montées de charge

[1] Office Artistique de la Région Nouvelle-Aquitaine (oara.fr, accédé le 19 octobre 2020)

[2] Maison de l'Économie créative et de la Culture en Nouvelle-Aquitaine (www.la-meca.com, accédé le 19 octobre 2020)

[3] www.facts-bordeaux.fr/LES-RESIDENCES/Aleas, accédé le 19 octobre 2020

[4] Eyes Voices Ears Rhythm (www.compagnie-ever.com, accédé le 19 octobre 2020)

[5] www.inria.fr/centre/bordeaux, accédé le 19 octobre 2020

[6] Studio de Création et de Recherche en Informatique et Musiques Expérimentales (scrime.u-bordeaux.fr, accédé le 19 octobre 2020)

[7] mbientlab.com/metamotionr, accédé le 19 octobre 2020

[8] Cet ordinateur a rendu l'âme deux jours avant les représentations, remplacé par un autre ordinateur du SCRIME.

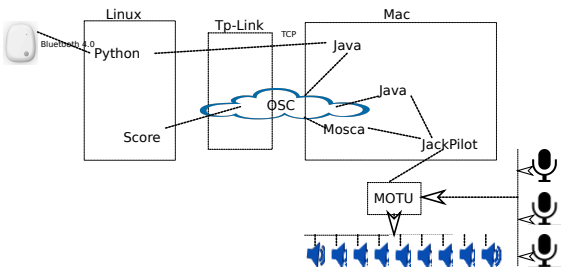

Figure 1. Connexions logicielles et matérielles

qui justifieraient l'utilisation de deux ordinateurs pour une répartition des charges.

3. BRIQUES LOGICIELLES

Entre la captation des voix, l'activité du danseur et les restitutions sonores sont intervenus plusieurs modules logiciels que nous énumérerons ici :

— des logiciels de bas niveaux liés à la carte son et aux informations délivrées par l'accéléromètre ;

— le logiciel *Jack* [9] pour la redistribution des entrées-sorties sonores ;

— la classe de *SuperCollider* Mosca [8], qui est un *plugin* pour contrôler la spatialisation. Mosca est maintenant développé et maintenu par le SCRIME [9].

— l'outil *OSSIA-Score* [10] du SCRIME [3], pour contrôler, entre autres, les objets temporels et la spatialisation – il joue le rôle de chef d'orchestre des briques logicielles ;

— du code Java contenant les outils développés pour les besoins du projet : outre les parties spécifiques à Aléas, le logiciel utilise les bibliothèques externes PortAudio [11] pour l'audio et illposed [12] pour l'interface OSC.

Mosca, *OSSIA-Score*, Java et l'accéléromètre utilisent OSC pour communiquer entre eux. OSC (*Open Sound Control*) [10] est un format de transmission de données entre ordinateurs au-dessus du protocole UDP. La plupart des langages de programmation propose une interface pour l'utilisation d'OSC.

Nous aurions pu pousser la *virtualisation* des données jusqu'à encapsuler les échantillons provenant des micros au travers de variables OSC. Pour ne pas prendre de risque quant à la latence et à la surcharge réseau, les données sonores transitent et sont redistribuées via le logiciel *Jack* qui est optimisé pour le traitement audio.

Chacune des briques logicielles expose les variables sur lesquelles elle va émettre des valeurs et annonce les variables sur lesquelles elle attend des valeurs. La brique logicielle liée à l'accéléromètre est particulière car elle ne fait que retransmettre, via des variables OSC, les données brutes venant de l'accéléromètre. Ces données ont été émises à une fréquence de 10 Hz.

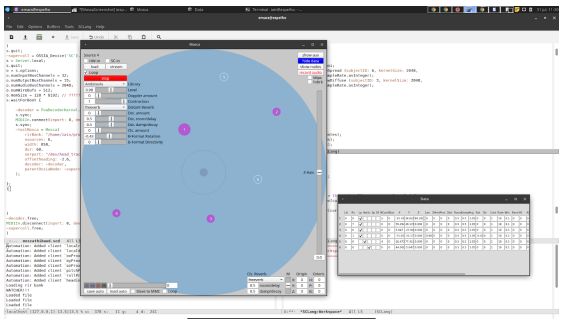

Figure 2. Interface graphique de Mosca

Nous donnons ici les caractéristiques des principales briques logicielles.

3.1. SuperCollider/Mosca

Le projet Mosca [13], créé avec le langage *Supercollider* par Iain Mott (professeur à l'université de Brazilia), est un moteur de rendu audio 3D intégrant une interface graphique optionnelle (voir figure 2), plusieurs protocoles de communications, un potentiel de simulation d'acoustique des salles et un système de suivi du mouvement de l'auditeur. L'une des particularités du programme est la liste des algorithmes de spatialisation disponibles pour chaque source sonore. Cette option permet aux utilisateurs d'entendre simultanément plusieurs techniques 3D et leurs implémentations par différents laboratoires autour du monde. L'interface présente ainsi un état de l'art basé sur un système de *plugins*, facilitant l'expérimentation avec des méthodes existantes ou encore en développement, sans avoir à modifier la base de code. Plusieurs des algorithmes testés ont depuis intégré les contributions du SCRIME et ont été ajoutés au répertoire officiel des *plugins Supercollider*. Mosca, ainsi que tous les programmes et bibliothèques annexes utilisés, sont libres et gratuits. Citons notamment :

— *Ambisonic Toolkit* [14] [2] (ATK) développé pour l'environnement Supercollider par Joseph Anderson (professeur à l'université de Seattle, Washington).

— HOALibrary [5], développé au CICM (Centre de Création Informatique et de Recherche Informatique et Création Musicale, Université de Paris 8), utilisant un ensemble de classes C++ et Faust pour le traitement ambisonique.

— *ADT* [6], développé par Aaron J. Heller (AI Center, SRI International, Menlo Park, Californie), est une bibliothèque Matlab/Octave mélangeant les techniques ambisoniques et VBAP pour l'encodage, la conversion et le décodage sur dispositifs irréguliers.

— *Ambitools* [7], développé par Pierre Lecomte (Laboratoire d'Acoustique du CNAM), vainqueur des

[9] `github.com/scrime-u-bordeaux/mosca`, accédé le 19 octobre 2020

[10] `github.com/OSSIA/Score`, accédé le 19 octobre 2020

[11] `github.com/EddieRingle/portaudio`, accédé le 19 octobre 2020

[12] `www.illposed.com/software/javaosc.html`, accédé le 19 octobre 2020

[13] `escuta.org/en/`, accédé le 19 octobre 2020

[14] `www.ambisonictoolkit.net`, accédé le 19 octobre 2020

Faust Awards 2016. Ambitools est un ensemble de programmes Faust pour la spatialisation ambisonique.

— SC-HOA [4] de Florian Grond (terminant un post-doctorat au laboratoire BIAPT de l'université McGill), permettant notamment l'intégration d'Ambitools dans l'environnement *Supercollider*.

— Plusieurs scripts et logiciels développés par Fons Adriensen (centre de recherche européen Huawei à Munich), permettant la création de réponses impulsionnelles à partir d'enregistrements de salles [1].

La variété des rendus possibles a été mise à profit pour le projet Aléas, où les sons diffusés étaient clairement séparés en 3 catégories : les séquences électroniques apportant le contexte et la structure (la bande orchestrale), les voix des chanteuses projetées en direct, les effets et transformations générés par les programmes Java. Pour les séquences pré-enregistrées, agissant comme fond sonore, un rendu plus diffus a été privilégié pour le choix des algorithmes, préférant l'homogénéité à la précision (ADT, ATK, VBAP élargi au maximum). Les voix des chanteuses ont nécessité un son plus chaud et flatteur (HOALib, Ambitools), et la variété des effets générés a imposé une sélection au cas par cas. Les distances virtuelles des sources ont aussi joué un rôle crucial quand l'atténuation induit par l'éloignement laissait la place à la position réelle de la voix dans la salle. A ces différences subtiles de rendu et de réalisme spatial est venue s'ajouter une variété de trajectoires, de taux de réverbération et d'effet Doppler pour travailler le relief de performance.

3.2. OSSIA-Score

Élaboré sur plus d'une décennie de recherches dans le domaine des automates temporels, le logiciel OSSIA-Score et les nombreux portages de son API pour différents environnements (ossia-pd, ossia-cpp, ossia-qml, ossia-max, etc.) a, notamment, fait l'objet de la thèse de Jean-Michaël Celerier, soutenue en 2018 [3].

L'interface utilisateur (voir figure 3) se présente sous la forme d'un séquenceur classique, rendant compte du passage du temps par un balayage des éléments graphiques de gauche à droite.

Cependant, le modèle diffère principalement par son arborescence, permettant une exécution parallèle d'intervalles temporels indépendants. Cette multiplicité des lignes chronologiques (*timelines*) et la hiérarchie qui les organisent permettent un ensemble d'opérations sur les objets temporels et un déroulement interactif de séquences prédéfinies.

Le programme se distingue aussi par le grand nombre de protocoles de communication supportés (OSC, OSC-Query, HTTP, Artnet, etc.) multipliant le potentiel d'interopérabilité et de répartition du système. *OSSIA-Score* s'apparente ainsi à un langage de programmation visuel.

Un objet temporel primitif est l'état (`State`), décrivant l'instantané, pouvant contenir un agglomérat unique de plusieurs paramètres, internes ou externes, ainsi que

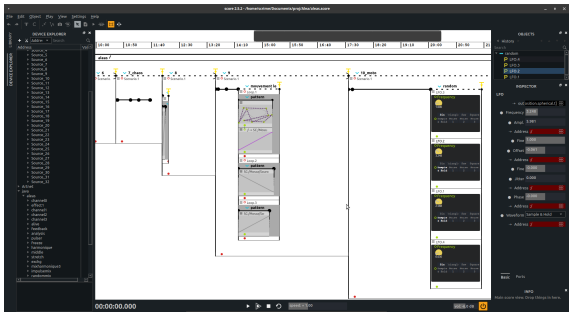

Figure 3. Interface OSSIA-Score

plusieurs processus : fonction de transfert, évaluation de code JavaScript, etc. Les états sont utilisés notamment pour enregistrer et rappeler des ensembles de préréglages (`Snapshot`).

Deux états peuvent être reliés horizontalement par un intervalle (`Interval`) définissant la durée absolue ou relative qui les séparent. Cette durée peut aussi contenir l'exécution de processus continus : synthèse audio, séquences MIDI, courbes d'automation, expressions arithmétiques, etc.

La synchronisation de plusieurs états est représentée par une connexion verticale (`Sync`). Par défaut, les états synchronisés sont contenus dans le même événement (`Event`), cependant, une fonction de division (`Split condition`) permet de créer des embranchements conditionnels.

Représentées aussi sous formes d'objets graphiques, les conditions (`Condition`) garantissent l'exécution des événements, des é

Les déclencheurs (`Trigger`) ne conditionnent quant à eux que la fin de l'exécution des intervalles et des processus qu'ils contiennent. Équivalent à l'instruction *when*, ils permettent de décrire la souplesse des durées.

Les boucles *while* sont représentées par le processus de boucle (`Loop`) et les processus scénarios (`Scénario`) dénotent l'appartenance (*ownership*).

Dans le cas qui nous occupe ici, aucun son n'était produit directement par *OSSIA-Score*, seules des données de contrôle y étaient traitées et communiquées aux moteurs audio (Mosca et le programme Java). La performance en 11 actes a été définie dans le logiciel par une succession de 11 scénarios dont les durées variables dépendaient de la longueur des extraits sonores pré-édités, de l'improvisation du danseur et des signes d'une des chanteuses (Laurie Batista).

3.3. Machine synchrone réactive

Comme nous l'avons vu, les interactions entre les briques logiciels se font au travers du protocole OSC. La bibliothèque que nous avons utilisée (*Java OSC*) s'utilise via un mécanisme de souscription : on peut attacher une fonction à une variable, ou à un groupe de variables OSC. Cette fonction, que nous appellerons une *alerte* (*call-back* en anglais), sera appliquée chaque fois que l'environnement extérieur affectera une nouvelle valeur à cette va-

riable. Ceci donne le côté *réactif* de la machine : les effets réagissent à certains événements externes.

Les observateurs des événements externes peuvent aussi générer des événements internes. Par exemple, un élément logiciel peut observer les données brutes venant de l'accéléromètre et produire un événement spécifique lorsque ces données dépassent un certain seuil : un détecteur d'impulsion. Pour des raisons d'efficacité les événements internes ne produisent pas des événements OSC ; en revanche, pour rester uniforme, les événements OSC sont systématiquement répercutés vers des événements internes.

A ce niveau, il reste possible d'utiliser la même technique d'alerte pour les événements internes : le déclenchement d'un événement interne va appliquer un ensemble d'alertes attachées à cet événement, chacune de ces alertes pouvant générer un ou plusieurs événements qui eux-mêmes iront déclencher d'autres alertes. Si les alertes sont exécutées *au plus tôt*, dès l'activation d'un événement, on obtient une évaluation des alertes en *profondeur d'abord*. Si les alertes sont mises en veille tant que l'alerte principale n'a fini son exécution, l'évaluation est en *largeur d'abord*.

Nous avons opté pour la deuxième stratégie. La machine sélectionne un ensemble d'alertes en attente et active ces alertes dans un ordre arbitraire. Si l'une ou plusieurs de ces alertes activent un événement, les alertes associées seront activés à cycle suivant. La sélection suivie des exécutions correspond à un instant *logique* de la machine. Cette décomposition en instant logique donne l'aspect *synchrone* de la machine.

La machine, schématisée dans la figure 4 comporte un *séquenceur*, qui enchaîne les instants logiques, et des emplacements mémoire contenant les valeurs courantes des événements.

Les emplacements mémoire contiennent une valeur courante (la dernière valeur émise par l'événement), un ensemble d'alertes (symbolisées par des étoiles) en attente de l'événement et éventuellement la valeur à émettre à l'instant suivant (écriture en attente). Le séquenceur, pour un instant logique, collecte les alertes associées aux emplacements mémoire ayant une écriture en attente en affectant la valeur courante dans la foulée, collecte aussi les alertes provenant des événements extérieurs (notamment ceux provenant d'événements OSC). Une fois la collecte effectuée, toutes les alertes sont exécutées dans un ordre arbitraire. Ainsi les écritures de la mémoire sont faites en *même temps* (aspect synchrone), ce qui fait que toutes les alertes collectées ont la même vision de la mémoire. Il ressort que, si on interdit deux pré-écritures d'un emplacement mémoire dans le même instant, l'état de la mémoire est indépendant de l'ordre dans lequel on actionne les alertes.

Lorsque les alertes, par causalité, s'enchaînent d'instant logique en instant logique, la fréquence de ces enchaînements est dépendante du temps de calcul des dites alertes. Si durant un instant logique aucune pré-écriture n'est établie, la machine se met en attente bloquante sur les événements externes : événements OST, déclenchement de timers, etc.

Pour cette sortie de résidence, la fréquence des instants logique a été calée sur la fréquence audio (44,1 kHz). Une alerte, attachée à l'emplacement mémoire d'un des micros, effectue une lecture bloquante sur le flux d'entrée et engendre une pré-écriture sur les mémoires dédiées aux micros. Cette pré-écriture déclenchera la même alerte l'instant logique suivant. La même alerte est responsable de la sortie physique des échantillons sur les flux de sortie.

Comme le flux d'entrée est tamponné (*bufferisé*), la lecture des échantillons provenant du microphone n'est effectivement bloquante qu'une fois sur n, où n est la taille du tampon d'entrée. Le graphique qui suit (figure 5) donne le temps passé entre deux instants logiques (fonction `System.nanoTime` de Java). L'espace entre les pics correspond à la taille du tampon d'entrée. Les autres pics, plus faibles et plus irréguliers, correspondent soit à du calcul local (une alerte attachée à un événement externe comme un réveil par exemple), soit à du calcul lié au langage de programmation (*garbage collection*), soit à l'interruption d'un autre processus du système d'exploitation.

Voici quelques exemples d'effets utilisés pour le projet Aléas :

— L'accéléromètre a été essentiellement utilisé comme déclencheur d'événements via un geste brusque du danseur. Par exemple, le danseur a été responsable du lancement de fichiers audio.

— Avec deux événements successifs venant de l'accéléromètre, le danseur détermine une portion du signal sonore provenant d'une chanteuse. Cette portion est mémorisée et restituée à l'envers à la fin de la section.

— Sur l'impulsion de l'accéléromètre, on retarde progressivement le signal sonore provenant d'une chanteuse, l'effet fait donc apparaître un *écho*. Au bout

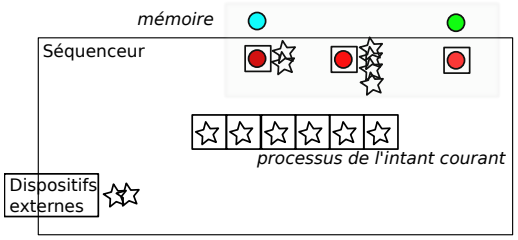

Figure 4. Structure générale de la machine

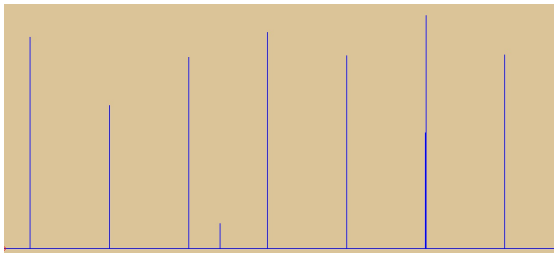

Figure 5. Temps entre deux instants logiques

d'un certain temps, l'effet inverse le processus pour re-synchroniser l'écho à sa source.

3.4. Détection temps réel d'une fréquence audio

Les effets produits par la machine sont en réaction, soit à l'accéléromètre attaché au poignet du danseur, soit aux voix des chanteuses. La machine est en continuelle écoute des signaux provenant des micros. Certains des effets utilisent directement le signal audio : écho, réverberation, spatialisation, inversion ; d'autres effets vont entrer dans les détails de la structure du son : amplitude, fréquence, timbre. Pour ce faire, nous utilisons un module de détection de fréquence.

Figure 6. Détection des périodes

Supposons qu'une portion du signal audio reçu par un des micros corresponde à la courbe se trouvant en haut de la figure 6. Le but de l'algorithme est de calculer les temps symbolisés par les traits verticaux en bas de cette figure. Ces instants sont déterminés de telle manière que les portions contenues dans deux intervalles de temps successifs soient *similaires*

Les trois principales méthodes pour estimer la similarité entre deux portions de fonctions sont basées sur la minimisation d'une distance. Ces trois différentes distances se retrouvent dans la figure 7.

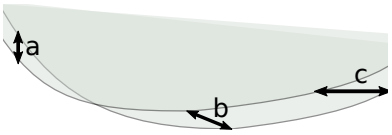

Figure 7. Distances entre deux courbes

On peut utiliser une distance verticale (la distance *a* sur la figure). Dans ce cas, on subdivise les deux portions de fonctions en n portions de tailles égales et on calcule la moyenne des distances verticales sur les points de la subdivision. Plus n est grand, plus la valeur est fiable, à l'infini on calcule la surface entre les deux courbes.

On peut utiliser la distance entre des points singuliers des deux courbes. Dans le cas de la distance *b* de la figure, les points singuliers sont les extremums locaux des deux fonctions (le minimum local pour l'exemple). Dans le cas de la distance *c*, les points singuliers sont ceux qui ont une ordonnée donnée. Lorsque l'ordonnée est 0, on cherche donc les points où les courbes croisent l'axe des abscisses, on parle alors de *0-crossing*. Le problème de cette méthode est que si on trouve un point singulier dans une courbe, il peut disparaitre dans la seconde : un extremum peut se dissoudre ou la courbe peut localement passer en dessous d'une certaine ordonnée.

L'algorithme que nous avons utilisé pour cette sortie de résidence est une variante de celui minimisant la distance *a* où l'on se permet de s'écarter de la version optimum en terme de distance si l'on peut retrouver, pas trop loin, un point à la fin de la deuxième fonction ayant la même ordonnée que le point de début, ceci afin de retrouver la propriété principale des fonctions périodiques $f(x) = f(x + \omega)$ où ω est la période de la fonction f et donc correspond à la longueur de l'intervalle que l'on cherche à calculer.

Une fois qu'un nouvel intervalle est trouvé, disons l'intervalle $[t1, t2[$ pour une fonction f, on va *normaliser* cette fonction par la fonction qui prend des valeurs entre 0 et 1 et rend des valeurs entre -1 et 1. la fonction normalisée sera notée $\hat{f}$. Les deux fonctions sont identiques à un changement d'échelle près. Le changement d'échelle sur l'axe des abscisses correspond à la fréquence du signal et le changement d'échelle sur l'axe des ordonnées correspond à l'amplitude du signal. La fonction $\hat{f}$ sera appelée (un peu abusivement) le timbre de f.

Nous venons de voir les possibilités de trouver une fréquence à partir d'une portion déjà calculée. Reste à déterminer une première portion du signal correspondant à la fréquence. Comme l'algorithme pour détecter une première fréquence est potentiellement beaucoup plus couteux (quadratique versus linéaire), nous avons opté pour la technique la plus simple, à savoir en observant les passages par l'origine (0-crossing).

Du point de vue de la machine réactive, le détecteur de fréquence est une alerte attachée à un micro, qui, de manière régulière, va écrire dans une variable contenant un triplet (a, f, τ) où a sera l'amplitude du signal, f sa fréquence et τ son timbre. Le signal d'origine, peut être reconstruit par $t \to a * \tau(f * t)$.

Une fois que les périodes successives sont disponibles, il est possible d"effectuer une transformée de Fourrier rapide (FFT) dont les résultats sont particulièrement pertinents. Nous avons ainsi accès aux diverses *harmoniques* d'une voix. D'une manière inverse, il est possible de (re)générer le signal audio d'origine.

Voici quelques effets utilisés pour le projet Aléas où intervient le module de détection de période et éventuellement la FFT :

— Pour deux triplets (a_1, f_1, τ_1), (a_2, f_2, τ_2), on peut calculer une valeur moyenne :

$$\left(\frac{a_1 + a_2}{2}, \frac{f_1 + f_2}{2}, t \to \frac{\tau_1(t) + \tau_2(t)}{2}\right) \quad (1)$$

Si on applique cette transformation aux triplets venant de deux micros, l'effet produit une *nouvelle* voix. Cet effet est particulièrement intéressant car si la transformation avait été faite directement au niveau du signal audio, les deux voix seraient restées différentiées.

— Une fois qu'une source a été décomposée en harmoniques, il est possible d'en extraire une composante particulière. Par exemple, un des effets a été de diffuser la 3e harmonique (quinte) d'une des chanteuses.

La spatialisation de cette voix a été faite au pôle opposé de la chanteuse, comme un effet miroir.

— La décomposition en harmoniques a aussi permis de *créer* une nouvelle voix en prenant successivement les harmoniques dans les trois voix des chanteuses.

— Dans l'effet précédent, les deux premières harmoniques ont le plus d'effet sur le résultat final. Pour obtenir une voix plus *radicale* on peut prendre les harmoniques de manière aléatoire.

— Un effet immédiat, que l'on peut obtenir après avoir décomposé une voix en un triplet (a, f, τ), est de figer cette voix en ne mettant plus à jour les nouvelles valeurs de ce triplet. Un des effets a été de maintenir, sur la décision du danseur via l'accéléromètre, les trois voix des chanteuses en trois triplets (a_1, f_1, τ_1), (a_2, f_2, τ_2) et (a_3, f_3, τ_3), puis, toujours sur l'impulsion du danseur, les trois fréquences f_1, f_2 et f_3 sont modifiées lentement pour converger vers un accord prédéfini (*si* mineur pour notre cas). Une fois l'accord établi, l'effet se dissout en diminuant les trois amplitudes a_1, a_2 et a_3.

4. TRAVAUX EN RESIDENCE

Au départ de la résidence, la structure orchestrale de la pièce était établie et les briques logicielles étaient fonctionnelles. La sortie de résidence a nécessité :

— un calibrage spécifique du MécaStudio afin de permettre la configuration de Mosca pour la spatialisation ;

— l'inter-connexion des briques logicielles. Autant la connexion entre Mosca et *OSSIA-Score* était bien rodée, autant la connexion entre les outils du SCRIME et le code Java n'avait jamais été testée ;

— un calibrage et une validation de ce qu'on pouvait attendre de l'accéléromètre porté par le danseur. Nous avions envisagé de contrôler certaines spatialisations via les mouvements du danseur, mais *in-fine* l'accéléromètre a été cantonné à un rôle de déclencheur ;

— la définition et l'implémentation des effets énumérés dans ce papier ;

— l'écriture de la chorégraphie du danseur ainsi que le rôle qu'il prendrait dans la représentation ;

— la mise en place du scénario en *Score*. La structure orchestrale initiale a été divisée en dix sections indépendantes, calibrées dans le temps, spatialisées et orchestrées par *Score*.

5. CONCLUSION

Nous laisserons les derniers mots à la direction artistique du projet qui conclura par son ressenti quant à l'utilisation de ces briques logicielles.

Dans ma démarche artistique qui se veut à la fois innovante et intuitive, ce projet ALEAS m'ouvre des perspectives très enthousiasmantes sur la façon de concevoir mes spectacles dans le futur. En effet, la découverte des logiciels pendant l'élaboration de la performance et surtout la possibilité de les éprouver en situation, pour modifier, transformer, augmenter les interactions entre les interprètes et le dispositif, me permettent d'imaginer très concrètement des manières totalement nouvelles, pour moi, de penser et d'écrire mes prochains projets scéniques.

Je pressens un potentiel énorme et ai déjà pu apprécier la souplesse de ces outils, leur capacité à s'adapter à mes besoins, à donner formes à mes idées voire de les dépasser et ouvrir des champs d'exploration que l'on ne soupçonne pas toujours, et cela dans un délai relativement court.

Bref, il s'agit là d'outils très performants, très malléables, et très stimulants en terme de créativité, notamment dans le cadre de la fabrication d'un objet scénique proposant une interaction véritable entre l'ordinateur et les artistes au plateau.

6. RÉFÉRENCES

[1] Adriaensen, F. « Acoustical Impulse Response Measurement with ALIKI », Proceedings of the Linux Audio Conference, Karlsruhe, Allemagne, 2006

[2] Anderson, J. « Authoring complex Ambisonic soundfields: An artist's tips & tricks », Proceedings of the Digital Hybridity and Sounds in Space Joint Symposium, University of Derby, Royaume-Uni. 2011

[3] Celerier, J-M. « Authoring interactive media : a logical & temporal approach », thèse de doctorat, sous la dir. de M. Desainte-Catherine, Université de Bordeaux, 2018

[4] Grond, F., Lecomte, P. « Higher Order Ambisonics for SuperCollider », Proceedings of the Linux Audio Conference, Saint-Etienne, 2017.

[5] Guillot, P., Paris, E., Deneu, M. « La bibliothèque de spatialisation HOA pour Max/MSP, Pure Data, VST, FAUST », *Revue Francophone d'Informatique et Musique* 3 (2013).

[6] Heller, A. J., Benjamin, E. M. « The Ambisonic Decoder Toolbox: Extensions for Partial-Coverage Loudspeaker Arrays », Proceedings of the Linux Audio Conference, Karlsruhe, Allemagne 2014.

[7] Lecomte, P. Gauthier, P. A. « Real-time 3D Ambisonics Using Faust, Processing, Pure-Data, and OSC », Proceedings of the International Conference on Digital Audio Effects, Trondheim, Norvège, 2015.

[8] Mott, I., Keller, T. « Three-dimentional sound design with Mosca », Anais do 18º Encontro Internacional de Arte e Tecnologia, Lisbonne, Portugal, 2019.

[9] Revell, L. « Realtime audio vs. linux 2.6 », Proceedings of the Linux Audio Conference Karlsruhe, Allemagne, 2006.

[10] Schmeder, A., Freed, A., Wessel, D. « Best Practices for Open Sound Control », Proceedings of the Linux Audio Conference Utrecht, Pays-Bas, 2010.

Texte édité par Corentin Guichaoua.