Numéros / n° 6 - Techniques et méthodes innovantes pour l'enseignement de la musique et du traitement de signal

« Leçons tirées de l'enseignement de l'informatique musicale à des musicologues »

Albert Gräf

Résumé

Nous présentons des expériences d'enseignement de la musique-informatique à des étudiants en musicologie et de diverses disciplines (informatique, mathématiques, sciences médiatiques) à l'université pluridisciplinaire JGU, dans le groupe de recherche Music-Informatics (Institut d'histoire de l'art et de musicologie). Nos cours couvrent un large éventail de sujets, allant des théories sous-jacentes (acoustique, traitement du signal numérique, théorie mathématique de la musique et certains domaines de l'informatique) à des sujets pratiques en technologie musicale et en programmation informatique. Réfléchir à qui et pourquoi nous enseignons est aussi important que de demander quoi enseigner et comment. Dans ce texte, nous réfléchissons à certaines de ces questions et à leurs interdépendances, et nous discutons également des outils logiciels et des langages de programmation utilisés.

Introduction

Ce texte témoigne des expériences d'enseignement de la musique et de l'informatique au niveau du premier cycle, principalement aux étudiants en musicologie, mais aussi d'autres disciplines telles que l'informatique, les mathématiques, les médias et la musique. À la JGU, le groupe de recherche Music-Informatics (informatique musicale) fait partie de l'Institut d'histoire de l'art et de musicologie. Le groupe de recherche a été fondé en 1991 et se trouve donc être parmi les premières institutions académiques allemandes travaillant dans ce domaine à l'intersection de la musique, des mathématiques, de l'informatique et de la technologie des médias. À l'origine, l'activité principale du groupe portait sur la recherche, mais des cours de musicologie systématique et d'informatique musicale ont été donnés depuis 1994. La JGU est une assez grande université (parmi les dix plus grandes d'Allemagne), combinant presque toutes les disciplines académiques sous un même toit. Cela offre de nombreuses possibilités de recherche et d'enseignement interdisciplinaires.

L'enseignement de la musique informatique et des sujets connexes dans un tel cadre interdisciplinaire pose des défis considérables. Les étudiants qui fréquentent les cours sont issus de milieux très différents. En outre, étant donné que la plupart d'entre eux aspirent à devenir des scientifiques plutôt que de futurs compositeurs, musiciens ou artistes multimédias, ils s'intéressent généralement davantage aux connaissances fondamentales (plutôt qu'aux méthodes spécifiques et aux compétences pratiques), ce qui doit leur permettre de comprendre non seulement l'informatique et la technologie des médias numériques, mais aussi leurs implications dans l'art contemporain, la musique et les sciences humaines. Ainsi, nos cours couvrent un large éventail de sujets, allant de la théorie sous-jacente (entre autres l'acoustique, le traitement du signal numérique, la théorie mathématique de la musique et certains aspects des sciences informatiques) à des sujets pratiques autour des technologies musicales et de la programmation informatique. La question de savoir à *qui* nous enseignons et *pourquoi* est aussi importante que celle de savoir *ce que* nous enseignons et *comment*. Dans ce qui suit, nous réfléchissons à certaines de ces questions et à leur interdépendance, et discutons également des outils logiciels et des langages de programmation utilisés.

1. L'informatique musicale au JGU

Nos étudiants viennent de différents départements de l'université, principalement de la musicologie et de la musique, mais aussi de l'informatique, des mathématiques et des sciences des médias. Il existe plusieurs cursus d'études disponibles, principalement une licence en Arts (BA), une autre pour l'enseignement (BEd) ainsi que des études de master correspondantes, et les étudiants de troisième cycle ont également la possibilité de faire un doctorat. Jusqu'en automne 2015, nous offrions également la possibilité de suivre des études traditionnelles pour l'obtention du Magister et du Diploma. À l'automne 2015, on comptait 282 étudiants (221 d'entre eux faisant un BA/BEd, 134 ayant leur sujet principal en musicologie). La plupart de ces étudiants sont donc amenés à suivre au moins un ou deux cours d'informatique musicale à un moment donné lors de leurs études (certains d'entre eux en suivent beaucoup plus, et nous avons fréquemment des étudiants invités, de sorte que l'on peut dire que l'offre de d'informatique musicale a beaucoup de succès). Depuis l'automne 2016, nous réalisons également des enseignements dans un nouveau cours d'études d'« Humanités Numériques » en collaboration avec l'Université des Sciences Appliquées de Mayence, afin que les étudiants de cette institution puissent également suivre des cours au niveau Master.

La plupart des cours prennent la forme de séminaires ou de travaux dirigés, avec un nombre de participants par cours allant de 5 à 40 étudiants. Typiquement, environ 20 à 30 étudiants suivent ces cours par semestre en moyenne. Nous proposons également des conférences sur une base plus ou moins régulière (habituellement au moins une tous les 4 semestres), pour donner aux étudiants intéressés la possibilité d'apprendre les fondements théoriques plus approfondis du domaine, mais en raison de la façon dont nos cours sont structurés, tous les étudiants n'y participent pas. Enfin, la plupart des concepts et notions de base sont également abordés dans les séminaires et les travaux dirigés en fonction des besoins.

Pour accueillir ces cours, et en particulier dans les formats de travaux dirigés plus pratiques, nous avons équipé un laboratoire de musique informatique (jusqu'à 10 places à ce jour) avec du matériel MIDI/OSC et des PC de bureau, fonctionnant principalement avec Linux et divers logiciels open-source. Le grand avantage est que les étudiants peuvent apprendre une grande variété de logiciels de pointe d'informatique musicale sans frais supplémentaires, ce qui leur permet d'obtenir facilement les logiciels sur leurs propres ordinateurs et donc de continuer leurs travaux en dehors de la salle de classe. Pour faciliter cela, nous préférons utiliser des logiciels open-source qui sont également multiplateformes, c'est-à-dire qui s'exécutent également sur MacOS et Windows.

1.1. Quelques questions

Il y a quelques questions clés qui doivent être abordées lors de la conception d'une offre de cours sur un sujet interdisciplinaire comme :

- **Qui ?** Qui devrait apprendre l'informatique musicale ? Qui va (ou devrait) s'y intéresser ?
- **Pourquoi ?** Pourquoi enseigner l'informatique musicale à tous ? Quels sont les bénéfices ?
- **Quoi ?** Quels sont les sujets importants ? Quels sont les prérequis (mathématiques, informatique, théorie de la musique, etc.) ?

Comment ? Comment mieux transmettre ces sujets ? Comment combler l'écart entre la théorie et la pratique ? Comment armer les étudiants avec les prérequis nécessaires ?

Je vais essayer de répondre ci-dessous au moins en partie à ces questions à partir de ma propre expérience. La plupart de mes réponses concernent bien sûr notre situation particulière, mais j'espère qu'au moins certaines de ces remarques pourront être utiles à d'autres personnes qui enseignent ces matières. Notez également que, n'étant pas un éducateur, je ne suis pas vraiment un expert sur ces questions, donc il ne s'agit que de mes propres observations (anecdotiques et plutôt non scientifiques), observations qui découlent de mes expériences d'enseignement sur le sujet depuis plus d'une décennie.

1.2. Qui ?

La plupart des instituts qui enseignent l'informatique musicale ont probablement comme public principal des compositeurs, des musiciens et des artistes multimédias. Notre situation à la JGU est un peu différente que dans la mesure où la majeure partie de nos étudiants est effectivement issue du département de Musicologie auquel appartient le groupe de recherche Music-Informatics. Les techniques modernes de la musique par ordinateur ne sont pas encore arrivées dans la recherche musicologique traditionnelle, le domaine est tout simplement trop jeune pour cela. Cependant, les bases théoriques du domaine ont certainement de nombreux liens avec la Musicologie Systématique. De plus, de nombreux musicologues travaillent dans les médias (tels que les maisons d'édition, la radio et la télévision) après avoir terminé leurs études, des domaines où une compréhension approfondie de la technologie des médias numériques modernes, y compris les techniques de production musicale, la notation musicale assistée par ordinateur ainsi que le Web et les technologies autour des bases de données est très importante actuellement.

Je dirais également que les informaticiens et les mathématiciens qui s'intéressent à la musique peuvent bénéficier des recherches menées dans ce domaine, puisqu'ils peuvent découvrir des applications importantes et largement répandues de l'ère numérique qui ne se trouvent généralement pas encore dans les programmes de leurs centres d'intérêt principaux. En particulier, les étudiants en informatique peuvent en apprendre beaucoup sur les logiciels et les interfaces de programmation liés à la musique et aux applications audio/vidéo, de même que les étudiants en mathématiques peuvent aborder des concepts mathématiques avancés appliqués au traitement du signal numérique et à l'analyse musicale. Le plus intéressant étant que ces applications peuvent peuvent se révéler ludiques dans la cadre de compositions ou dans la production de résultats réels que les étudiants peuvent regarder et écouter, montrer à leurs pairs, télécharger sur SoundCloud, YouTube et autres, etc.

1.3. Pourquoi ?

La question de la motivation est probablement la question la plus difficile parmi les quatre qui sont posées. Les étudiants ont leurs propres emplois du temps, qui dépendent des cours étudiés et de leurs projets pour l'avenir (post-universitaires), mais aussi de ce qu'ils trouvent intéressant, utile et (dernier point, mais pas le moindre) *amusant*. Heureusement, beaucoup d'étudiants dans nos cours *pratiquent* eux-mêmes de la musique, ou ils utilisent des technologies musicales d'une certaine façon, auquel cas il n'est pas trop difficile de trouver un sujet dans le domaine de l'informatique musicale qui suscite leur curiosité. Cependant, d'autres peuvent trouver les cours difficiles, car certains sujets impliquent nécessairement un peu de mathématiques et/ou de programmes informatiques qui ne sont clairement pas les sujets préférés de tous. Il faut donc trouver un équilibre entre des sujets plus techniques et des sujets tels que l'acoustique musicale, la psycho-acoustique ou l'histoire de l'informatique musicale.

Il est également intéressant de voir la motivation à faire de l'informatique musicale se transformer au fil du temps, alors que les médias numériques et l'environnement de calcul continuent d'évoluer. Lorsque j'ai commencé à enseigner cette matière au début du millénaire, la musique par ordinateur était encore un sujet assez ésotérique dont presque personne n'avait jamais entendu parler et qui n'était enseigné que dans les milieux spécialisés, du moins dans les universités allemandes. Avançons rapidement jusqu'en 2016 : alors que je dois encore expliquer à quelques étudiants ce qu'est l'informatique musicale, pratiquement tous les étudiants utilisent désormais des ordinateurs portables avec des dispositifs multimédias avancés et presque tous ont déjà entendu parler de logiciels de musique populaires comme FruityLoops ou GarageBand, bien qu'ils ne les aient peut-être pas encore utilisés. La plupart des étudiants en musicologie connaissent déjà des logiciels de notation musicale comme Finale ou Sibelius, il est donc beaucoup plus facile de partir de là et de leur présenter l'éventail des logiciels de musique. Cela aide également que la plupart des étudiants possèdent leur ordinateur portable, et que beaucoup de bons logiciels de musique sophistiqués soient facilement disponibles sous licence open-source sans frais supplémentaires de nos jours.

1.4. Quoi ?

Presque dès le début de son histoire, l'informatique musicale a concerné une large gamme interdisciplinaire de sujets, impliquant des sujets « classiques » (tels que la théorie mathématique de la musique, le traitement du signal) et les sujets « récents » (par ex. les interfaces pour l'expression musicale, les appareils mobiles). Le tableau suivant résume les sujets que je maîtrise assez bien et que j'offre assez régulièrement dans mes cours, sans ordre particulier, avec les sujets importants (souvent préférés par les

étudiants ou moi-même) mis en évidence en caractères gras. Je ne prétends certainement pas que cette liste est exhaustive, mais j'en ai discuté avec d'autres enseignants et praticiens sur le terrain, et je pense qu'elle couvre plusieurs des aspects importants.

composition algorithmique	remix
synthèse sonore	**traitement numérique du son**
théorie mathématique de la musique	design sonore
accords et tempéraments	**codes musicaux**
échelles et modes	technologies audio et vidéo
acoustique musicale	technologies des plug-ins
psycho-acoustique	technologies des contrôleurs
notation musicale	**programmation informatique**
analyse musicale	bases de données et technologies web

1.5. Comment ?

C'est une autre question difficile, et qui nécessite un peu de pensée Faustienne, afin de combler les différentes lacunes auxquelles nous sommes confrontés lors de l'enseignement de ce sujet :

- de *l'enseignant* vers *l'étudiant* (didactique) ;
- de *la science* vers *l'art* (interdisciplinarité) ;
- de *la théorie* vers les *applications* (pratique) ;
- des *savoirs de bases* vers *les savoirs experts* (spécialisation) ;
- des *non-programmeurs* vers *les programmeurs* (langages et outils).

Pour y parvenir, j'emploie les techniques classiques habituelles que tout le monde doit connaître :

Méthode scientifique : comme un peu partout en sciences, nous devrions permettre aux étudiants de faire leurs propres recherches plutôt que de les laisser suivre aveuglément les conseils de n'importe qui (y compris des enseignants). Cette démarche est plus facile dans les conférences et dans les séminaires, mais certaines recherches sont impliquées dans tous types de formats de cours. Heureusement, avec Internet, nous disposons aujourd'hui d'une masse d'information presque illimitée, mais cela rend d'autant plus cruciale l'acquisition par les étudiants d'une capacité à filtrer cette profusion et à *vérifier leurs sources* en se livrant à un examen scientifique, une vérification des faits et en effectuant des essais réels avec leurs logiciels préférés.

Formation pratique : il va sans dire que, dans notre domaine, le côté pratique est au moins aussi important que la théorie. Nous voulons donc permettre à nos étudiants d'appliquer réellement des connaissances théoriques dans leurs recherches et leurs travaux artistiques. Ceci est possible avec les formats des travaux dirigés. Heureusement, une grande partie des outils nécessaires est aujourd'hui facilement accessible pour les étudiants, même s'ils ne sont pas toujours aussi faciles à utiliser que nous aimerions qu'ils le soient dans un cadre éducatif.

En ce qui concerne les applications des connaissances théoriques, la programmation semble être le plus grand obstacle pour la plupart des étudiants (les mathématiques en deuxième position, selon mon expérience). Il n'y a pas de solution miracle, c'est simplement la pratique, la pratique, la pratique. Nous

devrions donc rendre cela aussi facile que possible ! Ce qui signifie que nous devons examiner les outils logiciels disponibles et voir comment nous pouvons les améliorer et les rendre plus accessibles pour les étudiants. Nous aborderons les aspects techniques de cette question dans la section suivante.

Un autre problème particulier auquel nous sommes confrontés est le fait que, comme nous l'avons déjà mentionné, nous avons des étudiants issus de milieux très différents dans la plupart des cours. Bien évidemment, dans un tel cadre, il est souhaitable que les étudiants travaillent en groupes ou en équipes, afin qu'ils puissent travailler sur des présentations et de petits projets ensemble et ainsi faire bon usage de leurs forces et talents respectifs et apprendre les uns des autres. Un exemple idéal serait une équipe composée d'un étudiant en musicologie maîtrisant le côté musical d'un projet, un étudiant en mathématiques ayant une bonne compréhension des notions mathématiques et des solutions impliquées, et un étudiant en informatique qui puisse concevoir et coder le logiciel résultant. J'ai trouvé que cela fonctionnait très bien dans les cours en travaux dirigés et aussi dans certains types de séminaires. Malheureusement, cette approche est parfois entravée par des règles d'examen qui nécessitent des contributions individuelles à évaluer par l'enseignant.

2. Outils

Enfin, parlons des outils logiciels que nous utilisons réellement. Dès le début, j'ai pris la décision d'essayer d'utiliser autant de logiciels open-source que possible, même si cela impliquait souvent de sacrifier certaines fonctionnalités et parfois de vivre avec des logiciels incomplets pendant un certain temps. Je l'ai fait à la fois pour des raisons pratiques (les étudiants n'ont pas besoin d'acheter des logiciels propriétaires coûteux s'ils veulent continuer à utiliser les programmes en dehors de la salle de classe) et pour des raisons philosophiques et éthiques (aider les développeurs open-source à toucher le public et inculquer aux étudiants la notion de partage dans la communauté open-source plutôt que de les encourager à exécuter des logiciels propriétaires piratés).

Un autre avantage intéressant présenté par cette approche est le fait que nous tâchons également d'expérimenter les derniers logiciels de pointe et les nouvelles fonctionnalités expérimentales, souvent issues des recherches actuelles (y compris les miennes), de sorte qu'il y ait toujours quelque chose d'intéressant et de novateur à essayer chaque semestre. En outre, le logiciel open-source a tendance à connaître des cycles de publication beaucoup plus rapides, et il est toujours agréable de corriger les bugs et les fonctionnalités manquantes dès qu'ils sont identifiés, parfois même pendant le semestre, alors que le cours est toujours en cours d'exécution.

2.1. Système d'exploitation

Comme je l'ai déjà mentionné, nous utilisons Linux dans notre laboratoire, car il nous permet facilement d'accéder à une foule de logiciels open-source intéressants, et il facilite également la compilation du dernier logiciel si nécessaire. Nous avons commencé avec SUSE Linux [(1)], mais nous sommes passés à Ubuntu [(2)], qui est encore assez populaire chez les étudiants qui connaissent Linux.

Cependant, depuis à peu près deux ans maintenant, nous utilisons Manjaro [(3)], un dérivé d'Arch Linux [(4)], car il fonctionne sur un modèle de « publication continue » qui offre toujours les dernières versions des logiciels disponibles à tout moment, sans jamais avoir besoin d'être réinstallé ou d'être mise à jour. Manjaro tend à offrir une meilleure stabilité en tant que « pure » Arch, car les principaux logiciels composant le système d'exploitation de base et l'environnement de bureau sont testés à fond avant d'être mis à la disposition du public. L'inconvénient est que cela introduit typiquement un délai de 1 à 2 semaines par rapport à Arch, mais cela semble raisonnable et ne pose pas beaucoup de problèmes.

Au début, l'utilisation de Manjaro dans le laboratoire était vraiment expérimentale, car il n'était pas clair dès le départ qu'une distribution Linux basée sur Arch soit adaptée à un cadre éducatif où les ordinateurs doivent toujours être en bon état de fonctionnement. Mais cela a bien fonctionné pour nous, de sorte qu'aujourd'hui, nous mettons à jour régulièrement le système tout au long du semestre. En fait, il s'est avéré que cette approche fonctionnait mieux qu'une distribution non dynamique telle que Ubuntu, puisque les bugs du système d'exploitation sont corrigés beaucoup plus rapidement (normalement dès que les développeurs en amont les corrigent, à quoi s'ajoute le délai de 1-2 semaines jusqu'à ce que les dépôts

stables Manjaro soient mis à jour, jusqu'aux derniers paquets d'Arch).

2.2. Software

Alors que nous utilisons Linux dans le laboratoire, la plupart des étudiants ne le font pas (beaucoup d'entre eux utilisent MacOS ou Windows). Par conséquent, nous essayons d'utiliser autant que possible des logiciels multiplateformes, afin que les étudiants puissent installer ces mêmes logiciels sur leurs ordinateurs. Voici une brève liste des logiciels les plus importants actuellement utilisés sur notre site :

- édition audio et vidéo : Audacity, Kdenlive ;
- station numérique audio (DAW) : Ardour, *Reaper*, *Tracktion* [(5)];
- logiciel de notation : Frescobaldi, Lilypond, MuseScore ;
- logiciel de synthèse : Qsynth/Fluidsynth ;
- programmation visuelle de flux de données : Pd (Pure Data) ;
- langages de programmation : Faust, Pure ;

La partie la plus intéressante ici est probablement le choix des environnements de programmation, à savoir Pd, Faust et Pure, auxquel je vais donc m'intéresser plus avant dans les sections suivantes.

2.3. Pd

Pd (alias Pure Data) est une sorte d'environnement de programmation visuelle (également connu sous le nom de langage de programmation de flux de données ? *dataflow* programming language) pour construire des algorithmes complexes de traitement du son et de contrôle, réalisés à partir de simples blocs de construction (Puckette, 2007). C'est la seule alternative open-source vraiment complète à Max, soutenue par une grande communauté. Le développeur principal de Pd est Miller Puckette qui a également écrit la version originale de Max que David Zicarelli a ensuite transformé en logiciel commercial qui est devenu Max/MSP (Puckette, 2002).

Pd permet aux étudiants de regarder « sous le capot » et de comprendre comment fonctionne le traitement du signal musical. En particulier, il leur permet de créer leurs propres effets, synthétiseurs et algorithmes de traitement de contrôle MIDI ou OSC à partir de zéro [(6)]. Il leur permet également de créer leurs propres petits outils de génération du son et de traitement du signal pour tout ce dont ils ont besoin dans leurs études, tels que des petits programmes de composition, des outils d'analyse du son et des expériences psycho-acoustiques. C'est pourquoi je le considère comme le couteau suisse de l'éducation musicale par ordinateur et que je le présente, plus ou moins partiellement, dans beaucoup, sinon la plupart de mes cours [(7)].

2.4. La programmation fonctionnelle

Pd est facile à utiliser, mais il lui manque la flexibilité d'un langage de programmation complet. Ses objets intégrés déterminent essentiellement ce que vous pouvez faire avec lui. Si vous devez aller au-delà, vous devez créer vos propres objets, appelés externes dans le langage Pd. Normalement, cela se fait dans le langage de programmation C, mais il existe des environnements de programmation qui facilitent cette tâche.

Notre principale exigence ici est que le langage soit facile à utiliser par des débutants, afin que ces derniers puissent apprendre à mettre en oeuvre au moins certains algorithmes basiques de traitement du signal. En outre, nous aimerions qu'il fonctionne avec Pd, mais parfois c'est également pratique si nous pouvons aussi prendre simplement les externes que nous avons développés et les exécuter en tant que

plug-in dans une station de travail audio. Voici quelques-unes des questions à prendre en compte lors du choix d'un langage de programmation à utiliser dans les cours d'informatique musicale :

- Le langage doit-il être impératif, orienté objet, fonctionnel ?
- Devons-nous utiliser un langage à usage général ou spécifique du domaine (DSL) ?
- Le langage doit-il être expérimental ou conventionnel ?

Il semble que, même aujourd'hui, la plupart des plug-ins de traitement du signal soient encore développés à l'aide de C et C++, qui sont des langages de type impératif (et orienté objet, pour C++), généraux et conventionnels. Cependant, d'après mon expérience, ces langages ne fonctionnent pas très bien dans un contexte éducatif, car ils sont trop compliqués pour les novices et nécessitent d'accorder beaucoup d'attention aux détails techniques, archaïques, du traitement du signal qui sont toujours des pièges majeurs pour les débutants et même pour les développeurs expérimentés.

J'utilise donc deux langages expérimentaux à la place, à savoir Faust de GRAME et mon propre langage de programmation Pure. Ce sont tous les deux des langages fonctionnels et ils fournissent ainsi de meilleures abstractions, à un plus haut niveau, telles que des fonctions et des flux d'ordre supérieur (c.-à-d. des listes potentiellement infinies), ce qui rend la programmation beaucoup plus agréable pour les programmeurs compétents et aussi plus facile pour les débutants. De plus, les deux langages sont bien intégrés au programme Pd et aux DAW, offrant notamment des interfaces Pd, LV2 et VST complètes pour des plug-ins (8).

2.5. Faust

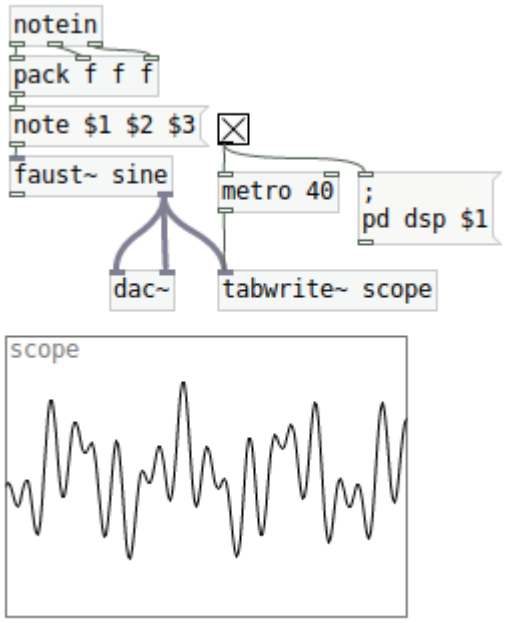
Faust du GRAME est un langage de programmation fonctionnel de domaine spécifique compilé pour créer des plug-ins de traitement du signal (Orlarey, Fober & Letz, 2004). Le compilateur Faust produit un code natif efficace pour une grande quantité d'environnements différents, parmi eux Pd, Pure, ainsi que les hôtes de plug-in LV2 et VST. Il existe également une bibliothèque externe nommée *Pd-Faust* écrite en Pure qui permet de charger *dynamiquement* les processeurs de signaux Faust dans Pd (ce qui signifie qu'ils peuvent être rechargés chaque fois que le programme change).

Nous n'irons pas très loin dans Faust ici, mais examinons brièvement un exemple simple, un synthétiseur monophonique (très basique) qui génère une simple onde sinusoïdale :

```
import("music.lib");
declare nvoices "8";
gate = button("gate");
gain = hslider("gain", 0.3, 0, 3, 0.01);
freq = hslider("freq", 440, 20, 2000, 1);
vol  = hslider("vol", 0.3, 0, 10, 0.01);
process = gate*gain*vol*osc(freq);
```

L'essentiel du programme est constitué des variables de contrôle requises. Les variables freq et gain définissent la hauteur et le volume de la note, tandis que la variable gate détermine quand la note est « activée » et « désactivée ». Globalement, on les appelle également les « commandes de voies », car ils déterminent les paramètres de chaque note individuelle et peuvent facilement être assignés aux valeurs de hauteur et de vitesse des données de note MIDI entrantes. Par ailleurs, la variable vol fournit une sorte de contrôle de volume général, qui peut être déclenché par une commande de l'utilisateur ou par des messages envoyés par un contrôleur MIDI. La fonction process, qui est la fonction principale d'un programme Faust, combine ensuite ces valeurs au signal sinusoïdal approprié, en utilisant la fonction osc de la bibliothèque Faust.

Figure . Un objet Faust dans un patch Pd



La déclaration nvoices dans ce programme indique que ce processeur de signal doit être utilisé comme un synthétiseur avec une polyphonie à 8 voix. Ce n'est pas le code de programme réel, mais des métadonnées consultatives qui peuvent être utilisées par l'environnement cible pour reconnaître le programme en tant que synthétiseur. Pd-Faust le fait automatiquement et transformera ce programme en un synthétiseur polyphonique avec allocation des voies et un traitement MIDI des notes (l'objet faust~ sur la figure 1). Bien sûr, ce n'est qu'un exemple très basique. Mais avec seulement quelques lignes de code supplémentaires, on peut ajouter quelques partiels et une courbe d'enveloppe appropriée pour le transformer en un instrument utilisable qui ressemble à un orgue électronique.

2.6. Pure

Alors que Faust est conçu pour le traitement des données numériques synchrones, ce qui est idéal pour programmer des instruments et des effets audio, *Pure* est un langage polyvalent qui est pratique lorsqu'une quantité importante de traitements symboliques portant sur des données de contrôle complexes est nécessaire (Gräf, 2016). Pure est implémenté comme un environnement interpréteur interactif, mais il est réellement compilé vers du *code natif efficace* à la volée, en utilisant la compilation *JIT* (juste à temps) via le framework de compilation LLVM [(9)].

Pure est également disponible en chargeur de plug-in dans Pd, appelé Pd-Pure, ce qui permet de programmer des objets Pd directement dans Pure. Ce chargeur supporte le *rechargement dynamique* des programmes Pure pour faciliter le *debugging* et le *live-coding*. Alternativement, les objets externes de Pure peuvent également être compilés dans des bibliothèques de codes natifs, de sorte qu'ils soient indiscernables d'objets externes Pd natifs écrits en C.

Nous ne pouvons donner qu'un aperçu très sommaire des caractéristiques les plus importantes de Pure. Pour plus de détails, nous renvoyons le lecteur intéressé à la documentation en ligne sur le site Web Pure.

Pure est un langage de programmation fonctionnel basé sur la *réécriture de termes*, c'est-à-dire *l'évaluation symbolique* des expressions. Ainsi, toutes les données en Pure prennent la forme *d'expressions* (également appelées termes), comme 12345 (un nombre entier), 3.1415 (un nombre à virgule flottante), "abc" (une chaîne de caractères), bang (une fonction ou un symbole constant). Les expressions composées sont formées en utilisant des applications de fonction, comme fib 21, note 60 127, 1:2:3:4:5:[] = [1,2,3,4,5] (la dernière désignant une liste formée avec la fonction qui utilise le symbole ':', qui dans ce cas prend la forme d'un opérateur *infixe*).

Les fonctions sont définies à l'aide *d'équations* et de *pattern matching*. Par exemple, voici une façon

possible de définir la *fonction Fibonacci* (qui assigne un entier n au nth numéro de Fibonacci) :

```
fib 0 = 0; fib 1 = 1;
fib n = fib(n-1) + fib(n-2) if n>1;
```

Les équations sont utilisées comme *règles de réécriture de termes* afin de réduire les expressions à leur forme la plus simple (*forme normale*). Cela fonctionne d'une manière tout à fait semblable à ce que font les systèmes d'algèbre informatique. Par exemple, l'évaluation de l'expression `fib 2` serait essentiellement exécutée comme suit (le symbole « => » indique les étapes de réécriture individuelles, et les sous-termes réécrits sont soulignés à chaque étape) :

```
fib 2 => fib 1 + fib 0 => 1 + fib 0 => 1 + 0 => 1
```

Il convient de mentionner ici que toutes les structures et fonctions de données en Pure sont *polymorphes*, c'est-à-dire qu'elles peuvent prendre comme arguments différents types de données comme on le souhaite. Ceci est également connu sous le nom de *typage dynamique*, par opposition au typage statique où les types d'arguments sont restreints. (Pensez, par exemple, à Python par rapport à Java, ou à Lisp par rapport à ML.) Cela rend les choses beaucoup plus faciles lors de la connexion dans un environnement dynamique comme Pd. En particulier, il permet aux messages Pd (nombres, symboles et listes) d'être mappés aux données Pure correspondantes et vice versa de manière très simple.

Il existe essentiellement deux approches différentes pour programmer des objets Pd en Pure : le modèle *d'acteur* et le modèle *de flux*. Le modèle *d'acteur* est essentiellement ce que Pd utilise lui-même comme modèle de calcul, donc c'est très proche de la façon dont les objets fonctionnent réellement dans Pd. Dans ce modèle,

- Un objet reçoit des données depuis ses *entrées* ;
- effectue certains calculs (éventuellement modifiant son *état* interne) ;
- et enfin envoie des données vers ses *sorties*.

Voici un exemple en Pure implémentant un objet « compteur » pour Pd, qui compte simplement 0, 1, 2... en envoyant le numéro suivant à son unique sortie chaque fois qu'il reçoit un bang (ou tout autre message) par son entrée :

```
counter = process with
  process _ = n when
    n = get counter;
    put counter (n+1);
  end;
end when
  counter = ref 0;
end;
```

Le modèle d'acteur n'est pas très élégant car il nécessite un état interne, c'est-à-dire des *effets de bord*. Dans l'exemple ci-dessus, l'état est implémenté en tant qu'une variable de référence `counter` de Pure, qui est essentiellement une cellule de mémoire stockant la valeur courante du compteur. La valeur du compteur est ensuite récupérée à l'aide de `get` et mise à jour avec `put` (qui sont toutes deux des fonctions prédéfinies dans la bibliothèque standard de Pure).

Mais il existe un meilleur moyen, mathématiquement plus agréable de décrire le fonctionnement d'un acteur de manière *purement fonctionnelle*, sans effets de bords : le *traitement des flux*. En Pure, tout comme dans d'autres langages de programmation fonctionnels tels que Haskell ou Alice ML, un flux est une liste potentiellement infinie dont les éléments ne sont produits qu'à la demande. Ceci utilise une technique de calcul connue sous le nom *d'évaluation paresseuse*. Cela signifie que la liste n'est jamais calculée dans son intégralité, ce qui entraînerait évidemment un calcul sans fin si le flux est infini. Au lieu de cela, les éléments ne sont produits que s'ils sont nécessaires au cours d'un calcul. Voici comment le flux de tous les nombres de Fibonacci peut être défini dans une seule équation imbriquée en Pure :

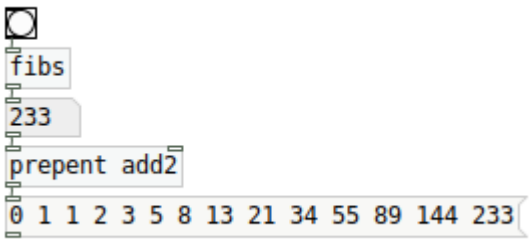
```
fibs = fibs 0 1 with fibs a b = a : fibs b (a+b) & end;
```

Notez la construction with ... end qui définit une fonction fibs, interne, « locale », qui est ensuite utilisée pour définir l'extérieur, « global » fonction. Toutes les actions réelles se produisent dans la fonction locale fibs qui prend l

Nous voulons maintenant transformer ce flux en un vrai objet Pd qui produit l'élément suivant du flux en recevant un message arbitraire (à peu près comme dans l'exemple counter ci-dessus). Pour cela, nous employons une petite fonction auxiliaire actor qui peut être écrite dans Pure elle-même, mais est omise ici pour des raisons de simplification ; le lecteur intéressé peut trouver une mise en oeuvre appropriée dans les actes de LAC 2009. (Gräf, 2009, p. 142). La fonction actor prend comme entrée une fonction de traitement de flux f, qui à son tour prend un flux de messages Pd comme entrée et produit le flux de messages de sortie correspondant. Le résultat est alors une autre fonction actor f implémentant l'objet selon le modèle d'acteur interne de Pd décrit ci-dessus. En utilisant cette fonction, l'objet fibs peut maintenant être défini à l'aide de la fonction Pure suivante :

```
using actor;
fibs = actor (f (fibs 0 1)) with
  f (x:xs) (y:ys) = x : f xs ys &;
  fibs a b = a : fibs b (a+b) &;
end;
```

Figure . Un objet Pure dans un patch Pd



La figure 2 montre cet objet fonctionnant dans un patch Pd. Bien sûr, ce n'est qu'un simple exemple qui, en soi, n'est pas encore musicalement très utile. Mais dans un cours d'informatique musicale, on pourrait ensuite demander aux élèves de modifier cette fonction, par exemple en prenant tous les nombres modulo un nombre donné, et en ajoutant un décalage, pour obtenir des motifs de notes intéressantes musicalement qui peuvent alors être produites en tant que données MIDI. D'autres flux périodiques de données musicales peuvent être ajoutés, par exemple pour obtenir les vélocités. Les étudiants peuvent ainsi commencer à explorer le langage en modifiant les exemples existants et en les transformant en musique, en utilisant les possibilités intégrées MIDI et de synthèse sonore de Pd. Pd-Pure permet de procéder de manière interactive et en direct en rechargeant l'objet Pure après chaque modification du programme. Avant qu'ils ne s'en rendent compte, les étudiants se trouveront en train de programmer une simple application de composition algorithmique en temps réel. C'est une excellente façon d'apprendre le langage, et il va sans dire que c'est beaucoup plus amusant que de regarder des chiffres ennuyeux dans un terminal.

Conclusion

Enseigner un sujet interdisciplinaire et complexe comme l'informatique musicale à un groupe d'étudiants également hétérogène n'est pas facile, mais cela est possible. Dans cet article, nous avons présenté quelques-unes des leçons que nous avons apprises au groupe de recherche Music-Informatics du JGU et nous avons esquissé certaines de nos approches ainsi que les outils logiciels que nous utilisons. Bien sûr, il n'y a pas de solution universelle, et tout le monde dans ce domaine a ses propres préférences en matière de style d'enseignement, de sujets et de sélection de logiciels. Mais j'espère que ces réflexions pourront

contribuer à un consensus sur ce qui est important dans ce domaine émergeant et ce qui pourrait éventuellement faire partie d'un programme d'études standard pour l'informatique musicale et les sujets connexes.

En ce qui concerne les outils de programmation, il existe bien sûr des solutions alternatives à celles qui sont proposées ici, même dans le monde open-source. En particulier, Csound (Lazzarini *et al.*, 2016) et SuperCollider (Wilson, Cottle & Collins, 2011) sont deux environnements bien connus pour la programmation en informatique musicale, avec de grandes communautés et de nombreuses ressources utiles en ligne. Il existe également un certain nombre de livres imprimés de grande qualité disponibles sur Csound, Pd et SuperCollider, ce qui rend les enseignements beaucoup plus faciles. Faust et Pure sont encore en développement actif en ce moment, donc les livres à leurs sujets sont difficiles à trouver, mais cela va s'améliorer lorsque ces langages et les outils environnants vont commencer à atteindre leur maturité. Pour le moment, les lecteurs peuvent se référer aux informations en ligne disponibles listées dans l'addendum « Ressources en ligne » ci-dessous.

Une dernière mise en garde est à ajouter : apprendre à programmer n'est jamais facile, et pour autant que je sache, personne n'a jamais appris à programmer en regardant quelqu'un le faire. Programmer des logiciels est une activité intellectuelle très exigeante qui nécessite un dévouement, une persévérance, une réflexion très précise, une concentration et une compréhension approfondie des langages de programmation et des environnements utilisés. Cela est vrai même pour un outil graphique assez intuitif comme Pd. Ainsi, alors que les langages et les outils dont j'ai parlé ici peuvent faciliter les premières étapes, ils ne transforment pas la programmation en jeu d'enfant comme par magie. Si les enseignants connaissent bien leur sujet, nous devons toujours nous rappeler combien nos premières étapes ont été difficiles, soyons patients avec les étudiants et donnons-leur beaucoup de commentaires positifs. En fin de compte, les étudiants décideront s'ils trouvent la programmation assez amusante et assez enrichissante pour s'y tenir. Tout ce que nous pouvons faire pour que cela se produise est de choisir des outils appropriés et de trouver un bon équilibre entre l'exploration ludique et l'enseignement des connaissances requises, afin de permettre aux étudiants de commencer à apprendre par eux-mêmes.

Enfin, une si grande partie de notre vie quotidienne dépend actuellement de logiciels, que je considère l'enseignement de la programmation, de manière à ce qu'elle « s'implante » dans l'esprit d'au moins quelques personnes, comme un objectif d'une importance vitale. Nous avons besoin de *beaucoup* plus de monde qui puisse comprendre ce qu'est un logiciel et comment il est créé, et ce dans tous les domaines. L'enseignement de l'informatique musicale aux personnes intéressées par la musique semble certainement un bon moyen de faire en sorte que cela se produise.

Ressources en ligne

---

1. Une distribution Linux populaire créée à l'origine par une société allemande, aujourd'hui appelée openSUSE, cf. https://www.opensuse.org/

2. Une distribution Linux populaire basée sur Debian, cf. http://www.ubuntu.com/

3. Un dérivé populaire de la distribution Arch, cf. https://manjaro.org/

4. Une distribution Linux en publication continue s'adressant aux utilisateurs experts, https://www.archlinux.org/

5. Cela inclut encore des logiciels DAW non-open source, mais peu coûteux (mis en évidence en italique ici), principalement parce qu'Ardour ne vient d'être que très récemment porté sur Windows, et nous avons donc besoin de solutions multiplateformes alternatives.

6. MIDI (Musical Instrument Digital Interface) et OSC (Open Sound Control) sont deux protocoles standard largement utilisés pour envoyer et recevoir des informations de contrôle dans les domaines musicale et multimédia. Voir https://www.midi.org/ et http://opensoundcontrol.org/

7. La version de Pd utilisée dans notre établissement est celle de Virginia Tech *Pd-L2Ork* qui comporte d'intéressantes améliorations en termes d'interface utilisateur ainsi qu'une vaste bibliothèque d'objets externes. voir http://l2ork.music.vt.edu/. Pd-L2Ork ne fonctionne actuellement que sur Linux, mais un travail est en cours pour créer une version multiplateforme fonctionnant également sur Mac et Windows, voir https://git.purrdata.net/u/jwilkes .

8. VST est une norme de plugin multiplateforme bien connue et largement utilisée par les logiciels commerciaux, tandis que LV2, surtout populaire sur Linux et autres systèmes de type Unix, est directement pris en charge par de nombreux programmes DAW open-source.

Pour citer ce document:
Albert Gräf, « Leçons tirées de l'enseignement de l'informatique musicale à des musicologues », *RFIM* [En ligne], Numéros, n° 6 - Techniques et méthodes innovantes pour l'enseignement de la musique et du traitement de signal, Mis à jour le 27/06/2018
URL: http://revues.mshparisnord.org/rfim/index.php?id=530