La programmation et l'art

Merwan Achibet — Florent Marchand de Kerchove

Licence 3 Informatique
UFR des Sciences et Techniques
Université du Havre

RÉSUMÉ. *L'art côtoie l'humanité depuis des milliers d'années et a su s'adapter aux spécificités de chaque époque. Des peintures rupestres aux films d'auteur, ses multiples supports témoignent de son universalité. À l'âge du multimédia, quelles sont les possibilités qu'offre la programmation à la création artistique ? Nous explorons principalement deux voies modernes d'expression créative : le langage Processing destiné aux artistes intéressés par l'interactivité numérique, et la demoscene, sous-culture du graffiti digital.*

ABSTRACT. *Art has been evolving alongside humanity for thousands of years and has adapted to the specificities of each era. From rupestrian paintings to independent movies, its numerous media are a testimony of its universality. In the multimedia age, what are the possibilities offered by computer programming to artistic creation ? We primarily focus on two modern ways of creative expression : the Processing language designed for artists interested in numerical interactivity, and the demoscene, a digital graffiti subculture.*

MOTS-CLÉS : *Programmation, art, Processing, demoscene.*

KEYWORDS: *Programming, art, Processing, demoscene.*

Introduction

La programmation informatique est le plus souvent considérée comme un travail d'ingénierie, nécessitant des connaissances avancées dans des domaines formels tels que les mathématiques, la logique et l'algorithmique. Des disciplines qui, dans la conscience populaire, sont rarement associées à la créativité ; bien au contraire ! Elles ont tendance à évoquer la rigidité, personnifiée par un enseignant austère en blouse et barbe grises, ressassant les prouesses de Thalès sans cesse, espérant ainsi enseigner à une classe assoupie. Et pourtant ! L'histoire de l'art contient des exemples d'intersection avec les mathématiques. Les œuvres de M.C. Escher en font probablement l'éminent représentant, mais on peut sans hésiter rappeler les travaux de Mandelbrot et Julia sur les fractales, ou ceux de de Vinci. Il n'y a donc pas de strictes frontières entre ces domaines à priori isolés. Par héritage, la programmation également s'autorise des échappées créatives, et nous en décrirons quelques occurrences.

L'avènement de l'informatique, et l'ubiquité des ordinateurs personnels résultante, ont changé le mode opératoire de nombreuses professions, artistes compris. Outre les expositions présentant des œuvres commentaires sur cette omniprésence, l'ordinateur est devenu pour beaucoup un outil. Dans le but de faciliter son utilisation à des fins artistique par des individus dont la formation est souvent perpendiculaire à celle d'un ingénieur, des programmes adaptés sont nécessaires. Le projet Design by Numbers de John Maeda, ainsi que son héritier le langage Processing, répondent à ces besoins. Nous en présenterons quelques utilisations.

Processing est un formidable programme pour des artistes qui ne portent pas un grand intérêt aux mécanismes de la machine, et davantage aux résultats. D'autres se délectent précisément de ces rouages ; ils explorent tous les détails techniques oubliés par la documentation officielle, et partagent leurs découvertes avec les autres membres d'une communauté appelée *demoscene*. Leurs productions sont, pour le non-initié, assimilables à des clips musicaux, mais elles diffèrent cependant en deux points majeurs : ce sont de véritables programmes non interactifs exécutés en temps réel, et la plupart de leur contenu est créé à partir de code et de formules mathématiques. Nous détaillerons cette communauté ainsi que certaines productions mémorables.

Enfin, nous aborderons une dernière occurrence de vagabondage de la programmation dans le territoire de l'art avec l'expérience intéressante que constitue le langage Piet.

1. Processing : la programmation artistique

1.1. *Le projet Design by Numbers*

Le projet Design by Numbers, issu du Media Lab du prestigieux Massachusetts Institute of Technology (MIT), voit le jour en 1999 sous la tutelle de John Maeda. Chercheur en informatique et designer, Maeda fusionne les deux univers et choisit d'orienter ses recherches vers l'art généré par ordinateur. Son objectif n'est pas d'ima-

giner de nouvelles méthodes d'art assisté par ordinateur telles qu'il en existent déjà de nombreuses, souvent liées à l'utilisation de logiciels spécialisés, mais de faciliter l'approche de la programmation aux artistes, et ainsi de leur permettre de travailler à un niveau plus bas, plus proche de la machine. Cette approche, bien qu'étant plus complexe à maîtriser, offre un large éventail de possibilités à l'artiste qui prendra la peine d'étudier le langage. Son objectif principal, en développant Design by Numbers, est de fournir un langage simple, permettant de démystifier l'outil informatique et de le faire découvrir aux artistes bricoleurs, mais aussi aux débutants en programmation.

En réalité, Design by Numbers n'est pas uniquement un langage de programmation. Il s'agit d'un environnement de création complet comprenant son langage propre mais aussi une visualisation des résultats générés intégrée, cela dans le but de centraliser les outils et donc de simplifier au maximum l'expérience de l'utilisateur (Maeda, 2001). L'artiste peut se concentrer sur son travail et ignorer les contraintes auxquelles pourrait faire face un programmeur classique.

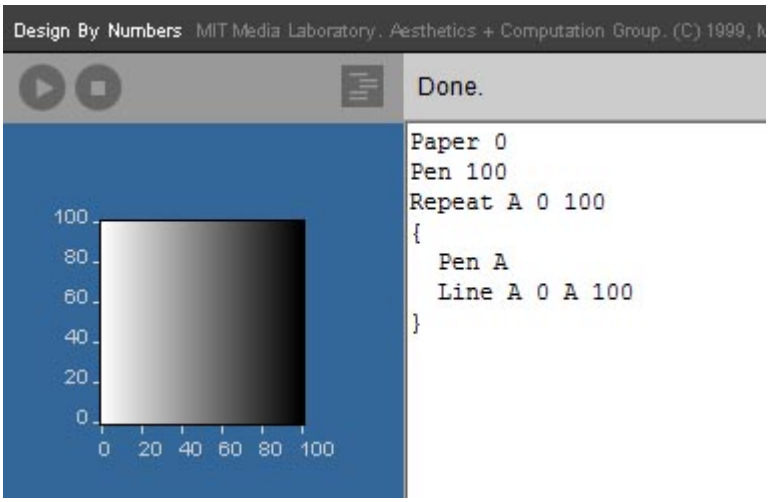


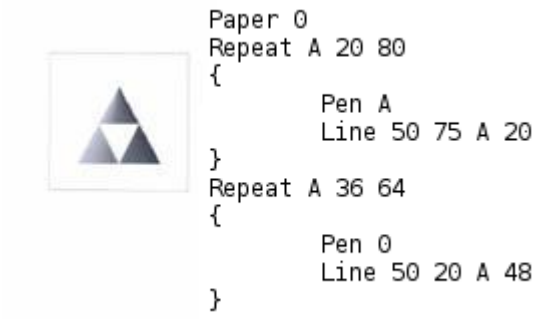
Figure 1. *L'interface de Design by Numbers.* [1]

L'analyse de tout extrait de code tiré d'un projet utilisant Design by Numbers illustre parfaitement cette idée de simplicité. Prenons par exemple le choix des couleurs. Dans d'autres langages, un utilisateur souhaitant dessiner d'une couleur particulière doit souvent préciser les pourcentages en rouge, vert et bleu de la couleur en question, et parfois d'autres informations, potentiellement obscures aux yeux d'un novice, telles que l'opacité de la couleur ou des informations concernant la surface qui accueillera le dessin. Au contraire, dans Design by Numbers, la tâche est simplifiée au possible et l'utilisateur détermine uniquement la couleur de son crayon par l'instruction `Pen x`, où `x` représente un nombre entier compris entre 0 et 100,

1. Image tirée de : `http://processing101.noisepages.com/2009/09/hello-world/`

0 représentant le blanc et 100 le noir. Toute valeur comprise entre ces deux limites correspondra à un gris proportionnellement concentré. L'instruction `Pen 50`, par exemple, permet à l'utilisateur de dessiner en gris. Une instruction équivalente est utilisée pour préciser la couleur du papier utilisé (la couleur de fond de l'image générée). `Paper 0`, par exemple, produira une image à fond blanc. On peut comparer la syntaxe de Design by Numbers avec celle du langage C associé à la bibliothèque SDL qui, pour obtenir le même résultat, nécessiterait l'instruction suivante, clairement moins appréhendable par un non-initié : `SDL_FillRect(screen, NULL, SDL_MapRGB(screen->format, 0, 0, 0))`. Une fois les couleurs déterminées, l'usager peut dessiner de façon tout aussi simple avec la fonction `Line x1 y1 x2 y2` qui, comme son nom le laisse supposer, trace une ligne entre le point de coordonnées `(x1, y1)` et le point de coordonnées `(x2, y2)`.

La figure 2 nous montre qu'il ne suffit que de quelques instructions basiques pour obtenir un résultat tangible. Dans cet exemple, Un grand triangle est tracé en dégradé (la couleur du stylo est modifiée à chaque tour de la boucle Repeat). Puis un triangle inversé plus petit est tracé en blanc au centre de l'image, créant ainsi l'illusion de trois triangles empilés. Bien que cette illustration ne soit pas esthétiquement exceptionnelle, elle représente à elle seule les capacités et la sobriété du langage.



```
Paper 0
Repeat A 20 80
{
        Pen A
        Line 50 75 A 20
}
Repeat A 36 64
{
        Pen 0
        Line 50 20 A 48
}
```

Figure 2. *Génération d'une pyramide de triangles en sept lignes de code. (Maeda, 2001)*

Bien sûr, cette simplicité d'utilisation a un prix, il n'est pas possible de dessiner en couleurs, les résultats générés restent en deux dimensions, et le potentiel interactif est fortement bridé. Un programmeur exigeant ne pourra se satisfaire uniquement des capacités limitées de Design by Numbers, c'est alors vers le langage Processing qu'il se tournera.

1.2. *Le langage Processing*

Le rapprochement entre programmation et art n'a pas toujours été évident. Longtemps, les programmeurs artistes durent se contenter des outils classiques de l'infor-

matique, parfois complexes, souvent obscurs, et qui présentaient la majeure partie du temps un obstacle de taille à la libération de leur créativité. Processing a permis de renverser la tendance en proposant un langage de haut niveau permettant de créer simplement des rendus graphiques très riches. Ce langage, dérivé de Java, est né des mains de Ben Fry et Casey Reas en 2001. C'est le successeur du projet Design by Numbers mené par John Maeda quelques années auparavant au sein du même laboratoire, alors que Reas et Fry n'étaient encore que ses étudiants.

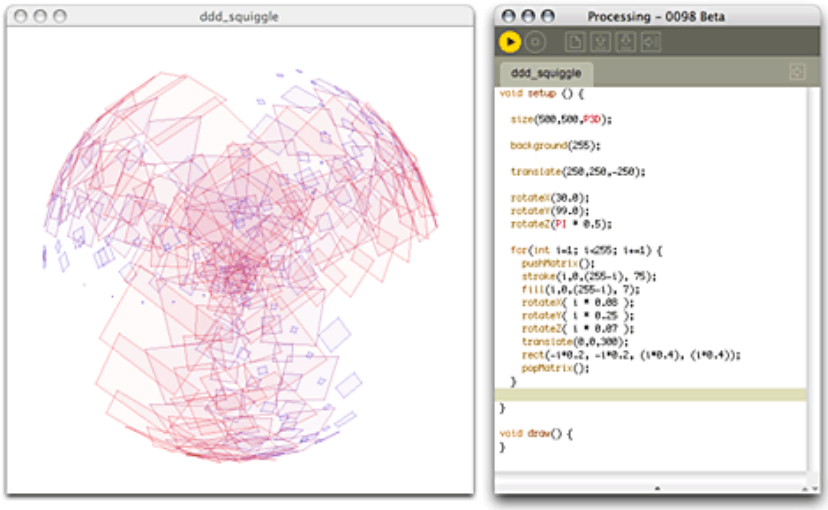


Figure 3. *L'interface de Processing.* [1]

À l'origine, le but de Processing était de faciliter la réalisation de prototypes en fournissant des instructions simples permettant de construire des affichages riches. Les programmeurs ne se souciaient plus ainsi des détails inhérents à la création d'une interface graphique et pouvaient se concentrer sur le cœur de leur projet. Néanmoins, le langage prit une autre direction suite aux réactions accueillantes de la communauté artistique et au fil des versions, les possibilités créatives ont augmenté. Aujourd'hui, Processing est compatible avec de grandes bibliothèques telles que OpenGL et peut générer du son, des scènes en trois dimensions, des œuvres interactives et autorise une liberté artistique complète. Des centaines de créations originales sont regroupées sur des sites internet communautaires tels que openprocessing.org et des artistes reconnus tels que Casey Reas (l'un des créateurs du langage) exposent leurs travaux dans des galeries d'art (figure 4), preuve que l'utilisation de la programmation est aujourd'hui totalement acceptée par le monde artistique et qu'elle ne se cantonne plus uniquement à la sphère virtuelle.

1. Image tirée de : `http://www.ecole-art-aix.fr/article408.html`

Figure 4. *Exposition de Casey Reas au Block Museum of Art de Chicago. (Reas, 2004)*

Le succès de ce langage n'est pas étranger à sa remarquable simplicité d'utilisation. Digne successeur de Design by Numbers, il conserve une syntaxe accueillante mais offre une approche plus conforme à celle qu'un programmeur classique pourrait expérimenter. Il s'agit en fait une surcouche du langage Java et, bien que les fonctions de dessin propres à Processing soient étudiées pour pouvoir être employées de façon intuitive, la syntaxe générale du langage reste inchangée (Shiffman, 2008). Avec Design by Numbers, la simplicité était aussi de mise, cela dans le but de permettre au plus grand nombre d'expérimenter sans avoir à passer des heures la tête plongée dans des livres de programmation. Bien que cette caractéristique soit liée à de nobles intentions, le programmeur avait tendance à se retrouver prisonnier de la simplicité et ses possibilités devenaient bien vite limitées. Grâce à l'utilisation de Java, l'usager ayant fait le tour des fonctions de base de Processing pourra se lancer dans un apprentissage plus poussé de son langage père. Java est actuellement l'un des langages les plus utilisés dans le monde et son spectre d'utilisation est très large, notamment grâce à une API et à des bibliothèques complètes et variées. Processing n'a donc pas de limite.

Body Navigation, du danois Ole Kristensen, illustre l'étendue des capacités de Processing (Kristensen, 2008). L'artiste a choisi de mélanger deux univers relativement éloignés, la danse contemporaine et la programmation, pour un résultat étonnant. Deux danseurs évoluent dans un univers dynamique qui s'adapte à leurs actions. Un projecteur accroché en hauteur projette l'image du sol sur lequel évoluent les danseurs. Leurs mouvements sont suivis grâce à des capteurs infrarouges et analysés par un programme écrit en Processing qui est chargé de modifier l'environnement en conséquence. Le premier artiste a avoir utilisé ce type de capteurs à des fins artistiques est

Merce Cunningham, pour Hand-drawn Spaces en 1997 (Bardiot, 2009). Un tel niveau d'interactivité ouvre de nouveaux horizons aux artistes prêts à tenter l'expérience de la programmation car chaque représentation est unique et imprévisible. Peut-on imaginer un ballet dont la musique changerait et serait composée instantanément en fonction des pas des danseurs ? La programmation permet d'ajouter cette notion d'incertitude et de conditionner l'inconnu.

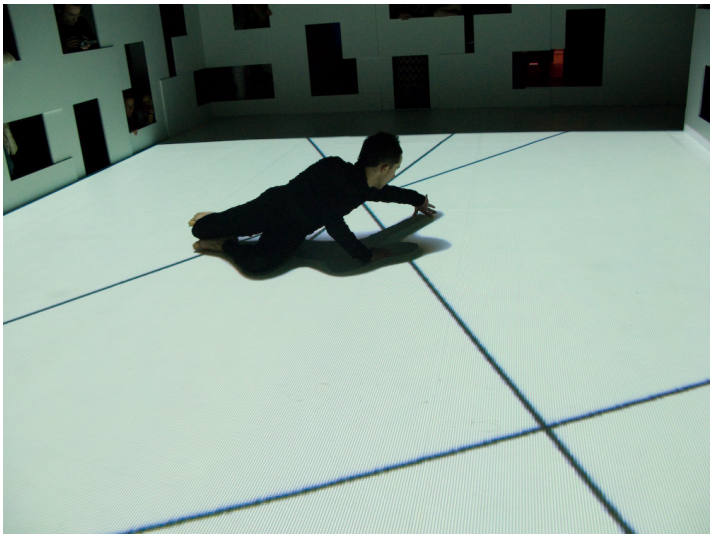


Figure 5. *Body Navigation (Kristensen, 2008).*

De nombreuses universités et écoles d'ingénieurs en informatique enseignent Processing en guise de cours d'introduction à la programmation, puisqu'il permet de faire abstraction des notions les plus complexes qui auraient tendance à faire fuir les débutants. Mais on retrouve aussi des cours de Processing dans certains écoles d'art, notamment l'École Supérieure d'Art du Havre, dans laquelle des étudiants artistes peuvent utiliser la programmation comme support de leur inspiration.

Suite au succès de Processing, plusieurs projets dérivés ont vu le jour, notamment Fritsing et Arduino. Ces variantes sont orientées programmation matérielle et permettent aux artistes qui désirent faire sortir leurs œuvres de la sphère informatique de travailler directement sur des puces électroniques pour, par exemple, créer des automates ou des installations interactives. Bien sûr, ces langages dérivés conservent la simplicité et l'efficacité qui font la force de Processing.

2. Demoscene : l'art de programmer

2.1. *Les origines*

Les programmeurs de la demoscene n'approuveraient certainement pas l'utilisation de Processing dans une de leur production. Les facilités apportées par ce langage masquent en effet, sous plusieurs couches d'abstraction successives, les savoureux détails dont ils se délectent. Ces *sceners*, comme ils sont appelés, préfèrent travailler au plus près de la machine, où le contrôle sur tous les composants est quasi total, et où les fins connaisseurs des aspects les plus obscurs du matériel brillent. Ou du moins, c'était le cas au siècle dernier.

En 1977, Apple révolutionne le marché de l'ordinateur personnel avec l'Apple II. De nombreuses vocations sont nées en passant de longues heures devant ses six couleurs. L'Apple II a également lancé l'industrie du jeu vidéo sur ordinateur, et bon nombre de grands noms de ce domaine ont produit leurs premiers opus sur cette machine. Tamás Polgár (Polgár, 2005) explique que ces jeux sont à l'origine des premiers écrans de crack. Les excellentes capacités de l'Apple II permettaient aux individus à la morale discutable qui déverrouillaient les protections logicielles d'ajouter un court écran composé de texte qui présentait leur groupe et vantait leurs mérites. Ces crackers agissent au sein de groupes, et sont principalement motivés par le défi posé par les protections, et par la reconnaissance au sein de leur communauté (Borzyskowski, n.d.). Les différents groupes sont en compétition permanente : le premier à défaire une protection prouve qu'il est supérieur aux autres, car ses membres sont plus malins. Pour asseoir cette prévalence, ils introduisent donc dans les logiciels cet écran de texte, appelé *intro*, qui est affiché au lancement du programme cracké.

Figure 6. *Cracktro de Duck Tales sur Amiga par Fairlight (1990).* [1]

1. Tiré de `http://flashtro.com/page.php?id=31`

Rapidement, les écrans statiques des groupes se ressemblent, et il faut trouver un nouveau moyen d'affirmer sa supériorité. De plus, ces écrans faisaient certainement pâle figure face aux programmes qu'ils précédaient, ceux-ci étant le plus souvent des jeux vidéos qui regorgeaient d'effets spéciaux graphiques et sonores. Les crackers souhaitent alors dynamiser leurs intros, mais ce sont pour la grande majorité des jeunes de 16 à 22 ans, certes doués en programmation mais pas en animation. Ils se servent donc d'effets graphiques générés à partir des formules mathématiques, et d'algorithmes astucieux, ne nécessitant pas un sens esthétique bien affirmé.

Dès 1985, le Commodore 64 domine le marché de l'ordinateur personnel dans les chaumières. C'est donc naturellement sur cette machine, qui comporte le plus grand catalogue de jeux, que les groupes de crackers prolifèrent. Ses capacités techniques confèrent au sceners une plus grande liberté d'expression : les effets graphiques se diversifient, et les egos s'affirment. La surenchère compétitive entraîne chaque groupe à surpasser les productions des autres, les poussant à exploiter toutes les possibilités des processeurs génériques et graphiques des machines pour produire un effet encore plus impressionnant, jusqu'à ce qu'un autre groupe le décortique, le copie, et enfin le détrône. Un exemple d'une telle surenchère est la « guerre des blobs » (Green, 1995). Les blobs, pour *blitter objects*, sont des éléments graphiques affichés à l'écran par l'intermédiaire du *blitter coprocessor* sur le Commodore Amiga (la seconde plate-forme de choix des crackers). Les intros comportaient parfois plusieurs de ces objets simultanément à l'écran, dans les limites de capacité de calcul et de mémoire de l'Amiga. La surenchère consistait alors à réussir à trouver des optimisations afin d'afficher le plus grand nombre de ces blobs à l'écran. Cette guerre pris fin lorsqu'un groupe réussi à en afficher une infinité, à l'aide de quelques buffers et de la persistance rétinienne. Ce dénouement permet d'illustrer une des caractéristique des ces productions : l'approximation des formules physiques ou mathématiques utilisées est souvent de rigueur. L'exactitude des résultats n'est pas aussi importante que dans une réalisation scientifique, surtout lorsque cette exactitude coûte des cycles processeur qui pourraient permettre l'inclusion d'un effet supplémentaire, et ainsi améliorer le rendu final de la production.

Vers la fin des années 1980, les créateurs d'intros se dissocient petit à petit des crackers. Ils peuvent ainsi se consacrer à des productions purement artistique, et entièrement légales. Les intros deviennent des collections d'effets appelées démos : la demoscene est née.

2.2. *L'évolution*

Le début des années 1990 voit deux tendances émerger dans le monde des démos. La première met en avant l'excellence technique : les démos qui s'y rattachent sont constituées d'une succession d'effets, sans transitions ou presque, et dénuées de ligne conductrice ou de message. Néanmoins, les techniques qui mettent en œuvre les effets présentés ne sont pas anodines, et requièrent des connaissances avancées en mathématiques et en programmation de la machine cible. Ces démos techniques sont

réservées à un public d'initiés : il faut avoir connaissance des capacités de l'Amiga pour apprécier la plupart des scènes de *Desert Dream* (Kefrens, 1993), *Enigma* (Phenomena, 1991), ou *Hardwired* (Crionics et The Silents, 1991).

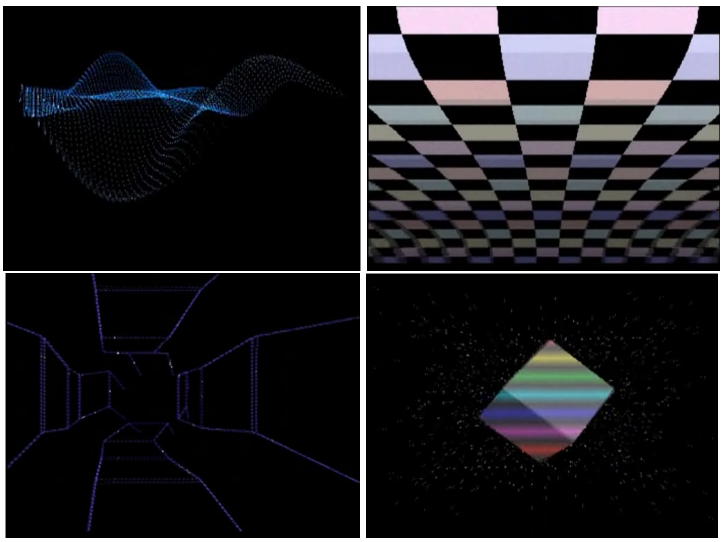


Figure 7. *Scènes tirées de* Desert Dream *(Kefrens, 1993).*

La seconde tendance regroupe les démos dans lesquelles le design et l'esthétique globale passent au premier plan. Les programmeurs sont toujours présents, mais ils fournissent moins d'efforts pour produire des effets temps réel spectaculaires. Les productions qui suivent cette tendance sont plus cohérentes, et plus courtes (environ 4 minutes) que les démos techniques (souvent plus de 10 minutes). Elles ont l'avantage d'être appréciables par tout le monde, jugées sur des critères esthétiques applicables à des clips musicaux. Cependant, ce type de démo polarise la communauté : d'un côté, les anciens de la demoscene, programmeurs et adeptes des démos techniques, de l'autre, un public plus jeune, plus consommateur que producteur, et moins versé dans les arcanes de la technique. *State of the Art* (Spaceballs, 1992), et *Baygon* (Melon Dezign, 1995) sont deux œuvres qui illustrent cette tendance.

Ultimement, c'est le compromis qui l'emporte. Il y a deux raisons à cela : la domination du PC vers 1995 rend les contraintes matérielles obsolètes, et les démos techniques finissent par lasser le public. À partir de cette période, les sceners apprennent à mêler effets et design, et à manipuler les transitions pour des productions plus unies. À ce moment, en partie à cause de ce changement d'orientation, et en partie parce que la plupart des membres qui avaient découvert la demoscene à ses débuts ont dix ans de plus, et d'autres préoccupations, la population de la communauté décline. Certains diront même que l'âge d'or est terminé. Les grands groupes demeurent, mais leurs membres changent. Il y a peu de nouveaux groupes car l'intérêt pour le mouvement diminue, et les connaissances demandées pour produire une démo respectable

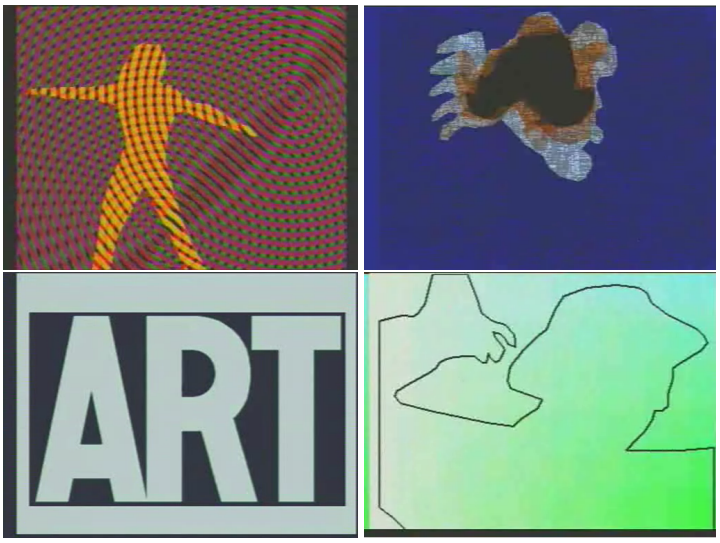


Figure 8. *Extraits de* State of the Art *(Spaceballs, 1992).*

augmentent. Pourtant, les démos gagnent en qualité, sans sacrifier la technicité. Les *sinescrollers*, *star fields*, *rotozoomers* laissent leur place aux techniques 3D de *raytracing*, à l'*ambient occlusion* et au contenu procédural.

Les démos de cette dernière décennie sont bien plus appréhendables par des non-initiés, et c'est probablement une des causes de la survie de la demoscene aujourd'hui. Le mode de diffusion des productions a évolué depuis les débuts, mais il est toujours principalement axé autour des *demoparties*. Ce sont des rassemblements, allant de plusieurs centaines à quelques milliers de personnes, destinés aux sceners. Ces *parties* se trouvent dans toute l'Europe, le territoire principal de la demoscene, et durant toute l'année ; une cinquantaine ont lieu par an. Durant un week-end, tous ces passionnés ont l'occasion de rencontrer les autres membres de la communauté en chair et en os. Mais le point fort de ces évènements est probablement la compétition.

Les groupes présents peuvent faire entrer leurs productions inédites dans une des nombreuses catégories pour les soumettre au jury et au public. Les plus populaires par catégorie remportent un prix, souvent en monnaie (jusqu'à 7000 euros à Assembly 2009 en Finlande). Les catégories séparent les démos essentiellement selon la taille de leur exécutable : 64ko, 40ko, 4ko, 1ko, et même 256 octets. Ces limites de taille sont héritées des débuts de la demoscene, et permettent d'ajouter des contraintes techniques même sur le PC. Grâce à ces tailles très restreintes pour un exécutable moderne, les programmeurs de démos sont passés experts en compression et en génération de contenu procédural. Les autres catégories peuvent concerner d'autres plate-formes que le PC : l'Amiga, l'Atari ST et le Commodore 64 reçoivent encore des démos ; les télé-

phones portables sont également une plate-forme de choix en raison de leurs capacités modestes.

L'autre mode de diffusion est bien sûr Internet et son ancêtre le *Bulletin Board System* ou BBS. Les groupes de crackers se servaient des BBS pour permettre à tout le monde d'y télécharger les copies des logiciels piratés qu'ils hébergeaient, et donc leurs intros. Internet a la même fonction : après la projection d'une nouvelle démo dans une *party*, elle est mise à disposition sur le site de celle-ci, mais aussi sur des sites communautaires et de partage comme pouët.net (Mandarine et scene.org, 2000). D'autres sites, comme Demoscene.tv (Association pour le développement de l'art numérique, 2005) et Capped.TV (Dolphins, 2008) proposent des streaming vidéos des démos les plus célèbres, pour permettre une visualisation rapide, et pour les proposer à un plus large public. Les démos étant, rappelons le, des programmes, elles sont souvent spécifiques à une machine, voire à un système d'exploitation (le plus populaire étant Windows). Certaines démos sont portées sur d'autres architectures, mais elles sont rares.

L'évolution des ordinateurs personnels a grandement complexifié leur mode opératoire. Il n'était pas rare pour un programmeur de démo sur Commodore 64 de connaître exactement quels effets une instruction assembleur aura sur les registres, et sur l'état du matériel. Aujourd'hui, ce matériel est bien plus complexe, et paradoxalement, le programmeur travaillant à plusieurs niveaux d'abstraction au dessus des composants, son travail est simplifié. En revanche, sa connaissance des détails s'en trouve amoindrie. La plupart des programmeurs de la demoscene utilisent désormais le C, C++ ou même le Java pour leurs productions, et occasionnellement un morceau critique d'une boucle est réécrit en assembleur. Néanmoins, les productions récentes sont les plus appréciées par le public contemporain : 23 démos sur les 25 les plus populaires sur pouët.net sont datées d'après 2000.

2.3. *L'état de l'art*

Nous allons maintenant étudier plus en détail deux productions. La première est *Heaven 7* (Exceed, 2000a), gagnante de la catégorie PC 64k à Mekka & Symposium en avril 2000. Il est peut être nécessaire d'insister sur l'importance de la contrainte de taille pour se rendre compte de l'effort fourni. La taille du code source de cet article pèse 42ko, les captures de la figure 9 prennent 73ko et, ironiquement, l'enregistrement vidéo de l'exécution de la démo dépasse les 64000ko ! Pourtant, cette démo parvient à produire près de 3 minutes de contenu dans l'équivalent en octets de 3 secondes de musique compressée au format MP3.

Le *making of* (Exceed, 2000b) nous révèle quelques effets de programmation 3D utilisés : *raytracing* temps réel, flou radial, soustraction de sphères, rayons 3D en rotation, réflections de sphères à surface miroir, surfaces texturées, textures à effet plasma, ombres mouvantes, particules en post-traitement (*post-processing*), et *bump mapping* animé. Tous les effets 3D de cette démo sont *raytracés* en temps réel par

Figure 9. *Scènes de* Heaven 7 *(Exceed, 2000a).*

un algorithme composé spécialement pour l'occasion. Grâce à l'utilisation de sous-échantillonage adaptatif (*adaptive sub-sampling*), tous les pixels n'ont pas à être calculés : les valeurs des moins décisifs sont interpolées. Mais l'astuce principale, utilisée dans la plupart des démos à taille limitée, est la génération procédurale des textures. Au lieu de conserver les textures dessinées à la main par un graphiste dans un fichier de type image, très coûteux en espace disque, seules quelques formules de création et de transformation sont stockées, et les textures finales sont générées en temps réel lors de l'exécution de la démo (souvent dans une étape de chargement placée en début d'exécution).

La seconde démo est *Elevated* (Rgba et TBC, 2009), arrivée en première place à Breakpoint en avril 2009 dans la catégorie PC 4k. 9 ans après *Heaven 7*, cet exécutable parvient à produire plus de 3 minutes et 30 secondes d'images photoréalistes dans 8 fois moins d'espace ! Les techniques ont changé : c'est une utilisation quasi exclusive des *shaders* qui permet ce gain de taille. Ici, il n'y a plus aucune texture générée, toute la géométrie, les couleurs, et les lumières sont calculées par la carte graphique. À partir de seulement deux triangles, que les shaders géométriques se chargent de subdiviser, on atteint un million de triangles qui composent une scène. Il n'est même pas nécessaire de posséder une carte dernier cri pour l'exécuter : ce serait réduire le public potentiel ; une carte de moyenne gamme suffit pour assister au spectacle de la plus haute résolution. Pour réussir à rentrer dans 4ko cependant, *Elevated* a été écrit entièrement en assembleur. Le prototype a tout de même été conçu en C. Les mouvements de la caméra sont également procéduraux, avec la possibilité de régler quelques paramètres. Leurs tremblements sont le fait d'un léger bruit ajouté au chemin défini mathématiquement. C'est encore une brillante démonstration d'optimisation, car un chemin de caméra est ici entièrement déterminé en 2 octets (Quilez, 2009).

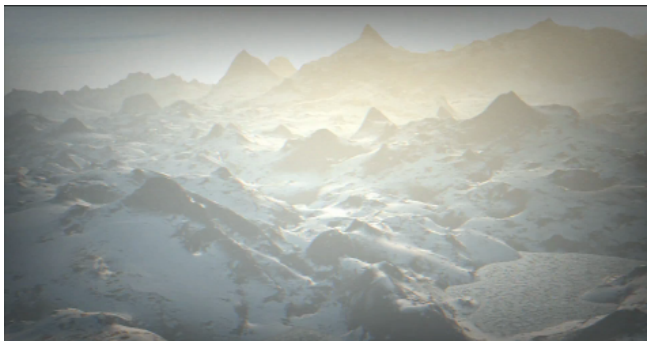


Figure 10. Elevated *(Rgba et TBC, 2009).*

Nous pourrions continuer à énumérer et décrire d'autres productions significatives de ces dernières années, mais leur nombre est trop important pour la place qui nous est allouée dans ce journal. Le site pouët.net recense plus de 53000 œuvres, et environ 10000 groupes. Nous avons exposé suffisamment d'éléments pour appuyer notre propos : toutes ces créations illustrent adéquatement les intersections de l'art et de la programmation. Il s'agissait d'abord d'art au sens de maîtrise technique, mais les œuvres plus récentes savent être plus émotionnelles. Mais que ce soit pour une démo ou un programme en Processing, c'est le résultat qui est présenté au public comme œuvre, et non le code source. Le langage Piet remédie à cela.

3. Piet : l'inversion des rôles

Dans les exemples précédents, il a été montré que la programmation pouvait se révéler un outil extrêmement puissant et qu'il lui était possible de repousser les limites que certains media imposent et donc de grandement contribuer à l'évolution du monde artistique. La programmation permet de faire de l'art, mais l'art peut-il faire de la programmation ? C'est à cette question que répond Piet, un langage de programmation dit ésotérique, qui sort des sentiers battus et propose de programmer non pas en transmettant des instructions formées de lettres, mais par des instructions en pixels.

Quand l'on souhaite compiler un programme avant de l'exécuter, son code source est transmis à l'ordinateur qui se charge de le traduire en langage machine, le seul dialecte qu'un ordinateur puisse véritablement comprendre. Dans un langage classique, le code source en question est composé d'instructions en langage "humain", elles-mêmes composées de mots puis de lettres. Dans Piet au contraire, le code source est une image dans laquelle la couleur de chaque pixel, ici appelé codel, aura une signification bien particulière. Le langage doit son nom à l'artiste néerlandais Piet Mondrian dont les œuvres ont inspirées l'auteur du langage, David Morgan-Mar, pendant sa phase de conception.

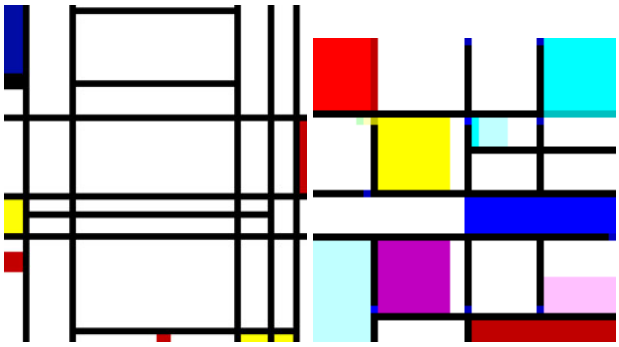


Figure 11. *Un tableau de Piet Mondrian (Composition 10)* [1] *et un programme Piet* [2].

L'idée de base de Piet est que chaque pixel de l'image source, aussi appelé codel, est associé à une couleur appartenant à une palette prédéfinie (Morgan-Mar, 2008). En associant des codels de différentes couleurs et de différentes teintes, et en fonction de leur position, on peut effectuer des opérations arithmétiques de base permettant de manipuler des valeurs numériques tel que pourrait le faire n'importe quel autre langage dit classique et donc de créer tout type de programme.

Figure 12. *Programme résolvant le problème des tours de Hanoï.* [2]

Bien sûr ce type de langage n'a pas pour vocation d'être adopté à grande échelle par la communauté des programmeurs, sa nature rendant tout programme complexe à écrire (ou plutôt à dessiner) et quasiment impossible à comprendre à moins d'en être l'auteur. Mais le concept même reste original et Piet permet de réunir art et programmation dans un sens tout à fait inattendu. C'est ici le code source, habituellement cryptique pour toute personne dénuée de formation adéquate, qui devient œuvre d'art, et non son exécution.

1. Tiré de `http://www.ibiblio.org/wm/paint/auth/mondrian/`
2. Tiré de `http://www.dangermouse.net/esoteric/piet/samples.html`

Conclusion

Avec l'avènement de l'ordinateur personnel durant les années quatre-vingt, de nombreuses passions ont vu le jour. Les bricoleurs de l'époque prirent vite conscience du potentiel de l'outil informatique et, presque aussitôt, la demoscene et ses brillants programmeurs apparurent pour dompter la machine et tenter de la maîtriser à la perfection. La plupart des œuvres produites, bien qu'époustouflantes aux yeux des connaisseurs, pouvaient laisser dubitatif le spectateur qui n'était pas adepte de la pratique. Ce sont néanmoins ces programmeurs pionniers qui ont ouvert la voie, alors que l'ordinateur n'était encore qu'une machine de travail ou de jeu.

Quelques années plus tard, le laboratoire du MIT créa Design by Numbers, dans le but de permettre au plus grand nombre de dessiner en programmant. Il est ici difficile de parler d'art, le projet de John Maeda restant très limité, mais on peut le qualifier d'expérience. Ses résultats ont démontré qu'il existait une véritable demande pour ce type d'idée : un public d'artistes-programmeurs potentiellement intimidés par la sémantique menaçante des langages traditionnels, mais prêts à utiliser un langage plus simple et plus adapté à leurs besoins. Ainsi naquît Processing, qui a su trouver un équilibre entre liberté artistique et complexité.

La programmation permet aujourd'hui de laisser libre cours à son imagination. La demoscene et sa constante course à l'optimisation, aidée par la puissance sans cesse renouvelée des ordinateurs qui permet aujourd'hui de magnifiques rendus, attirera le programmeur par vocation, avide de détails techniques et connaissant sur le bout des doigts les arcanes de la machine. L'artiste, quant à lui moins intéressé par le processus de conception et davantage par le résultat final, ira vers Processing et la famille de langages qui l'entoure afin de tirer parti de son interface intuitive et de son potentiel étendu. Enfin, le curieux se tournera vers l'ésotérique langage Piet, dans lequel l'exécutable est anodin, car c'est le code source la véritable œuvre d'art : un tableau abstrait.

Nous avons ainsi présenté différents produits issus du croisement de l'art et de la programmation, mais notre questionnement va plus loin. L'agencement de nos trois parties représente une importance croissante du code source : avec Processing il est simplifié et caché pour laisser la place aux résultats ; dans les démos il est rarement divulgué, mais très convoité ; avec Piet il est l'objet de toute l'attention. Nous sommes alors amenés à considérer l'élaboration d'algorithmes plus seulement comme une science, mais comme un véritable art. Dans les mots de Donald E. Knuth, tirés de l'introduction du premier volume de *The Art of Computer Programming* :

> The process of preparing programs for a digital computer is especially attractive because it not only can be economically and scientifically rewarding, it can also be an aesthetic experience much like composing poetry or music.

Références

Association pour le développement de l'art numérique, « Demoscene Tv », http://www.demoscene.tv, 2005. Accédé le 13 mars 2010.

Bardiot C., « Merce Cunningham, la danse et l'ordinateur », *Patch*, Octobre, 2009.

Borzyskowski G., « The Hacker Demo Scene and it's Cultural Artifacts », n.d.

Crionics, The Silents, « Hardwired », http://pouet.net/prod.php?which=981, 1991. Accédé le 14 mars 2010.

Dolphins, « Capped.TV », http://capped.tv, 2008. Accédé le 13 mars 2010.

Exceed, « Heaven 7 », http://pouet.net/prod.php?which=5, 2000a. Accédé le 14 mars 2010.

Exceed, « Heaven 7 Making of », http://exceed.hu/h7/, 2000b. Accédé le 14 mars 2010.

Green D., « Digital Graffiti », *Wired*, 1995.

Kefrens, « Desert Dream », http://pouet.net/prod.php?which=1483, 1993. Accédé le 13 mars 2010.

Kristensen O., « Body Navigation », http://3xw.ole.kristensen.name/works/body-navigation/, 2008. Accédé le 7 mars 2010.

Maeda J., *Design by Numbers*, MIT Press, 2001.

Mandarine, scene.org, « pouët.net », http://pouet.net/prod.php?which=25304, 2000. Accédé le 13 mars 2010.

Melon Dezign, « Baygon », http://pouet.net/prod.php?which=1644, 1995. Accédé le 14 mars 2010.

Morgan-Mar D., « Piet », http://www.dangermouse.net/esoteric/piet.html, 2008. Accédé le 7 mars 2010.

Phenomena, « Enigma », http://pouet.net/prod.php?which=394, 1991. Accédé le 14 mars 2010.

Polgár T., *Freax*, vol. 1, CSW-Verlag, 2005.

Quilez I., « behind elevated », http://iquilezles.org/www/material/function2009/function2009.pdf, 2009. Accédé le 14 mars 2010.

Reas C., « TI », http://reas.com/iperimage.php?section=works&work=ti_s&id=0, 2004. Accédé le 13 mars 2010.

Rgba, TBC, « Elevated », http://pouet.net/prod.php?which=52938, 2009. Accédé le 14 mars 2010.

Shiffman D., *Learning Processing, A Beginner's Guide to Programming Images, Animation, and Interaction*, Morgan Kaufmann, 2008.

Spaceballs, « State of the Art », http://pouet.net/prod.php?which=99, 1992. Accédé le 13 mars 2010.