Médias de programmation.

***Casey Reas**, in «Code - The Language of our Time», Ars Electronica, 2003.*

La conception de logiciel est un facteur déterminant de notre culture moderne et devient de plus en plus un fondement de notre réalité.
Les citoyens du monde se ressemblent, passant leurs vies face aux surfaces réfléchissantes de leurs mobilophones ou de leurs ordinateurs de bureau.
Leurs esprits et leurs mains opèrent dans un espace contenu entre la réalité et les règles arbitraires des menus, cliquant et étirant des fenêtres.

Les artistes utilisent le logiciel pour commenter nos structures sociales et politiques de plus en plus numériques, et pour défier les acceptations fondamentales du code machine. Indépendamment du contenu ou de l'intention de leur travail, les artistes contemporains expriment leurs idées par l'intermédiaire du logiciel. Dans le bouillonnement continuel des différentes disciplines des arts électroniques (cybernétique, réalité virtuelle, CAVE [1], vie artificielle, Net-art, réalité augmentée), le logiciel fournit les fondations sur laquelle la signification et le contenu sont construits.

Avec la revitalisation du concept de «software art» dans des festivals tels que Transmediale et READ_ME, une discussion critique émerge autour du rôle du logiciel dans notre pratique en matière de culture et d'art.
Ce texte en est une extension et se concentre sur le concept de logiciel comme moyen capable d'expressions inédites, et de langages de programmation comme matériaux doués de propriétés spécifiques.

Qu'est-ce qu'un logiciel ?

Un logiciel est écrit grâce à des langages de programmation, des séquences de caractères alphanumériques et des symboles composés selon des règles syntaxiques rigides [2]. Si vous n'êtes pas un familier des programmes d'ordinateurs, en voici pour référence quelques fragments :

Perl

```
        opendir(DIR, $dir) || die $!;
        @files = readdir(DIR);
        closedir(DIR);
        foreach $file (@files) {
        if($file =~ «.xml») {
                handle(«$dir/$file»);
        }
}
```

[1] NdT: La technologie CAVE est un environnement immersif de réalité virtuelle en 3D.

[2] Il existe des exceptions de langages de programmation appelés «langages de programmation visuels» qui permettent à des structures d'être définies à l'aide de symboles graphiques.

C++

```
                main() {
        int c;
        c = getchar();
        while(c != EOF) {
        putchar(c);
                c = getchar();
        }
}
```

LISP

```
(define (square x)
        (* x x))
(define (sum-of-squares x y)
        (+ (square x) (square y)))
```

Par l'écriture de logiciel, les programmeurs décrivent les structures qui définissent des «processus». Ces structures sont traduites en un code qui est exécuté par une machine et les processus sont interprétés, impliquant activement les composants électroniques de l'ordinateur.
Harold Abelson, informaticien au MIT (Massachusetts Institute of Technology) explique : «Des processus manoeuvrent ces choses abstraites appelées «données». L'évolution d'un processus est dirigée par une succession de règles que l'on appelle un «programme». Les gens créent des programmes pour diriger des processus».

C'est ce processus actif de lecture, de manipulation, et de stockage de données qui offre au logiciel ses capacités uniques.

Le logiciel est un moyen.

Le logiciel a permis une manière de construire un pont entre l'art du passé et des arts électroniques du présent et futur. Comme le souligne Roy Ascott, nous avons établi une transition entre «contenu, objet, perspective, et représentation» et «contexte, processus, immersion, et négociation».

L'aspect le plus singulier du logiciel comme moyen est qu'il permet une réponse. Un tel objet réactif a la capacité d'interagir avec son environnement. Myron Krueger, pionnier de la réalité artificielle, suggère un certain nombre de métaphores intéressantes concernant les interactions entre une personne et un logiciel : dialogue, amplification, écosystème, instrument, jeu, et récit. Je suis pour ma part intéressé au développement d'expressions logicielles plus fondamentales que celles évoquées par Ascott et Krueger. Ces expressions sont à la base du logiciel comme moyen et incluent la

forme dynamique, le geste, le comportement, la simulation, l'auto-organisation, et l'adaptation.

Chaque langage est unique.

Comme il y a beaucoup de différentes langages humaines, il existe beaucoup de différents langages de programmation.
De la même manière que différents concepts peuvent être véhiculés par des langues humaines diverses, différents langages de programmation permettent à des programmeurs d'écrire diverses structures de logiciel.

Comme certaines expressions sont intraduisibles d'une langue humaine à l'autre, des structures de programmation ne peuvent souvent pas être traduites d'un langage machine à un autre. Certains langages de programmation ont été établis spécifiquement pour des applications commerciales (COBOL), d'autres pour l'exploration de l'intelligence artificielle (LISP), ou encore la manipulation de données (Perl), et beaucoup de structures écrites dans ces différents langages ne peuvent être exprimées comme telles par d'autres langages. L'animateur abstrait et programmeur Larry Cuba décrit ainsi son expérience : «Chacun de mes films a été fait sur un système différent en utilisant un langage de programmation différent. Un langage de programmation vous donne la puissance d'exprimer certaines idées, tout en limitant vos capacités à en exprimer d'autres».

Les langages de programmation sont des matériaux.

Il peut être utile d'envisager chaque langage de programmation comme un matériau possédant ses moyens et ses contraintes propres. Les différentes langues sont appropriées selon le contexte. Certains langages sont faciles à employer mais obscurcissent le potentiel de l'ordinateur. D'autres sont très compliqués, mais permettent un contrôle total en offrant un accès complet à la machine. Certains langages de programmation sont flexibles et d'autres sont rigides. Les langages flexibles comme Perl et Lingo sont bons pour créer rapidement des programmes courts, mais ils deviennent souvent difficiles à entretenir et à comprendre quand les programmes deviennent longs.

La programmation à l'aide de langages rigides tels que le Assembly 68008 ou le C exige un soin extrême et une attention pénible jusqu'au moindre détail, mais les résultats sont efficaces et robustes. Autant les bois de sapin et de chêne ont une apparence et une utilisation différente, autant les programmes logiciel écrits en différents langages ont des formes esthétiques distinctes. Par exemple, des programmes semblables écrits en Java ou en Flash présentent de singulières différences qui n'échappent pas aux habitués des deux.

La programmation est exclusive.

Beaucoup de gens perçoivent les programmeurs comme des personnes d'un genre unique, différent de tous les autres.
Une des raisons pour lesquelles la programmation reste coincée dans les limites de ce type de personnalité est que les langages de programmation sont généralement créés par des gens aux esprits semblables.

Il est cependant possible de créer différents genres de langages de programmation impliquant des personnes dont l'esprit est visuel et spatial. Certains langages alternatifs ouvrent l'espace de programmation à des gens qui pensent différemment. LOGO était un de ces langages alternatifs précoces, conçu vers la fin des années 60 par Seymour Papert comme concept de langage pour enfants. Grâce à LOGO, les enfants peuvent programmer beaucoup de différents médias comprenant une tortue-robot et des images graphiques à l'écran. Un exemple contemporain est l'environnement de programmation MAX développé à l'IRCAM par Miller Puckette dans les années 80. Max a généré un enthousiasme auprès de milliers d'artistes qui l'emploient pour créer des applications audiovisuelles et des installations.

Les interfaces graphiques (GUIs - Graphic User Interfaces) ont rendus possible l'utilisation d'un ordinateur à des millions de personnes ; les environnements alternatifs de programmation contribueront à engendrer de nouvelles générations d'artistes créant des logiciels.

Expressions logicielles.

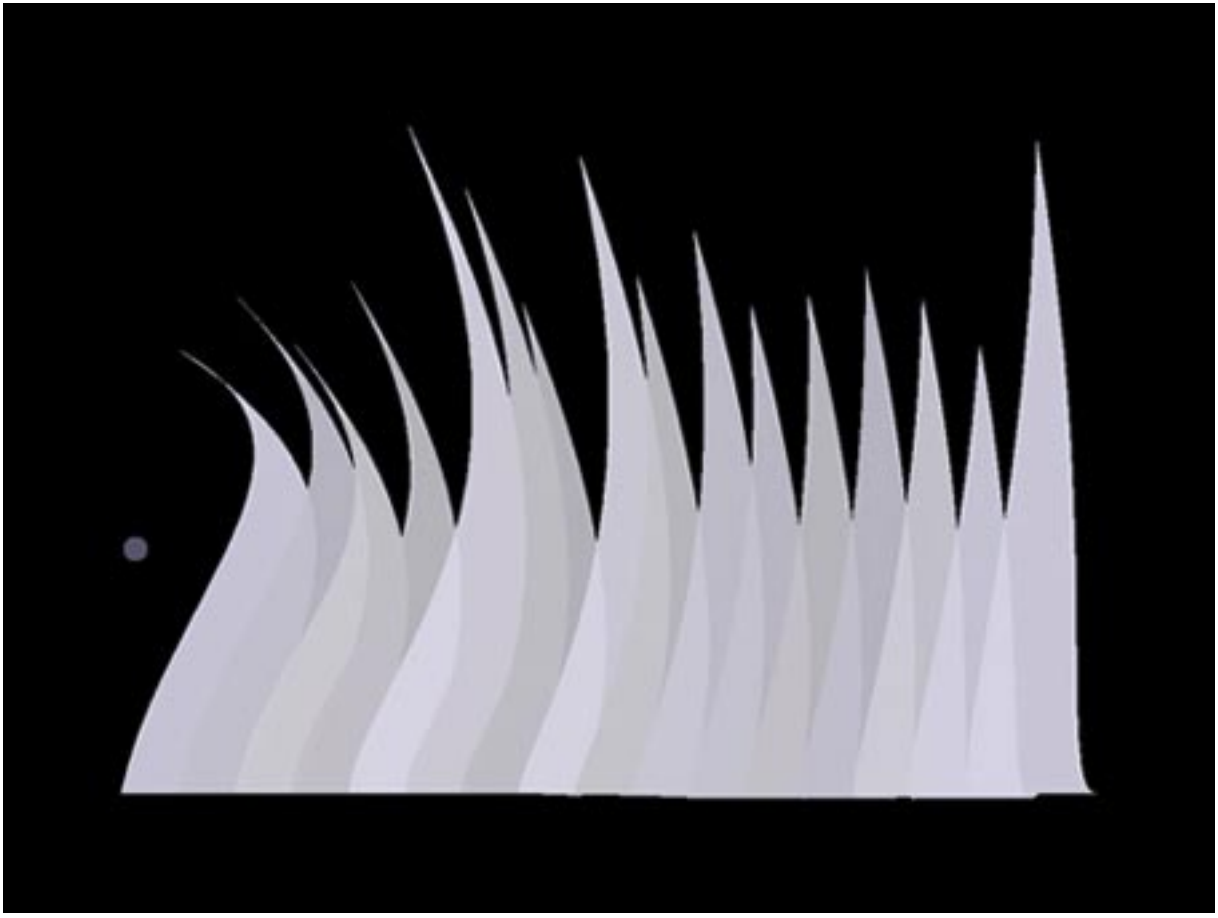
Quand des programmes sont exécutés par une machine, ils sont des processus dynamiques et non des textes statiques à l'écran.
Des noyaux d'expressions numériques incluant forme dynamique, geste, comportement, simulation, auto-organisation, et adaptation émergent de ces processus. De telles expressions, ou d'autres plus basiques, sont les fondements sur lesquels sont élaborées des idées ou des expériences plus complexes. Chacune de ces expressions est décrite ci-dessous et illustrée par un exemple provenant du «Aesthetics & Computation Group» du MIT (Massachusetts Institute of Technology). Ces exemples ont été créés par d'hybrides artistes/programmeurs entre 1998 et 2001 et fournissent des démonstrations claires de ces expressions logicielles.

Forme dynamique.

Une forme dynamique est une forme qui change au bon moment. Si cette forme réagit aux stimuli, elle est réactive. «Scratch» de Jared Schiffman (figure 1) démontre les qualités de base d'une forme dynamique. Dans ce logiciel, la position d'un cercle contrôlable affecte sans interruption la découpe de chaque élément visuel. «Scratch» augmente la communication

visuelle de la forme en ajoutant des couches de mouvement de manière réactive et fluide. En général, la forme peut réagir à n'importe quel signal de son environnement, incluant les dispositifs d'entrée de données habituels comme la souris, un micro ou une camera video, jusqu'à des interfaces plus surprenants comme un capteur de radiations ou encore un sonar.

Figure 1: Jared Schiffman : «Scratch».

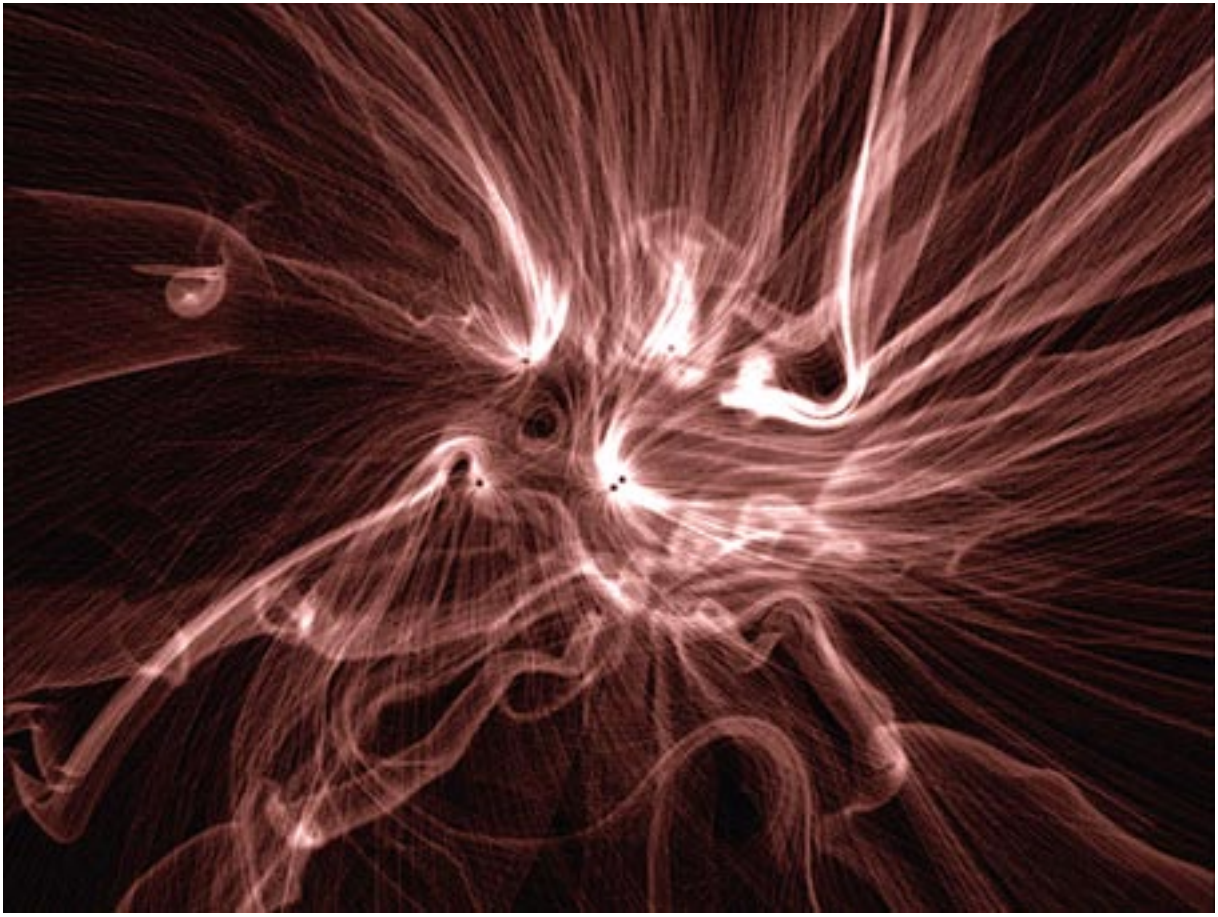


Le geste.

Le geste est un composant important de tout medium continu et le logiciel possède la capacité de transcrire et d'interpréter un geste.
«Aves» de Golan Levin (figure 2) est un ensemble d'applications qui amplifient les gestes de la main en traduisant leurs données en son et en image. Une application transpose la structure de chaque geste en sons qui reflètent leur degré de courbure. Une autre empile les gestes pour créer des textures sonores graduelles qui s'activent et varient selon la présence du curseur.

L'interprétation des gestes est alors plus complexe, mais permet de nouvelles occasions pour provoquer l'interaction. La reconnaissance de l'écriture manuelle est une application d'interprétation de geste. Certaines installations ou jeux-video utilisent une forme plus basique d'identification de geste pour permettre de contrôler l'action avec des mouvements complexes.

Figure 2: Golan Levin: «AVES (AudioVisual Environment Suite)».



Le comportement.

Le comportement est un mouvement possédant l'apparence d'une intention. La combinaison de comportements simples peut suggérer une personnalité ou une humeur. Le comportement peut être créé en écrivant intuitivement des programmes ou en mettant en application des modèles biologiques.

Dans le projet «Trundle» (figure 3), l'objet physique est géré par un programme qui détermine comment il devra se déplacer en présence de stimuli dans son environnement. «Trundle» scrute l'environnement à la recherche de personnes, mais quand il trouve quelqu'un, il tente de s'échapper. «Trundle» est curieux et timide. Une rangée de capteurs sur le corps de «Trundle» surveille continuellement son environnement immédiat et envoient des signaux au microcontrôleur qui détermine comment les moteurs devraient tourner. En général, le comportement peut être employé pour impliquer activement l'esprit par la personnification d'objets, le développement de caractères, la communication d'une humeur, et ajouter une épaisseur psychologique à une oeuvre logicielle.

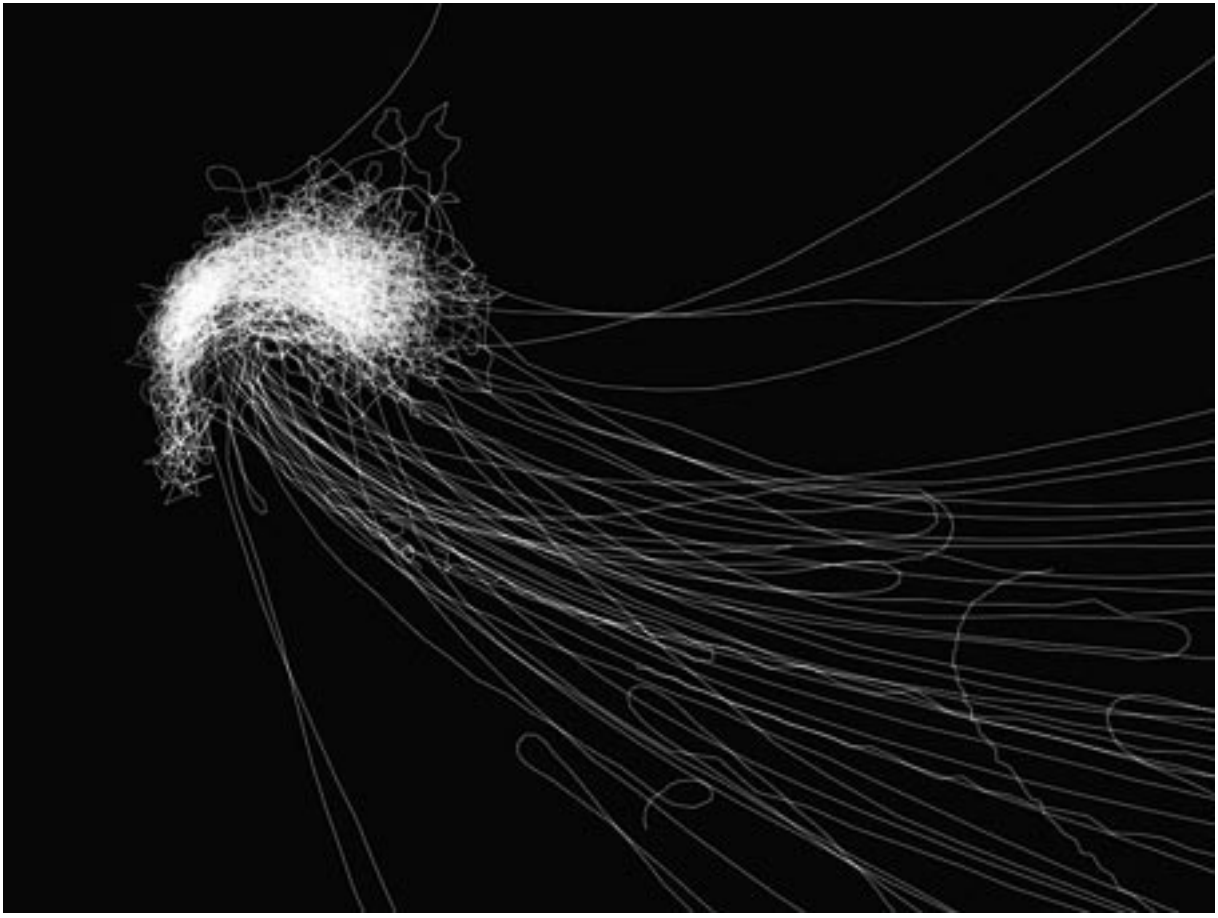
Figure 3: Casey Reas: «Trundle».

La simulation.

La simulation du monde physique fournit un point d'accès facile à la perception des travaux logiciels. Nos sens ont évolué pour répondre aux règles de la nature. Un des premiers jeux d'ordinateur, Pong, était une simulation hautement abstraite du tennis. La technologie moderne et les communautés scientifiques utilisent des modèles inspirés de la réalité comme base de conception d'objets physiques et d'orientation de recherche.

«Floccus» de Golan Levin (figure 4) crée un groupe de lignes élégantes, connectées chacune à une liste de ressorts simulés. Le mouvement ondulant créé par cette simulation simple engendre une crispation chez les spectateurs quand elle est combinée avec la réponse. Les lignes s'étendent et se contractent selon la force, la masse, et l'accélération. Dans un logiciel, la simulation peut dépasser une simple imitation de la perspective, des matériaux, et des lois de la physique - les processus inhérents aux phénomènes naturels peuvent être aussi bien simulés.

Figure 4: Golan Levin: «Floccus».



Auto-Organisation.

La capacité des éléments à s'auto-organiser rend possibles les phénomènes d'émergence. La structure émerge des interactions entre de nombreux processus autonomes. «Valence» de Ben Fry (figure 5), lit le texte d'un livre mot après mot et l'organise dans l'espace selon un système de règles. Un volume complexe émerge des relations entre divers mots dans le texte. De petites entorses aux règles de l'interaction peuvent avoir des effets potentiellement grands sur le traitement de la visualisation.

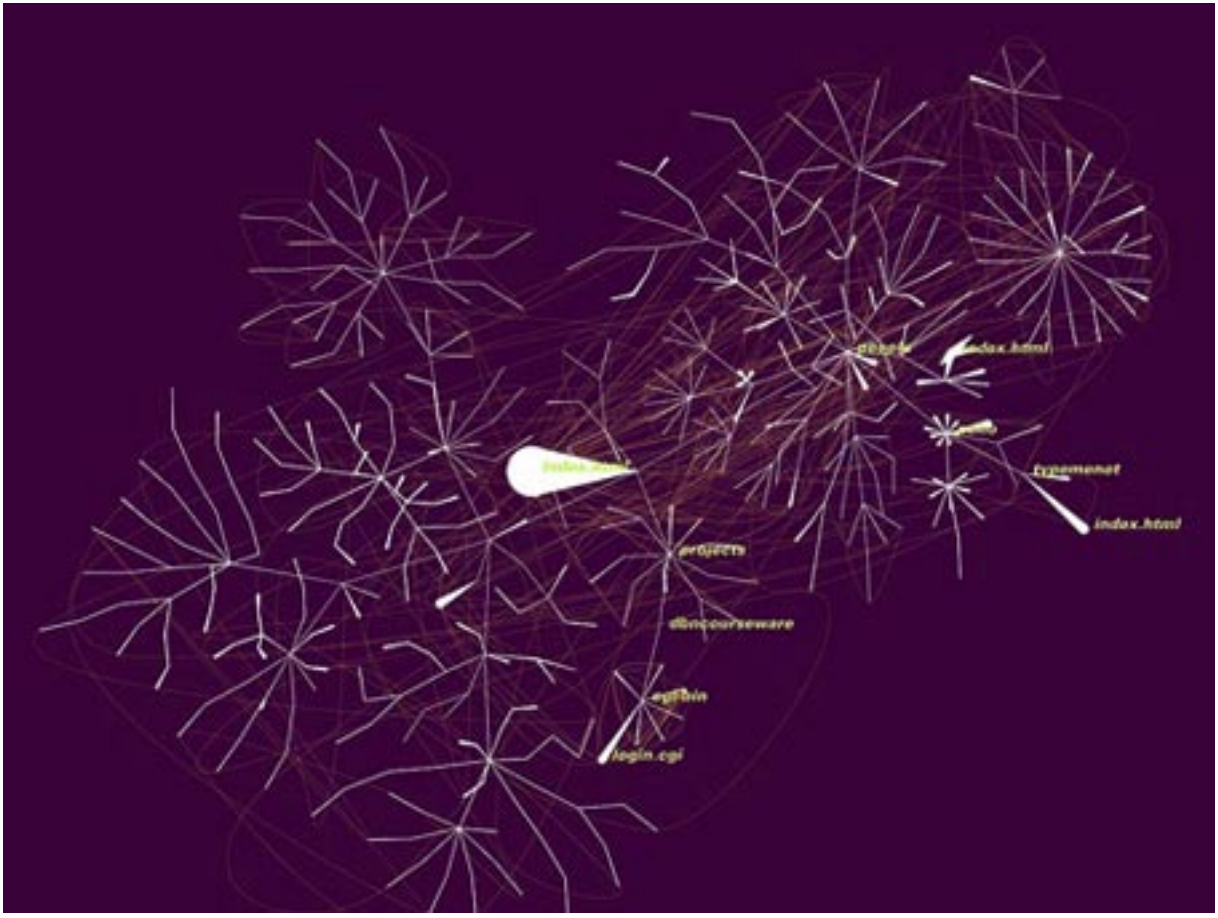
Figure 5: Ben Fry: «Valence».

Adaptation.

L'adaptation est la capacité à changer. Pour que le logiciel s'adapte, il doit posséder une représentation de lui-même et se rendre compte de son contexte. «Anemone» de Ben Fry (figure 6) peut surveiller sa densité et modifier sa structure pour garder l'équilibre.

«Anemone» est la visualisation d'un trafic sur un site web et, au fur et à mesure que les heures et les jours passent, le logiciel enlève des sections de sa masse pour tenir compte du développement de nouvelles sections sans empêcher la lisibilité de l'information. Écrire un logiciel qui s'adapte vraiment à son contexte est un vrai défi et les résultats pertients sont rares. En utilisant un interprète, il est possible pour un programme de s'auto-modifier en cours d'exécution.

Figure 6: Ben Fry: «Anemone».



Programmer.

Bien que les logiciels soit constamment utilisés dans les arts électroniques, des individus choisissent de construire leur code de manière radicalement différentes, allant de l'écriture dans des langages de bas niveau jusqu'à des collaborations avec des programmeurs. Certains artistes utilisent le code comme outil pour créer des oeuvres dans d'autres médias.

Ils emploient des applications disponibles dans le commerce pour produire des impressions, des videos ou pour faire des croquis approximatifs qui sont exécutés dans des médias analogiques. D'autres collaborent avec des programmeurs professionnels, créant des caractéristiques qui sont ensuite mises en application par les programmeurs. Beaucoup d'autres créateurs utilisent des environnements de programmation développés pour des designers et des artistes. Ils travaillent avec des environnements de programmation de scripts et/ou visuels tels que Director, Flash, et Max qui facilitent la construction de logiciels pour les non-programmeurs. Un plus petit groupe d'artistes travaillant le logiciel utilise des langages de programmation développés pour des programmeurs professionnels. Ils utilisent en général des langages comme C, Java, et Peri et développent souvent leurs propres outils-maison pour travailler dans ces environnements. Il n'y a pas UNE seule manière correcte de travailler avec du logiciel.

C'est un choix personnel, une mélange équilibré de contrôle et de simplicité. La maîtrise de la programmation demande beaucoup d'années de travail intensif, mais la compréhension des principes de base est à la portée de chacun. À mon avis, chaque artiste utilisant du code devrait y être instruit.

Que signifie l'instruction dans le contexte du logiciel ?

Alan Kay, innovateur en matière de réflexion sur le code comme matériau, écrit ceci: «La capacité à «lire» un medium signifie l'accès aux matériaux ET aux outils créés par d'autres. Vous devez posséder les deux pour être instruits. Dans l'écriture imprimée, les outils que vous produisez sont rhétoriques; ils démontrent et convainquent. Dans l'écriture pour ordinateur, les outils que vous produisez sont des processus; ils simulent et décident».

Ces processus qui simulent et décident sont l'essence du logiciel et ils peuvent seulement être entièrement compris en les construisant. Les artistes capitalisent en écrivant leur propre logiciel. Avec la croissance du web, la popularité d'environnements de scripts comme Flash, et la chute des prix du matériel, beaucoup plus d'artistes explorent la programmation.Le champ de la programmation audio-visuelle est un excellent exemple de cette tendance. De petites compagnies de logiciel comme Cycle '74 développeur de Jitter, sont très sensibles à leur communauté d'artistes et stimulent le développement d'outils accessibles. Jitter est une bibliothèque sophistiquée

de structures visuelles pour l'intégration de l'image avec le son.

Beaucoup d'artistes ont été bien au-delà de compter sur les développeurs pour leurs outils. Les Pink Twins, un duo de musiciens/programmeurs de Helsinki, ont créé Framestein, un logiciel de calcul video lié à PD (Pure Data), logiciel open-source de traitement en temps réel. Le collectif d'artistes allemand Meso est même allé encore plus loin avec VVVV, une bibliothèque ambitieuse d'outils de synthèse visuelle en temps réel. Certains artistes développent leurs propres outils logiciels, et après une période d'amélioration, choisissent de le faire partager à la communauté.

Synthèse.

Au cours des trente dernières années, des artistes ont développé des pratiques innovantes à l'aide de moyens logiciels, mais ils n'ont seulement exploré qu'une petite gamme des possibilités conceptuelles. Historiquement, les langages et les environnements de programmation ont encouragé une méthodologie spécifique, qui n'a cependant pas engagé la majorité des artistes créant des oeuvres interactives ou logicielles. De nouveaux outils émergent et encouragent les artistes à commencer à travailler directement avec le médium logiciel. La prolifération de l'instruction au logiciel parmi les artistes augmentera une meilleure utilisation du logiciel et contribuera à de nouvelles formes de logiciels et d'environnements de développement. Ces matériaux et environnements ont le potentiel d'étendre la création de logiciel à une communauté créative et critique encore plus vaste.

Casey Reas,
«Programming media», in «Code - The Language of our Time», Ars Electronica, 2003.
(Traduit de l'anglais par Marc Wathieu).

- *http://www.aec.at/en/archiv_files/20031/FE_2003_reas_en.pdf*

Bibliographie :

Abelson, Harold, Gerald Sussman, and Julie Sussman :
«Structure and Interpretation of Computer Programs».
MIT Press, Cambridge, MA. 1985

Ascott, Roy :
«Moist Ontology».
Published in The Art of Programming.
Sonics Acts Press, Amsterdam. 2002

Cuba, Larry :
«Calculated Movements».
Published in Prix Ars EIectronica Edition '87: Meisterwerke der Computerkunst.
Verlag H.S. Sauer. 1987

Kay, Alan :
«User Interface: A Personal View».
In «The Art of Human-Computer Interface Design»,
edited by Brenda Laurel Addison-Wesley Publishing Co, Reading MA. 1989.

Krueger, Myron : «Responsive Environments».
Published in «Multimedia, From Wagner to Virtual Reality».
Edited by Randall Packer and Ken Jordan. W.W Norton & Co, Inc., New York. 2001.

Casey Reas *est artiste, co-créateur (avec Ben Fry) du logiciel Processing et professeur à UCLA (Los Angeles). Il est représenté par la galerie Bitforms à New-York.*