

interface logiciel dans le domaine du design graphique

analyse des méthodes
de représentations
visuelles des logiciels
de design graphique.

Nicolas Baldran

Mémoir de Master thesis orientation Media Design HES-SO
HEAD—Genève, dirigé par Daniel Sciboz, soutenue le 6 mars 2019.

Responsable de département: Alexia Mathieu.

Note: Les schémas et images placés dans le corps de texte sont des représentations de concept de ma propre interprétation. Aucun de ces schémas n'est directement extrait de documents originaux. Si tel est le cas, ils sont placés sur les pages comportant des images accompagnant le texte, et sont référencés par «fig-».

Introduction p.7

Interfaces graphiques p.23

- Pygmalion et le concept d'icône
- Xerox Palo Alto Research Center
- Le rôle des interfaces graphiques à différents stades de travail du design graphique.
 - Recherche et expérimentation par tâtonnement
 - Répétition de tâches ou d'actions
 - Mise à jour et partage entre logiciel
- En conclusion

Interfaces textuelles p.51

- Langage informatique
- Le rôle des interfaces textuelles à différents stades de travail du design graphique.
 - Recherche et expérimentation par tâtonnement
 - Répétition de tâches ou d'actions
 - Mise à jour et partage entre logiciel
- En conclusion

Pour un environnement de travail p.65

- Aider à comprendre
- Manipulation direct et WYSIWYG
- Dialogue entre trois entités, l'utilisateur, l'interface textuelle, l'interface graphique
- En conclusion

Conclusion p.79

Bibliographie p.83

0— introduction

Lors de sa conférence en 2009, *Art That Looks back at you*, Golan Levin projette une photographie d'une table de l'un de ses étudiants. On peut y voir l'usure due au déplacement d'une souris pour contrôler un ordinateur. Il fait remarquer que dans cette trace, on peut reconnaître l'écran de cet ordinateur, avec en haut à gauche, la position du menu OS X fig .¹. C'est le constat que la manière dont nous utilisons nos ordinateurs personnels est profondément ancrée dans le monde physique. Analyser la manière dont les logiciels peuvent être perçus nous permettra de comprendre comment le designer graphique peut appréhender son ordinateur. Pour que ce dernier devienne malléable et acquiert une certaine « souplesse » pour que le designer graphique puisse réellement en faire son outil au sens de Pierre Damien Huyghe¹, il faut analyser la manière dont les mécanismes de cette machine lui sont représentés. Comment le designer peut-il appréhender ses programmes et son ordinateur? Qu'impliquent les interfaces visuelles dans le travail du designer graphique ?

D'après les écrits de Michel Beaudouin-Lafon² et de Jean Caelen³ nous pouvons identifier plusieurs modes d'interaction entre l'humain et l'ordinateur que je regroupe ainsi :

- vocal ou sonore, tel que le fait Siri sur les appareils Apple, ou les directives annoncées par le GPS d'une voiture;
- écrites, comme l'utilisation d'un clavier ou l'affichage d'erreurs sur une feuille par une imprimante;
- gestuel ou physique, par exemple la souris, le trackpad, les écrans tactiles, la reconnaissance faciale ou les vibrations dans une manette de console;
- ou visuel, par l'affichage de graphiques, images, animations.

¹ Entretien avec Pierre-Damien Huyghe, « faire franchir un pas à une technique », Back Office n° 1, Design et pratique du numérique, ed. B42 et Fork, Paris, 2017, p. 84.

² M. Beaudouin-Lafon, « Interaction homme-machine », Laboratoire de Recherche en Informatique, 2007. [En ligne]. Disponible sur: <https://www.lri.fr/~mbl/pdf/mbl-encycl-o6a.pdf>. [Consulté le : 15-nov-2018].

³ J. Caelen, « Interaction et multimodalité », dans Troisième colloque Hypermédiyas et Apprentissages, E. Bruillard, J.-M. Baldner, et G.-L. Baron, Ed. Châtenay-Malabry, France : ENS Éditions, 1996, p. 11-32.

Les interactions homme-ordinateur se font dans deux sens. Il y a des interactions qui vont de l'homme vers l'ordinateur et des interactions qui vont de l'ordinateur vers l'homme. Ces échanges d'interactions se font grâce à des composants qui relient les deux entités que sont l'ordinateur et son utilisateur. Ce sont des périphériques informatiques. Dans le référentiel de la machine, ces composants peuvent être émetteurs ou récepteurs d'interactions. Ces éléments composent l'interface entre l'homme et l'ordinateur qui, toujours dans le référentiel de l'ordinateur, composent respectivement l'interface d'acquisition et l'interface de restitution.

«Une interface agit comme une jonction entre deux objets distincts, leur permettant d'interagir, par un ensemble de règles définies. Dans le champ du numérique, il peut s'agir d'interface utilisateurs permettant à un être humain d'interagir avec un ordinateur [...], de protocoles de communication entre logiciel et périphérique matériel (pilote) ou encore entre logiciels»⁴.

Notre manière d'interagir avec l'ordinateur se fait essentiellement via des logiciels. «Un logiciel est un ensemble de programmes»⁵, un ordinateur peut donc être décomposé en une succession de programmes. Comme ces programmes communiquent entre eux, un ordinateur peut donc être décomposé en une succession d'interfaces. Joseph Weizenbaum⁶, souligne le fait que l'interface étant une représentation

4 E. Gay, K. Donnat, et A. Masure, Éd., «glossaire», dans Back office : design graphique et pratiques numériques | graphic design and digital practices., B42., Paris, France, 2017, p. 136.

5 *ibid.*

6 «Un programmeur travaillant pour un ordinateur en n'utilisant que l'assembleur n'aura jamais besoin d'apprendre le langage déterminé par l'ordinateur lui-même, c'est à dire son langage machine. Dans un certain sens, le programmeur ne perçoit jamais la machine à laquelle il s'adresse ; il ne perçoit et ne travaille qu'avec un artifice symbolique qui, pour lui, devient la machine»

J. Weizenbaum et M.-T. Margulici, «Fonctionnement des ordinateurs», dans Puissance de l'ordinateur et raison de l'homme : du jugement au calcul, Boulogne-sur-Seine : Ed. d'informatique, 1981, p. 67.

d'un programme, elle apporte inévitablement un niveau d'abstraction. Ces représentations ont été classifiées par Alan Cooper selon trois paradigmes :

- le paradigme technologique, qui « est basé sur la compréhension du fonctionnement des choses : objectif difficile »⁷ ;
- le paradigme de la métaphore, qui « est basé sur le fait de deviner intuitivement comment fonctionnent les choses : méthode problématique »⁸ ;
- et le paradigme idiomatique, qui « est basé sur le fait d'apprendre comment accomplir les choses : processus normal et humain »⁹.

Stephen D. Burd nous présente une classification des niveaux d'abstraction de langages de programmation par rapport à l'ordinateur ^{fig . 2 fig.2.2}. Si une interface graphique présente des fonctions d'un programme écrit avec un langage dit de 2e génération ou supérieur, elle vient comme une interface supplémentaire qui éloigne l'homme de sa machine.

7 A. Cooper, « The Myth of Metaphor », Visual Basic Programmer's Journal, 1995. [En ligne]. Disponible sur : <https://tafein2009.files.wordpress.com/2009/09/the-myth-of-metaphor.pdf>. [Consulté le : 15-nov-2018].

8 Ibid.

9 Ibid.

machine

langage machine

Langage d'instruction binaire pour le CPU.

interface textuelle
de bas niveau

Langage
d'assemblage,
langage dit
«de deuxième
génération», lisible
par l'homme.

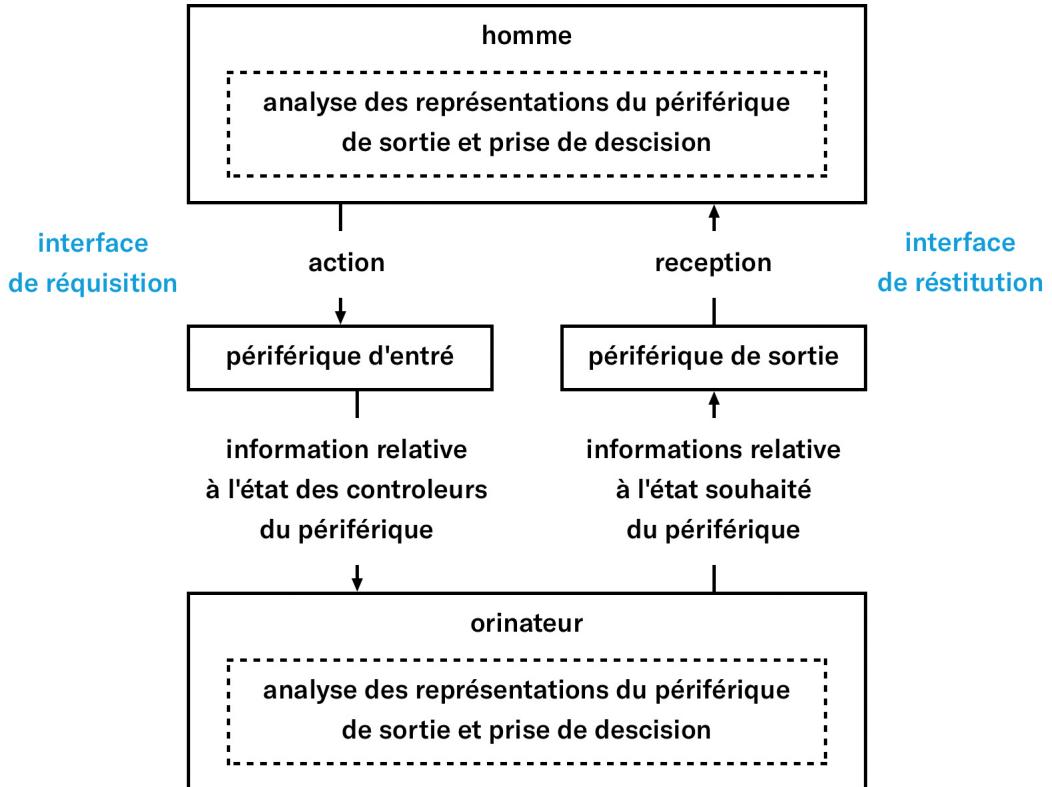
interface textuelle
de haut niveau

Langage suivant la
seconde génération,
par exemple
FORTRAN, JAVA,
C++, JAVASCRIPT,
PYTHON.

interface graphique

..... perception du logiciel

designer graphique



Ces schémas sont des interprétations que je fais des mécanismes d'interactions humain-ordinateur, qui découlent des concepts de Stephen D. Burd et Joseph Weizenbaum. Il nous permet de constater que les interfaces de représentation ont la caractéristique d'être le lieu de prise de décision par l'utilisateur, par rapport à « l'état » des processus de calcul de la machine qui lui est représenté.

Alors que des projets tels que *Synopsis Video Synthesizer*^{fig .3} de Denis Gallant et Rob Schafer, qui peuvent questionner la nature de nos périphériques de saisie, nous nous concentrerons sur les interfaces de représentations, qui joue un rôle particulier dans la représentation que nous nous faisons de l'ordinateur. De plus, si des domaines comme le jeu vidéo utilisent par exemple l'interface sonore (comme dans le mouvement des audio games et le jeu Soundvoyager^{fig .4} par exemple) force est de constater que les logiciels professionnels de création graphique sollicitent principalement l'interaction visuelle via des écrans pour leurs interfaces. Ainsi, nous nous limiterons au sujet des interfaces visuelles affichées sur écrans.

À l'heure où l'ordinateur tient une place prépondérante dans le travail du designer graphique, il est important de comprendre quelles sont ces représentations, comment elles sont construites et comment elles fonctionnent. « L'interface façonne la manière dont l'utilisateur conçoit l'ordinateur lui-même. [...] L'interface leur impose sa propre logique. »¹⁰ Si les programmes dits « créatifs » sont essentiellement désignés comme des « outils », il faut analyser comment nous les percevons pour avoir un regard critique sur ceux-ci. L'enjeu étant de comprendre dans quelles limites nos logiciels tiennent du dispositif, comme le questionne Anthony Masure¹¹. Une interface se limite-t-elle qu'à des choix d'actions présentés à l'utilisateur d'un logiciel ? N'est-elle qu'une forme plus ou moins complexe de liste de tâche dont ce dernier est capable ? Les langages de programmation sont-ils moins directifs sur les usages d'un programme, ou simplement moins explicites qu'une interface graphique ?

Nous nous poserons donc la question de savoir ce qu'imposent les méthodes de représentations de l'interface visuelle d'un programme sur l'interaction entre celui-ci et le designer graphique ?

On peut facilement séparer les interfaces visuelles selon deux critères :

- les interfaces graphiques, qui découlent principalement des concepts initiés au Xerox Palo Alto Research Center (Xerox PARC);
- les interfaces textuelles, dont l'échange se fait par des caractères, regroupant les interfaces en ligne de commande (CLI pour Commande Line Interface) et les interfaces de programmation textuelle (API pour Application Programming Interface).

Ma démarche a été d'analyser les interactions offertes par chacune lors de différentes étapes de travail du designer graphique. Comment les paramètres d'un logiciel sont-ils présentés, accessibles et manipulables ?

10 L. Manovich, « l'interface », dans Le langage des nouveaux médias, Les Presses du réel, 2010, p. 158.

11 A. Masure, « Adobe - Le créatif au pouvoir ? », STRABIC.FR, 24-juill-2011. [En ligne]. Disponible sur : <http://strabic.fr/Adobe-le-creatif-au-pouvoir>.

Nous verrons dans un premier temps les interfaces graphiques. Nous nous intéresserons ensuite aux interfaces textuelles. La démarche sera identique pour les deux catégories. Chacune introduite par un historique puis quelque peu détaillée, nous les analyserons ensuite selon trois critères que je déclare importants pour le designer graphique :

- la phase de recherche et d'expérimentation par tâtonnements;
- la récursivité et la répétition de tâches, liées au fait que le designer graphique crée des concepts qui doivent pouvoir se décliner¹²;
- l'utilisation de plusieurs programmes et le besoin « d'étendre » des fonctionnalités qui ne sont pas présentes dans un logiciel.

Nous terminerons sur le constat que les interfaces textuelles sont souvent présentées à l'écran via des programmes utilisant des interfaces graphiques. Lorsqu'un utilisateur utilise une interface textuelle de programmation, son intérêt est de passer par un logiciel qui l'aidera un minimum à comprendre la structure du texte qu'il manipule *fig. 5*. Par cette idée « d'environnement » où l'interface graphique et l'interface textuelle coexistent, nous ferons un état des lieux des recherches et outils dédiés au designer graphique.

12 Karl Gerstner met par exemple en évidence le fait que le designer peut être amené à créer des processus de création plutôt qu'un résultat fini :

« Le processus créatif doit être ramené à une suite de choix. Faire du design graphique consiste dans la sélection et la combinaison des éléments déterminants d'un problème. Vu sous cet angle, le design graphique requiert avant tout une méthode. »

Karl Gerstner, « Designing programmes », traduit par A. Manaranche, « Karl Gerstner », Index Grafik, 22-févr-2016. [En ligne]. Disponible sur : <http://indexgrafik.fr/karl-gerstner/>. [Consulté le : 02-déc-2018].

fig .1 Extrait de la conférence de Golan Levin, «Art that looks back at you», 2009, consultée le 13 novembre 2018, [en ligne], www.ted.com/talks/golan_levin_ted2009

fig .2 S. D. Burd, Programming language evolution. Extrait de Stephen D. Burd, «Systems Architecture», éd. Course Technology, 6e édition, juin 2015, p. 370.

fig.2.2 S. D. Burd, Programming language characteristics. Extrait de Stephen D. Burd, «Systems Architecture», éd. Course Technology, 6e édition, juin 2015, p. 371.

Représentation de la classification des langages de programmation par «génération», en fonction de leur niveau d'abstraction par rapport à la machine; extrait du livre de Stephen D. Burd, «Systems Architecture».

L'abréviation «XGLs» signifie «X Generation Languages», soit «langage de génération X» en français, abrégé «LXG».

Nous pouvons remarquer que la tendance générale est la création de langage de plus en plus abstrait par rapport au langage machine.

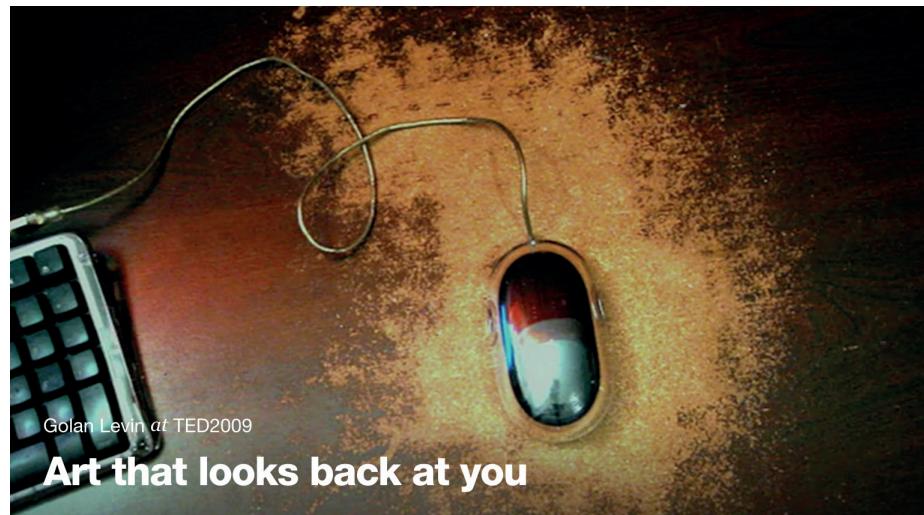


fig.1

Generation	Description or characteristics
First	Binary instructions for a CPU
Second	Mnemonic instructions for a CPU
Third	Instruction explosion, machine independence, and usually standardization
Fourth	Higher instruction explosion, interactive and graphical I/O support, database support, limited nonprocedural programming, and proprietary standards
Fifth	High instruction explosion, nonprocedural programming, expert systems, and artificial intelligence applications
Object-oriented	High instruction explosion, support for modern code development, and code reuse methods, such as inheritance and message passing
Scripting	Very high instruction explosion; interpreted execution, used mainly for Web applications

fig:2

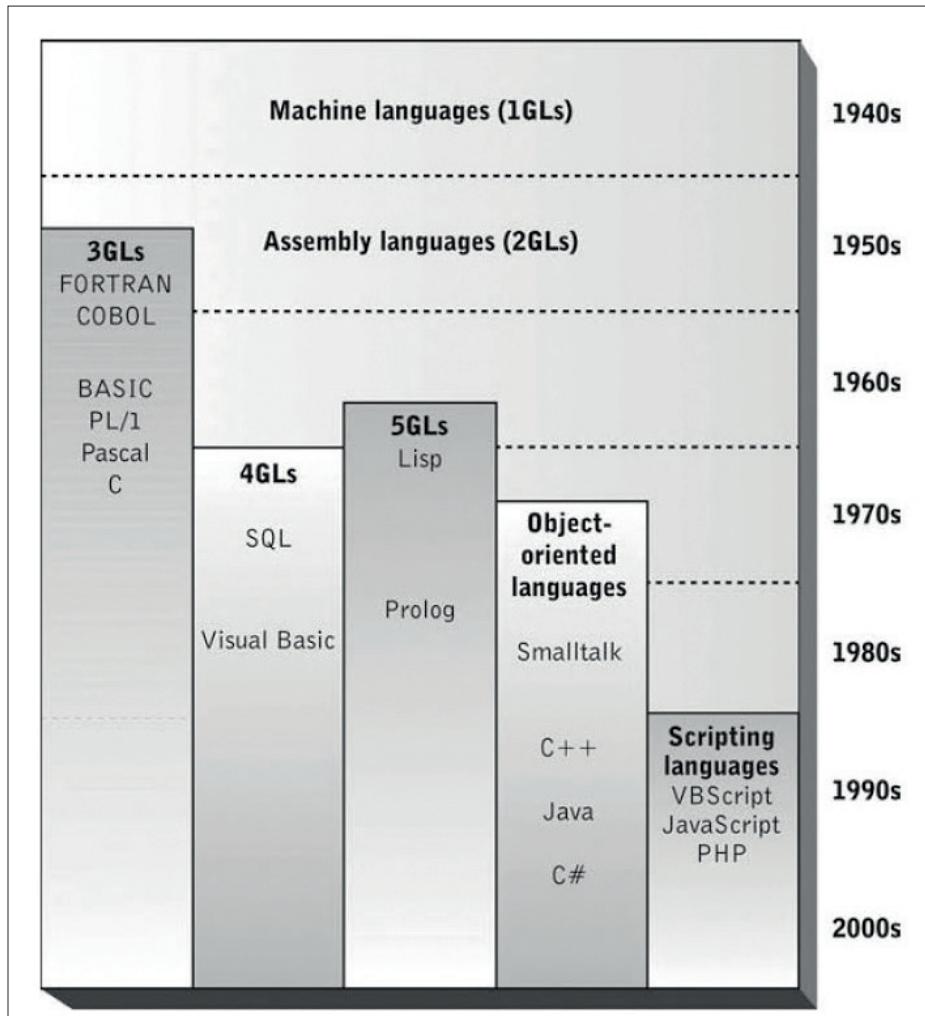


fig:2·2

fig .3 Synopsis Video Synthesizer est un projet conçu par Denise Gallant, et Rob Schafer à partir de 1975.

À l'aide de périphériques de contrôles (à gauche) ce système permet la manipulation vidéo (couleur, déformation) en offrant une interaction différente que celle d'un dispositif de pointage.

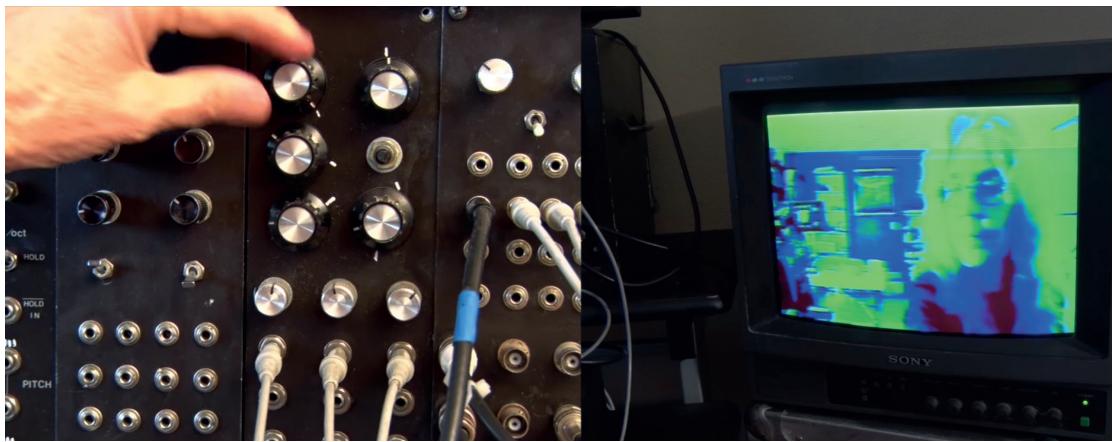
fig .4 Chaîne IGN, « Bit Generations: Sound Voyager Game Boy Gameplay », Youtube, 21-mai-2011. [En ligne]. Disponible sur: <https://youtu.be/pHznCLRbO7c>. [Consulté le: 15-nov-2018].

Skip Ltd., Soundvoyager. Japon : Nintendo, 2006.

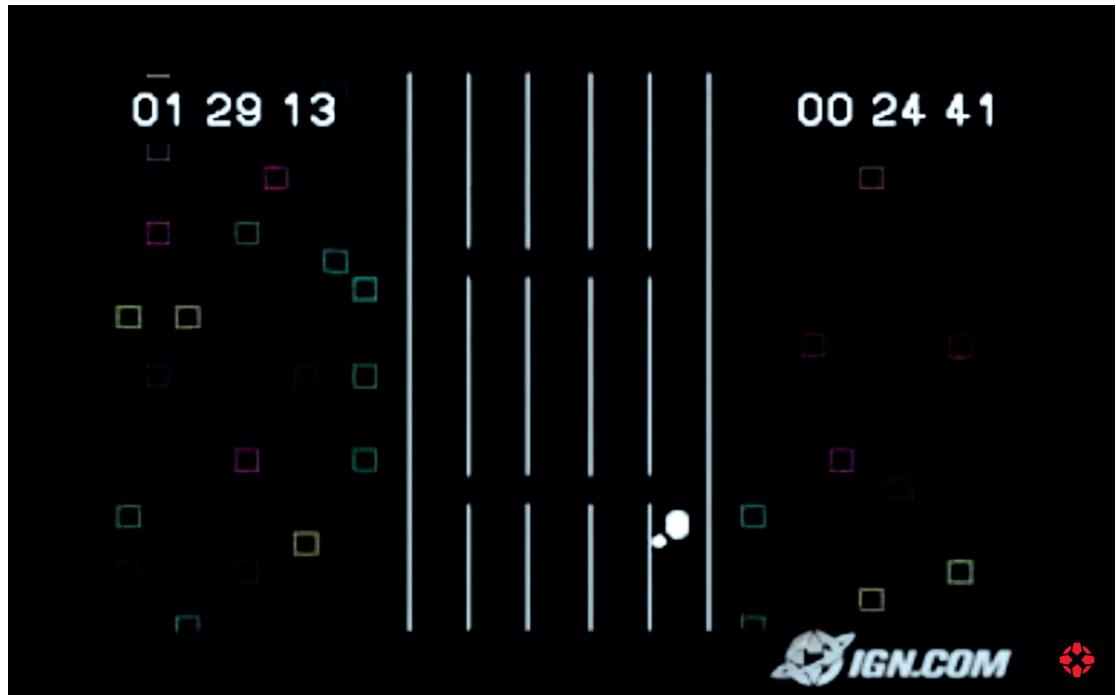
Soundvoyager peut se jouer entièrement sans regarder l'écran de la Game Boy Advance puisqu'il utilise des retours sonores plutôt que visuels comme mode d'interaction entre le programme et son utilisateur. Les informations sur l'environnement dans lequel le joueur évolue sont principalement données à travers des différences d'intensité ou de canal (gauche/droite).

Il faut avoir des écouteurs pour profiter au mieux de l'expérience.

Il a été distribué seulement au Japon, dans la série bit Generations.



fig·3



fig·4

fig . 5 Exemple de Daniel Shiffman pour créer une onde sinusoïdale simple avec le programme P5.js (Processing Foundation).

Nous pouvons constater la mise en forme opérée par le programme Atome (GitHub inc.), à droite, pour simplifier la lecture de l'interface textuelle (le langage de programmation JavaScript présenté de manière « brute » à gauche).

```
1 var xspacing = 16; // Distance between each horizontal location
2 var w; // Width of entire wave
3 var theta = 0.0; // Start angle at 0
4 var amplitude = 75.0; // Height of wave
5 var period = 500.0; // How many pixels before the wave repeats
6 var dx; // Value for incrementing x
7 var yvalues; // Using an array to store height values for the wave
8
9 function setup() {
10    createCanvas(710, 400);
11    w = width/16;
12    dx = (TWO_PI / period) * xspacing;
13    yvalues = new Array(floor(w/xspacing));
14 }
15
16 function draw() {
17    background(0);
18    calcWave();
19    renderWave();
20 }
21
22 function calcWave() {
23    // Increment theta (try different values for
24    // 'angular velocity' here)
25    theta += 0.02;
26
27    // For every x value, calculate a y value with sine function
28    var x = theta;
29    for (var i = 0; i < yvalues.length; i++) {
30      yvalues[i] = sin(x)*amplitude;
31      x+=dx;
32    }
33 }
34
35 function renderWave() {
36    noStroke();
37    fill(255);
38    // A simple way to draw the wave with an ellipse at each location
39    for (var x = 0; x < yvalues.length; x++) {
40      ellipse(x*xspacing, height/2+yvalues[x], 16, 16);
41    }
42 }
```

fig5

1— interfaces graphiques

Sketchpad, réalisé par Ivan Sutherland en 1963 dans le cadre de sa thèse de doctorat au MIT, est considéré comme le programme étant à l'origine de l'interaction graphique¹³. Il permet de tracer des formes géométriques sur un écran cathodique à l'aide d'un stylo optique. L'utilisateur pouvait tracer et manipuler (rotation, taille) des primitives géométriques (cercle, carré, ligne). Sketchpad introduit aussi le dessin par contrainte, avec la possibilité de tracer des arcs de cercle, des lignes parallèles ou la mise à l'échelle de manière homothétique d'un élément graphique fig .6.

Pygmalion et le concept d'icône

David Canfield Smith, qui travaillera sur la première interface iconique moderne implantée sur le Xerox Star¹⁴, est à l'origine du concept d'icône, qu'il décrit dans sa *Pygmalion: An Executable Electronic Blackboard*, en 1975 :

« Les entités avec lesquelles on programme dans *Pygmalion* sont ce que j'appelle des <icônes>. Une icône est une entité graphique porteuse de sens à la fois comme une image visuelle et comme un objet machine. Les icônes contrôlent l'exécution des programmes de l'ordinateur, parce qu'il leur est associé du code, des données, ainsi qu'une image pour apparaître à l'écran. C'est ce qui distingue les icônes de, disons, les lignes et les rectangles dans un programme de dessin, qui n'ont pas une telle sémantique. Pygmalion est à l'origine du concept d'icônes tel qu'il apparaît maintenant dans les interfaces graphiques utilisateurs sur les ordinateurs personnels. Après avoir terminé ma thèse,

13 D. Bissell, « The father of computer graphic », in BYTE Magazine - June 1990, État-Unis: McGraw-Hill Inc., 1990, p. 380-381. Accessible en ligne, « The father of computer graphics », GULdebook. [En ligne]. Disponible sur: <https://guidebookgallery.org/articles/thefatherofcomputergraphics>. [Consulté le: 02-déc-2018].

14 Henry Lieberman, « A Creative Programming Environment », dans HCI Remixed, Tom Ericson and David MacDonald, éd. MIT Press, Londre, Angleterre, 2008, p. 37-42.

j'ai rejoint le projet d'ordinateur <Star> de Xerox. La première chose que j'ai faite a été la refonte des icônes orientés programmation de *Pygmalion* en des icônes orientées bureaux qui représentent documents, dossiers, classeurs, boîtes aux lettres, téléphones, poubelles, etc. Ces icônes contiennent à la fois des données (ex. les icônes des documents contiennent du texte) et des comportements (ex. quand une icône de document est déposée sur une icône de dossier, le dossier stocke le document dans le système de fichiers). Cette idée a ensuite été adoptée par l'ensemble de l'industrie des ordinateurs personnels et des stations de travail»¹⁵.

Les icônes reposent sur l'idée de présenter les choses de manière comportementales et intrinsèques à l'aide de représentations visuelles. Cette représentation permet une retranscription des données et fonctions d'un programme moins abstraite pour l'utilisateur qu'une interface textuelle qui elle, décrirait plus l'action ou la donnée par rapport à l'ordinateur¹⁶. Les icônes permettent d'appréhender l'environnement d'un programme de manière plus intuitive. Il y a deux types d'icônes¹⁷:

- les icônes de données, qui représentent des objets sur lesquelles il est possible d'effectuer des actions;
- les icônes de fonctions, qui représentent des objets qui effectuent des actions.

Xerox Palo Alto Research Center

En 1968, Douglas Engelbart présente un événement nommé *The mother of All Demos*. Il y présente pour la première fois au grand public la visioconférence, la téléconférence, le courrier électronique, le système de navigation hypertexte et le modèle d'interface basée sur le paradigme de la métaphore avec la « métaphore du bureau », composé de

15 David Canfield Smith, «Pygmalion : An Executable Electronic Blackboard », traduit par D. Cunin, « Pratiques artistiques sur les écrans mobiles : création d'un langage de programmation », Université Paris 8, Saint-Denis France, 2014.

16 Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard et Kevin Mackey, « The Xerox Star : a retrospective », IEEE Computer, État-Unis, 1989, p. 11-29.

17 David Canfield Smith, Charles Irby, Ralph Kimball et Eric Harslem, « The Star user interface : an overview », Proceedings of the AFIPS 1982 National Computer Conference, p. 515-52.

→ « The Star user interface : an overview », GUILdebook : Graphical User Interface gallery, consulté le 5 janvier 2018, www.guidebookgallery.org/articles/thestaruserinterfaceanoverview

fenêtres, dossiers, fichiers, corbeilles. Au sein du Xerox PARC (Palo Alto Research Center Incorporated) créé en 1970, ces notions seront implantées pour la première fois dans l'ordinateur Xerox Alto en 1973 puis reprises dans le Xerox Star en 1981. Ce dernier est accompagné d'une souris et d'éditeurs de documents qui permettent une visualisation fidèle lors de la phase de conception d'un rendu final ^{fig.} 7 (souvent imprimé)¹⁸.

Popularisées par l'Apple Lisa, nos interfaces actuelles reposent principalement sur les principes d'interface utilisateur graphique (GUI) du Xerox PARC¹⁹:

- les concepts de fenêtre, d'icône, de menu et de l'interaction avec un dispositif de pointage, interfaces dites de type WIMP (Windows Icon Menu Pointer);
- «ce que vous voyez est ce que vous obtenez», dit WYSIWYG (What You See Is What You Get), qui repose sur un affichage précis d'un document lors de son édition par rapport au résultat final (alors principalement imprimé) tel que des graphiques incorporés, numéros de page, entêtes et bas de page par exemple;
- la manipulation directe, le principe de présenter les fonctions d'un logiciel dans des emplacements distincts, permettant ainsi de cibler physiquement une action plutôt que de faire un «appel» de celle-ci²⁰;

18 Description détaillé par Anthony Masure. A. Masure, «Manifeste pour un design acentré», dans Design et humanités numériques, Paris, France : B42, 2017, p. 183.

19 Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard et Kevin Mackey, «The Xerox Star: a retrospective», IEEE Computer, État-Unis, 1989, p. 11-29.

20 «What, exactly, is direct manipulation? Consider the following passage from a description of Apple's Macintosh:

«Imagine driving a car that has no steering wheel, accelerator, brake pedal, turn signal lever, or gear selector. In place of all the familiar manual controls, you have only a typewriter keyboard.

Anytime you want to turn a corner, change lanes, slow down, speed up, honk your horn, or back up, you have to type a command sequence on the keyboard. Unfortunately, the car can't understand English sentences. Instead, you must hold down a special key with one finger and type in some letters and numbers, such as "S20:TL:A35," which means, "Slow to 20, turn left, and accelerate to 35."

No doubt you could learn to drive such a car if you had sufficient motivation and determination. But why bother, when so many cars use familiar controls? Most people wouldn't.»

Actually, it isn't familiarity that makes real cars easier to drive than the hypothetical "computer-car" would be – cars are certainly not familiar to those who are just learning to drive them. What makes real cars easier to drive is the directness of their controls. Real cars have distinct interfaces to the speed control (the accelerator pedal), the direction control (the steering wheel), the gears (the gearshift handle), the radio (several knobs and buttons), etc. Each interface is specially designed for controlling its respective function. In contrast, the hypothetical "computer-car" has only one control: a keyboard.»

- le concept d'icône, qui consiste en une représentation des types de données (image, texte...) et des actions (imprimante, corbeille) de composants d'un programme;
- le paradigme de la métaphore avec la métaphore du bureau, qui utilise la représentation par icônes et la manipulation directe de ces éléments en s'inspirant de la manière dont l'utilisateur le ferait sur un bureau physique, comme le fait de prendre l'icône d'un fichier, et de le déposer dans l'icône corbeille.

Nous avons ici tous les éléments qui décrivent les interfaces des logiciels que la majorité des designers utilisent, telles que Photoshop (1990), Illustrator (1987), InDesign (1999) ou leurs homologues open source respectifs Gimp(1995), Inkscape(2003) et Scribus(2003).

Le rôle des interfaces graphiques à différents stades de travail du design graphique.

Recherche et expérimentation par tâtonnement. Le concept de manipulation directe, que l'on a défini au chapitre précédent, nécessite la retranscription d'un programme par une grammaire liée à l'espace. Ainsi, si nous prenons l'exemple du logiciel Illustrator, nous pouvons constater des zones de menu, de choix d'outils par des icônes et d'autres fenêtres dédiées à un ensemble de fonctions précises. Les interfaces graphiques de type WYSIWYG peuvent se décomposer en deux types de zones ^{fig .8}:

- rendu, projetant le résultat final obtenu par le logiciel;
- paramètres, permettant de modifier la zone de rendu.

Description de l'Apple Macintosh de cet extrait : L. Poole, « A Tour of the Mac Desktop », MacWorld, Vol. 1, No. 1, 1984, p. 16-21.

Cette zone de rendu est elle même composée de plusieurs éléments. Si nous prenons l'exemple de Photoshop, la zone de rendu est composée de différents calques. Des outils de manipulation vectorielle comme Illustrator se présentent eux en ensembles d'éléments qui peuvent être de forme graphique ou textuelle par exemple. Ce qu'il est important de noter c'est le fait que la zone de rendu est décomposable en ensembles «d'objets». Chacun de ces objets possède des propriétés.

La sélection de l'un de ces objets permet de visualiser les propriétés de celui-ci dans des emplacements dédiés. Pour exemple, si nous sélectionnons un cercle dans Illustrator, nous pouvons avoir rapidement des informations sur celui-ci *fig .9*. Ces propriétés sont le plus souvent éditables par l'usage de curseurs, de menus ou de zones en attente d'une valeur textuelle *fig .10*. D'autres propriétés sont éditables via des menus. L'utilisateur peut accéder à différents types d'actions et voir lesquelles sont disponibles. Souvent, si l'objet sélectionné ne peut recevoir une action proposée dans un menu, celle-ci sera présentée de manière à faire comprendre que l'action est possible, mais pas sur cet élément *fig .11*. Ce mode de représentation qu'offre l'interface graphique permet de se faire rapidement une idée sur les domaines d'actions d'un programme. L'utilisateur peut facilement sélectionner des éléments dans la zone de rendu, modifier, assigner des propriétés, des actions sur celui-ci, et en voir directement le résultat.

Une autre particularité de l'interface graphique offrant beaucoup de possibilités est de placer l'utilisateur dans des modes de présentations différents selon l'action en cours. Si nous sélectionnons l'effet de déformation «Ondes», une nouvelle fenêtre s'ouvre avec des paramètres à modifier *fig .12*. Ceci illustre parfaitement le fait qu'un logiciel est composé de programmes. Nous sommes à présent dans un sous-logiciel capable seulement d'appliquer un type d'effet avec des variables modifiables mises à disposition du designer graphique par le programmeur. L'utilisateur est devant à une interface qui ne lui présente que ce dont il a besoin.

Nous pouvons donc voir que par ces principes de présentation, l'interface graphique peut faciliter la tâche du designer lors de phases de recherche. Il peut se référer à des noms d'actions et de propriétés qui ont du sens par rapport à ce qu'il est en train de faire, grâce au paradigme de la métaphore. Cependant, le designer graphique se retrouve avec une liste de propriétés et d'actions. Il est face à des choix prédéfinis. Miller Puckette explique ainsi que la conception des logiciels «joue un rôle essentiel dans ce que l'artiste sera finalement

en mesure de créer». L'utilisateur n'est pas amené à étendre les fonctionnalités d'un programme, mais à faire avec, même si « la majeure partie de ce qu'il veut faire n'est certainement jamais venue à l'esprit du développeur ».

Le logiciel peut donc tendre au dispositif, puisque selon la définition de Giorgio Agamben, on se retrouve avec des logiciels qui ont « la capacité de capturer, d'orienter, de déterminer, d'intercepter, de modeler, de contrôler et d'assurer les gestes, les conduites »²¹ de son utilisateur.

Répétition de tâches ou d'actions.

Nous avons vu que la zone de rendu de nos interfaces est la présentation d'un ensemble d'éléments graphiques avec leurs propriétés propres. Ceci permet par exemple de facilement dupliquer une forme. Ce principe permet au designer graphique de réutiliser des éléments qu'il a déjà créés. Cependant, la question est de savoir si l'assignation de valeurs à ces propriétés peut être automatisée.

After-effect propose un système de chainage de propriétés entre éléments ^{fig .13}. Ce qui permet par exemple d'assigner des positions identiques sur plusieurs éléments.

Dans cette même logique d'automatisation, la plupart des logiciels font appel au concept de « style ». Par exemple, comme nous le voyons dans Indesign ou l'application Sketch, il est possible d'établir des styles liés à des éléments typographiques. Ainsi, les propriétés de plusieurs éléments sont « partagées » ^{fig .14}. Photoshop propose aussi un moyen d'enregistrer les actions faites avec le dispositif de pointage ^{fig .15}. L'utilisateur peut enregistrer une suite d'exécutions comme des déplacements l'application d'effets. Ces enregistrements peuvent être exécutés à d'autres moments ou dans d'autres fichiers.

Comme nous pouvons le constater, si l'assignation de propriétés peut être réalisée, elle est cependant très liée au paradigme de la métaphore. After-Effect trace une corde entre des propriétés, Photoshop enregistre des actions de déplacement, et les styles sur des éléments textuels est présenté dans une interface identique à l'assignation d'une propriété propre à un seul élément.

« Ne soyez pas dogmatique sur la métaphore du Bureau et la manipulation directe.

21 Giorgio Agamben, Qu'est-ce qu'un dispositif ?, traduit de l'italien par Martin Rueff, éd. Payot & Rivages, Paris, 2007, p.31.

La manipulation directe et la métaphore du bureau ne sont pas les meilleures façons de tout faire. Se souvenir et taper est parfois mieux que de voir et de pointer. Par exemple, si les utilisateurs veulent ouvrir un fichier parmi plusieurs centaines dans un répertoire (dossier), le système doit les laisser taper son nom plutôt que de les forcer à faire défiler le répertoire en essayant de le repérer afin de pouvoir le sélectionner.»²²

La représentation par icônes et l'utilisation intensive de la souris (due à la manipulation directe) que nous imposent les interfaces graphiques peuvent se révéler limitées dans certaines situations. Dans la dernière mise à jour de Photoshop (version CC 2018), les survols prolongés d'une fonction représentée par une icône lance une vignette vidéo qui présente ce qu'il est possible de faire avec celui-ci fig .16.

Dans Illustrator, il est possible de « détacher » un menu d'outils, pour pouvoir le placer à un endroit sur l'écran, ce qui facilite les allers-retours vers celui-ci.

Sketch Runner est une extension pour l'application Sketch qui permet d'afficher un champ de recherche dans lequel il est rapide, grâce à la saisie textuelle, de trouver et lancer une commande du logiciel. Les développeurs de cette extension expliquent que cette fonctionnalité permet « [d'] optimiser votre flux de travail quotidien »²³.

Nous constatons ici que les développeurs ajoutent des briques aux interfaces graphiques pour en corriger des problèmes.

Malheureusement cette méthode nécessite qu'un développeur prenne en considération le problème. De plus, le designer graphique doit se plier à une réponse apportée par ce développeur. La volonté de répondre aux actions possiblement exécutables par l'utilisateur avec un logiciel est le risque de la mise en place de méthodes de travail liées à la productivité, la « bonne pratique », plutôt qu'à la créativité.

« Je me souviens que j'avais des pinceaux favoris qui étaient marqués pour moi par la nature souple de leur résistance, par la qualité de leurs poils. On peut plus ou

22 Traduit depuis Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard et Kevin Mackey, « The Xerox Star : a retrospective », IEEE Computer, État-Unis, 1989, p. 11-29.

23 Citation de la page principale du site Sketch Runner, [en ligne], www.sketchrunner.com

moins travailler avec cela, faire travailler cette souplesse. Ces outils-là, en tant qu'ils sont travaillables, deviennent des appareils. Le risque avec les logiciels est que l'on gagne en efficacité avec des choses qui sont terriblement prédisposées, dont on fait vite le tour. Souvent, quand on dit «c'est mon outil de travail», on parle en réalité d'un instrument.»²⁴

Mise à jour et partage entre logiciels.

Comme nous l'avons vu, les interfaces graphiques permettent au designer la mise en forme des éléments en interagissant dans la zone de rendu. L'utilisateur y manipule directement le fond (texte, image) pour sa mise en forme. Ce principe intuitif au premier abord pose problème lors de la mise à jour du contenu d'un document. Comme l'interface présente le fond et la forme de manière indissociable, la mise à jour du premier modifie la seconde. Il faudra donc, après modification, appliquer les styles visuels qui correspondent.

Chaque logiciel à interface graphique est «spécialisé» dans un domaine. Sketch App, pour le dessin vectoriel et la création de maquettes d'interface graphique; Photoshop pour le travail d'image; Illustrator pour le dessin vectoriel. Si l'utilisateur d'un logiciel «sort» des fonctionnalités offertes par une application, il devra continuer son travail sur une autre. Par exemple, pour créer une animation vectorielle, il faudra tout d'abord créer l'illustration avec un logiciel de création vectoriel. Ce fichier devra ensuite être rouvert dans un logiciel adapté à l'animation, comme Adobe After-Effects. Le second logiciel devra donc être en mesure d'interpréter le fichier créé par le premier. Il faudra donc convertir les descriptions enregistrées dans un format de fichier vers un autre. Si nous prenons l'exemple d'un fichier vectoriel créé avec Illustrator, il sera possible de l'importer dans Adobe After-Effects. Par une conversion, les éléments dessinés dans le fichier Illustrator peuvent apparaître à l'intérieur du logiciel After-Effects et y rester modifiables. Cependant, si les formes telles que cercles, lignes ou rectangles sont correctement recréées, un texte provenant d'Illustrator sera transformé en image dans After-Effects. Il y a donc eu une perte d'information lors du passage d'un logiciel à un autre.

24 Entretien avec Pierre-Damien Huyghe, «faire franchir un pas à une technique», Back Office #1, Design et pratique du numérique, ed. B42 et Fork, Paris, février 2017, p. 84.

Regardons le fichier généré par deux logiciels de création vectorielle [fig .17](#), Illustrator et Inkscape pour une même illustration. Les fichiers générés par ces applications à interface graphique sont peu compréhensibles. Le fait que l’interface ne présente jamais le code à son utilisateur signifie que les concepteurs n’ont pas eu la nécessité de s’assurer que le code généré soit lisible. Et cela peut en partie expliquer pourquoi la conversion des fichiers - pour les faire passer d’un logiciel à l’autre (même réfléchis par le même groupe comme c’est le cas avec Illustrator et Adobe After-Effects) - est toujours si complexe.

Un plug-in²⁵ réalisé par Google, Sketch2AE, permet d’importer des illustrations vectorielles créées avec l’application Sketch App dans After-Effects [fig .18](#). Grâce à ce plug-in, un texte dans Sketch App est toujours un texte modifiable dans After-Effects. En regardant le code généré par ce plug-in, nous pouvons voir qu’il crée une déclaration générique de ce qui a été mis en place dans l’application vectorielle. Ceci montre qu’un langage commun peut lier différents programmes à interfaces graphiques entre eux. Si nous revenons sur les fichiers générés précédemment avec Illustrator et Inkscape [fig .17](#), nous pouvons voir que le fichier généré par Inkscape est plus lisible que sa version Illustrator. C’est parce que Inkscape s’appuie sur le langage SVG, «acronyme de «Scalable Vector Graphics» — graphisme de vecteurs extensibles [qui] est un format de fichier libre/ouvert et normalisé pour le graphisme vectoriel»²⁶. Ainsi, par l’utilisation d’un langage ouvert, et donc beaucoup mieux documenté, la conversion d’un fichier vers un autre logiciel est plus accessible.

Adobe propose un ensemble de logiciels, spécifique à certaines tâches. Photographie, dessin graphique, mise en page, etc. Ces logiciels sont conçus pour fonctionner ensemble, et offrir la possibilité aux entreprises de créer une chaîne de production, qui dans une logique de productivité sur fond de taylorisme, confine les professionnels dans des logiciels hyper spécialisés. Eric Schrijver²⁷ soulève l’importance d’une

25 Un plug-in est un programme qui fonctionne au sein d’un ou plusieurs logiciels, pour leur ajouter une ou plusieurs fonctionnalités. Un plug-in pourrait par exemple ajouter une fonctionnalité d’effet sur une image dans un logiciel de photographie. Les plug-ins sont souvent un moyen pour les éditeurs de logiciel d’offrir à leurs utilisateurs la possibilité d’étendre un logiciel selon leurs besoins.

26 «About», Dessiner en toute liberté | Inkscape, consulté de 25 janvier 2018, [en ligne], www.inkscape.org/fr/developper/apropos-svg

27 Eric Schrijver fait partie du collectif Open Source Publishing (OSP). OSP pratique la conception graphique en utilisant uniquement des logiciels libres et open sources, et invite leurs utilisateurs à participer à leur élaboration. → www.osp.kitchen/about

participation active dans le développement open source de la part des designers. Tant que ceux-ci se posent comme simples consommateurs, les logiciels open source, s'ils s'attaquent aux problèmes liés au logiciel propriétaires, ne restent pour le moment que des versions communautaires de produits commerciaux, sans venir inventer de nouveaux concepts.

En conclusion.

Les concepts apportés par le Xerox PARC ne semblent pas avoir évolué. Comme le souligne Lev Manovich, nos interfaces graphiques « reposent encore que sur d'anciennes métaphore et grammaires d'action »²⁸. Les ordinateurs sont beaucoup plus popularisés que dans les années 70, et les concepts pour leurs utilisations n'ont peut-être plus besoin de métaphore liée au monde physique pour être compréhensibles. La question n'est pas de remettre en cause l'interface graphique, mais plutôt son vocabulaire.

Les interfaces graphiques proposent un environnement qui doit répondre à deux cas de figure distincts du designer qui sont :

- la conception d'un projet, avec des recherches, des expérimentations, voir même l'ouverture vers de nouveaux programmes;
- la déclinaison d'un projet, le moment où les règles établies en phase de recherche doivent être appliquées à un ensemble.

Le fait de proposer le même type d'interface pour ces deux problématiques de travail montre la limite des logiciels trop concentrés à ne proposer que des interfaces basées sur le paradigme de la métaphore. Cette obstination pour le paradigme de la métaphore tend à créer des logiciels de plus en plus pensés pour une utilisation purement finale, ainsi qu'une <bonne pratique>. Pierre Damien Huyghe explique que la prédisposition des programmes instrumentalise l'outil logiciel.

«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

²⁸ Lev Manovich, « interface homme machine », *Le langage des nouveaux média*, traduction de Richard Crevier, éd. Les presses du réel, 2010, p. 192.

attractifs. Même après une utilisation intensive, ce type d'application gardera globalement les mêmes caractéristiques.»²⁹

Le fait de ne pas avoir accès au code peut rendre compliqué le partage entre logiciels, car les informations de mise en forme n'ont pas comme objectif premier d'être compréhensibles par l'homme, mais par un programme. Cependant, rien n'empêche le concepteur de s'appuyer sur un langage commun, suivant une norme préétablie, facilitant l'échange entre logiciels, et intelligible.

29 Muriel Puckett « Étude de cas sur les logiciels pour artistes : Max/MSP et Pure Data », dans Art++, sous la dir. de David-Olivier Lartigaud, éd. Hyx, 2011, page 181.

fig . 6 Ivan Sutherland faisant une démonstration de Sketchpad.

À droite, exemple de création de cercles sur un écran cathodique à l'aide d'un stylo optique.

Extrait d'une démonstration de Sketchpad par Ivan Sutherland, « Ivan Sutherland : Sketchpad Demo (1/2) », Youtube, publié le 17 nov. 2007, [en ligne], consulté le 9 décembre 2018.

fig . 7 Exemple d'interface WYSIWYG avec le Xerox Star.

Le rendu à l'écran est fidèle au rendu imprimé. « XEROX STAR with WYSIWYG Editor », [en ligne], www.intentsoft.com/wysiwyg-see-get

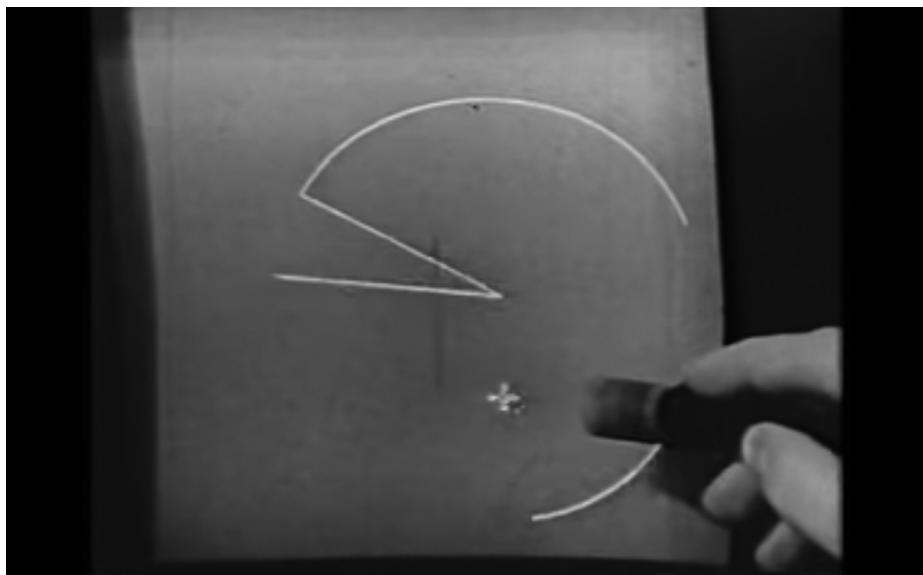
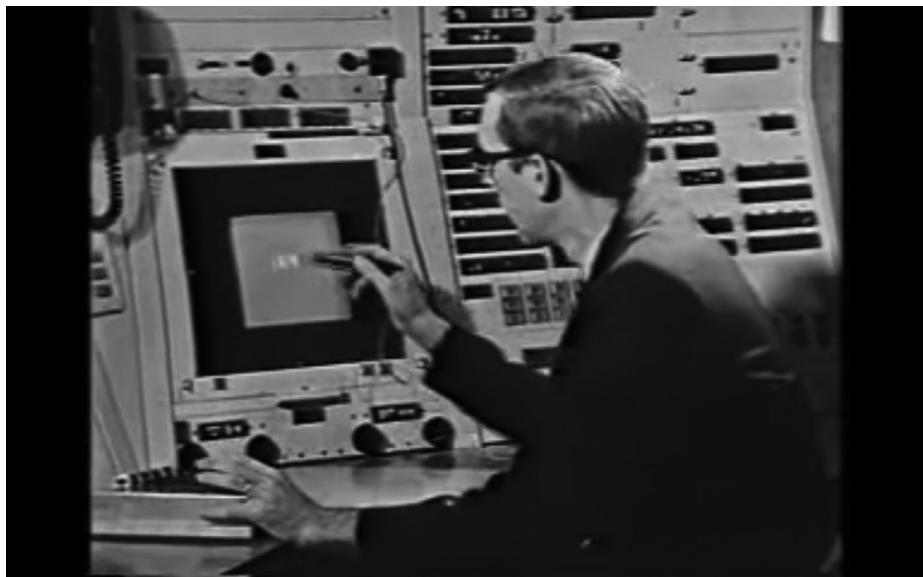


fig 6

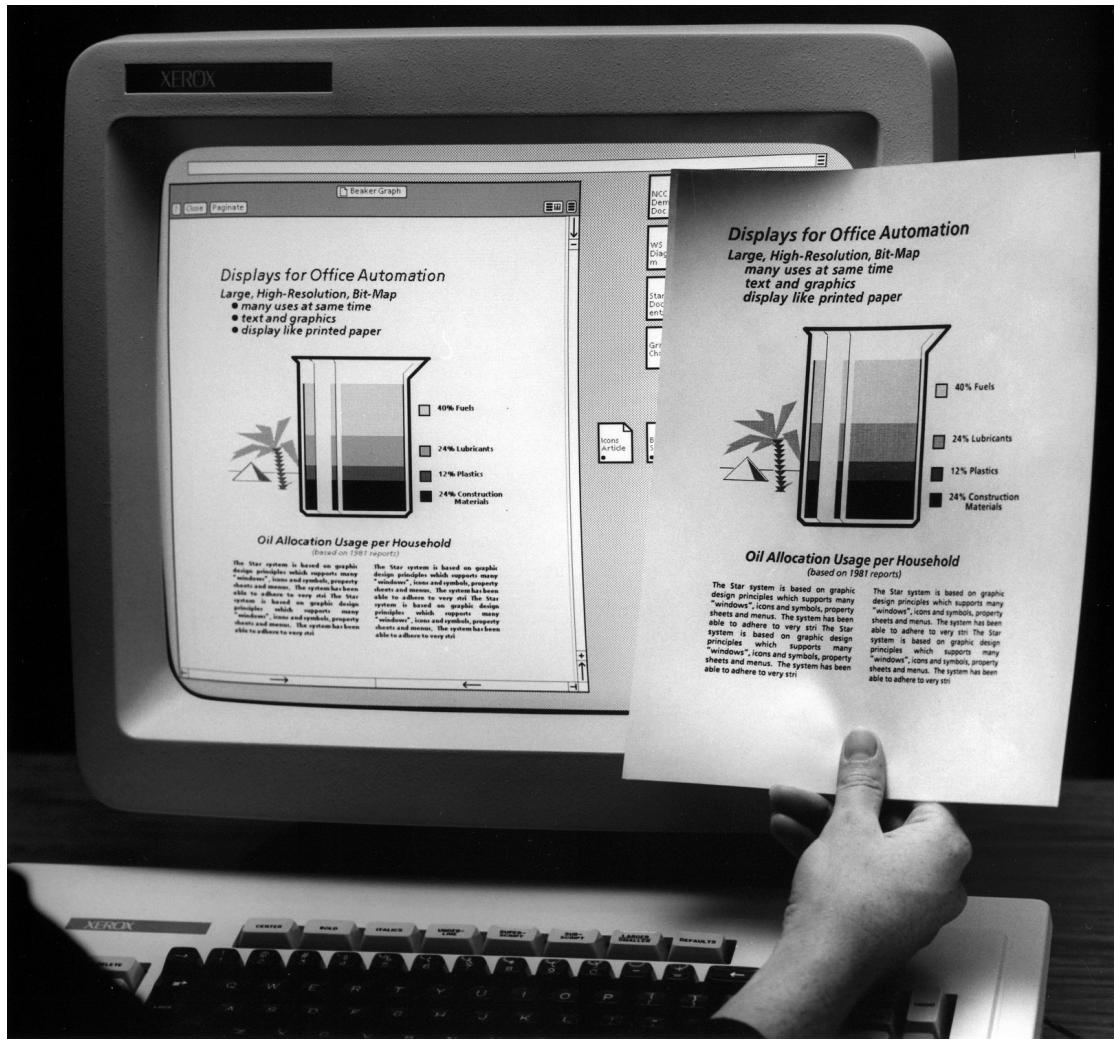


fig.7

fig .8 Représentations des zones de paramétrages et de rendues dans une interface graphique. Capture d'écran de l'interface Illustrator CC 2018.

fig .9 Exemple de la représentation des propriétés d'un élément graphique dans une interface graphique. Capture d'écran de l'interface Illustrator CC 2018.

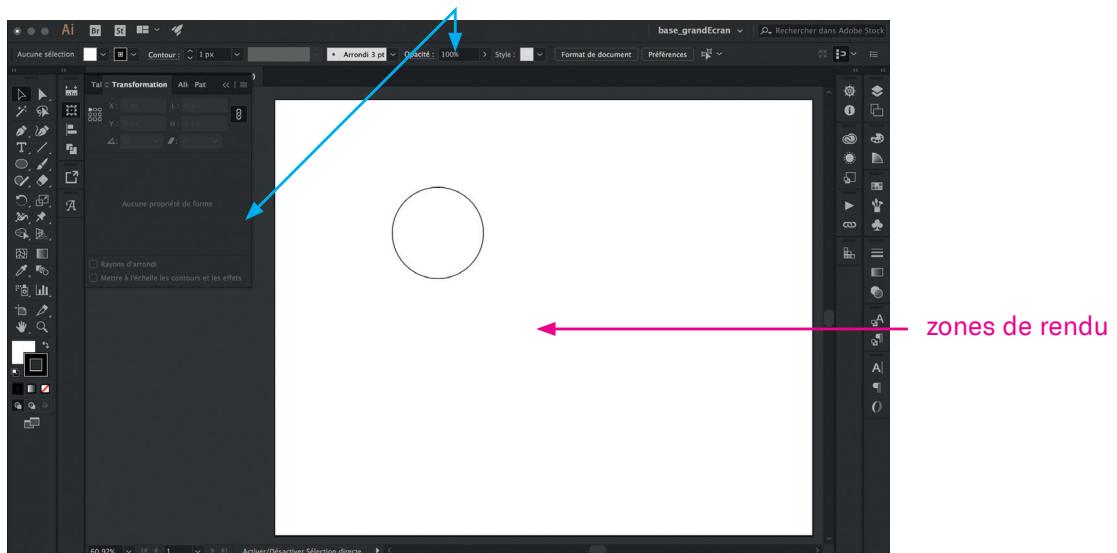
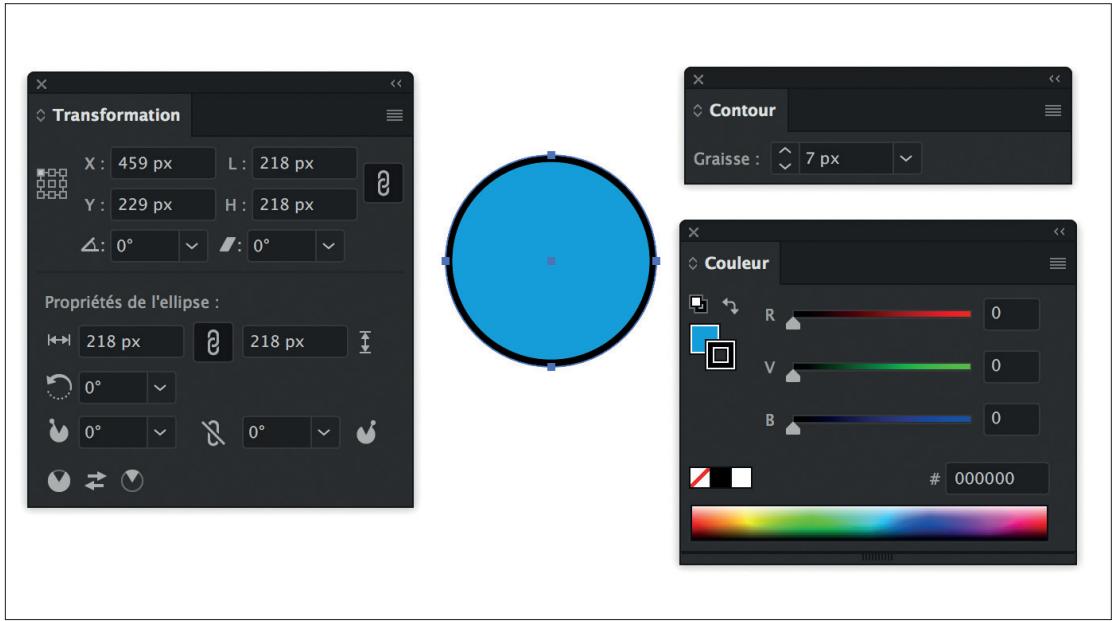


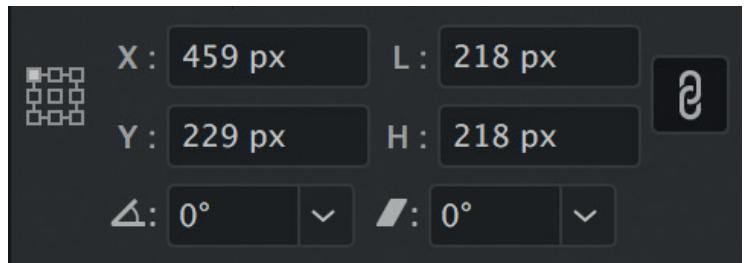
fig.8

fig .10 Dans Illustrator, les paramètres modifiables sont représentés dans des encarts.

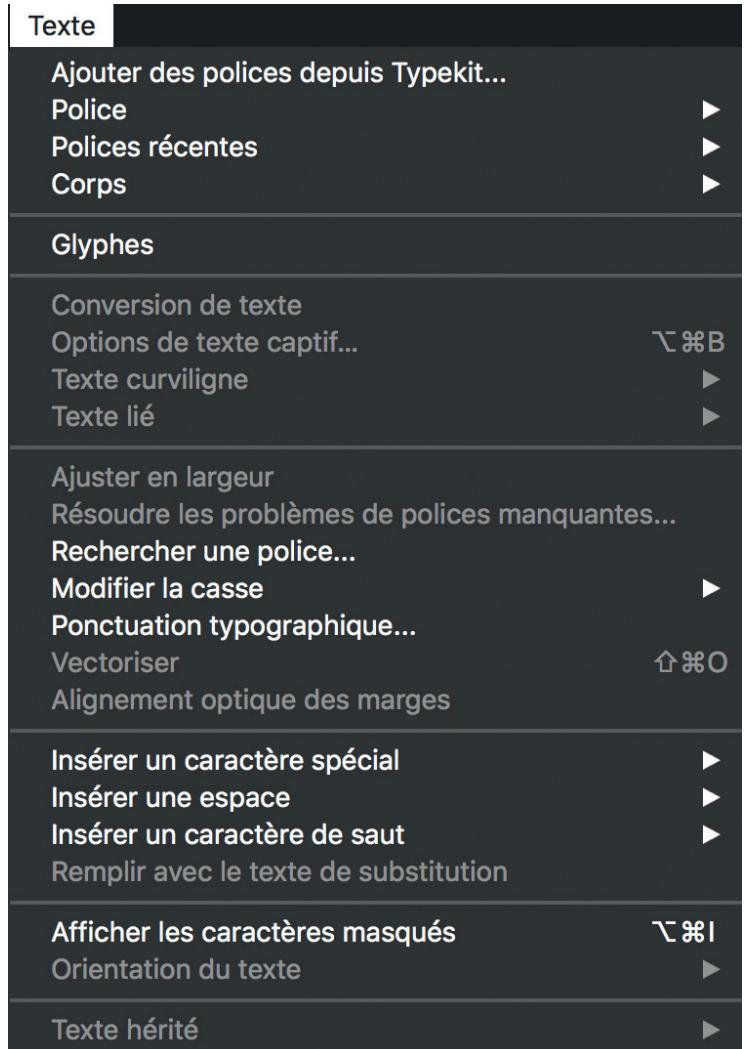
Ces paramètres possèdent des descriptions textuelles ou imagées. Capture d'écran de l'interface Illustrator CC 2018.

fig .11 Exemple de représentation classique d'options possibles, mais inaccessibles dans les menus d'interfaces graphiques.

Ce qui n'est pas applicable par rapport au contexte sélectionné par l'utilisateur est présenté en gris clair, alors que les actions utilisables sont en blanc. Capture d'écran de l'interface Illustrator CC 2018.



fig·10



fig·11

fig .12 Fenêtre du filtre « Déformation/Onde » dans Photoshop.

Une nouvelle interface se présente à l'utilisateur, dans la fenêtre « Onde ». Les autres fonctions du logiciel sont inaccessibles. Capture d'écran de l'interface Photoshop CC 2018.

fig .13 Exemple d'assignation de propriétés d'un élément lié à un autre.

Ici, en traçant un trait (bleu) entre le paramètre « position » du « calque_2 » au « calque_1 », la position de ce dernier sera identique à la position du « calque_2 ». Capture d'écran de l'interface Adobe After-Effects CC 2018.

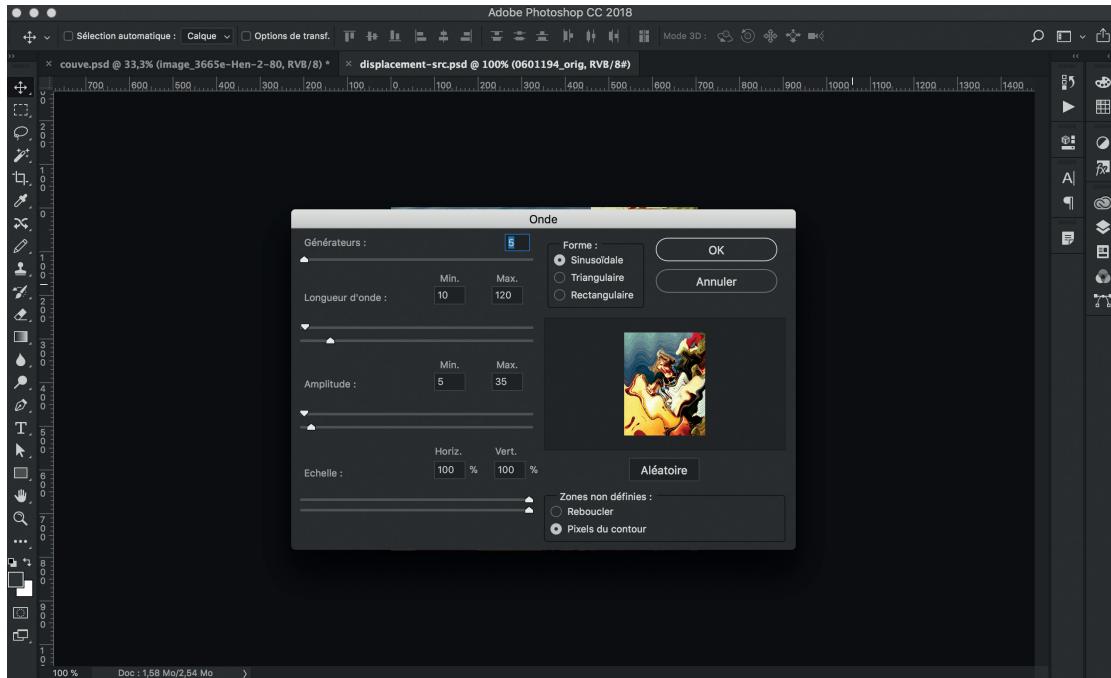


fig.12

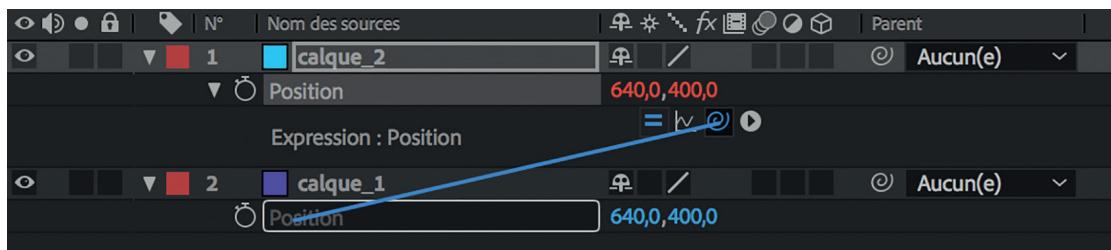
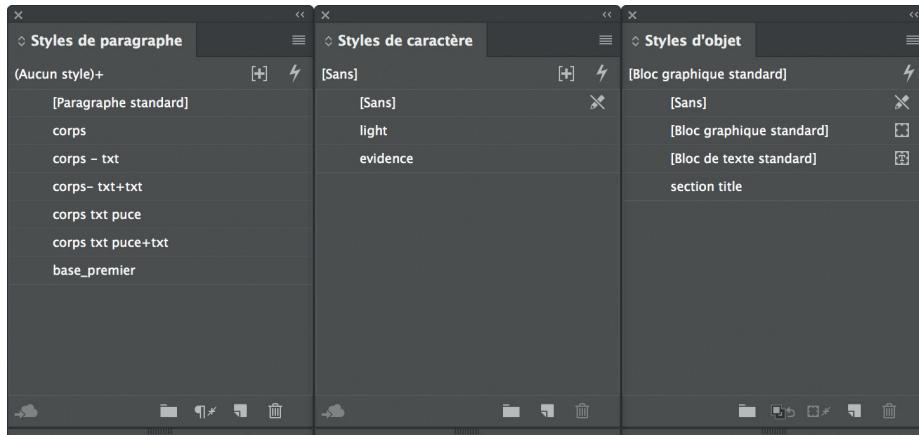


fig.13

fig.14 Représentations graphique des styles de paragraphes sur les interfaces Photoshop CC 2018 (en haut) et Sketch App (en bas).

fig.15 Capture le la fenêtre «Action» de l'interface Photoshop CC 2018. Cette section permet à un utilisateur d'enregistrer ses actions dans le logiciel.



No Text Style

h2 center
≡ Helvetica Neue Medium — 25pt

h3 center
≡ Helvetica Neue Medium — 15pt

h3 center white
≡ Helvetica Neue Medium — 15pt

h3 left
≡ Helvetica Neue Medium — 15pt

p center
≡ Helvetica Neue Medium — 10pt

p left
✓ ≡ Helvetica Neue Medium — 10pt

p left red
≡ Helvetica Neue Medium — 10pt

Create new Text Style
Organize Text Styles...

This screenshot shows the "Text Style" panel in Sketch. It lists various text styles with their corresponding font, size, and color. The "p left" style is currently selected, indicated by a checkmark. Other styles listed include "h2 center", "h3 center", "h3 center white", "h3 left", "p center", and "p left red". At the bottom of the panel, there are buttons for "Create new Text Style" and "Organize Text Styles...".

fig.14

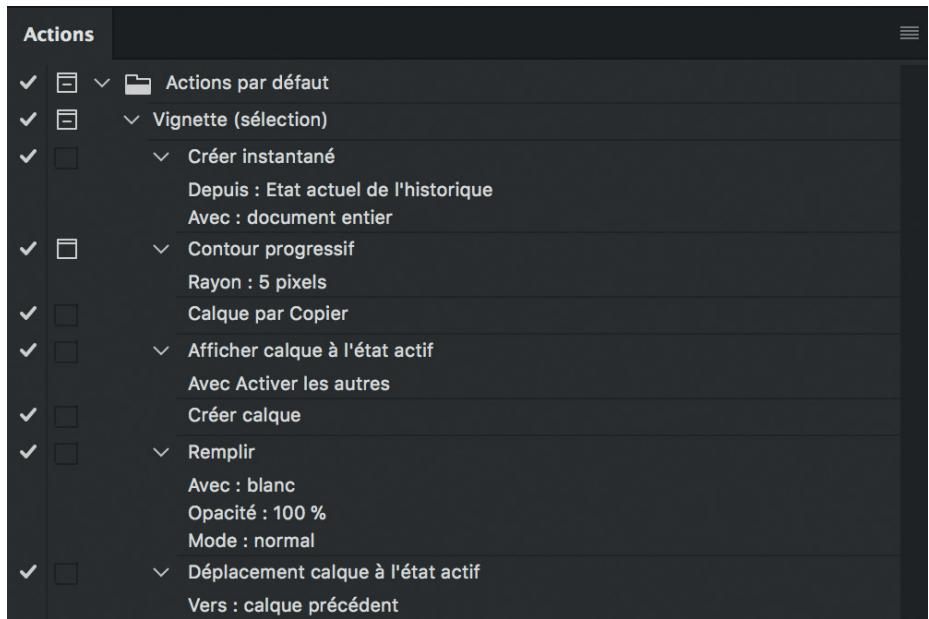


fig:15

fig .16 Dans l'application Photoshop CC 2018, le survole d'un icône déclenche une vidéo miniature pour présenter sa fonction.

fig .17 Fichier Illustrator (en haut) et Inkscape (en bas) pour la création d'une illustration vectorielle identique.



fig·16

```

7325 0 D
7326 0.232028678059578 0.170626372098923 0.181155100464821 0.00830090790987 0.8 0.8 0.8 Xa
7327 133.104461669922 447.455856323242 m
7328 456.157498667969 447.455856323242 L
7329 456.15749667969 251.877853393555 L
7330 133.104461669922 251.877853393555 L
7331 133.104461669922 447.455856323242 L
7332 f
7333 %_ArtDictionary :
7334 %_XMLUID : (rect3068) ; (AI10_ArtUID) ,
7335 %_0 /Int (SVG_ShapeType) ,
7336 %_ ;
7337 %_
7338 0 Ae
7339 u
7340 0.55986875295639 0.453986406326294 0.448859363794327 0.326848238706589 0.4 0.4 0.4 Xa
7341 42.1248247367148 447.455834597706 m
7342 78.390690298716 420.298661469213 L
7343 78.390690298716 271.634216425744 L
7344 42.1248247367148 251.877833575367 L
7345 42.1248247367148 447.455834597706 L
7346 f
7347 %_ArtDictionary :
7348 %_XMLUID : (path3087) ; (AI10_ArtUID) ,
7349 %_7 /Int (SVG_ShapeType) ,
7350 %_XMLNode :
7351 %_Dictionary :
7352 %_XMLNode :
7353 %_Dictionary :
7354 %_ ; (xmlnode-attributes) ,
7355 %_Array :
7356 %_ ; (xmlnode-children) ,
7357 %_ 2 /Int (xmlnode-nodeType) ,
7358 %_(cccc) /String (xmlnode-nodeValue) ,
7359 %_(sodipodi:nodetypes) /String (xmlnode-nodeName) ,
7360 %_ ; (sodipodi:nodetypes) ,
7361 %_XMLNode :
7362 %_Dictionary :
7363 %_ ; (xmlnode-attributes) ,

```

```

1223     xml:space="preserve"><tspan
1224         style="font-size:16px;fill:#ffffff"
1225         y="225.5"
1226         x="401.66632"
1227         id="tspan3962"
1228         sodipodi:role="line">Hover</tspan></text>
1229     </g>
1230     <g
1231         id="g5548"
1232         transform="translate(-11,-58.000001)">
1233         <path
1234             inkscape:connector-curvature="0"
1235             id="path3035"
1236             d="m 175.73908,640.17588 390.88494,0"
1237             style="fill:none;stroke:#aaaaaa;stroke-width:0.73804379px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1"
1238             transform="translate(0,-152.36217)" />
1239         <path
1240             sodipodi:nodetypes="cssssc"
1241             inkscape:connector-curvature="0"
1242             id="path3036"
1243             d="m 36.936349,640.55313 -33 c 0,-2.216 1.784,-4 4,-4 l 131.000001,0 c 2.216,0 4,1.784 4,4 l 0,33"
1244             style="fill:#ffffff;stroke:#aaaaaa;stroke-width:1;stroke-miterlimit:4;stroke-opacity:1;stroke-dashoffset:0"
1245             transform="translate(0,-152.36217)" />
1246         <path
1247             sodipodi:nodetypes="cssssc"
1248             inkscape:connector-curvature="0"
1249             id="rect3809-1"
1250             d="m 180.8158,639.86282 -33 c 0,-2.216 1.784,-4 4,-4 l 131,0 c 2.216,0 4,1.784 4,4 l 0,33"
1251             style="fill:#cccccc;stroke:#aaaaaa;stroke-width:1;stroke-miterlimit:4;stroke-opacity:1;stroke-dashoffset:0"
1252             transform="translate(0,-152.36217)" />
1253         <text
1254             style="fill:#cccccc;stroke:#aaaaaa;stroke-width:1;stroke-miterlimit:4;stroke-opacity:1;stroke-dashoffset:0"
1255             d="m 324.8158,639.86282 -33 c 0,-2.216 1.784,-4 4,-4 l 131,0 c 2.216,0 4,1.784 4,4 l 0,33"
1256             id="path3848"
1257             inkscape:connector-curvature="0"
1258             sodipodi:nodetypes="cssssc"
1259             transform="translate(0,-152.36217)" />
1260         <text
1261             sodipodi:linespacing="125%">

```

fig17

fig .18 Exemple du code (ci-contre) généré par le plug-in Sketch2AE pour décrire le contenu réalisé dans l'application Sketch (ci-dessous).

Sketch2AE est composé de deux plug-ins développés par Google, dont l'un fonction au sein de l'application Sketch et l'autre d'After-Effects. Ils permettent de récupérer dans After-Effects du contenu créé dans Sketch. Pour fonctionner, le plug-in intégré à Sketch génère un code descriptif de ce que nous voulons transposer dans After-Effects, ici le code à gauche. Ce code doit ensuite être copié dans le plug-in intégré à After-Effects et reproduira les éléments graphiques décrits dans ce code.

L'intérêt ici est de montrer comment des informations sont transportées d'un logiciel à un autre. On peut par exemple voir ici que l'on retrouve dans le code à gauche, le nom de la font utilisée dans l'image à droite (GTAmerica-Regular). On retrouve le contenu du texte, la taille est la position de celui, etc.

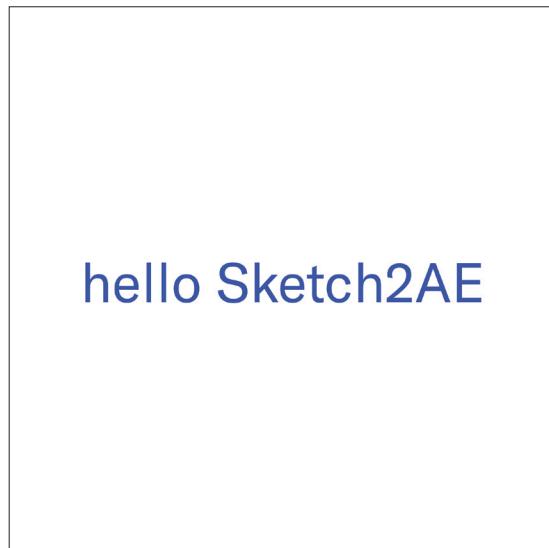


fig.18

```
[  
  {  
    "type": "MSArtboardGroup",  
    "name": "Artboard",  
    "backgroundColor": [  
      1.000000,  
      1.000000,  
      1.000000,  
      1.000000  
    ],  
    "size": [  
      500,  
      500  
    ]  
  },  
  {  
    "type": "MSTextLayer",  
    "name": "hello Sketch2AE",  
    "isVisible": 1,  
    "stringVal": "hello Sketch2AE",  
    "size": [  
      370,  
      60  
    ],  
    "opacity": 100,  
    "position": [  
      65,  
      220  
    ],  
    "fontName": "GTAmerica-Regular",  
    "fontSize": 50,  
    "tracking": 0,  
    "justification": 2,  
    "lineHeight": 57,  
    "hasFill": 1,  
    "textColor": [  
      0.262745,  
      0.243137,  
      0.878431,  
      1.000000  
    ],  
    "fill": null,  
    "stroke": null,  
    "shadow": null,  
    "innerShadow": null,  
    "blendMode": 0  
  }  
]
```


2— interfaces textuelles

Comme le constate John Maeda, « dessiner avec la souris sur l'écran d'ordinateur est très ludique, mais [...] le tracé d'un trait au crayon ne se différencie nullement du même tracé avec la souris »³⁰.

Muriel Cooper questionne dès les années 70 les méthodes de création liée aux interfaces graphiques prochent des paradigmes développés au Xerox PARC. Elle voit ces concepts comme un état de transition pour les interfaces³¹. À travers le Visual Language Workshop qu'elle cofonde avec Ron MacNeil en 1973 au MIT, et le MIT Media Lab (1985), elle recherche de nouveaux principes ou vocabulaires pour caractériser un nouveau médium³². Muriel Cooper prévoit les problèmes liés à la quantité importante de données auxquelles nous avons accès aujourd'hui. Elle décrit le designer graphique comme le possible créateur de filtres pour la représentation de celles-ci³³. En s'inspirant de l'ouvrage *Designing Programmes* de Karl Gerstner, elle défend le fait que puisqu'il « est impossible de connaître les contenus à l'avance, le designer doit concevoir son système en tenant compte des informations potentielles »³⁴. Le Designer graphique est créateur de « système », de « principes » graphiques, et elle fera travailler ses étudiants directement avec la programmation.

C'est dans ce contexte que John Maeda, qui fut étudiant de Cooper, a exposé l'intérêt d'une interface textuelle à travers son ouvrage *Design By Numbers* et mis au point un logiciel homonyme. Conçu dans un but pédagogique il initie les designers graphiques et artistes à la « créa-

30 John Maeda, « Design By Numbers » cité par P. Donaldson, « La création d'algorithmes dans le design génératif », HEAD—Genève, Genève, Suisse, 2014, p. 17.

31 Muriel Cooper, « Computers and Design », *Design Quarterly*, n° 142, Minneapolis, États-unis, Walker Art Center, 1989, p. 17.

32 Muriel Cooper à propos du Visual Language Workshop : « we're exploring or researching new design characteristics or principles or vocabularies that will characterize what is essentially a new medium » dans « Cooper, Muriel. Conversation with Ellen Lupton », 1994, Ellen Lupton, [en ligne], consulté le 5 janvier 2018, www.elupton.com/2010/07/cooper-muriel

33 Nolwenn Maudet, « Muriel Cooper Information Landscape », Back Office n° 1, Design graphique et pratique numérique, éd. B 42, Paris, France, 2017, p. 105.

34 *ibid.*

tion computationnelle »³⁵. Pour en simplifier l’appréhension, *Design By Numbers* fait fortement appel au paradigme de la métaphore. Nous retrouvons dans cette interface textuelle, les concepts de page, de pinceau, de couleur à sélectionner.

```
paper 0
pen 100
line 20 20 40 80
```

Nous voyons que la page (paper) est blanche (0). On prend un stylo (pen). On lui applique la couleur noire (100) et on trace une ligne avec les coordonnées [x: 20, y: 20] pour le point de départ, et [x: 40, y: 80] pour le point d’arrivée (line 20 20 40 80).

Langage informatique

L’utilisation d’un logiciel grâce à une interface textuelle nécessite un langage établi, un langage informatique. Par exemple, les interfaces de ligne de commande sont généralement utilisées à travers le langage Bash sur les systèmes UNIX, *Processing* utilise le langage Java. Ces langages peuvent être de natures différentes. Nous en retiendrons trois pour les analyses suivantes :

- les langages de programmation, qui permettent la création d’algorithmes, avec des concepts de variables, fonctions, boucles, conditions;
- les langages de balisage, qui permettent la représentation de données structurées et hiérarchisées;
- les langages de feuille de style, qui permettent la déclaration de mise en forme d’un contenu lui-même déclaré dans un langage de balisage.

Si les langages de balisage et de feuille de style permettent ensemble de créer une représentation visuelle, seuls les langages de programmation peuvent créer du contenu, à travers un algorithme.

35 John Maeda, « What Is DBN? », Design By Numbers, [en ligne], consulté le 13 novembre 2018, www.dbn.media.mit.edu/whatisdbn.html

Le rôle des interfaces textuelles à différents stades de travail du design graphique.

Recherche et expérimentation par tâtonnement.

Une interface textuelle ne présente pas les possibilités du programme. L'utilisateur a besoin de connaître le langage avant de pouvoir l'utiliser. Il devra donc se documenter pour assimiler les interactions possibles avec un logiciel qui se présente sous cette forme.

Une autre limitation de l'interface textuelle lors de la phase d'expérimentation, sa nature descriptive qui constraint à un travail par projection³⁶. Effectivement, une fois que l'utilisateur a décrit ce qu'il attend du programme, il doit lui « donner » ses instructions pour qu'il puisse interpréter cette description.

Processing est un logiciel de création graphique interfacé avec un langage de programmation homonyme. Il a été créé par Ben Fry et Casey Reas en 2001, lorsqu'ils étaient étudiants au MIT Media Lab au sein du groupe de recherche en esthétique et calcul de John Maeda. La plupart des concepts de *Processing* découlent des travaux de Muriel Cooper et plus directement du projet *Design By Numbers* de John Maeda. Le logiciel lui reprend ainsi les notions de métaphore vues en introduction. Si nous voulons dessiner un triangle blanc avec *Processing*, il faudra tout d'abord configurer les propriétés de dessin. Pas de contours, avec « `noStroke()` » et un remplissage blanc, avec « `fill(255)` ». Ensuite, nous pourrons dessiner le triangle, « `triangle(18, 18, 18, 360, 81, 360)` ».

```
noStroke();  
fill(255);  
triangle(18, 18, 18, 360, 81, 360);
```

36 David Canfield Smith, « Pygmalion : A creative Programming Environment », 1975.

Bret Victor critique cette approche en expliquant qu'elle ne permet pas facilement de reprendre du code d'un cas de figure à l'autre, puisque les propriétés d'une forme sont justement dépendantes du contexte dans lequel elle a été créée :

Jürg Lehni, avec Paper.js, se positionne avec une logique différente de Processing. Chacun des éléments tracés est un objet avec des propriétés qui lui sont directement assignées. Ainsi, si je duplique un élément graphique dans un nouveau programme, celui-ci aura exactement les mêmes paramètres. Nous sommes plus dans une logique de programmation que l'on appelle « programmation orientée objet ». Ce modèle repose sur le fait de décomposer la rédaction d'un programme sous forme de « briques » représentant une idée, un concept. Chacune de ces briques est un objet qui a une structure et un comportement qui lui est propre et paramétrable, comme un objet du monde physique.

Nous constatons que dans ce modèle de programmation, on retrouve certains des grands principes qui construisent les interfaces graphiques vu précédemment. Jürg Lehni cherche d'ailleurs à avoir des logiques de représentation proches de ces interfaces. Il permet par exemple de rassembler des éléments graphiques dans des groupes et dans des calques, à la manière d'Illustrator. Ce qui est intéressant c'est que l'interface textuelle de *Paper.js* a été pensée pour des designers graphiques habitués à travailler avec des logiciels à interface graphique. Jürg Lehni crée une continuité logique entre l'utilisation de son application et des logiciels conventionnels de design graphique, là où

Processing crée des modes de réflexion différents. Ce qui peut s'avérer plus efficace dans une phase de recherche puisque l'utilisateur peut se projeter, « comment je ferais dans Illustrator », et l'écrire avec Paper.js.

Les interfaces textuelles ont souvent une approche plus idiomati-
que de la manière dont le logiciel qu'elle représente fonctionne.
Créons et appliquons un filtre svg sur une image, et comparons-la
à la manière dont nous pourrions obtenir un résultat identique avec
Photoshop *fig .19*. Si Photoshop permet d'expérimenter comme nous
le ferions dans le monde physique, l'interface textuelle nous permet
d'expérimenter comme un ordinateur nous permet de nous projeter.
Avec le filtre svg, nous sommes en mesure de comprendre que l'image
d'origine est construite selon des valeurs rouge (R pour Red), vert (G
pour Green) et bleu (B pour Blue), et que de les transposer, entre diffé-
rents canaux (d'entrées et de sorties) permet d'arriver à de nombreux
résultats *fig .20*. Comprendre comment une application fonctionne
peut pousser à expérimenter avec.

Cependant, tout comme des choix par menu, l'interface textuelle offre elle aussi des possibilités, qui selon les langages, peuvent être aussi limitées que la sélection par menu. Les filtres CSS peuvent prendre seulement l'une de ces valeurs :

- normal;
- multiply;
- screen;
- overlay;
- darken;
- lighten;
- color-dodge;
- color-burn;
- hard-light;
- soft-light;
- difference;
- exclusion;
- hue;
- saturation;
- color;
- luminosity.

Les filtres CSS sont un exemple extrême, mais qui illustre le fait que l'interface textuelle reste un moyen d'accéder uniquement à ce que le programme peut faire.

Nous voyons que même textuelle, l'interface est toujours la présentation de programmes sous-jacents. Ainsi, il y a toujours une limite finie sur ce qui est présenté à son utilisateur.

Répétition de tâches ou d'actions.

L'interface textuelle est descriptive. Pour mettre à jour plusieurs éléments, le changement de valeur d'une propriété est ainsi très vite répercuté. Les langages de programmation permettent grâce aux méthodes de boucles et de conditions de créer des éléments plusieurs fois et de leur assigner des paramètres conditionnels.

Cependant, toutes les interfaces textuelles ne peuvent pas fonctionner de cette manière. Si nous prenons l'exemple des langages de feuille de style ou des langages de balisage, il n'est pas possible de créer des boucles ou des conditions. Il faudra réécrire le texte décrivant l'action que nous voulons répéter, ou passer par un langage de programmation pour générer ledit texte. Par exemple, le langage de programmation SASS permet de générer du CSS. Il prend en charge les notions de boucles, de conditions et de variables. L'interface textuelle n'est donc pas la garantie d'une simplification dans des tâches répétitives.

LaTeX est un langage de balisage qui permet de hiérarchiser un document. Le concept du logiciel du même nom est de permettre à un utilisateur de se concentrer sur le fond, est de laisser la mise en page se faire par LaTeX, au travers de « thèmes ». Si un mot doit être mis en gras, ou en italique, nous devons le placer entre les balises adaptées :

```
\textit{text italique},  
\textbf{text gras}
```

Il est possible de changer les paramètres associés à ces balises, comme le type de famille de caractères, de cette manière :

```
BoldFont={Minion Pro Bold},  
ItalicFont={Minion Pro Italic}
```

Cette logique est identique dans le rapport entre le HTML (interface de hiérarchisation de contenu, langage de balisage) et le CSS (interface textuelle de mise en forme, langage de feuille de style). Le site

Zen Garden illustre les possibilités de mise en forme avec CSS³⁷ par rapport à un seul document HTML. Il met en évidence le fait que le designer graphique peut établir des règles de mise en forme par rapport à un contenu formaté. Si le contenu est amené à être modifié, la manière dont il sera représenté restera identique.

L'interface textuelle semble donc permettre une séparation du fond et de la forme grâce à son approche déclarative (en opposition à la manipulation directe des interfaces graphiques vues précédemment). Il est donc possible de déclarer une mise en forme générique liée à un type de contenu dont la hiérarchie est préétablie.

Cependant, la forme est liée au langage de hiérarchisation. Reprenons l'exemple de couple CSS/HTML. Disons que nous voulons que les mots en gras («strong») soient en rouge, et que les mots à mettre en emphases («em») soient en bleu :

```
<em>text emphase</em>
<strong>text en gras</strong>
```

```
em {
    color: blue;
}
strong {
    color: red;
}
```

text emphase **text en gras**

Si à présent nous voulons pouvoir placer du texte en *bold* dans du texte en emphase, nous placerions la partie dans la partie :

```
<em>text emphase et <strong>text en gras emphase</strong></em>
```

```
em {  
    color: blue;  
}  
strong {  
    color: red;  
}
```

text emphase et text en gras emphase

Or si nous si nous voulons que le texte en *bold* dans du texte en emphase devienne lui aussi bleu, il faudra mettre à jour la déclaration de style faite avec le langage CSS:

```
<em>text emphase et <strong>text en gras emphase</strong></em>
```

```
em {  
    color: blue;  
}  
strong {  
    color: red;  
}  
em > strong {  
    color: blue;  
}
```

text emphase et text en gras emphase

En résumé, contrairement à une interface graphique, la mise en forme d'un contenu se fait dans un contexte différent de celui où on le modifie. Nous avons du code pour le fond (ici me HTML), et du code pour la forme (le CSS). Cependant, ces deux contextes sont dépendants l'un de l'autre, et nous pouvons voir que, comme à travers une interface graphique, il est toujours difficile de séparer fond et forme.

mise à jour et partage entre logiciels

Nous avons vu que les interfaces textuelles étaient toujours liées à un langage informatique. Si besoin est de passer par une autre application, il sera possible d'importer un programme dans un autre (si les langages le permettent), ou bien d'exécuter un programme extérieur.

```
import processing.pdf.*;           importation d'un programme  
launch("/Applications/Process.app"); execution d'un programme
```

Ainsi, contrairement aux logiciels d'interfaces graphiques, il n'est pas nécessaire ici de sortir de l'environnement de l'interface. L'utilisateur fait appel à un programme, ou logiciel.

Si nous reprenons les exemples de latex ou du couple CSS et HTML, où fond et forme sont manipulables séparément, l'interface de rendu visuelle reste liée à la logique de l'interface textuelle de hiérarchisation. Il persiste donc une certaine dépendance entre fond et forme.

Comme dans un logiciel à interface graphique de type WIMP, WYSIWYG de manipulation directe, il est nécessaire de passer par une phase de conversion pour passer d'un programme à un autre, comme en témoigne le logiciel *Pandoc*³⁸.

En conclusion.

La communication via interface textuelle avec un logiciel peut être de natures très différentes. Nous avons vu que les langages informatiques n'étaient pas forcément des langages de programmation. Ainsi, comme nous l'avons vu avec le cs, des problèmes liés à la récursivité, la duplication d'actions peuvent apparaître.

L'interface textuelle, à l'inverse de l'interface graphique, ne présente pas les possibilités d'un programme. L'utilisateur d'une interface textuelle se retrouve dans un lieu unique. Là où l'interface graphique

³⁸ Pandoc est un logiciel qui permet de convertir des fichiers d'un format de balisage à un autre. Il permet par exemple la conversion d'un document HTML en PDF.

va pouvoir présenter à la fois un rendu, des options, des valeurs de propriétés sur un élément précis, sélectionné, l'interface textuelle est une suite déclarative d'un état général.

L'approche déclarative d'une interface textuelle permet de décrire des actions plutôt de réaliser ces actions. Quand les fichiers de travail de logiciels qui utilisent une interface graphique sont possiblement difficiles à lire et/ou à interpréter, les logiciels qui passent par l'interface textuelle s'appuient sur des fichiers intelligibles.

Les logiciels à interface graphique sauvegardent un état, l'interface textuelle sauvegarde une définition de cet état.

`fig .19` Cr ation d'un effet identique avec Photoshop (en haut) et un filtre
SVG (en-bas).

`fig .20` Explication du fonctionnement des filtre SVG comme vu `fig .19`



`fig19`

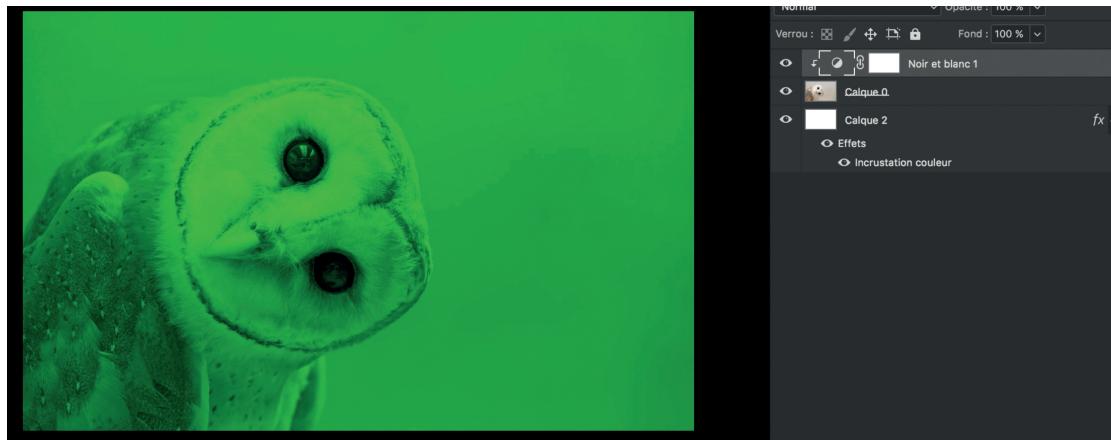


fig.20

3— pour un environnement de travail

Nous avons vu les avantages et inconvénients des interfaces textuelles et graphiques par rapport au travail du designer graphique. Cependant, l'édition par interface textuelle se fait généralement au travers d'interfaces graphiques. Dans un tel environnement, est-il possible d'allier les points forts des deux types d'interfaces pour améliorer les méthodes de conceptions graphiques assistées par ordinateur?

Smalltalk a été à l'origine conçu par Alan Kay. Smalltalk-80, première version de Smalltalk accessible en dehors de Xerox Parc, est distribuée avec un «environnement de travail interactif» [Interactive Programming Environment]. Ce programme d'édition «est une interface polyvalente, conçue pour faciliter la création et la manipulation» lors de la réalisation d'un programme avec le langage Smalltalk-80. On y retrouve des fonctions de compilateur, débogueur et éditeur de texte. Différentes zones d'affichage et d'interactions pour séparer différentes fonctions comme l'édition de texte ou la visualisation d'un graphique, ainsi que la présence de menus. L'utilisateur peut également interagir avec le programme à l'aide d'un clavier et d'une souris³⁹.

39 Adele Goldberg, «Smalltalk-80. The Interactive Programming Environment», Xerox Palo Alto Research Center, 1984.

«The user interface to the Smalltalk-80 system is a multipurpose interface, designed to facilitate text and graphics creation and manipulation, program development, and information storage and retrieval. You, as the user create, manipulate, save, or retrieve, either a visual form or specific information within such a form. You interact with the system on a bitmapped display screen, using a typewriter keyboard and a pointing device. The Smalltalk-80 system includes utilities typical of a computer's operating system: compiler, debugger, text editor. These are brought together on the display screen in the form of a collection of rectangular areas or views of information. One rectangular area might view the text of a program under development. Changing the text is accomplished by pointing to the parts to be changed, and typing new text or issuing an editing command such as delete. When the text is considered correct, an accept or compile command is issued. If any syntactic errors are detected, they will be indicated in the text in the window so that they can be corrected right away. Once all syntactic errors are corrected, the text is compiled into object code and linked into the system, as the system is running. The new code can then be tested by typing an appropriate expression, selecting it, and issuing an evaluate command. This process is immediate; no exchange of editing, compiling, filing, or executing modes is required.»

HyperCard, développé par Apple de 1987 à 2004, est aussi un logiciel de création qui est assisté par un environnement graphique conçu pour l'interface textuelle du langage (HyperTalk). Ce qui est intéressant est le fait qu'une zone est réservée à la représentation visuelle, ou nous retrouvons les icônes d'outils tels que le crayon *fig .21*.

Dans un éditeur de code comme Visual Studio Code par exemple, nous pouvons voir que l'utilisateur a accès à un ensemble de fonctionnalités pour l'assister dans l'utilisation de son interface textuelle *fig .22*.

Si un concept d'environnement peut assister le designer dans l'utilisation d'interfaces textuelles, il est important que celui-ci soit adapté à sa pratique, à l'image d'HyperCard. Dans cette logique, nous pouvons citer Adobe Animate CC. L'environnement de travail est familier à l'ensemble des outils Adobe. L'utilisateur peut créer du code dans une fenêtre textuelle, gérer des animations avec des « timelines » proches de la logique d'After-Effects, et dessiner directement dans la fenêtre de rendu *fig .23*.

Cependant, est-il possible de rendre convivial un environnement de programmation, avec des principes et des modes de représentation que le designer connaît, en étant dans des concepts de visualisations nouveaux ?

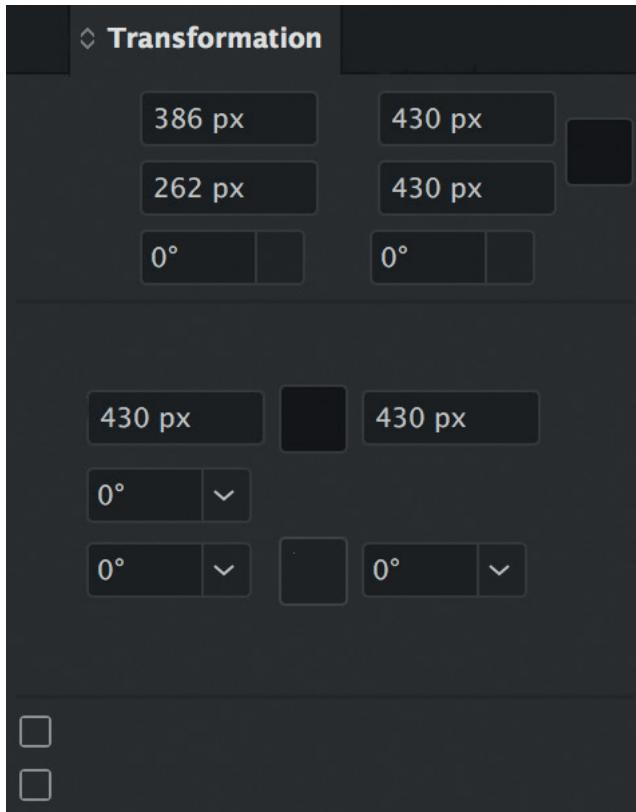
Aider à comprendre.

Lors de sa conférence « Inventing on Principle »⁴⁰ de 2012 à l'événement Canadian University Software Engineering Conference (CUSEC), Bret Victor fit la démonstration de modèles d'environnement pour l'assistance de création visuelle à l'aide d'un langage de programmation. « Learnable programming. Designing a programming system for understanding programs »⁴¹ est un article qui en détaille les principes. Il démontre l'intérêt d'une interaction entre interface graphique et interface textuelle. L'accent est mis sur un dialogue entre ces deux interfaces.

40 Bret Victor, « Inventing on Principle », conférence CUSEC 2012, Youtube, [en ligne], www.youtube.be/PUV66718DII

41 Bret Victor, « Learnable programming. Designing a programming system for understanding programs », Bret Victor, beast of burden [en ligne], 2012, consulté le 13 novembre 2018, www.worrydream.com/#!/LearnableProgramming

Dans une logique d'assistance, l'éditeur doit pouvoir aider l'utilisateur à comprendre l'interface textuelle qu'il manipule. Bret Victor montre par exemple que le survol d'une propriété éditée dans un langage pourrait laisser apparaître un étiquetage donnant des détails sur ce qu'elle contrôle ^{fig .24}. L'interface textuelle n'est pas nécessairement complexe, il faut qu'elle soit légendée, expliquée. Pour reprendre une remarque d' Anthony Masure⁴², dans les interfaces graphiques vues en première partie, si les propriétés éditableables n'étaient pas expliquées, nous ne pourrions pas les comprendre:



Ce que je veux mettre en évidence ici c'est le fait que ce n'est pas l'interface graphique qui nous apporte des précisions sur ce qu'on manipule, mais les informations et légendes qui la composent.

42 A. Masure, « Adobe - Le créatif au pouvoir? », STRABIC.FR, 24-juill-2011. [En ligne]. Disponible sur: <http://strabic.fr/Adobe-le-creatif-au-pouvoir>.

Manipulation directe et WYSIWYG.

Gal Sasson a créé une fonctionnalité pour l'éditeur Processing (programme d'édition du langage homonyme), accessible via le menu «Sketch/Tweak». Celle-ci permet de changer les valeurs des propriétés dans le texte et de voir en temps réel les modifications visuelles dans la zone de rendu de Processing. Ces interactions se font grâce à des composants graphiques. Par exemple, la modification de la valeur d'une couleur se fera en cliquant sur celle-ci dans l'interface textuelle, ce qui fera apparaître une palette de couleurs [fig .25](#).

Yining Shi a créé un environnement pour P5.js. Il permet de venir interagir directement avec les éléments dessinés dans la zone de rendu, avec l'interaction de la souris [fig .26](#), à la manière dont on pourrait le faire dans Illustrator [fig .27](#).

En utilisant l'interface de débogage de Google Chrome, il est possible de survoler des éléments de code avec le curseur [fig .28](#). Ceci met en surbrillance dans la zone de rendu visuelle, l'élément qui correspond au texte survolé. Cette méthode permet à un utilisateur de comprendre ce avec quoi il va interagir.

Nous voyons que grâce à l'interaction entre l'environnement et le langage textuel, il est possible de retrouver ses marques pour se lancer dans des recherches par tâtonnement comme le designer graphique peut le faire avec les interfaces vues en première partie. De plus, ces systèmes permettent à l'utilisateur de créer avec les mêmes systèmes de pensée qu'il possède avec ses outils habituels.

Dialogue entre trois entités, l'utilisateur, l'interface textuelle, l'interface graphique.

Dans le concept d'environnement de programmation, le dialogue se fait entre trois entités. L'interface graphique, l'interface textuelle et l'utilisateur de ces deux interfaces qui communiquent ensemble. Ainsi, si le langage de l'interface textuelle se doit d'être compréhensible pour l'homme, il doit aussi «aider» l'interface graphique.

Par exemple, des langages statiquement typés demandent à leurs utilisateurs de déclarer, lors de leur écriture, le type d'élément manipulé. par avec le langage Typescript:

```
var un_nombre: number  
var une_chaine_de_caractere: string
```

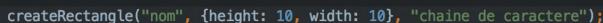
On peut ainsi voir que «un_nom» est un élément qui est un «number», donc un numéro. Si l'utilisateur met une chaîne de caractère (type «string») dans «un_nom», nous pouvons déduire qu'il y a une erreur. Grâce à cette méthode, l'éditeur est lui aussi capable de dire qu'une erreur est présente, et de mettre à jour l'interface graphique pour le signaler:



```
createRectangle( name: "nom", size: {height: 10, width: 10}, position: "chaine de caractere");
```

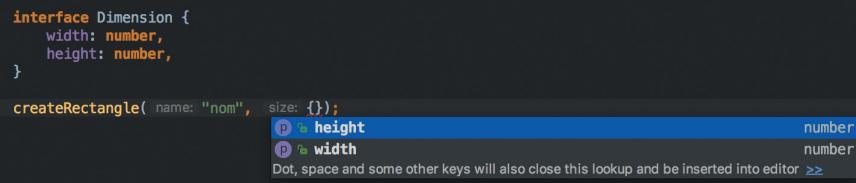
Argument type "chaine de caractere" is not assignable to parameter type GraphicPosition

J'ai pris ici l'exemple de Typescript, car c'est un langage qui est extrêmement proche de JavaScript, à la différence que ce dernier n'est pas statiquement typé. Si nous regardons la même erreur avec le langage JavaScript, nous voyons que l'éditeur n'est pas en mesure de nous dire qu'une erreur est présente dans les lignes de code :



```
createRectangle("nom", {height: 10, width: 10}, "chaine de caractere");
```

En allant encore plus loin, un langage statiquement typé comme Typescript permet de définir des types, eux-mêmes composés de sous types. Pour être plus clairs, expliquons qu'un type «Dimension» est composé d'une information de largeur (numérique), et une information de hauteur (également numérique). On peut ainsi déclarer une infinité de types. L'éditeur IntelliJ est par exemple capable de comprendre ces définitions, et de nous présenter les options possibles en fonction de l'emplacement du curseur dans le code :



```
interface Dimension {
    width: number,
    height: number,
}

createRectangle( name: "nom", size: {});
```

Possible completions for 'height':
Possible completions for 'width':
number
number
Dot, space and some other keys will also close this lookup and be inserted into editor >>

Nous pouvons ainsi retrouver dans une interface textuelle, l'ensemble des possibilités d'un programme, sans devoir nous en souvenir, grâce à une présentation d'options sous forme de liste.

En conclusion.

Il est intéressant de noter qu'avec les recherches de Bret Victor et l'interaction possible entre une interface graphique et une interface textuelle, c'est que nous pouvons nous trouver dans une approche idiomatique de l'utilisation d'un programme. L'ensemble des solutions proposées permettent au designer de créer avec une interface textuelle de la même manière qu'il le faire avec des logiciels de la suite Adobe. En simplifiant l'accessibilité à un langage de programmation, un environnement de travail cohérent pour le designer lui permettrait de voir l'interface textuelle comme un outil en plus, qu'il peut facilement prendre en main.

«Globalement, nous sommes passés [du paradigme] de la technologie à l'usage de la métaphore, et nous nous rendons à présent compte de l'importance de la conception idiomatique. Bien que chacune de ces trois phases soit évidente dans le domaine du logiciel contemporain, le paradigme de la métaphore est le seul à avoir été popularisé. Nous lui payons donc un lourd tribu et, trop souvent, nous entravons la création d'interfaces vraiment efficaces par un usage erroné des métaphores.»⁴³

43 Alan Cooper, «The myth of metaphor», Visual Basic Programmer's Journal, 1995. Traduction de Marc Wathieu, «Le mythe de la métaphore», multimedialab.be [en ligne], consulté le 13 novembre 2018, www.multimedialab.be/doc/alan_cooper.htm

fig .21 Outils de dessin et zone de rendu du logiciel HyperCard. Extrait de «hypercard tutorial», Youtube, [en ligne] www.youtu.be/AmeUt3_yQ8c

fig .22 Capture d'écran de l'interface Microsoft Visuel Studio Code. L'interface graphique présente plusieurs zones pour créer un environnement de développement.

fig .23 Les interfaces Adobe Animate CC et Illustrator CC sont visuellement proches.

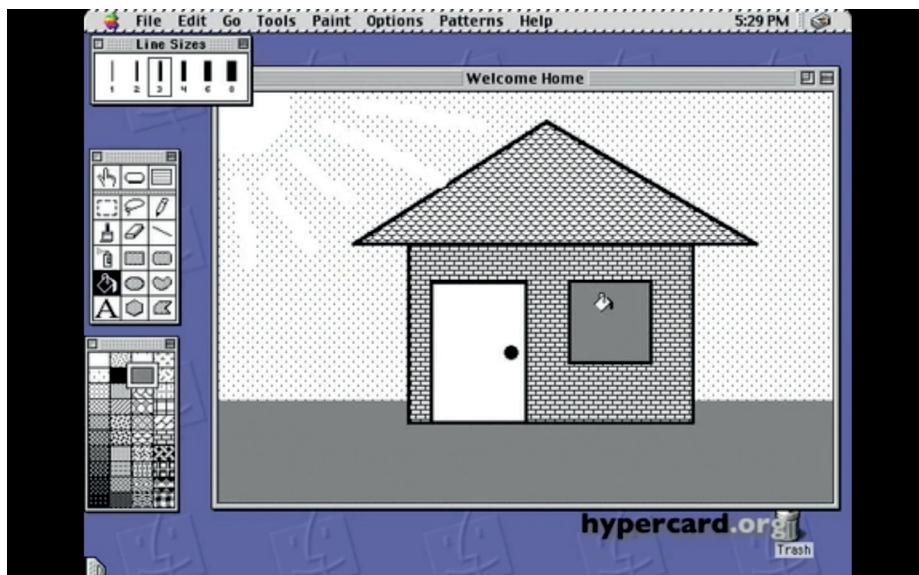


fig.21

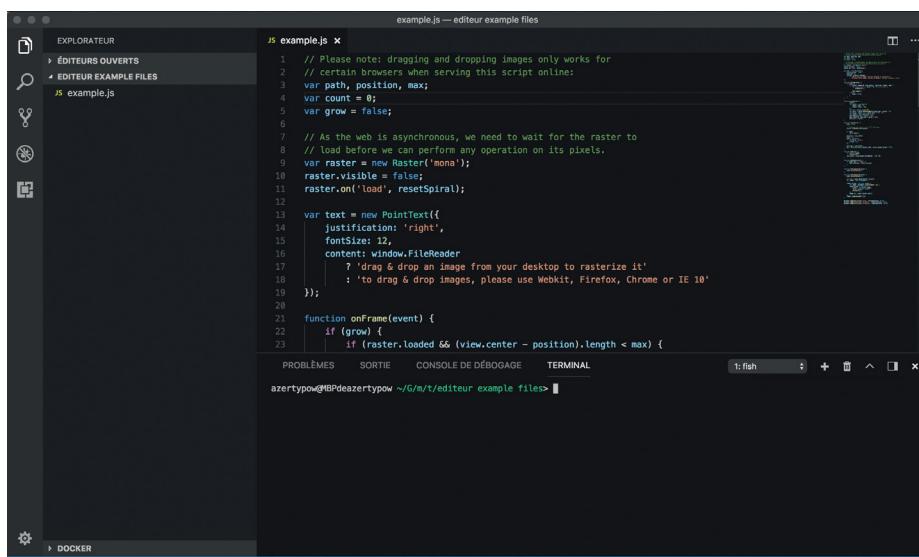


fig.22

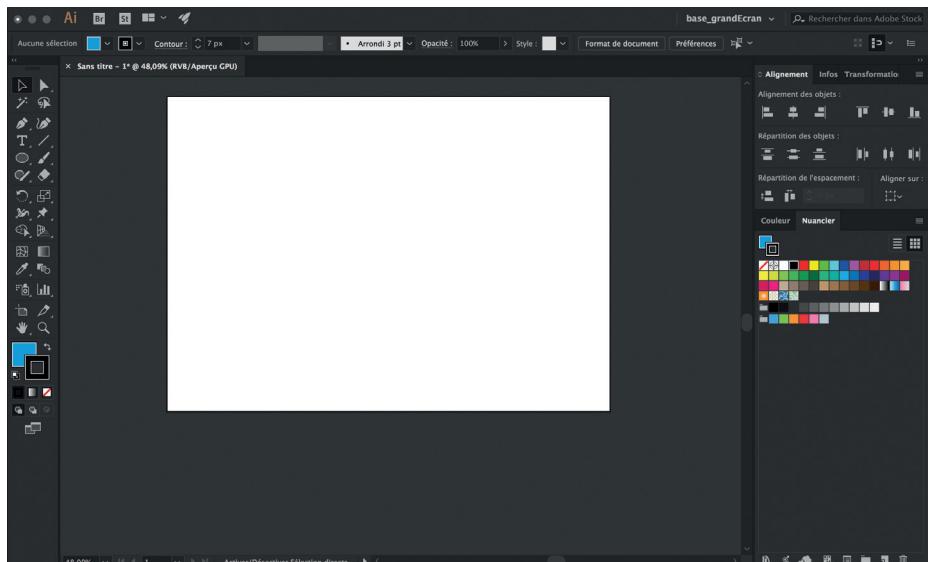
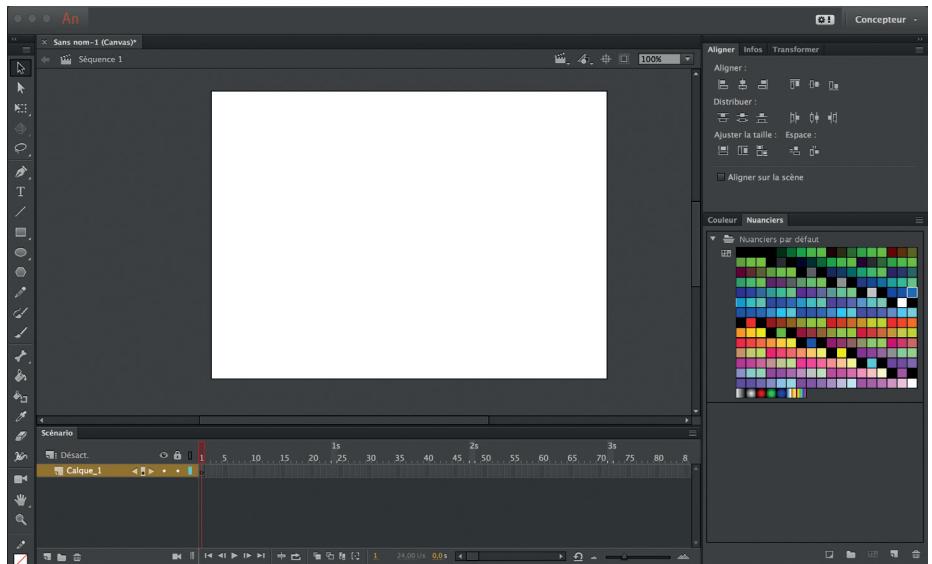


fig:23

fig .24 Illustration du concept « d'étiquetage » de Bret Victor, extrait de « Learnable programming. Designing a programming system for understanding programs », Bret Victor, beast of burden [en ligne], 2012, consulté le 13 novembre 2018, www.worrydream.com/#!/LearnableProgramming

fig .25 Capture d'écran de l'environnement de programmation

Processing, en utilisant le mode « Tweak » créé par Gal Sasson. Ici, l'utilisateur change la couleur du cercle grâce à la palette de sélection. La zone de rendu (à gauche) et l'interface textuelle (à droite) sont automatiquement mises à jour.

fig .26 Extrait video de l'environnement de programmation dédié à P5.js par Yining Shi. Extrait de son article « p5.playground : an Interactive Programming Tool for p5.js (in progress) », Yining Shi, www.1023.io/p5-inspector



fig:24

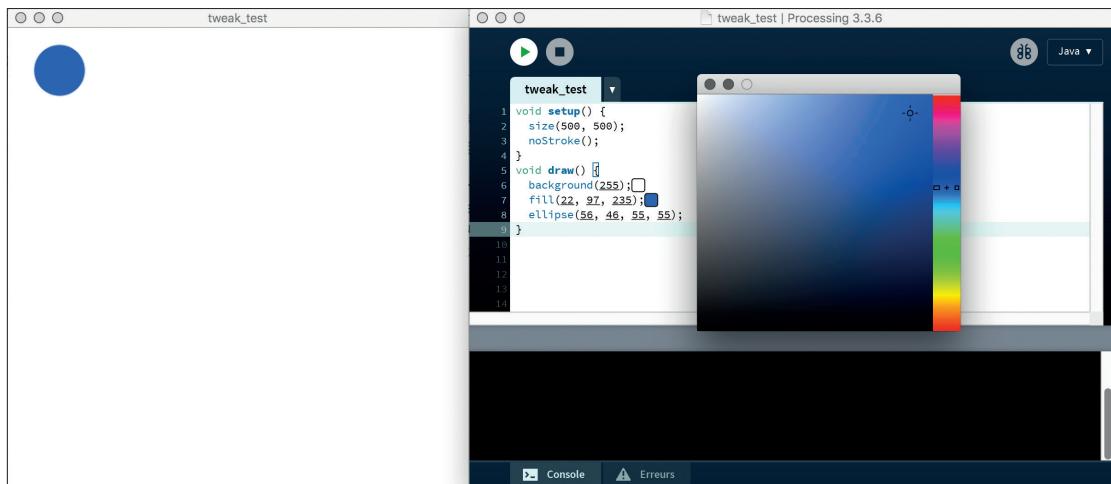


fig:25

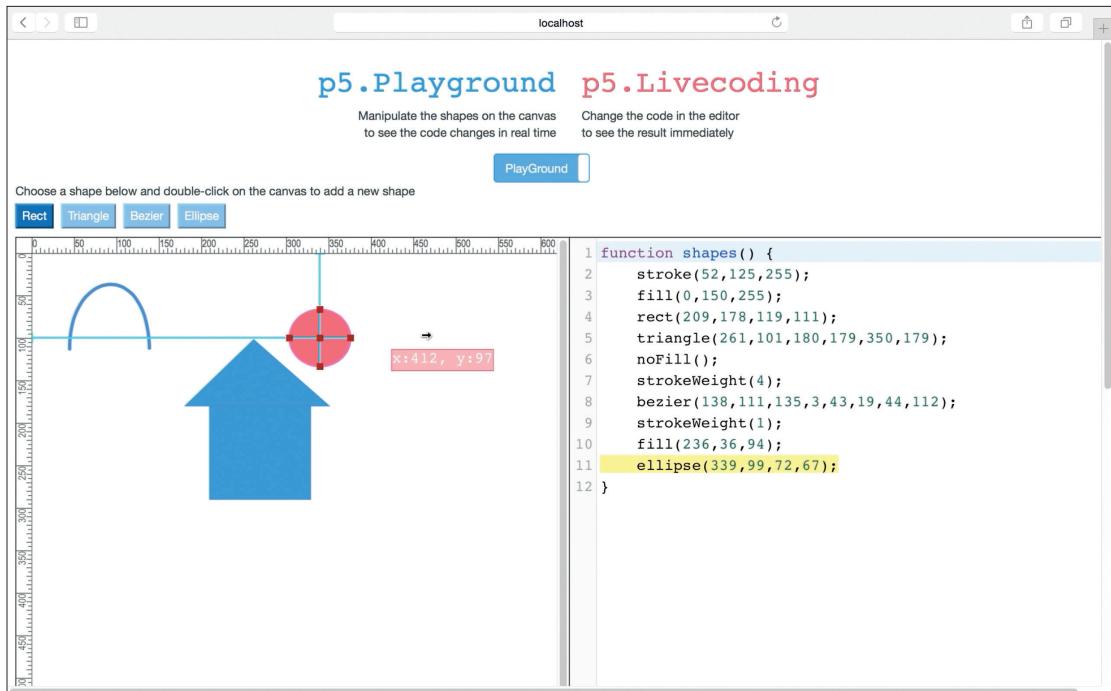


fig.26

fig .27 Concep de création vectoriel dans Illustrator. Yining Shi,
«p5.playground : an Interactive Programming Tool for p5.js (in progress)»,
Yining Shi, www.1023.io/p5-inspector

fig .28 Interface de débogage (à droite) et zone de rendu (à gauche) de l'application Goocle Chrome.

Le survole avec la souris d'un élément de code dans la zone de débogage met en surbrillant l'élément graphique qui lui correspond; ici le carré bleu, dans la zone de rendu.

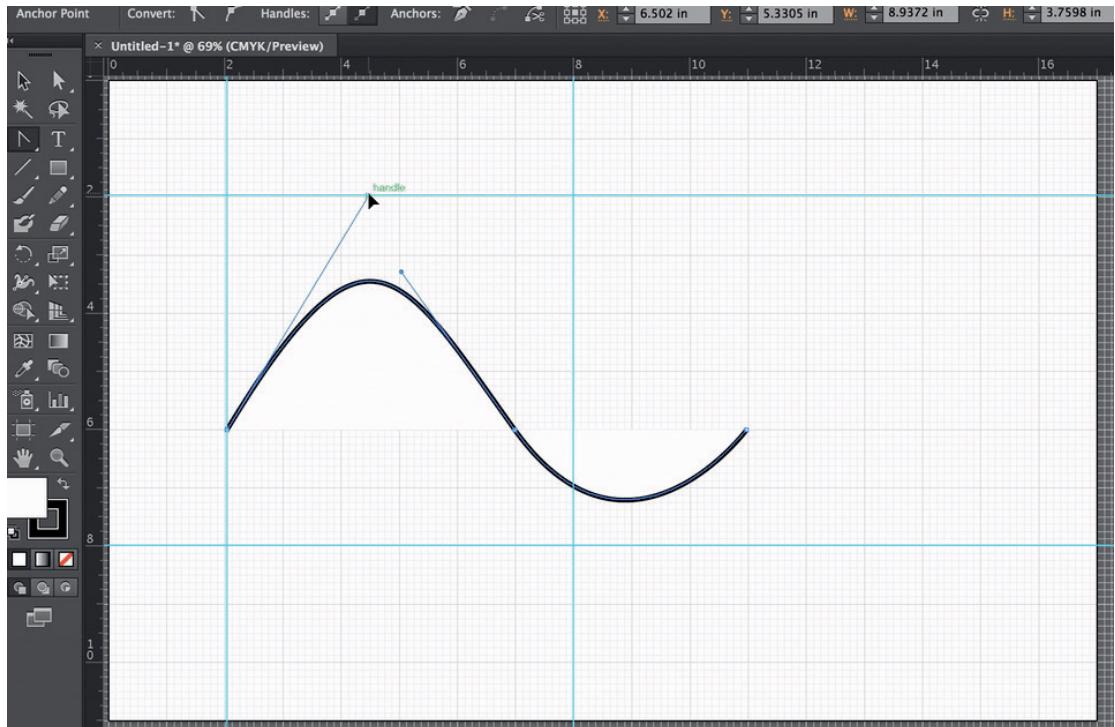


fig.27

A screenshot of a browser's developer tools. On the left, a 3x3 grid of colored divs is displayed. The top-left div is white with a black border, containing the text "div.rect | 100x100". The other eight divs are colored: orange, grey, blue, and orange again. On the right, the "Elements" tab shows the corresponding HTML code:

```
<!DOCTYPE html>
<html lang="fr">
  <head></head>
  <body>
    <div class="rect"></div>
    <div class="rect"></div> == $0
    <div class="rect"></div>
  </body>
</html>
```

The line "... <div class="rect"></div> == \$0" is highlighted in blue, indicating it is selected.

fig.28

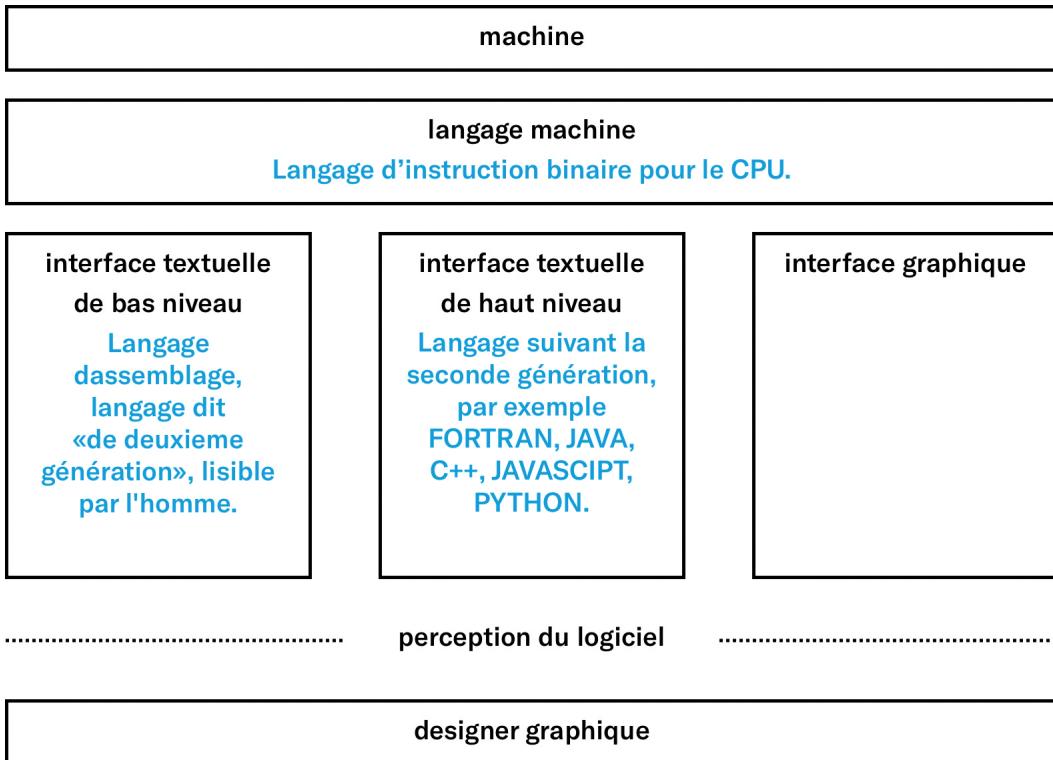
4— conclusion

Qu'elle soit textuelle ou graphique, une interface est inévitablement une couche d'abstraction par rapport au programme qu'elle représente. Chacune peut répondre à des paradigmes et des modes de représentation qui peuvent avoir leurs avantages et inconvénients.

Si comme Muriel Cooper et Karl Gerstner nous désignons la pratique du design comme un établissement de règles, de conception logique, comme un « processus plutôt que comme un produit unique »⁴⁴, alors l'interface textuelle a l'avantage d'être descriptive, et de pouvoir écrire un système. Cependant, la manipulation directe, le WYSIWYG et la présentation des possibilités d'un programme sont des points qui comme nous l'avons vu ont un intérêt pour le designer graphique dans les phases d'expérimentation de son travail. Le constat que l'utilisation d'interfaces textuelles peut se faire via des interfaces graphiques nous permet d'envisager une méthode de travail pouvant réunir plusieurs types d'interfaces dans un même environnement. Les recherches de Bret Victor et les travaux qui en découlent, comme ceux de Yining Shi, nous montrent l'intérêt d'un tel environnement adapté aux tâches du designer. Une telle conjecture permettrait de placer les interfaces

⁴⁴ Muriel Cooper, « Computers and Design », *Design Quarterly* n° 142, 1989, éd. Walker Art Center, Minneapolis, États-unis, p. 14.

graphiques au même « niveau » que les interfaces textuelles. C'est réduire les strates dans l'interactivité entre l'humain et l'ordinateur; et réduire cette « distance » constatée en introduction:



Si la logique de programmes Unix en ligne de commande permet qu'une « opération effectuée par un premier programme puisse servir de données de base à un second »⁴⁵ et que cette « capacité de «chaîner» les commandes par le biais de pipes [tubes] provient du fait que tous les programmes Unix partagent le même format de données entrantes et sortantes : le text brut»⁴⁶ ceci n'est pas dû à l'interface textuelle, comme nous l'avons vu, mais à une norme entre plusieurs programmes. Si le second programme ne comprend pas le langage du texte, aussi brut soit-il, qu'il reçoit du premier programme, il ne pourra pas l'interpréter. Là où de nombreux formats de fichiers permettent de transporter une

45 Eric Schrijver, « Culture hacker et peur du WYSIWYG », Back Office n°1, Design graphique et pratiques numérique, 2017, Co. éd. Fork Éditions & Éditions B42, Paris France, p. 43.

46 *ibid.*

information hiérarchisée sur un contenu (le fond), il serait intéressant de convenir d'une norme qui puisse contenir une description générique de style (la forme), à la manière du plug-in Sketch2AE de Google.

L'interaction d'un logiciel via une interface textuelle garantit que les informations stockées dans un fichier de travail soient intelligibles (contrairement aux fichiers Illustrator vus en première partie par exemple). Ainsi, les intentions d'interaction de l'utilisateur avec un logiciel peuvent être comprises sans passer par une étape de rendu. Il est donc certain que la conversion vers un autre format de fichier est plus facile. Si une norme commune de déclaration de style entre les applications est hypothétique, le passage par une interface textuelle peut être la garantie pour le designer de ne plus se retrouver bloqué dans un logiciel propriétaire.

bibliographie

Giorgio Agamben, Qu'est-ce qu'un dispositif? (2006), éd. Payot & Rivages, coll. «Petite Bibliothèque», Paris, France, 2007.

Don Bissell, «The father of computer graphic», Byte, États-Unis, 1990, p. 380-381.

Michel Beaudouin-Lafon, «40 ans d'interaction homme-machine: points de repère et perspectives», Interstices, 2007.

Jean Caelen. Interaction et multimodalité. Bruillard, E., Baldner, J.-M., Baron, G.-L. Troisième colloque Hypermédias et Apprentissages, May 1996, Châtenay-Malabry, France. INRP ; EPI, pp.11-32, 1996.

David Canfield Smith, «Pygmalion: A creative Programming Environment», 1975.

Alan Cooper, «The myth of metaphor», Visual Basic Programmer's Journal, 1995. Traduction de Marc Wathieu, «Le mythe de la métaphore», multimedialab.be [en ligne], consulté le 13 novembre 2018, www.multimedialab.be/doc/alan_cooper.htm

Muriel Cooper, «Computers and Design», Design Quarterly, n° 142, Minneapolis, États-unis, Walker Art Center, 1989, p. 17.

D. Cunin, «Pratiques artistiques sur les écrans mobiles: création d'un langage de programmation», Université Paris 8, Saint-Denis France, 2014.

Karl Gerstner, Designing Programmes, éd. Lars Müller Publishers, Zurich, Switzerland, 1964.

Adele Goldberg, «Smalltalk-80. The Interactive Programming Environment», Xerox Palo Alto Research Center, 1984.

Entretien avec Pierre-Damien Huyghe, «faire franchir un pas à une technique», Back Office n° 1, Design et pratique du numérique, ed. B42 et Fork, Paris, 2017, p. 84.

Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard et Kevin Mackey, «The Xerox Star: a retrospective», IEEE Computer, État-Unis, 1989, p. 11-29.

Joseph A. Konstan, «Drawing on SketchPad: Reflections on Computer Science and HCI» dans HCI Remixed, Tom Ericson and David MacDonald, éd. MIT Press, Londre, Angleterre, 2008, p. 23-27.

Henry Lieberman, «A Creative Programming Environment», dans HCI Remixed, Tom Ericson and David MacDonald, éd. MIT Press, Londre, Angleterre, 2008, p. 37-42.

Golan Levin, [vidéo] «Art that looks back at you», ted [en ligne], 2009, consultée le 13 novembre 2018, www.ted.com/talks/golan_levin_ted2009

J. Maeda, Design by Numbers, 1. paperback ed. Cambridge, Mass.: MIT Press, 2001.

John Maeda, «What Is DBN?», Design by Numbers, [en ligne], consulté le 13 novembre 2018, www.dbn.media.mit.edu/whatisdbn.html

Lev Manovich, Le langage des nouveaux média, traduction de Richard Crevier, éd. Les presses du réel, 2010.

Anthony Masure, Design et humanités numériques, éd. B42, coll. «Esthétique des données», Paris, France, 2017.

Nolwenn Maudet, «Muriel Cooper Information Landscape», Back Office n° 1, Design graphique et pratique numérique, éd. B 42, Paris, France, 2017, p. 105.

Muriel Puckett «Étude de cas sur les logiciels pour artistes: Max/MSP et Pure Data», dans Art++, sous la dir. de David-Olivier Lartigaud, éd. Hyx, 2011.

Eric Schrijver, «Culture hacker et peur du WYSIWYG», Back Office n° 1, Design graphique et pratique numérique, éd. B42, Paris, France, 2017, p. 37.

Ivan Edward Sutherland, «Sketchpad: A man-machine graphical communication system», 1963.

Bret Victor, «Inventing on Principle», conférence CUSEC 2012, Youtube, [en ligne], www.youtu.be/PUv66718DII

Bret Victor, «Learnable programming. Designing a programming system for understanding programs», Bret Victor, beast of burden [en ligne], 2012, consulté le 13 novembre 2018, www.worrydream.com/#!/LearnableProgramming

Joseph Weizenbaum, Puissance de l'ordinateur et raison de l'homme: Du jugement au calcul, traduit de l'américain par Marie Thérèse Margulici, éd. d'Informatique, Boulogne-sur-Seine, France, 1981.

Stephen D. Burd, «Systems Architecture», éd. Course Technology, 6e édition, juin 2015.

