MIGRER DES ŒUVRES AVEC ELECTRONIQUE TEMPS REEL VERS FAUST

Alexander Mihalic *Laurent Pottier*
Université Jean Monnet
CIEREC, EA 3068
F-42023, Saint-Etienne, France
alexander.mihalic@univ-st-etienne.fr laurent.pottier@univ-st-etienne.fr

RÉSUMÉ

Dans le cadre du projet ANR ASTREE (préservation et synthèse de processus temps réel)[1], le CIEREC étudie les possibilités de portage d'outils de traitement et de synthèse audionumérique vers le langage Faust qui permet de garantir leur pérennité et leur préservation. Nous avons donc fait des comparaisons entre ces deux environnements et effectué le portage de certains outils existants dans d'autres environnements (principalement Max/MSP) vers Faust [4]. Nous étudions également des techniques de validation qui permettent de vérifier l'efficacité de ces portages.

1. INTRODUCTION

La question de la préservation des œuvres numériques peut s'effectuer à différents niveaux.

Le premier niveau consiste à sauvegarder des pièces, en numérisant leurs enregistrements provenant de supports analogiques (disque vinyle ou bande magnétique le plus souvent). Dans le cas de pièces sur support (musique acousmatique, musique sur bande, musique concrète) on peut penser que cette sauvegarde peut suffire à sauvegarder l'œuvre. Elle n'est pas forcément satisfaisante, notamment pour le musicologue qui cherche à comprendre aussi comment l'œuvre a été réalisée et avec quels outils.

Il est donc important d'essayer de sauvegarder les outils, qui permettraient de reconstruire ou de rejouer la pièce. C'est le problème du portage, de la migration des outils d'une plateforme vers une autre, nécessitant parfois de réécrire des instruments qui n'existent plus.

Il est également intéressant de sauvegarder une représentation des processus de traitement de signal qui sont mis en œuvre dans une pièce ainsi que les paramètres qui sont transmis à ces processus au long de la pièce. Il ne s'agit alors pas uniquement de trouver le code qui permettait de programmer les outils de traitement de signal, mais de donner une description mathématique et physique de ces traitements, de façon à pouvoir éventuellement les réimplementer avec d'autres langages totalement différents du langage originel utilisé.

Nous avons choisi d'utiliser un langage de programmation spécialisé dans le traitement du signal qui permette d'obtenir une documentation mathématique et textuelle du processus utilisé : le langage Faust développé à GRAME.

Nous cherchons à faire un portage de la partie DSP de divers outils vers le langage Faust, ce qui peut nous garantir ainsi une pérennité de ces outils. Cela nous permet de tester Faust pour savoir si cet environnement répond effectivement à nos attentes, et donc s'il peut être conseillé aux développeurs et aux réalisateur qui souhaitent construire des outils ayant une durée de vie supérieure aux autres, et ainsi assurer la pérennité des œuvres.

Notre travail porte sur plusieurs points.

Nous avons d'abord réalisé une étude des usages, à la fois sur Max/MSP, un langage qui permet de réaliser des programmes temps réel à pérennité réduite, dont le portage doit être réalisé régulièrement, et sur Faust, le langage qui doit nous garantir la pérennité.

Cette étude des usages nous a permis de mesurer la diversité et la nature des outils disponibles dans ces deux environnements.

Dans un deuxième temps, nous avons cherché à comparer ces deux langages pour savoir dans quelles limites des portages seraient possibles entre ces deux langages. Nous avons donc cherché à trouver les primitives communes aux deux langages, à les comparer et ensuite à répertorier les primitives manquantes, dans Faust par rapport à Max/MSP.

Dans un troisième temps, nous avons commencé à mettre au point les primitives manquantes dans Faust, soit par programmation manuelle des outils manquants (sous réserve de disposer d'une description suffisamment précises de ces modules), soit par transcription automatique, grâce à un travail réalisé par l'équipe de Jérôme Barthélémy à l'Ircam.

Nous avons alors testé la validité de ces modules en comparant les versions Max/MSP et les versions Faust.

Enfin, nous avons tenté de mettre en œuvre des modules DSP réalisé avec Faust pour mettre à jour des pièces du répertoire qui puissent être rejouées en temps réel.

[1] ASTREE, Analyse/Synthèse de processus Temps-Réel, est un programme de recherche labellisé et subventionné par l'Agence Nationale de la Recherche (ANR-08-CORD-003) qui regroupe des chercheurs de l'IRCAM, de GRAME, de MINES ParisTech et du CIEREC.

Certaines de ces pièces sont des pièces temps réel, programmées initialement dans Max/MSP - *Laccampu* (2002) de Thierry Pécou, portage par Laurent Pottier, *En Echo* (1997) de Philippe Manoury, portage par Alain Bonardi -, d'autres sont des pièces sur support, mais dont la structure nous a semblé permettre une réimplantation pour le concert avec interprétation des processus de synthèse par des musiciens sur scène - *Turenas* (1972) de John Chowning, portage par Laurent Pottier.
Nous décrirons quelques-uns des résultats obtenus et des problèmes rencontrés lors des recherches menées au CIEREC dans le cadre du projet ANR ASTREE.

2. COMPARAISON DES ENVIRONNEMENTS

Le premier point important à noter est que Faust ne gère que les fonctions DSP synchrones. Si Faust dispose de certains outils *Interface-Utilisateur* (bouton, curseurs) et permet de recevoir des messages via OSC, il a toutefois besoin d'un système hôte qui gère la temporalité des variations des paramètres du son, sauf si ceux-ci sont intégralement contrôlés par des interprètes en direct ou par des processus synchrones (voir l'installation *SonikCub*[2]). En contrepartie, Faust dispose de compilateurs qui lui permettent de réaliser des *Plug-Ins* ou des fonctions externes pour la plupart des environnements audionumériques existants (VST, Supercollider, Max/MSP, Pure Data, Flash, Csound, Jack).

La comparaison avec Max/MSP se limite donc aux objets DSP.

Nous avons donc recensé les objets élémentaires de traitement de signal de Max/MSP et avons étudié leur utilisation dans différents types de patchs simples (« maxhelp », « msp-tutorial », « ircam-jimmies ») ou plus complexes (patchs utilisés dans des œuvres du répertoire).

Maxhelps		Max Tutorials		Jimmies	
261	*~	184	*~	43	dac~
157	ezdac~	57	cycle~	25	gain1~
124	cycle~	53	line~	17	noise~
108	number~	36	dac~	15	*~
62	phasor~	25	+~	15	cycle~
61	scope~	24	meter~	11	adc~
40	+~	21	buffer~	8	meter~
34	line~	18	ezdac~	6	stgain1~
32	filtergraph~	16	sig~	5	-~
32	selector~	12	adc~	5	biquad~
30	sig~	11	fftin~	5	clip~
27	sfplay~	11	gain~	5	line~
23	meter~	10	gate~	4	+~
23	noise~	9	ezadc~	4	phasor~
20	buffer~	9	tapin~	4	sig~
19	-~	9	tapout~	3	comb1~
17	gain~	8	fftout~	2	adc1~
17	spectroscope~	7	-~	2	buffer~
13	saw~	7	phasor~	2	delay2~
13	snapshot~	7	selector~	2	hilbert~
11	tapout~	7	sfplay~	2	number~
9	adc~	6	number~	2	record~
9	tapin~	6	partial~	2	vu1~

Figure 1 : tableau de fréquence d'apparition des objets DSP dans les patchs de plusieurs répertoires de Max/MSP (extrait).

[2] *SonikCub*: installation de Trafik et Yann Orlarey.

2.1. Les oscillateurs

Même si des objets qui semblent équivalents existent dans les deux environnements, il nous a fallu valider cette équivalence. Par exemple, la comparaison des oscillateurs présents dans Max/MSP et dans Faust révèle des approximations qui induisent des décalages de phase à long terme, qui varient selon la fréquence.

Les seules fréquences pour lesquelles on ne constate pas de déphasage sont celles qui obéissent à la formule suivante :

$$Freq = \frac{11025}{2^n} x \qquad (1)$$

où n et x sont les nombres entiers[3].

Pour ces fréquences, on peut comparer le résultat produit par des oscillateurs de Max/MSP (*cycle~*) et de Faust (*osc*) en soustrayant le signal produit par les deux. On obtient alors un signal nul.

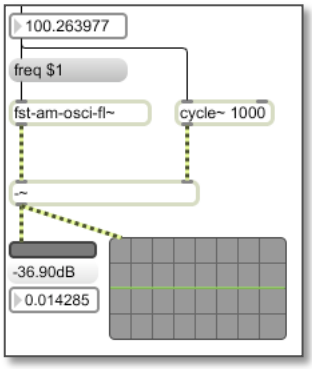

Figure 2 : soustraction entre le signal produit par la fonction cycle~ dans Max/MSP et un oscillateur sinusoïdal créé dans Faust.

Nous avons ensuite fait le portage de certains objets de Max/MSP vers Faust.

2.2. Les filtres – synthèse soustractive

Dans le cas de fonctions élémentaires du langage, comme l'objet *biquad~* dans Max/MSP, dont nous avons eu besoin pour le portage de la pièce *Laccampu*, le code de la fonction n'est pas disponible. Nous avons toutefois trouvé dans la littérature le texte de Robert Bristow-Johnson[4][1] qui décrit les paramètres de ce type de filtre. Nous avons implémenté ces filtres dans Faust et comparé leur action avec les filtres de Max/MSP. En soustrayant le signal produit par le filtre de Max/MSP et la fonction écrite en Faust et compilée pour Max/MSP, nous avons trouvé un signal nul confirmant l'équivalence des deux types d'objets (voir figure 5).

[3] Pour un taux d'échantillonnage de 44100 Hz. A noter que des comparaisons effectuées entre Max/MSP et PureData ou SuperCollider ont montré des différences encore plus importantes.

[4] http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

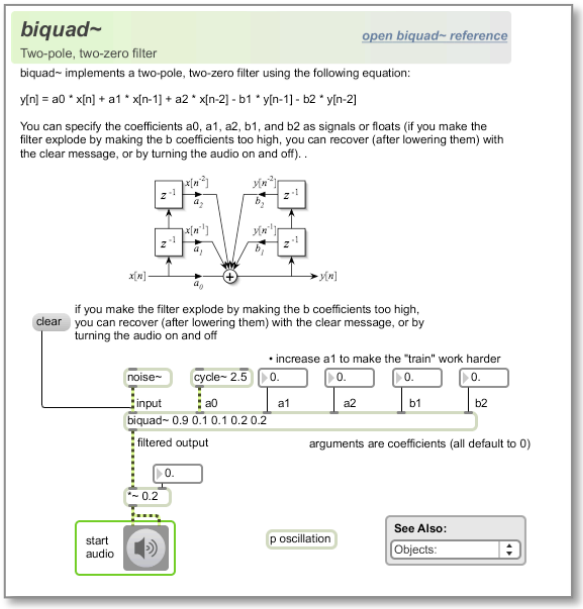

Figure 3 : l'objet biquad~ dans Max/MSP

2.3. Traduction automatique : réverbération, synthèse FM

Dans le cas d'objets MSP issus de sous-patchs DSP, l'équipe de Jérôme Barthélémy a mis au point à l'Ircam un programme permettant une traduction automatique d'un patch graphique MSP vers un programme Faust (sous réserve que les fonctions présentes dans le patch MSP aient leur équivalent connu dans Faust).

Le portage des fonctions *rev4~* (réverbération) et *simpleFM~* a ainsi pu être réalisé et dans les deux cas, l'absence de différence entre les signaux produits par les fonctions Max/MSP et Faust a pu être constatée[5].

```
simpleFM_(in0, in1, in2) = ((in0), ((((in0)):e:osc_g, ((((((in0),(in1)):e),(in2)):h)):f)):b:osc_c:a
with {
        a = _; //<-----[outlet, ]
        b = +; //<-----[newobj,+~]
        osc_c = oscrc; //<-----[newobj, cycle~]
        e = *; //<-----[newobj, *~ #1]
        f = *; //<-----[newobj, *~]
        osc_g = oscrc; //<-----[newobj, cycle~]
        h = *; //<-----[newobj,  *~ #2]
];
process = simpleFM_;
```

Code généré par le programme de traduction automatique.

[5] Pour la synthèse FM, avec la réserve d'utiliser des fréaquences qui ne provoquent pas de différences de phase, comme indiqué par la formule (1).

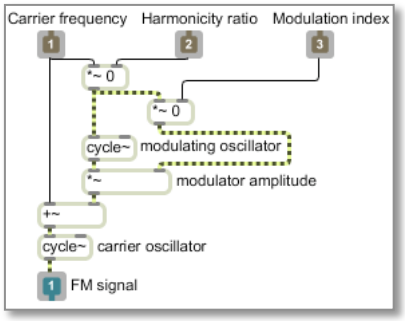

Figure 4 : patch de synthèse FM dans Max/MSP

3. L'EXEMPLE DE *TURENAS*

La pièce *Turenas* a été entièrement construite par synthèse FM [2], avec une spatialisation quadriphonique et en utilisant une réverbération mise au point par John Chowning [3].

Cet exemple nous a permis de tester le potentiel de l'utilisation des processus Faust dans Max/MSP pour une production grandeur nature [5].

Quatre synthétiseurs FM ont ainsi été écrits en Faust, disposant tous d'enveloppes d'amplitudes intégrées[6]. Les figures 6 et 7 montrent respectivement le synthétiseur VLN programmé en Max/MSP et l'objet externe correspondant programmé en Faust et compilé pour Max/MSP. Nous avons noté un gain en performance de calcul de 30% (on passe de 35% du CPU à 25%).

Les difficultés rencontrées dans la programmation Faust résident en partie dans la prise en compte des initialisations afin que lorsque des objets polyphoniques sont créés, les valeurs envoyées ne provoque pas de ruptures. Cela nous a notamment amené à créer des entrées « signal » pour tous les paramètres de la synthèse, afin de pouvoir réaliser des transitions progressives d'une valeur à une autre.

Ce type de portage nous a permis de tester Faust en situation de concert, et les résultats ont été très encourageants.

Ce portage a permis également d'étudier en détail les techniques de synthèse et tous les processus mis en œuvre par John Chowning pour réaliser cette pièce. Enfin, il nous a amené à réfléchir aux moyens à prendre, pour transformer une œuvre figée (enregistrée sur bande) en une œuvre interactive (jouée par des interprètes). De nombreuses questions ont dû être résolues : quel type d'interprètes choisir, quels paramètres contrôler, quelles interfaces gestuelles mettre en œuvre etc. [6]

4. CONCLUSION

L'utilisation de Faust pour la mise au point de dispositifs de traitement de signal s'est révélée très intéressante. D'une part les fonctions écrites en Faust

[6] Pour une étude détaillée du synthétiseur VLN, voir le site musimediane : http://www.musimediane.com/numero6/POTTIER/index.html

sont extrêmement optimisées et nécessitent des temps de calcul souvent inférieurs à ceux requis pour des patchs programmés en Max/MSP. Par ailleurs, le code est particulièrement concis, même s'il est écrit dans une syntaxe qui le rend difficile à lire pour des utilisateurs non initiés, habitués à des langages comme le C. Avec un peu de pratique, il se révèle très élégant, vraiment bien adapté à la programmation DSP et finalement assez intuitif. Sa documentation automatique permet de conserver les formules utilisées et de pouvoir ultérieurement les utiliser dans d'autres environnements. Enfin, le code écrit en Faust permet des réaliser des *Plug-Ins* pour des environnement très variés ce qui est un atout majeur pour le programmeur. Par ailleurs, même si historiquement Max/MSP est un environnement qui dispose d'une communauté d'utilisateur très importante, comme les codes des applications réalisées par ces utilisateurs sont rarement accessibles (objets externes écrits en C), ils n'évoluent pas toujours avec les différentes versions du programme et finissent souvent par disparaitre. Avec Faust, qui dispose d'une licence GPL, la communauté s'organise et se développe, puisqu'il existe déjà des bibliothèques variées pour Faust (math.lib, music.lib, effect.lib, filter.lib, osc.lib, maxmsp.lib, reduce.lib, guitarix.lib…) et que des développements sur Faust sont réalisés dans plusieurs centres importants de l'informatique musicale, par Julius O. Smith (Université Stanford), Albert Gräf (Université Johannes Gutenberg), Stefan Kersten(TU Berlin), Tiziano Bole…

5. REFERENCES

[1] BRISTOW-JOHNSON, Robert, « Cookbook formulae for audio EQ biquad filter coefficients » http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

[2] CHOWNING, John, « The synthesis of complex audio spectra by means of Frequency Modulation », J. Audio Eng. Soc., 21:7, 1973. Paru également dans Curtis Roads & John Strawn (dir.), *Foundations of Computer Music*, Cambridge, Massachusetts : MIT Press, 1985.

[3] CHOWNING, John, « The simulation of moving sound sources », *J. Audio Eng. Soc.,* 19:1, 1971. Paru également dans *Computer Music Journal* 1:3, 1977.

[4] ORLAREY, Yann ; Fober, Dominique ; Letz, Stephane, « FAUST: an Efficient Functional Approach to DSP Programming », *New Computational Paradigms for Computer Music*, France : Delatour, 2009.

[5] POTTIER, Laurent, « *Turenas* (1972) de John Chowning, vers une version interactive », *Musimediane*, n°6, 2011. http://www.musimediane.com/numero6/POTTIER/index.html

[6] POTTIER, Laurent, « Musiques électroniques, vers de nouveaux modes de jeu », E-Formes 2, Arts & écritures numériques, au risque du jeu, dir. M. Maza et A. Saemmer, Saint-Etienne : PUSE, 2011, p. 155-165.

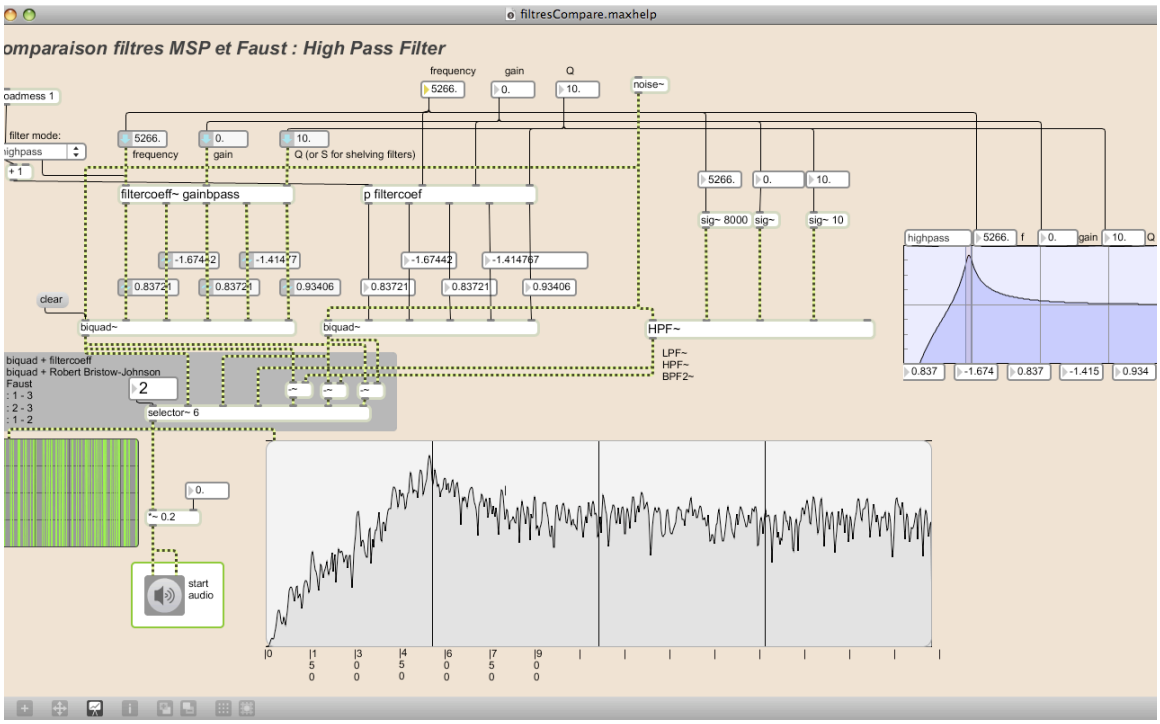

Figure 5 : comparaison entre 3 signaux : le filtre biquad~ contrôlé par l'objet filtercoeff~, ce même filtre contrôlé par les équations de Robert Bristow-Johnson et le filtre HPF programmé dans Faust.
La soustraction de deux des trois signaux produit toujours un résultat nul.

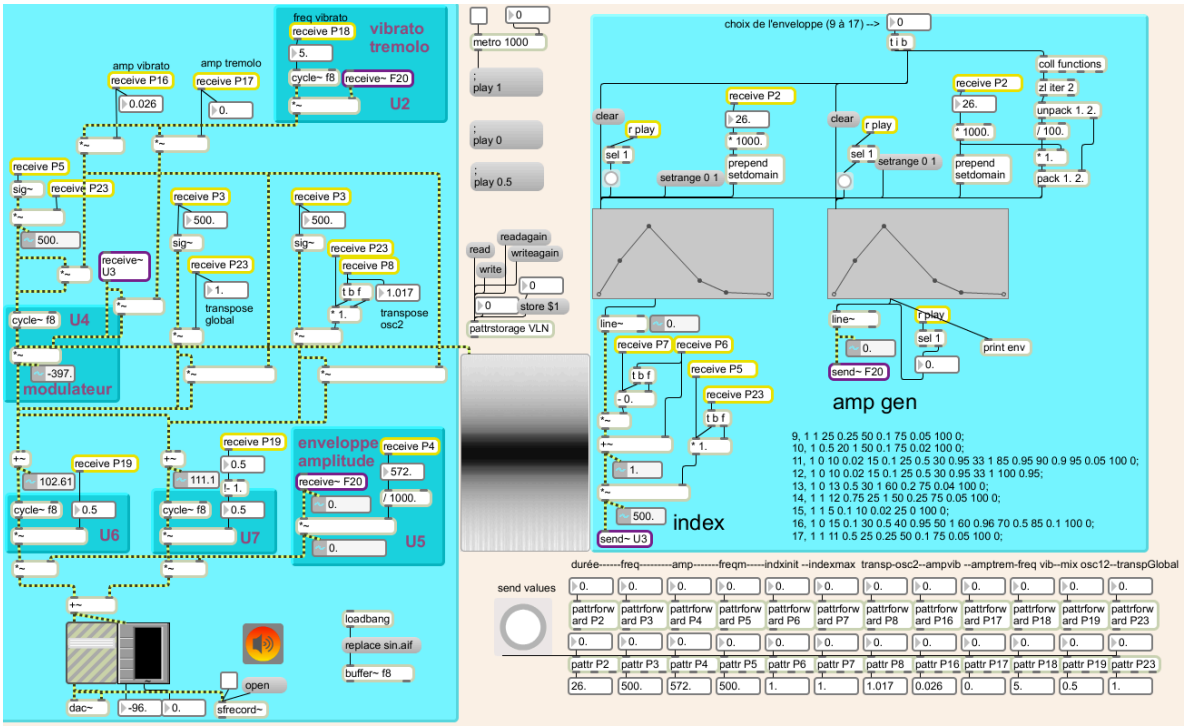

Figure 6 : le synthétiseur VLN programmé dans Max/MSP
consommation CPU : 34-36%. pour 32 voix de polyphonie (Maxbookpro 2.53GHz vectorsize = 512)

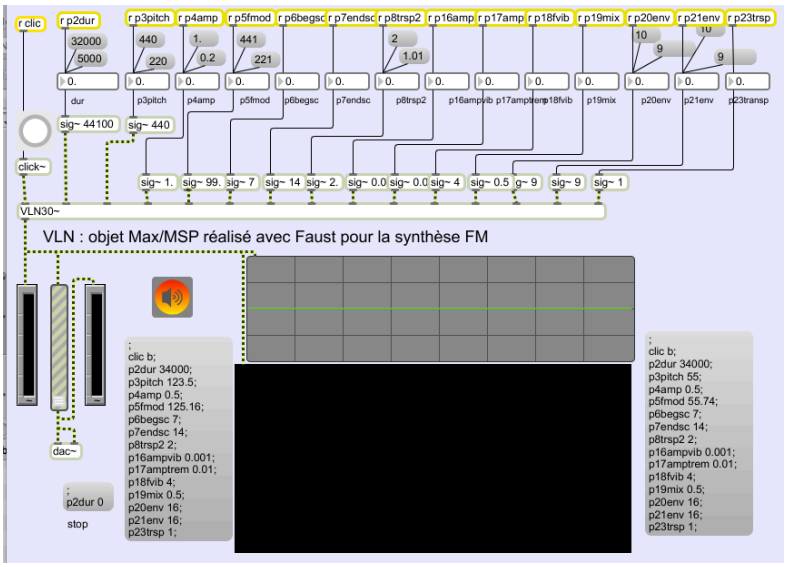

Figure 7 : le synthétiseur VLN programmé dans Faust et compilé pour Max/MSP
consommation CPU : 23-25%. pour 32 voix de polyphonie (Maxbookpro 2.53GHz vectorsize = 512)

```
vln(trig, dur, p3pitch, p4amp, p5fmod, p6begscal, p7endscal, p8trsp2, p16, p17, p18fvib, p19mix, p20env, p21env, p23trsp) =
        env(trig, nsamples, p20env, 0, p4amp) <:
        *(p19mix * osci(U4modul + (p3pitch * p23trsp) + (p3pitch * p23trsp) * U2vibra * p16)),
        *((1 - p19mix) * osci(U4modul + (p3pitch * p23trsp * p8trsp2) + (p3pitch * p23trsp * p8trsp2) * U2vibra * p16))
        :> +
with {
        nsamples = max(1, dur * SR / 1000.0) ;
        U4modul = (U3indx + (U3indx * p17 * U2vibra)) * osci((p5fmod * p23trsp) + (p5fmod * p23trsp * p16 * U2vibra));
        U2vibra = env(trig, nsamples, p20env, 0, 1) : *(osci(p18fvib));
        U3indx = env(trig, nsamples, p21env, p6begscal * p5fmod * p23trsp, p7endscal * p5fmod * p23trsp);
} ;

process = vln;
```

Code en Faust du synthétiseur VLN (excepté pour la partie « enveloppe »)