# UNE MÉTHODE D'ÉCRITURE MUSICALE DANS UN CONTEXTE TEMPS RÉEL SOUS SUPERCOLLIDER

*Sébastien Clara*
Université de Lyon
sebastien.clara@univ-st-etienne.fr

**RÉSUMÉ**

*Phantom Limb*[1] est une improvisation dirigée pour un claviériste et les sonorités qui la matérialisent sont exclusivement d'origine synthétique. La composition de cette pièce est le fruit de mon travail de recherche doctorale et elle valide l'usage artistique des outils développés à cette attention. Le dispositif de cette pièce est constitué principalement d'un clavier MIDI standard, d'un système d'automatisation et de trois niveaux de contrôle. Les réactions automatiques du dispositif dépendent de ce que joue le claviériste et de la manière dont il joue.

Cet article me donne l'occasion de présenter ma méthode d'écriture musicale sous SuperCollider dans un contexte temps réel. Après un bref aperçu de l'informatique musicale temps réel du XX$^{e}$ siècle et une présentation des spécificités de ce langage de programmation du point de vue d'un utilisateur, je détaillerai l'architecture de ma pièce afin de m'intéresser à son ossature, à son articulation temporelle, à son système d'automatisation et à ses trois niveaux de contrôle.

## 1. INTRODUCTION

En 2002, James McCartney rédige un article pour décrire la genèse et présenter les capacités de son langage de programmation [7]. Le constat technique qu'il établit à partir des solutions audionumériques de la fin du XX$^{e}$ siècle est à l'origine de ses motivations pour démarrer le développement de SuperCollider.

SuperCollider est un langage orienté objet similaire à Smalltalk. Tous ses éléments sont des objets et ces derniers sont organisés en classes. La classe UGen fournit l'abstraction des unités de génération (*unit generator*). L'assemblage de différentes unités produit un synthétiseur ou un traitement particulier. Dans son article, James McCartney dénomme ces structures sous le terme générique d'instrument. Pour ma part, j'emploierai le terme d'instrument ou de module.

Dans SuperCollider, un module est construit de manière fonctionnelle. Lorsqu'on écrit une fonction dans ce but, son exécution instancie et connecte les unités de génération dynamiquement. Avec SuperCollider, la construction d'un dispositif audionumérique n'est pas statique et ce langage offre la faculté de redéfinir complètement la structure modulaire de ses instruments à la volée et cela sans générer des clics sonores imprévus.

Les possibilités d'interaction offertes par les outils modernes me semblent aujourd'hui une option indispensable. En effet, les pionniers de l'informatique ont œuvré pour obtenir cette souplesse d'interaction entre l'homme et la machine. Les premiers ordinateurs travaillaient par traitement par lots. Ce fonctionnement consistait à fournir un programme aux ordinateurs sous forme de cartes perforées. L'ordinateur traitait ces informations sans que l'opérateur puisse intervenir et à la fin de son traitement, l'ordinateur délivrait le résultat qu'il soit réussi ou erroné.

Dans son ouvrage consacré à l'éthique des *hackers* [6], Steven Levy relate l'émergence de cette culture au sein du M.I.T. et sa consolidation sur la côte ouest des États-Unis avant qu'elle n'essaime sur le reste de la planète. Il montre aussi l'intérêt de plusieurs générations d'informaticiens au M.I.T. durant les années 1960 pour accroître l'interaction homme-machine temps réel afin de dépasser les usages de leur temps.

Avant d'exposer dans la quatrième section de cet article ma méthode d'écriture dans un contexte temps réel, je souhaite revenir sur l'informatique musicale temps réel du XX$^{e}$ siècle dans la deuxième section et montrer dans la troisième section la contribution du langage de James McCartney du seul point de vue de l'utilisateur.

1 *Phantom Limb* une est pièce électronique interprétée et elle a été composée en 2018 : https://soundcloud.com/s-bastien-clara/phantom-limb, consulté en avril 2019.

## 2. L'INFORMATIQUE MUSICALE TEMPS RÉEL DU XX$^{E}$ SIÈCLE

La première version du langage Music développé par Max Mathews date de 1957 et l'architecture de sa descendance hérite des technologies de l'époque de sa création. En effet, les ordinateurs de cette période demandent plusieurs minutes pour calculer une seconde de son. Dans ces conditions, l'interaction musicale entre une machine numérique et un humain est impossible. À la fin des années 1960, Mathews et Moore mettent au point le système Groove pour pallier cette carence. Cependant, Groove est un système hybride. L'ordinateur n'est toujours pas capable de calculer des sons en temps réel, mais il peut générer du contrôle en temps réel. L'ordinateur commande alors un synthétiseur analogique [9].

De même, Max est le produit des contraintes techniques de l'époque de sa naissance et son architecture est fortement influencée par les travaux de Max Mathews sur l'audionumérique temps réel [8]. Durant les années 1980, Miller Puckette utilise ses premiers travaux sur le sujet pour les consacrer au contrôle temps réel du processeur audio analogique 4X développé au sein de l'IRCAM.

D'autre part, Miller Puckette se conforme au paradigme de raccordement[2]. En effet, de nombreux autres langages graphiques de raccordement dédiés à la musique ou à d'autres usages étaient apparus lorsque Miller Puckette a commencé à développer l'interface graphique de Max [8]. Cette interface permet aux utilisateurs des synthétiseurs modulaires de s'approprier son logiciel en exploitant leur expérience et elle offre un cadre plus accueillant qu'un langage de programmation textuel aux musiciens intéressés par l'informatique musicale, mais sans culture scientifique *a priori*.

Durant les années 1980, plusieurs fabricants de synthétiseurs se concertent pour mettre au point une norme commune de communication entre les différents équipements musicaux. Ces discussions aboutissent à la spécification MIDI 1.0 et elle sera publiée en 1983 par l'International MIDI Association. Cette nouvelle norme est rapidement adoptée par l'industrie et le contrôle MIDI devient une fonctionnalité standard du synthétiseur moderne.

Vers le milieu des années 1980, les ordinateurs personnels deviennent largement abordables et certains modèles sont équipés par défaut d'une interface MIDI, notamment l'Atari ST. Dans un contexte temps réel, les ordinateurs étaient employés comme interface entre les contrôleurs et les sources sonores afin de stocker et d'utiliser des préréglages ou des séquences MIDI durant la performance. Puis à l'aide de logiciels tel que Max, l'utilisateur a eu accès aux données MIDI et il a pu concevoir des traitements temps réel sur cette ressource. Cette dernière démarche a permis d'augmenter les capacités d'un dispositif par la mise en place d'une nouvelle relation entre le musicien et la machine.

Au tournant du XX$^{e}$ et du XXI$^{e}$ siècle, les ordinateurs deviennent suffisamment puissants pour calculer du son en temps réel et cette augmentation des capacités des unités de calcul a eu pour conséquence d'amoindrir considérablement l'importance des équipements dédiés à la génération du son au profit de solutions exclusivement logicielles. Une attention particulière a été accordée à l'analyse des données (son, geste ou vidéo) tout en générant simultanément du son en temps réel. Des solutions multimodales ont été développées pour exploiter au maximum les données produites par un musicien et affiner le potentiel expressif d'un dispositif numérique.

SuperCollider a été conçu dans ce contexte de dématérialisation de tous les équipements dédiés ou non à la musique. L'ordinateur devient le point d'ancrage des contrôleurs afin de faciliter son usage en temps réel, mais certains s'en dispensent, notamment les *livecodeurs*[3]. James McCartney n'a plus les mêmes contraintes technologiques que Max Mathews ou Miller Puckette pour créer son langage de programmation. Les instruments sont alors numériques et ils peuvent être multipliés à la volée dans la limite des capacités de l'ordinateur.

## 3. LES SPÉCIFICITÉS DE SUPERCOLLIDER

SuperCollider est un langage de programmation orienté objet dédié au traitement, à la synthèse sonore temps réel et à la composition algorithmique. SuperCollider est un logiciel gratuit et son code source est librement consultable. Sa première version date de 1996 et elle a été développée par James McCartney. Depuis 2002, SuperCollider est publié sous licence GPLv2 et il rassemble une large communauté de chercheurs, d'artistes et de musiciens.

2 Les équipements électroniques dédiés à une tâche particulière peuvent être considérés comme des boites noires. Les manuels détaillent les paramètres de contrôle et les différents types d'entrées-sorties, mais le fonctionnement interne n'est généralement pas divulgué. Dans le domaine du son, les studios d'enregistrement disposent d'équipements standardisés de traitements du son et qui sont empilés dans des armoires dédiées. L'une des activités des ingénieurs du son est de les relier ensemble afin de réaliser un traitement particulier. Cette activité de raccordement pour traiter le son est similaire chez les guitaristes électriques. Ils ont en effet la possibilité de traiter le son capté de leur instrument par une série de pédales d'effets avant d'atteindre l'amplificateur et le haut-parleur de diffusion. Le raccordement des équipements est une tâche fondamentale pour la musique électronique et la complexité du raccordement est corrélée au nombre d'équipements à raccorder.

3 Le défi des *livecodeurs* est de développer des instruments numériques et de les jouer durant le temps de la représentation. Le clavier informatique et un langage de programmation optimisé pour cet usage particulier sont les interfaces principales entre le musicien et la musique qu'il produit. Chez le *livecodeur*, la dextérité de la programmation en temps réel est exaltée. La coutume veut que celui-ci débute sa prestation avec une page numérique vierge. De plus, l'écran de son ordinateur est projeté pour que le public puisse voir le *livecodeur* à l'œuvre. Cette démarche de transparence permet de donner des clés de compréhension de la musique au spectateur, mais aussi de mettre en valeur les compétences et la virtuosité des *livecodeurs* [4].

La pièce que je présente dans cet article n'aurait jamais vu le jour sans le support de la communauté[4], la documentation du langage[5], sa base de données d'exemple[6] et les nombreuses bibliothèques conçues par les utilisateurs[7]. Bien que la syntaxe de SuperCollider ait quelque peu évolué depuis la publication en 2011 d'un ouvrage consacré à ce langage [10], et bien que la création d'interface graphique ainsi que son environnement de développement intégré aient été unifiés sous différents systèmes d'exploitation avec l'interface de programmation applicative Qt, l'ouvrage de 2011 reste la référence pour se l'approprier.

Dans son article manifeste pour repenser les outils de l'informatique musicale [7], James McCartney énumère quelques abstractions disponibles dans les langages de programmation modernes, abstractions qu'il a implantées dans SuperCollider. Mais pour cette section, je présente quelques éléments du langage qui me semblent intéressants pour les utilisateurs.

### 3.1. Client/serveur

Depuis la version 3 de SuperCollider, le moteur audio et l'interpréteur du langage ont été séparés et ils ont été soumis à une architecture client/serveur. Les éléments constitutifs de cette architecture utilisent le protocole *Open Sound Contro*l (OSC) pour communiquer entre eux. En 2002, McCartney indique qu'il est possible d'optimiser les ressources des machines qui possèdent plusieurs processeurs en multipliant le moteur audio et en répartissant les processus sonores entre les différents serveurs. Cependant, Tim Blechmann a implémenté un moteur audio alternatif au début des années 2010 qui permet d'exploiter au mieux les processeurs multi-cœurs et d'optimiser le fonctionnement temps réel à faible latence de SuperCollider [1].

Toutefois, cette architecture client/serveur est fort utile pour répartir les clients sur le réseau et mutualiser les ressources de diffusion connectées à la machine dédiée au moteur audio, spécificité appréciée du *livecoding* de groupe. De même, il est possible de synchroniser facilement des événements entre les différents clients et d'aboutir à la création d'une musique pulsée. Dans ce registre, une communauté de *livecodeurs* ambitionne de créer une musique algorithmique consacrée à la danse ou du moins de créer des événements festifs ouverts à tous autour de cette expression musicale particulière[8].

De plus, cette architecture permet à n'importe quelle application de communiquer directement avec le serveur et de le contrôler en s'abstenant d'employer le client de SuperCollider.

4 https://www.birmingham.ac.uk/facilities/ea-studios/research/supercollider/mailinglist.aspx, consulté en avril 2019.

5 http://doc.sccode.org/, consulté en avril 2019.

6 http://sccode.org/, consulté en avril 2019.

7 https://github.com/supercollider-quarks, consulté en avril 2019.

8 https://algorave.com, consulté en avril 2019.

### 3.2. Paradigme du raccordement

Toutes les instances des modules en cours d'exécution sont classées dans une arborescence de nœuds définissant un ordre d'exécution. Il existe deux types de nœuds : *Synth* et *Group*. *Synth* est un ensemble d'unités de génération et il peut produire ou traiter des signaux audio ou de contrôle. Pour plus de commodité, j'appelle ces structures instrument ou module. *Group* est un ensemble de nœuds représentés sous forme de liste chaînée et les nœuds d'un groupe peuvent être à la fois des *Synths* ou d'autres *Groups*.

Un module génère du son ou du contrôle et il transmet son signal au moyen de bus, objet qui est aussi typé entre l'audio et le contrôle. Une pensée modulaire traditionnelle est alors possible pour organiser ses différents modules et il est possible de redéfinir son organisation à la volée sans générer des clics sonores imprévus. Par conséquent, les instruments produits avec SuperCollider peuvent être dénommés sous le terme pratique de *patch* sans abus de langage.

James McCartney regroupe ses modules suivant leur nature (contrôle, synthétiseur, effet et mixeur). Pour ma part, je les regroupe par pistes. Chaque piste dispose de son bus audio et tous les bus convergent vers la piste de contrôle global (*master*, cf. figure 1). J'ai choisi cette organisation par habitude, mais aussi parce qu'elle facilite la résolution du problème de raccordement. Les informations d'état des modules retournées par SuperCollider sont affichées par groupes et je trouve la lecture de ces informations plus claire lorsqu'un groupe correspond à une piste.

### 3.3. Séquence

La tradition compositionnelle occidentale implique de séquencer des événements sonores dans le temps sur des portées à l'aide d'une écriture symbolique. Selon Hugues Dufourt, le travail d'écriture consiste à refouler l'impulsion première et à le fonder sur la cohérence des actes, sur des structures logiques, sur des conduites opératoires [5].

Pour faciliter ce travail d'écriture, SuperCollider dispose de ressources pour séquencer des événements sonores grâce au concept de flux. Un flux est un objet qui retourne l'élément suivant d'une séquence lorsqu'on le lui demande. Cette abstraction a été encapsulée afin d'obtenir une abstraction de plus haut niveau pour gérer l'infrastructure séquentielle de différents flux. Un motif (*pattern*) définit le comportement de différents flux et cette abstraction nous ouvre des possibilités riches de variations suivant différents modèles.

J'ai utilisé ce type d'abstraction uniquement pour gérer et séquencer des listes de données issues de ma réappropriation des *Duos pour un pianiste* de Jean-Claude Risset [3] et j'ai utilisé deux modules d'augmentation de ce type dans ma pièce *Phantom Limb*.

## 4. UNE ÉCRITURE DU TEMPS RÉEL

Le contrôle de *Phantom Limb* est réalisable sur trois niveaux. Le premier niveau est établi manuellement par une séquence d'événements. Elle me permet de composer et d'articuler la forme globale de ma pièce. Pour chaque étape, j'attribue si nécessaire différents timbres au claviériste, aux différentes couches de l'accompagnement et aux articulations (cf. 4.2).

Le deuxième niveau concerne les modules d'augmentation. Ces derniers façonnent l'allure de mes différentes étapes par le déclenchement automatique de processus en réponse aux données délivrées par le claviériste [3].

Mon dernier niveau de contrôle porte sur les cinq paramètres des sons joués par le claviériste, mais aussi de l'accompagnement. Suivant l'usage, ces variations affectent la forme de l'étape ou la microforme sonore (cf. 4.3).

Cette structuration sur trois niveaux de contrôle et le système d'automatisation me permettent d'accroître par prolifération l'usage des données saisies et de hiérarchiser l'implication de mes fonctions de transfert. Je peux augmenter les gestes du claviériste, mais aussi lui répondre et conduire ainsi un discours. En effet, je ne cantonne pas la musique électronique à un rôle d'ornementation, mais je l'utilise aussi comme un élément distinct du discours musical.

La syntaxe de SuperCollider est riche et il existe différents moyens d'obtenir le même résultat. Pour les prochains exemples, j'emploierai en général celle qui contient le plus de sucre syntaxique afin de fournir des exemples les plus clairs possibles. De même, j'utiliserai des exemples simples pour exposer ma méthode d'écriture afin de focaliser l'attention moins sur le fond que sur la forme.

### 4.1. Définition et usage d'un instrument

Pour commencer, je définis un instrument construit à partir d'ondes en forme de dent de scie. Le signal de sortie sera composé d'une onde principale et de trois ondes de battement. Je n'ajoute pas de traitement à cet instrument, car par la suite je pourrais en ajouter dynamiquement. Cependant, je ne traiterai pas ces capacités offertes par SuperCollider, car elles complexifieraient les exemples.

```
(
SynthDef(\saw, {
   arg out=0, freq=330, amp=0.25, ampBeat=1,
   pan=0, attack=0.1, release=0.1, gate=1;

   var sig, freqBeat;
   var env;

   env = EnvGen.kr(Env.asr(attack, amp, release),
gate, doneAction:2);

   // Un signal en dent de scie accompagné de
trois signaux de battement
   sig = Saw.ar(freq, mul:0.5);
   freqBeat = Saw.ar(freq * [0.99, 1.001, 1.008],
mul:0.5 * ampBeat);
   sig = Pan2.ar(sig + freqBeat.sum, pan);

   // Diffusion sur le bus donné en argument
   Out.ar(out, sig * env);
}).add;
)
```

Un usage simple d'un instrument est d'affecter une instance de celui-ci à une variable afin de la contrôler en temps réel et de la libérer le moment venu sans générer de clic intempestif.

```
// Instanciation de l'instrument avec les
paramètres par défaut
i = Synth(\saw)

// Mise à jour de l'instance
i.set(\freq, 220)

// Libération de l'instance
i.release(0.5)
```

Néanmoins, si je désire gérer plusieurs instances avec des variables, d'une part le nombre de variables peut devenir difficile à organiser et d'autre part la versatilité dynamique d'un programme ainsi écrit sera limitée. Pour stocker et modifier les paramètres de mes instances, je les affecte alors à un objet collection qui me permettra d'associer un identifiant à chaque instance.

```
// Initialisation d'une collection
c = ()

// Instanciation des instruments
c.put(\saw1, Synth(\saw))
c.put(\saw2, Synth(\saw, [\freq,220]))

// Mise à jour de la 1ere instance
c.at(\saw1).set(\freq, 110)

// Libération des instances
c.removeAt(\saw1).release(0.5)
c.removeAt(\saw2).release(0.5)
```

### 4.2. Séquence des événements musicaux

Maintenant que j'ai déterminé comment gérer mes différentes instances dynamiquement avec un objet collection, je vais aborder l'écriture de ma séquence d'événements. Pour cela, chaque événement est rédigé dans une fonction et chaque fonction est affectée à une cellule d'un tableau.

```
(
var c = ();

a = [
{ // étape 0
   c.put(\saw1, Synth(\saw));
},

{ // étape 1
   c.put(\saw2, Synth(\saw, [\freq,220]));
},

{ // étape 2
   c.at(\saw2).set(\amp, 0.15);
   c.at(\saw1).set(\freq, 110);
```

```
},

{ // étape 3
    arg intruNum = 5;
    var fadeTime = 5;

    c.size.do {c.keysDo { |key|
c.removeAt(key).release(fadeTime) }};

    intruNum.do{|i| c.put(("saw"++i).asSymbol,
        Synth(\saw, [\attack,fadeTime,
\freq,exprand(80, 4000),
\amp,rrand(0.05,0.3)])) };
},

{ // étape 4
    c.size.do{ c.keysDo{ |key|
c.removeAt(key).release(0.1) }};
}
];
)
```

La fonction de l'étape 3 de mon précédent exemple dispose d'un argument et il permet de choisir le nombre d'instances de mon instrument à charger au moment de son exécution. J'ai utilisé un argument pour montrer un aspect dynamique supplémentaire de SuperCollider et en effet, toutes les ressources du langage sont convocables pour instancier, désallouer ou contrôler mes différents modules. Cependant, je n'utilise pas mes fonctions d'étape avec des arguments. Lorsque je désire exécuter plusieurs fois le même traitement, j'écris une fonction à part et je l'appelle dans mes fonctions d'étape en déterminant à ce moment la valeur de l'argument. Je peux aussi exécuter une séquence ou déclencher un traitement particulier à la réception d'un message MIDI ou OSC. Je dédie mes fonctions d'étape uniquement à la composition de ma musique électronique en termes de séquence, de paramètres sonores et d'automatisation du contrôle.

```
// Exécution des étapes
a[0].value
a[1].value
a[2].value

a[3].value // Cette étape peut être exécutée
plusieurs fois
a[3].value(rrand(3,7)) // Une variation

a[4].value
```

Pour contrôler mon programme, j'ai réalisé une interface graphique relativement dépouillée afin de faciliter l'appropriation de ma pièce par un musicien (cf. figure 1). Cette interface permet d'exécuter mes fonctions d'étape séquentiellement et de contrôler le gain des différentes pistes. Cette interface graphique minimale libère le musicien des contingences techniques et il peut se concentrer alors sur sa performance.

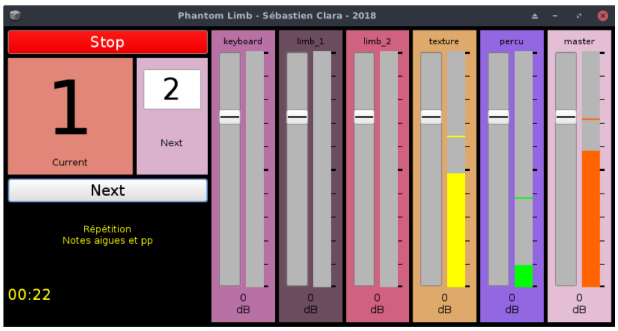


**Figure 1**. Interface graphique de *Phantom Limb*.

### 4.3. Automatisation

*Phantom Limb* est une improvisation dirigée pour un claviériste et les sonorités qui la matérialisent sont exclusivement d'origine synthétique. Le dispositif de cette pièce est constitué principalement d'un clavier MIDI standard, d'un système d'automatisation et de trois niveaux de contrôle. Les réactions automatiques du dispositif dépendent de ce que joue le claviériste et de la manière dont il joue.

Le premier niveau d'automatisation est composé de deux modules d'augmentation réalisés à partir de ma réappropriation des *Duos pour un pianiste* de Jean-Claude Risset [3], mais je ne reviendrai pas sur leur facture dans cet article. Toutefois, par rapport à cette première réalisation, j'ai implémenté une nouvelle opération basée sur la prédiction par reconnaissance partielle utilisant un modèle de Markov à ordre variable. Cette fonction n'est pas présente par défaut dans SuperCollider, mais elle est issue d'une librairie développée par Nick Collins et dont l'objectif est de rechercher des informations musicales[9].

Cette nouvelle option utilise les informations délivrées par le claviériste en temps réel pour générer des réponses et elle rend le dispositif instrumental de *Phantom Limb* interactif. Cependant, ce traitement interactif impose au claviériste une attention particulière afin qu'il puisse réagir avec à-propos aux propositions inattendues du dispositif. Dès lors, la rédaction d'une partition traditionnelle a été exclue. Néanmoins, je fournis un schéma pour indiquer au claviériste la forme globale de la pièce.

Indépendamment de ces modules d'augmentation, j'ai ajouté un accompagnement constitué d'une percussion et d'une texture sonore. Pour varier les paramètres sonores de cet accompagnement, je n'utilise pas des courbes d'automation ou des oscillateurs basses fréquences. De même, je n'utilise pas le son généré par le dispositif instrumental comme source de contrôle pour moduler les paramètres des synthétiseurs ou des traitements. Mais pour construire ce deuxième niveau de variation automatique, j'ai préféré utiliser les informations délivrées par le claviériste durant sa performance. Dans SuperCollider des fonctions

9 http://composerprogrammer.com/code/SCMIR.zip, consulté en avril 2019.

d'interruption MIDI ou OSC sont définissables par l'utilisateur. Pour ma pièce et les exemples suivants, j'ai utilisé les messages MIDI de type *noteOn*, *noteOff* et *bend*.

De manière analogue à mes précédents exemples, je sauvegarde dans une collection puis dans une liste les instances de l'instrument déclenchées par le claviériste lors de la réception d'un message *noteOn*. L'information contenue dans le message MIDI me renseigne sur la hauteur et la vélocité de la note jouée. J'utilise ces données pour paramétrer l'instance de mon instrument, mais je corrèle aussi la vélocité à l'amplitude des fréquences de battement afin de varier automatiquement le timbre de l'instrument.

```
node = (
   theSynth: Synth(\saw, [
        \freq, pitch.midicps,
        \amp, velocity/127,
        \ampBeat,velocity.linlin(10,120, 0.05,1)
   ]),
   note: pitch,
   amp: velocity/127
);

pitchKeyboard.addFirst(node);
```

Pour retrouver l'instance dans la liste, j'utilise la hauteur de la note comme discriminant lors de la réception d'un message *noteOff*. De plus, je module la durée de relâchement de l'instance suivant une estimation de la vitesse d'exécution du claviériste. Cette estimation est calculée à partir des durées entre l'attaque des trois dernières notes. Si le résultat de ce calcul entre dans un intervalle défini, l'estimation est mise à jour. Sinon les données sont écrasées par la dernière. Cette condition me permet de me prémunir quelque peu d'une estimation erronée lorsque je joue un accord.

```
node = pitchKeyboard.select({|n| n.note==pitch});

node.do({|n|
   n.theSynth.release(
tempo.linlin(minTempo,maxTempo, 1.5,0.03) );
   pitchKeyboard.remove(n);
});
```

J'utilise la liste où je sauvegarde les instances déclenchées par le claviériste pour obtenir d'autres informations. Dans l'exemple suivant, je calcule la moyenne[10] des amplitudes des notes jouées pour déterminer la valeur maximale de l'intervalle en demi-ton du message MIDI *bend*.

```
bendAmp = pitchKeyboard.collect({|n|
n.amp}).mean.linlin(0.1,1, 0.9,3);

pitchKeyboard.do({|n|
   n.theSynth.set(\freq, (n.note +
args[0].linlin(0,16383,
bendAmp.neg,bendAmp)).midicps)
});
```

10 Pour effectuer des calculs plus complexes, des méthodes supplémentaires de statistique sont disponibles dans la librairie MathLib. Nous pouvons installer celle-ci en utilisant le gestionnaire de paquet de SuperCollider, http://doc.sccode.org/Guides/UsingQuarks.html, consulté en avril 2019.

Cette liste me permet aussi de contrôler le nombre d'instances en cours d'exécution et si ce nombre dépasse un certain seuil, je libère l'instance la plus ancienne avec un temps de relâchement rapide afin de ne pas produire un clic sonore intempestif.

```
if(pitchKeyboard.size > maxNumVoices, {
   pitchKeyboard.pop.theSynth.release(0.01);
});
```

L'exemple suivant est la réalisation complète des précédents commentaires. Il est composé d'une fonction destinée à estimer le tempo tenu par le claviériste et de trois fonctions d'interruption MIDI. Je peux écrire ces dernières fonctions dans les cellules de mon tableau d'étape afin de modifier leur définition, si cela est musicalement intéressant.

```
(
var maxNumVoices = 8;
var tempo = 100, minTempo = 15, maxTempo = 350;
var lastDate, computeTempo;
var pitchKeyboard = List();

MIDIIn.connectAll;
MIDIdef(\noteOn).free; MIDIdef(\noteOff).free;
MIDIdef(\bend).free;

computeTempo = {
   var tmpTempo, maxNote = 3;

   lastDate =
lastDate.add(Date.getDate.rawSeconds);

   if(lastDate.size > maxNote,
{lastDate.removeAt(0)});

   if(lastDate.size > 1, {
        tmpTempo = (lastDate.size-1) * 60 /
(lastDate.last - lastDate.first);
        if((minTempo > tmpTempo).or(tmpTempo >
maxTempo), {
                lastDate = [lastDate.last]
        }, {
                tempo = tmpTempo
        });
   });
};

MIDIdef.noteOn(\noteOn, {arg velocity, pitch,
chan, src;
   var node;

   computeTempo.value;

   node = (
        theSynth: Synth(\saw, [
                \freq, pitch.midicps,
                \amp, velocity/127,
                \ampBeat,velocity.linlin(10,120,
0.05,1) ]),
        note: pitch,
        amp: velocity/127
   );

   pitchKeyboard.addFirst(node);

   if(pitchKeyboard.size > maxNumVoices, {
        pitchKeyboard.pop.theSynth.release(0.01);
   });
});
```

```
MIDIdef.noteOff(\noteOff, {arg velocity, pitch,
chan, src;
    var node;

    node = pitchKeyboard.select{|n|
    n.note==pitch};

    node.do({|n|
        n.theSynth
.release(tempo.linlin(minTempo,maxTempo,
1.5,0.03));
        pitchKeyboard.remove(n);
    });
});

MIDIdef.bend(\bend, {arg ...args;
    var bendAmp = pitchKeyboard.collect({|n|
n.amp}).mean.linlin(0.1,1, 0.9,3);

    pitchKeyboard.do({|n|
        n.theSynth.set(\freq, (n.note +
args[0].linlin(0,16383,
bendAmp.neg,bendAmp)).midicps)
    });
});
)
```

## 5. CONCLUSION

La composition de *Phantom Limb* est le fruit de mon expérience[11], mais aussi de mon travail de recherche doctorale. L'objectif de mon projet de thèse est de réaliser une panoplie de captation du geste et de dispositifs de production de sons électroniques pour la musique vivante.

La composition de *Phantom Limb* est une mise en œuvre des développements réalisés durant ma thèse et elle valide leur pérennité puisque j'ai utilisé d'autres outils pour la concevoir. Cependant, je ne peux pas me passer de l'usage de SuperCollider pour mon travail compositionnel. Au fil du temps, ce langage de programmation est devenu mon outil exclusif en informatique musicale. Je l'utilise principalement pour la création sonore temps réel et temps différé, l'extraction et l'analyse de données, mais aussi pour la composition algorithmique.

Néanmoins, les solutions dédiées à l'informatique musicale et à la création sonore sont légion[12] et je pourrais utiliser d'autres outils pour effectuer les mêmes tâches. De plus, certains logiciels sont conçus spécialement pour la composition musicale et l'exécution temps réel[13]. Dans ce dédale de solutions, comment ai-je choisi SuperCollider ?

La réponse à cette question relève d'une simple volonté de frugalité technologique et cela à des fins de maintenance, de préservation et de transmission de mes pièces. De plus, SuperCollider possède de nombreuses fonctionnalités par défaut et de nombreuses bibliothèques sont partagées gratuitement par la communauté qui ouvrent le champ des possibles de ce langage.

Avec un unique environnement de développement, je peux aborder la plupart des domaines de l'informatique musicale et jusqu'à présent, je n'ai pas souffert de limitation. L'usage d'un outil me permet d'envisager mes pièces comme des applications autonomes. Lorsque ma pièce est composée et que la robustesse de son programme est éprouvée, sa mise en place technique lors de moments sensibles comme le concert en est facilitée. De même, un musicien peut se réapproprier mon travail pour répéter et l'interpréter sans assistance technique majeure.

## 6. RÉFÉRENCES


[1] Blechmann, T. "Supernova – a scalable parallel audio synthesis server for SuperCollider", *Proceedings of ICMC*, Huddersfield, 2011.

[2] Clara, S. "Heterogeneous data orchestration. Interactive fantasy under SuperCollider", *Proceedings of LAC*, Saint-Étienne, 2017.

[3] Clara, S. "Étude et appropriation de l'augmentation instrumentale proposée par Jean-Claude Risset dans l'élaboration de ses duos pour un pianiste", *Proceedings of JIM*, Amiens, 2018.

[4] Collins, N. McLean, A. Rohrhuber, J. Ward, A. "Live coding in laptop performance", *Proceedings of Organised Sound*, Cambridge, 2003.

[5] Dufourt, H. "De la notation à l'ordinateur", *Révolutions industrielles de la musique*, Cahier de médiologie / IRCAM n° 18, Fayard, Paris, 2004.

[6] Levy, S. *L'éthique des hackers*. Globe, Paris, 2013.

[7] McCartney, J. "Rethinking the Computer Music Language: SuperCollider", *Computer Music Journal*, Vol. 26, N° 4, 2002.

[8] Puckette, M. "Max at seventeen", *Computer Music Journal*, Vol. 26, N° 4, 2002.

[9] Roads, C. Mathews, M. "Interview with Max Mathews", *Computer Music Journal*, Vol. 4, N° 4, 1980.

[10] Wilson, S. Cottle, D. Collins, N. *The SuperCollider Book*, The MIT Press, Cambridge, 2011.


11 *Danaus plexippus*, pièce mixte pour quatre percussionnistes à la suite de mon master 2, validé à Paris 8 en 2014.

*Les noces d'Iphigénie*, pièce acousmatique multicanal composée en 2015. J'ai organisé mes processus en utilisant la même méthode présentée dans cet article.

*Peyote*, pièce mixte pour flûte, saxophone et piano composée en 2016. J'ai rédigé un article pour montrer mon usage des données issues de l'analyse audio pour contrôler certains processus [2].

12 Sonic Visualiser, Praat, Virtual ANS, Cecilia, Pure Data, Faust, etc.

13 Ossia Score, Antescofo, Integra Live, INScore, etc.