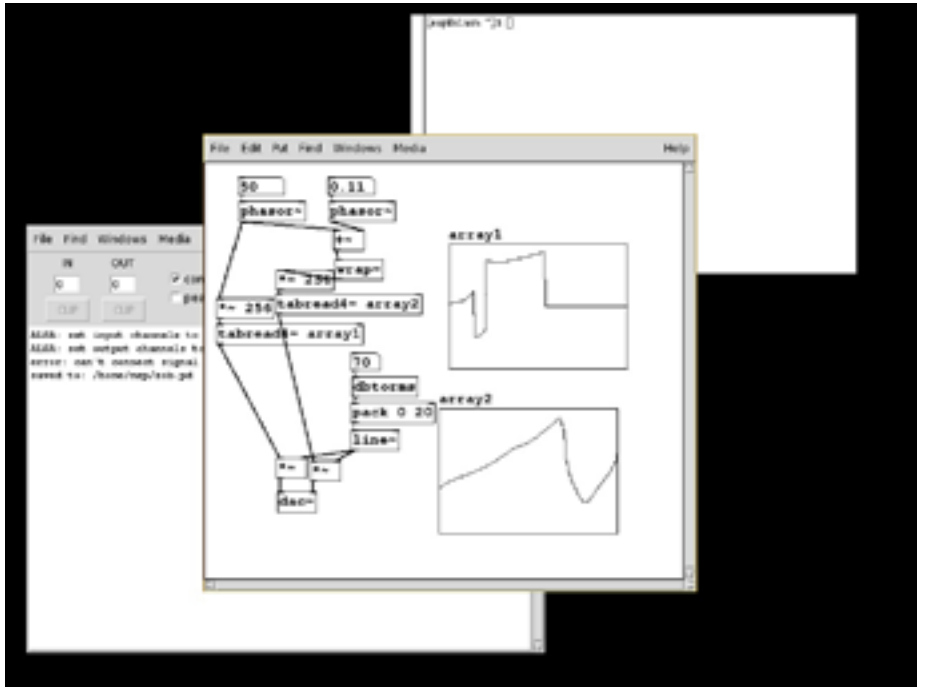

Dans Pure Data, les programmes appelés *patches* consistent en des réseaux d'objets interconnectés. Le *patch* présenté ici, qui utilise la technique de modulation de fréquence, sert à créer un son musical. (Capture d'écran issue de l'interface graphique personnelle de Miller Puckette.)

http://www-crca.ucsd.edu/~msp/
Site de Miller Puckette.

http://puredata.info
Site de la communauté Pure Data.

http://cycling74.com/products/maxmspjitter/
Page de présentation de Max/MSP chez Cycling '74, compagnie fondée par David Zicarelli.

Étude de cas sur les logiciels pour artistes : Max/MSP et Pure Data

Miller Puckette

Imaginons que vous ayez longtemps travaillé dans un secteur spécifique, quel qu'il soit. Cela fera-t-il forcément de vous la personne la mieux placée pour en décrire les rouages ? À la lumière de cette interrogation, il convient de voir le texte qui suit, traitant de logiciels et d'art et produit par quelqu'un qui conçoit ces mêmes logiciels, comme le résultat d'un point de vue particulier. Un point de vue qui, s'il se base indéniablement sur des années d'expérience, reste toutefois loin d'être objectif.

Les réflexions que je propose ici trouvent leur origine dans ce que j'ai vécu au cours des vingt-sept dernières années, durant lesquelles je me suis efforcé de concevoir des logiciels pour des artistes désireux de créer des œuvres d'art interactives. Alors qu'au départ je me concentrais principalement sur l'utilisation de l'ordinateur lors des concerts, le projet a progressivement pris de l'ampleur, au point d'inclure d'autres médias (vidéo et graphisme, réseaux, robotique) et d'autres modes de présentation (installations, productions enregistrées en studio). Je tiens néanmoins à préciser qu'à titre personnel, je m'intéresse aux nouvelles possibilités offertes dans le domaine de la musique.

Bien que l'on puisse considérer les logiciels en eux-mêmes comme de l'art, et/ou que l'art puisse aborder la question des logiciels ou de l'outil informatique dans sa globalité, j'adopterai une position plus conventionnelle consistant à les envisager, au même titre

que l'ordinateur, comme des outils aidant à la création artistique, que l'on rencontrera dans l'atelier de l'artiste mais qui ne seront pas directement à la vue du public. Quoi qu'il en soit, leur conception joue un rôle essentiel dans ce que l'artiste sera finalement en mesure de créer, et c'est pourquoi elle devrait dans l'idéal reposer sur une certaine appréciation de ce qu'il fait ou de ce à quoi il souhaite parvenir.

Les logiciels et l'artiste

Les *software artists* créent bien entendu des logiciels (du *software*). Cependant ils doivent tous, à un moment ou à un autre, avoir recours à des outils. Qu'ils correspondent alors aux interrupteurs à bascule sur le panneau d'un ordinateur PDP-8 ou à l'interface homme-machine dernier cri, on en revient toujours à une certaine forme d'interface, et par la même occasion à la nécessité de trouver quelqu'un qui saura les élaborer en tenant compte de leur utilité pour l'artiste. Mais dans ce cas, comment peut-on permettre à ce dernier de faire les choses plus facilement, quand on sait que la majeure partie de ce qu'il veut faire n'est certainement jamais venue à l'esprit du développeur ?
Dès lors qu'il est impossible de mesurer à quel point l'action de ce développeur favorise la création artistique, lui-même se retrouve totalement incapable de déterminer de façon systématique si un avant-projet est meilleur qu'un autre. Tout enseignement que l'on pourra tirer de telles situations risque donc de s'avérer quelque peu brouillon, voire parfois contradictoire. En effet, à défaut d'indicateurs et d'exemples précis, tout ce que l'on peut offrir, ce sont des intuitions et des généralités.
Aujourd'hui survit encore un idéal romantique de l'artiste technophobe, s'attachant exclusivement aux sentiments et ne pouvant œuvrer qu'à l'écart des avantages pratiques offerts par les matériels et techniques. Or les artistes travaillant dans le domaine des médias électroniques ne peuvent se permettre de penser en ces termes, dans la mesure où le maniement de ces médias dans une optique artistique et originale nécessite une compréhension totale et une connaissance approfondie de leur fonctionnement. Dans le cas particulier des arts utilisant l'ordinateur, les packs de logiciels de montage sonore et de traitement d'images – très complets et faciles d'utilisation – se sont multipliés, tant et si bien que le paysage informatique se retrouve désormais envahi de leurs produits : du travail de pro, que ce soit sur l'image ou le son. Les artistes voulant sortir des sentiers battus pour s'aventurer sur d'autres terrains se voient donc contraints de jouer un rôle plus actif dans l'utilisation de leurs outils.
Ces logiciels de création artistique par ordinateur peuvent être classés dans un spectre qui s'étend des applications non évolutives aux environnements de développement – tout véritable outil logiciel se trouvant alors quelque part entre ces deux extrêmes. Dans le cadre de formes d'art plus créatives, plus expérimentales, on peut aisément supposer que l'artiste devra s'appuyer sur la partie du spectre correspondant aux « environnements de développement », et de la même façon, arriver à la conclusion que ce côté requiert davantage de subtilité de la part de l'utilisateur.

Or il s'avère difficile, et en définitive peut-être même illusoire, d'établir une nette distinction entre les deux extrêmes. En effet, si on prend l'exemple des logiciels de traitement d'images, appartenant aux dites « applications non évolutives », on remarque qu'ils sont parfois dotés d'une capacité d'écriture de scripts ; les environnements de développement, pour leur part, ne sont en réalité rien de plus que des types ou des ensembles d'applications spécifiques.
En d'autres termes, les applications « accomplissent » des tâches, là où les environnements de programmation se contentent d'attendre qu'on le fasse nous-mêmes. Un environnement de programmation offre à l'utilisateur un écran vide sur lequel projeter son imagination. En revanche, une application destinée à l'« utilisateur final » présente tous les choix que son créateur a pu mettre en œuvre en ajoutant à l'interface graphique de petits composants attractifs. Même après une utilisation intensive, ce type d'application gardera globalement les mêmes caractéristiques.
Si on étudie attentivement la question, on se rend compte qu'il est tout simplement faux de prétendre que les environnements de programmation requièrent davantage d'efforts ou de maîtrise. Quelle que soit la finalité d'un pack de logiciels, au cours de son développement il a toujours tendance à devenir de plus en plus complexe. Il reste toutefois dans la limite des capacités cognitives de l'utilisateur, qui atteint à son tour, par l'utilisation de ses divers constituants, un niveau de compétences qui ne dépend que du temps qu'il peut y consacrer.
Adoptons maintenant une vision différente des logiciels et de leur « portée », c'est-à-dire de l'ensemble des possibilités qu'ils offrent en tant qu'outils. Supposons que l'on compare deux packs de logiciels, les packs A et B. En somme on peut considérer que chacun prend un flux de 1 et de 0 (apportés par l'utilisateur) et les interprète pour en faire un produit fini, par exemple un fichier audio. Dans l'éventualité où A et B ne seraient pas des packs trop spécialisés, chaque donnée de sortie, quelle qu'elle soit, pourrait être générée soit par A soit par B. Cependant, si on limite les données d'entrée à une taille précise, disons à un méga-octet, le nombre de sorties possibles n'est plus que de $2^{8000000}$ (et même un peu moins en réalité, dans la mesure où de nombreuses chaînes de caractères en entrée peuvent très bien générer exactement les mêmes données de sortie). Ainsi, si on part du principe que le concepteur et l'utilisateur s'entendent pour offrir au second des performances aussi étendues que possible, et ce quelle que soit l'approche de base adoptée dans la conception du logiciel, alors le nombre de choses que l'utilisateur pourra faire restera plus ou moins identique.
Il se peut également que certaines chaînes de caractères en entrée soient plus accessibles que d'autres pour l'utilisateur. Si les données sont interprétées à l'aide d'un langage de programmation par exemple, il devra certainement consacrer du temps à la maîtrise de quelques petites astuces, ce qui lui reviendra plus cher en termes de coût par *bit*[1].

1. Le « coût par *bit* » est une unité de mesure permettant d'évaluer l'investissement, financier ou personnel, de l'utilisateur dans l'achat puis la maîtrise d'un matériel informatique (NdT).

On pourrait, du moins en principe, corriger ces variations en remplaçant la longueur totale des entrées binaires que l'utilisateur apporte par une autre variable reflétant le temps passé à élaborer les données d'entrée, ou peut-être même l'entropie de Shannon[2] du flux binaire. Dans n'importe quel cas de figure, en théorie, il y a toujours un large échantillon de sorties possibles, et dont la taille sera similaire qu'il s'agisse du pack de logiciels A ou B. Concevoir un bon outil informatique à finalité artistique reviendrait par conséquent à exploiter au maximum la diversité du paysage couvert par un échantillon de taille limitée. Malheureusement, cela laisse en suspens la définition du mot « diversité », qui s'avère difficile à cerner. On sait bien entendu à quoi elle fait référence, mais une idée originale peut alors surgir et offrir des possibilités que personne n'avait imaginées jusque-là, à tel point que ce qui semblait être au départ un paysage particulièrement divers paraître ensuite n'occuper qu'une niche assez réduite au sein de ces nouvelles possibilités.

Il existe cependant d'autres moyens de mesurer l'utilité d'un outil logiciel. En effet on pourrait se contenter, entre autres, d'évaluer son côté ludique. À l'instar de la variable « diversité » présentée ci-dessus, c'est un critère que la plupart des informaticiens s'emploient à ignorer. Ce n'est qu'une impression, mais d'après moi lorsqu'on utilise quelque chose d'amusant, on est amené à faire travailler ses méninges d'une manière ou d'une autre. Ainsi, d'après moi, ce qui divertit l'artiste est davantage susceptible de le guider sur des terrains inconnus et potentiellement productifs.

Bien entendu, c'est l'utilisateur, et non pas le concepteur, qui évalue le caractère ludique du logiciel. Or, si certains pensent que c'est le cas du jeu vidéo *Grand Theft Auto* par exemple, cela ne signifie pas qu'il soit en lui-même un outil d'une quelconque utilité pour des artistes. Il faudrait donc en quelque sorte des logiciels qui soient distrayants du point de vue de ceux qui créent, et non pas du point de vue de ceux qui consomment, ce qui obligerait d'une certaine façon le concepteur à deviner ce qui amusera les artistes.

Que le développeur le veuille ou non, le logiciel se fait irrémédiablement le reflet de sa mentalité et de sa personnalité. Emacs est ainsi à l'image de Richard Stallman et Csound à l'image de Barry Wercoe. Utiliser un logiciel revient donc à passer du temps avec son créateur, et le choix d'un pack au détriment d'un autre reflète la compatibilité qui existe entre la personnalité du développeur et la nôtre. En l'occurrence, pour qu'un logiciel ait du succès, il faut que la personnalité qu'il projette soit attrayante et directe, et non pas obscure et prétentieuse. Une fois de plus il s'agit là d'un aspect que les spécialistes sont incapables de mesurer ou de favoriser, par manque d'outils adaptés.

Hormis les rares occasions où on l'utilise en public, l'ordinateur nous sert à effectuer du travail de bureau. Qu'on soit en train de lire et d'écrire un e-mail ou bien de créer un nouveau *patch*[3], l'opération de base ainsi que son effet sur notre humeur et nos pensées sont les mêmes. Le logiciel, en tant qu'outil de bureau, devrait être fonctionnel et non pas monopoliser l'attention. Son créateur, qui en devient pour ainsi dire un collègue de bureau, ne devrait parler que lorsqu'on s'adresse à lui.

Nos habitudes de travail, et peut-être notre esprit lui-même, sont corrompus par notre expérience de l'ordinateur. Le concepteur doit en être conscient, et éviter alors les comportements ou les habillages qui ne contribuent pas de façon essentielle à ce que veut faire l'utilisateur. Par exemple le recours massif, voire endémique, aux couleurs et aux animations dans les logiciels d'aujourd'hui sert principalement à polluer l'espace qui sépare l'artiste de l'ordinateur. Au contraire, les logiciels devraient dans l'idéal proposer une surface qui soit, esthétiquement parlant, la plus neutre possible.

Implication du logiciel dans des questions plus fondamentales

Au-delà de ce problème visant à rendre les logiciels immédiatement utiles (auquel on pourrait réfléchir en imaginant un seul artiste travaillant sur un unique projet et utilisant d'une manière ou d'une autre un ordinateur), il convient de se demander comment l'ensemble des artistes – avec toutes leurs machines et tous leurs logiciels – évolue dans le temps et participe à la construction de notre culture. Tout concepteur consciencieux se doit en effet de considérer les implications de son travail tant au niveau global que local. Les artistes sont les gardiens et les garants de notre culture. Être entouré d'art digne de ce nom permet de penser plus librement, de prendre conscience de soi et de l'environnement dans lequel on évolue. Ainsi, bien que la présence d'une scène artistique vivante ne soit en aucun cas un critère suffisant pour déclarer la société comme étant elle-même en bonne santé, la situation inverse poserait de graves problèmes, un peu à l'image d'une carence chez un individu dont les habitudes alimentaires négligeraient un minéral essentiel. Indirectement, il est important que les artistes disposent de logiciels de qualité : le concepteur joue alors un rôle secondaire mais primordial dans l'évolution de la scène artistique.

Le travail de l'artiste pourrait sous divers aspects être considéré comme porteur d'une certaine originalité, telle qu'elle est mise en valeur par notre culture occidentale – à savoir en accord tacite avec cette croyance selon laquelle notre civilisation entreprend une longue marche qui la mènera de l'obscurantisme et l'ignorance vers les lumières de la connaissance. Autre fantasme occidental, celui qui veut que l'artiste soit d'une manière ou d'une autre reconnu pour cette originalité, et de façon plus générale que les idées restent la propriété de leur auteur. Aux États-Unis, on dit souvent que l'encouragement

2. L'entropie de Shannon est une fonction mathématique qui permet d'évaluer la quantité d'information contenue ou transmise par une source, qu'il s'agisse d'une langue, d'un signal, de données informatiques… (NdT)

3. Les *patches* sont des programmes additionnels mis à disposition par les éditeurs pour rectifier, mettre à jour ou compléter un logiciel. Le terme *patch* relève également du vocabulaire de Pure Data, dans lequel il désigne les regroupements d'objets graphiques réalisés par l'utilisateur afin d'élaborer des programmes (NdT).

financier de l'« originalité » (par la protection de la propriété intellectuelle) agit comme une incitation à l'innovation. Bien que les auteurs de telles allégations fassent souvent référence à la découverte de nouveaux médicaments ou au développement de nouvelles armes, cette incitation s'appliquerait également aux artistes, dans la mesure où le « divertissement » y est désormais l'un des produits nationaux majeurs en termes économiques. En encourageant les artistes à être plus créatifs, on pourrait dégager de leurs idées une valeur financière plus élevée si elles se concrétisaient sous la forme d'objets susceptibles d'être possédés, achetés et vendus.

L'originalité peut également se manifester à travers le logiciel pour artiste lui-même, qui peut alors être loué pour un aspect précis de son fonctionnement ou dans sa globalité. Ce n'est cependant pas le cas de la plupart des autres outils qu'utilise l'artiste : on peut mettre en évidence la qualité d'un pinceau par exemple, mais pas son côté innovant. Pourtant, l'originalité peut dans ce cas être problématique, car on s'empresse d'intégrer dans ces outils des idées originales au risque d'en déstabiliser la base logicielle. Nous sommes aujourd'hui tiraillés entre la volonté de dépasser les limites de ce qui se fait en matière de logiciel (ce qui implique vraisemblablement de trouver des solutions novatrices et originales aux problèmes) et le besoin de stabilité au niveau des outils (qui permettrait de préserver notre stock de documents électroniques, œuvres d'art y compris). L'œuvre d'art elle-même se mêlant de plus en plus à la technologie par laquelle elle s'exprime, elle devient de plus en plus difficile à perpétuer face aux bouleversements incessants que subissent les logiciels, orchestrés au nom de la recherche de solutions innovantes à nos nombreux tracas.

Pour emprunter la métaphore de David Zicarelli[4], l'artiste et le concepteur sont deux espèces vivant dans un écosystème, dépendantes l'une de l'autre et toutes deux quasiment incapables d'agir sur les forces supérieures qui les animent. L'utilisateur et son ordinateur sont étroitement liés, chacun exerçant sur l'autre un contrôle partiel.

Le concepteur guide toujours l'artiste dans une direction ou une autre. Lorsqu'il essaie de créer des fonctions « utiles » et faciles à exécuter, cela a toujours pour effet de privilégier celles qu'il avait lui-même jugées utiles au départ. Il traîne ainsi derrière lui l'artiste, sans que celui-ci ne puisse rien y faire (mis à part héberger un autre logiciel parasite).

D'un autre côté, on pourrait adopter le point de vue inverse selon lequel l'artiste serait l'éleveur et le concepteur serait le chien. En faisant son choix parmi des programmes et en les combinant, l'artiste assure la descendance d'un certain nombre de « gènes » logiciels au détriment des autres. Le concepteur n'a en réalité aucun pouvoir : quel que soit le produit de son dur labeur, l'artiste se contente de choisir ce logiciel ou bien un autre, interrompant ainsi la lignée de tous ceux qu'il ne souhaite pas utiliser.

4. David Zicarelli, « Music Technology as a Form of Parasite », dans *Proceedings of the 1992 International Computer Music Conference* (San José), International Computer Music Association, San Francisco, 1992, p. 69-72.

Un troisième point de vue[5] conçoit l'art, et plus généralement la culture à laquelle il appartient, comme n'étant rien de plus qu'un plumage dont les représentants de l'espèce humaine se parent pour attirer un partenaire sexuel. Par conséquent nous serions tous, artistes et développeurs, choisis sur la base de critères qui n'ont en fait rien à voir avec les compétences, ni même avec l'esthétique, mais plutôt avec la quantité d'énergie résiduelle que nous pourrions consacrer, après nous être nourris, aux activités culturelles.

La conception de Max/MSP et de Pure Data

Cela fait plus de vingt ans que je travaille sur les programmes Max/MSP et Pure Data[6], une période au cours de laquelle je me suis également forgé les opinions exposées dans cet article. On ne peut donc pas dire qu'à l'époque de la conception de Pure Data (sur lequel je me concentre à l'heure actuelle), ces considérations étaient si clairement définies. En réalité, les nombreux artistes avec lesquels j'ai eu la chance de travailler pendant toutes ces années ont eu, à chaque nouvelle collaboration, une influence bien plus marquée sur le développement du logiciel. Il est possible d'étudier de près Pure Data, afin de voir si – et à quel point – il reflète mes positions sur les logiciels et l'art, mais il y a de grandes chances que ces opinions résultent, au moins en partie, de l'expérience qu'a constituée l'écriture du logiciel. En d'autres termes, c'est sans doute le code source qui a déterminé, dans une certaine mesure, ce que j'en attendais moi-même – ce qui va totalement à l'encontre des procédés de spécification et réalisation tels qu'ils sont mis en œuvre par les ingénieurs en logiciel.

Le critère le plus fondamental dans l'élaboration de Pure Data a sans doute été cette volonté de conserver un caractère généraliste. Sa conception tend vers celle d'un environnement de programmation, bien qu'en comparaison avec des langages de programmation destinés au développement d'applications optimisées et compilées tels que le langage C, elle s'apparente davantage à un langage de script tel que le *shell* Unix (l'interpréteur de commandes d'Unix). De plus, l'idée communément répandue selon laquelle Pure Data était un langage de programmation a amené, par le passé, à le comparer à ceux qui en sont véritablement, ce qui n'a fait que semer le trouble[7]. Si on souhaite décrire Pure Data de façon adéquate, il convient de le considérer comme un mécanisme d'interconnexion, grâce auquel on assemble des applications en configurant de plus petites primitives (des « objets ») telles que des additionneurs ou des oscillateurs. La plupart des utilisateurs s'appuient sur les objets existants, dont environ deux cents sont fournis avec Pure Data, et peuvent en trouver des milliers d'autres n'importe où sur

5. Geoffrey Miller, *The Mating Mind: How Sexual Choice Shaped the Evolution of Human Nature*, Anchor Books, New York, 2000, 528 p.

6. Miller Puckette, « Max at 17 », dans *Computer Music Journal*, n° 26/4, MIT Press journals, Cambridge (MA)/Londres, 2002, p. 31-43.

7. Peter Desain, Henkjan Honing, « Letter to the editor: the mins of Max », dans *Computer Music Journal,* n° 17/2, MIT Press journals, Cambridge (MA)/Londres, 1993, p. 3-11.

Internet. (L'ensemble de primitives fournies dans ce cas est beaucoup plus complet que ce que fournit par exemple le langage de programmation C. La taille et la variété de l'échantillon sont plus ou moins comparables à une API [interface de programmation d'application] telle que OpenGL.) Dans la conception de Pure Data, le souci particulier d'allier équitablement le côté généraliste et la facilité d'utilisation est implicite.

Si Pure Data est aussi extensible, c'est précisément parce qu'il a été conçu comme un outil d'interconnexion – un facteur qui a également contribué à son expansion dans le domaine des arts visuels, ce qu'on n'attendrait pas forcément d'un logiciel conçu pour la création musicale. Le fait qu'il ne s'adresse à la base à aucun medium en particulier (à moins que le temps qui passe ne soit lui-même considéré comme un medium) a facilité l'introduction de packs tels que GEM[8], qui permet aux *patches* Pure Data de faire appel à OpenGL. Pour Pure Data ce n'était pas tant une aubaine que le résultat d'une préparation minutieuse.

Sa conception reflète l'environnement mouvementé de trois laboratoires de création musicale (au sein du MIT, puis de l'IRCAM et enfin de l'Université de Californie à San Diego), pour lesquels j'ai travaillé au cours de ma carrière dans la musique assistée par ordinateur. Tout au long de cette période, j'ai été impliqué dans diverses productions musicales dans le cadre desquelles je travaillais dans l'urgence et en me reposant fortement sur l'utilisation de logiciels, que ce fût en studio ou sur scène. La plupart de ces productions ayant consisté en des performances live sur scène, ma principale préoccupation est devenue depuis ce jour la fiabilité. En tant que développeur de Pure Data je passais également un nombre incalculable d'heures à l'utiliser, ce qui explique son côté dépouillé. Les habillages (*skins*) tout en couleurs et les animations sont des éléments que le programmeur intègre uniquement pour impressionner l'utilisateur non-programmeur. Pour ma part, je n'avais nullement l'envie, ni la possibilité, de m'auto-impressionner en présentant en surface de Pure Data un quelconque élément qui n'aurait pas fait partie intégrante de son noyau dur, tout simplement parce que c'est de ce point de vue que je l'imaginais. J'étais moi-même un représentant de la base d'utilisateurs à laquelle le logiciel devait s'adapter, et c'est pourquoi il invite aujourd'hui chacun à apprécier directement ses fonctionnalités intrinsèques sans fournir davantage d'efforts que ce qui m'avait semblé nécessaire au départ.

Pure Data étant un outil de création musicale, ses caractéristiques mettent en évidence un intérêt profond pour la question du temps et de la causalité. La musique émane de l'instrument à mesure que l'on en joue, et les gestes par lesquels le musicien en fait sortir les sons ne peuvent être compris en dehors du laps de temps qui s'écoule alors que ces gestes se produisent. J'ai conçu Pure Data de sorte qu'il permette de faire réagir

8. Mark Danks, « Real-Time Image and Video Processing in GEM », dans *Proceedings of the 1997 International Computer Music Conference* (Thessalonique), International Computer Music Association, San Francisco, 1997, p. 220-223.

l'ordinateur comme un instrument de musique. C'est cette considération qui l'amène à devoir traiter principalement des flux de données, plutôt que des mécanismes de stockage et d'extraction. (Le stockage de données est en effet l'une de ses faiblesses les plus flagrantes.) La façon particulière dont les données, les contrôles et le temps y interagissent est le résultat de plus d'une décennie de réflexion et d'expérimentation, et c'est sur cette interaction que s'est basé mon travail. Pure Data incarne ainsi l'expression la plus pure que je puisse imaginer de l'idée de réactivité dans un programme informatique.

De nombreuses fonctions considérées comme essentielles dans un langage de programmation moderne ne figurent pas dans Pure Data. L'exemple le plus évident se trouve sans doute au niveau de son espace de nommage « à plat » : c'est un logiciel qui ne comprend pas de portée (qu'elle soit lexicale ou dynamique), pas de notion de « variables locales ». L'omission est intentionnelle, car l'idée de portée serait appropriée dans un environnement encourageant la construction de systèmes vastes et complexes. Les systèmes auxquels Pure Data s'adapte le mieux sont plus réduits, et ont pour but de réaliser des œuvres d'art ou des morceaux de musique isolés. Autre élément commun aux langages de programmation et dont il ne dispose pas : un ensemble standard de mécanismes permettant de créer des boucles, de basculer vers des sous-programmes, et ainsi de suite. Contrairement à la portée et aux variables locales, je n'avais pas du tout l'intention d'omettre ces fonctionnalités : il s'avère tout simplement difficile d'imaginer comment les intégrer naturellement dans un outil graphique tel que Pure Data.

Une autre absence notable est celle de toute forme de sémantique préalable. À la place des tonalités, de la dynamique, des indications rythmiques et de toutes les autres mesures qu'on peut trouver sur un éditeur de notation musicale, Pure Data n'offre que des chiffres et, de temps en temps, une chaîne (*string*), des données qui peuvent aussi bien « déclencher » des actions que les paramétrer. (Il est également possible de déclencher une action sans la moindre donnée en utilisant la commande modestement appelée « bang ».) Ce manque de sémantique préalable rappelle davantage le langage C que le *shell* Unix et constitue l'aspect sur lequel Pure Data atteint son « niveau » le plus bas. Le but était d'éviter de faire une quelconque suggestion à l'utilisateur sur le nombre d'éléments qui devraient entrer en jeu dans un projet musical ou artistique.

Bien que les structures de données et les mécanismes de stockage de Pure Data constituent assurément sa plus grande faiblesse, cela ne signifie pas que la volonté n'était pas là et que ces critères ont été négligés, car le nom du logiciel lui-même reflète mon récent désir de m'attacher à la structure et à la gestion des données (*data*). De même, l'aspect procédural de Pure Data n'est pas foncièrement différent de celui qui régit Max/MSP, et les caractéristiques les plus innovantes du premier sont liées aux données. Pourtant, avant de m'attacher à cette question, il fallait attendre que tout le reste soit définitivement et nécessairement arrêté. Ainsi, le défi a consisté à adapter les approches traditionnelles en matière de gestion des données dans des environnements logiciels à une structure

qui n'était pas fondamentalement tournée vers ces mêmes données. C'est, au niveau de la conception du logiciel, la question qui me préoccupe le plus dans l'immédiat.
Pure Data présente également un défaut majeur, dans la mesure où il s'avère difficile de l'adapter au multitraitement. Ce phénomène était très important dans les années 1980 et au début des années 1990, lorsque les systèmes monoprocesseurs n'étaient pas assez puissants pour traiter de sérieuses applications de musique assistée par ordinateur. Une des premières versions de Max sur lesquelles j'ai travaillé était d'ailleurs explicitement conçue pour le multitraitement. Du point de vue de l'utilisateur, en adapter un *patch* pour pouvoir utiliser plusieurs processeurs efficacement était si compliqué que lorsque des processeurs plus performants et capables de remplacer les multiprocesseurs de l'époque sont arrivés, ce fut un véritable soulagement. De ce fait, dès le début de la conception de Pure Data, je me suis concentré sur les processeurs uniques. Or ces derniers temps les prix en la matière ont tellement baissé que la plupart des nouveaux PC sont équipés au minimum de deux processeurs. Dans le domaine de l'art numérique, l'une des priorités absolues devrait être de trouver un meilleur moyen de tirer parti de ces nouvelles possibilités, mais il semblerait que Pure Data ne soit jamais en mesure d'aborder correctement la question.
L'une des dernières particularités de Pure Data est qu'il est par essence un logiciel « libre », une qualité qui s'avère indissociable de nombre de ses constituants. La facilité avec laquelle il peut être adapté afin de fonctionner sur un PDA ou sur un contrôleur intégré provient en partie du fait que les utilisateurs peuvent ne prendre qu'une partie du logiciel, sans l'utiliser dans son ensemble. Son statut de logiciel libre est une garantie contre le *feature creep*[9], dans la mesure où les utilisateurs peuvent ajouter eux-mêmes les fonctionnalités qu'ils désirent, ce qui fait qu'il n'y a pas d'impératif à les intégrer en amont au cœur du logiciel. Cela permet également d'éviter les modifications et suppressions incompatibles avec la base logicielle, car un utilisateur qui n'a vraiment besoin que de l'ancienne version 0.21 de Pure Data peut simplement se contenter du code source. La frontière entre l'utilisateur et le développeur est abolie, ce qui augmente considérablement la flexibilité.

Ce que j'ai appris de toute cette expérience

Étant donné que je connais désormais quelque chose au développement d'environnements logiciels pour la création artistique, je devrais être bien placé pour commencer à écrire la prochaine bible de la création artistique assistée par ordinateur. Malheureusement, il est fort probable que je ne le fasse jamais. D'abord parce que je dois m'occuper de mettre à jour Pure Data, qui je l'espère continuera d'être compatible avec les *patches* existants pendant encore quelques décennies, afin que les artistes aient au final une interface capable de gérer des œuvres d'art sur la durée. Puis parce que je suis las d'écrire des logiciels, et que j'aimerais faire autre chose (peu importe ce que ça sera, d'ailleurs). Mais la raison la plus importante, c'est que ce que j'ai appris concerne l'écriture de logiciels typiques des années 1980 et 2000. En effet, les logiciels de l'an 2015 seront *a priori* radicalement différents.
La mise au point d'environnements logiciels interactifs pour des créations artistiques en temps réel constituait dans les années 1980 un champ problématique alors grand ouvert, mais aujourd'hui presque clos. Cela ne signifie pas qu'il y ait des solutions idéales ; disons qu'il existe des solutions adéquates, en présence desquelles on peut difficilement parvenir à une solution parfaite. À l'instar du clavier de type 'ERTY, les applications logicielles qui existent à l'heure actuelle, en partant du principe qu'elles peuvent être mises à jour de façon régulière, nous obligeront par leur simple présence à apprendre à les maîtriser, et ce sera cette « base d'utilisateurs » qui les fera perdurer.
On entend souvent dire que les techniques informatiques sont en perpétuelle évolution, mais ce n'est pas totalement le cas. Dans certains nouveaux domaines de recherche, l'évolution et la découverte sont réelles, mais dans d'autres tels que les langages de programmation et la conception de processeurs, l'évolution n'est plus possible du fait de la présence de solutions satisfaisantes. Quelles que soient les avancées qui nous attendent, elles auront lieu dans des domaines nouveaux, inexplorés, et c'est pour cette raison que les enseignements résultant de la conception de Pure Data ne seront que très peu utiles aux personnes qui souhaiteront écrire de nouveaux logiciels. Certaines des leçons exposées dans cet article seront sans doute encore applicables à l'avenir, mais pour l'heure il est impossible de savoir lesquelles, ni même dans quelle mesure les autres n'arriveront pas à s'imposer.
Il est possible au moins de considérer que certaines choses ne seront plus ce qu'elles étaient par le passé. Comme je l'ai dit ci-dessus, la frontière entre « utilisateurs » et « développeurs » est de plus en plus mince et risque de disparaître, ou dans tous les cas d'évoluer. Les modèles de diffusion évoluent également d'une façon qui pourrait bouleverser les possibilités offertes aux artistes. Les transports deviendront de plus en plus chers alors que le prix des communications baissera progressivement. On ne peut prédire l'influence que cela aura sur la vie artistique, mais les effets seront considérables.
Le développement de Pure Data a joué un rôle dans l'apparition d'une tendance qui risque de perdurer : on ne perçoit plus les logiciels comme des entités fixes qui opèrent sur des « documents » modifiables tels que des suites bureautiques sur des feuilles de calcul. La « culture » qui est en jeu s'incarne progressivement dans la fonctionnalité elle-même et de moins en moins dans le document. Les idées qu'on échange se retrouvent moins sous la forme de données passives et davantage dans la capacité de créer, modifier et utiliser les données de façon innovante. La notion de fonctionnalité, telle qu'elle est

9. Le *feature creep* correspond à la prolifération incontrôlée des fonctionnalités d'un logiciel au cours de son développement, répondant à des impératifs marketing ou commerciaux (NdT).

véhiculée par les logiciels, correspond en soi à de l'information, d'un genre qu'il est difficile de définir mais qui reste porteuse d'un potentiel énorme.
La culture se transmettant de plus en plus via la fonctionnalité et non pas via des documents passifs, la domination du paysage par quelques logiciels omniprésents vit ses derniers instants. Alors que certains environnements seront toujours utilisés de par l'« effet 'ERTY », le travail sera de plus en plus souvent effectué en dehors de ces environnements fixes, ou dans les interstices qui subsistent entre eux. La glorieuse époque caractérisée par l'émergence des « environnements » – époque à laquelle on se faisait une réputation en inventant la feuille de calcul ou le système d'exploitation multitâche – touche à sa fin. Sans doute les gestes forts et d'une grande résonance culturelle se font-ils plus rares de nos jours, mais il se peut également qu'ils passent à l'avenir par quelque chose de radicalement différent du logiciel. Celui-ci finira par être aussi intéressant que l'est, par exemple, la voiture d'aujourd'hui : toutes sont différentes, mais on les considère davantage pour leur utilité que pour leur beauté.