Visual Programming and Creative Code: A Maker Perspective in Software studies

Master thesis

MA Mediastudies
New Media & Digital Culture
Utrecht University
Netherlands

Jelke de Boer
3884023

Abstract

If software is the cornerstone of (post) modern society then it is the algorithm that serves as the primary tool of the information age. The field of software studies investigates how software affects culture and society. The initial efforts have contributed to the understanding of the "nature" of code and widespread 'popular' or 'mainstream' productivity software. This thesis shifts attention to visual programming, a popular tool in artistic performance, interactive installations and electronic music production. How can visual programming be understood as an artistic tool? As an overarching theme three concepts will be addressed: Materiality, Agency and Sociality. Building upon actor network theory it will be argued that, as a tool, visual programming should be understood an act of creative coding that attempts to make hardware performative. Furthermore, it will also be argued that aside of the arts, engineering and design, a new tradition of creators has emerged from maker culture.

Keywords

Software studies, Visual Programming, Artistic Tools, Creative Code, Agency, Performance, DIY, Maker Culture, Actor Network Theory, Design Fiction.

Acknowledgments

The list of people that somehow deserve to be mentioned for their support is long, including a network of people that covers all three major Educational institutions in the Utrecht area; Utrecht University, the University of Applied Sciences Utrecht and the University of the Arts Utrecht. The Utrecht area offers a wonderful climate for students, lecturers and artists, three roles that I have tried to combine following the New Media & Digital Culture program.

Of course my gratitude goes out to all staff members of the NMDC program. A special word of thanks goes out to Dr. René Glas for his support and feedback and to Dr. Mirko Tobias Schäfer and Dr. Imar de Vries for their inspiring lectures and the thoughtful discussions that we have had. I would also like to thank my colleagues at the University of Applied Sciences Utrecht; and more specific Erik Hekman, Ronald van Essen and Kees Winkel for the inspiring discussions and support

Finally, my gratitude goes out to Guinevere, Nova and Tibbe and their great patience, enduring my absence.
Thank you

Jelke de Boer

Table of content

Introduction

In 2013 media and performance artist Lisa Park presented the Eunoia project, a digital performance using EEG technology to manipulate sound and the motion of water. The code used in the Eunoia project was created in Max MSP[1], a visual programming interface aimed at live performance. The project has been presented at several international interactive art festivals, and found its way into the science and technology sections in popular media and the technology related special interest titles such as Wired Magazine. The article in Wired magazine reads:

> Increasing her frustration levels modulates the volume of the music; the more frustrated Park becomes, the more intense the vibration of the water. When Park is calm or meditative, the frequency of the vibrations slows down. She pans the sound waves around the room by controlling her levels of excitement and engagement. In that way, Park is effectively choreographing the movement with her mind (Stinson, 2014).

At least part of the interest in the Eunoia project can be explained by the use of the EEG device, a technology that captures the imagination of the technology-enthusiast audience. From a media studies perspective one might say that the Eunoia project appeals to "the dream of ideal communication" (De Vries, 2005) though popular media seem to favour the dystopian imaginary: "if you ignore the black sensors wrapped around her head and watch the pools of water below, it looks as though she's pulling off some serious Carrie-style mind manipulation" (Stinson, 2014). Of course, and as always, at closer examination the dream shatters. Still the performance is far from cheap trickery; it's a genuine invitation to the experience of a shared imagination.

In the media coverage of the Eunoia project most attention may have gone out to the EEG device, still it is of crucial importance not overlook the role of the software code that ties the performance together. Or, as Cramer (2002) points out in a broader sense:

[1] Max MSP visual programming software (https://cycling74.com/)

> The term “digital art” has been associated primarily with digital images, music or audiovisual installations using digital technology. The software which controls the audio and the visuals is frequently neglected, working as a black box behind the scenes. “Interactive” room installations, for example, get perceived as a interactions of a viewer, an exhibition space and an image projection, not as systems running on code (Cramer, 2002, 1).

My interest is directed towards Max MSP, a visual programming environment that has gained a wide popularity among “laptop” musicians, artists in genres such as (interactive) installation art and (live) performance and the practice of “experience design” for physical (exhibition) spaces. It is important to note that there is a difference between visual programming and the syntax based coding style commonly associated with software code and the algorithm. The visual programming interface does not represent code as linguistic syntax but as a visual mapping of objects that represent actions. This visual representation seems to appeal to artists, musicians and others involved in live performance. Park (2015) emphasizes the importance of the visual metaphor:

> What I like about Max compared to other processing languages such as C ++, is that it has the visual component. I find that easier to work with and it makes more sense to me. Since I was trained in visuals since I was kid, I feel more comfortable using Max compared to any other coding program (Park, 2015).

The visual representation of code may be more appealing and easier to grasp for the visual artist, the electronic musician might also recognize the process of “patching” the modular synthesizer. Max MSP was originally designed as a tool for the creation of software sequencers, thus allowing the electronic musician to control a broad range of hardware synthesizers from a single (customized) computer interface. As such, the Max MSP interface can be related to the early electronic musicians of the 1950’s and 1960’s whom aimed at creating new musical genres. One might add that these early electronic musicians were drawing from engineering rather than instrumental virtuosity. However, that does not mean that visual programming is

limited to music production. If the modular synthesizer allowed for new approaches to music production and sound design then it seems fair to assume that creative coding has opened a new kind of performative space (Saphiro, 2014). Visual programming in Max MSP thus allows for new kinds of arrangements combining audio, visuals, physical objects and human bodies in and through code. Of course creative coding is not limited to visual programming, still investigating visual programming may shed light on the creative and artistic use of software code.

Research question

The emerging field of software studies has expressed a great concern for the role of software and the algorithm in (post) modern society. Much attention has gone out to the complex relations in digital environments and the distribution of power trough software interfaces and digital code. Still the code itself is often perceived as a linguistic or mathematical abstraction. In this thesis I will offer an alternative perspective, suggesting that code may also be understood as creative, visual and embedded in a context of artistic maker practices. I will do so addressing the following research question:

How can visual programming be understood as an artistic tool in contemporary maker practices?

I will argue that through visual programming the computer can be explored as *"a metamedium, whose content would be a wide range of already-existing and not-yet-invented media"* (Kay & Goldberg, 1977, 40; Manovich, 2013, 105). Furthermore I will argue that, as a tool, visual programming allows the artist, maker or creator to infuse machinery with subjective imagination and the narratives of design fiction (Bleecker, 2009). I will ground my approach in actor network theory (Latour, 1996, 1999) and in material semiotics (Law, 2009) using (digital) materiality, agency and sociality as key concepts in addressing the research question.

Materiality, Agency & Sociality

If code is the material of the 21th century artist then the importance of digital materiality seems self-explanatory. Still the relations between artists, the layers of (visual) code and the physical infrastructure in which the code "lives" are complex. Or, as Cramer & Gabriel (2001) conclude when reflecting upon their role as members of the jury for the "artistic software" award at the Transmediale art festival:

> The software artists we reviewed seem to conceive of generative systems not as negation of intentionality, but as balancing of randomness and control. Program code thus becomes a material with which artist work self-consciously. Far from being simply art for machines, software art is highly concerned with artistic subjectivity and its reflection and extension into generative systems (Cramer & Gabriel, 2001, 4).

Indeed, in the context of artistic practices the material "nature" of code invokes questions of intentionality, subjectivity and agency. I will build my argumentation around digital materiality and agency, but to fully grasp visual programming as an artistic tool it is also important to gain an understanding of how digital code *perceived* from the maker perspective. I will analyse the Max MSP visual programming interface, but I will do so recognizing that (visual) programming can be associated with a variety of distinct cultural practices. Of course there is a great difference between musicians, artists and others involved in performance or maker practices. Still my assumption is that, even though these practices may vary in many ways, they do share some fundamental cultural logic.

As such my approach seem in line with the three primary facets of software studies as identified by Mackenzie (2006): "(1) agency, the concept of who does what; (2) materiality, or what counts as the basic stuff that exists; and (3) sociality; the concept of how we belong together, how attachments form and collective social life coheres" (2006, 171). For Mackenzie these concepts underline the importance of software code in contemporary society as algorithms, network protocols and digital infrastructures reshape power relations in production, consumption and

distribution. My aim is different - and perhaps more modest - as what is of interest here is how software code is perceived and produced from the specific cultural position of the artist, designer or maker. From this specific perspective digital code could - and should - be understood as creative rather than productive.

Structure & Methodology

Agency and digital materiality and cultural logic will provide the overarching structure of this thesis. But first I will situate my research within the field of software studies. I will build upon Lev Manovich's (2013) analysis of the software tools most frequently associated with creative practices: Adobe Photoshop and the collection of tools bundled in the Adobe Creative Suite. I will argue that software may provide a bridge between culture and technology (Rieder & Schäfer, 2008) and clarify my understanding of the artist, designer and maker in contemporary (techno) culture. Finally I will introduce Max MSP discussing the MIDI protocol and the use of hardware devices in (digital) performance and music production.

In the second part of this thesis I will turn to the question of agency. Reviewing the Eunoia project introduced earlier I will discuss the interactions between performer, code, hardware and audience arguing that creative code has opened a new performative space (Saphiro, 2014) in which the artist, designer or maker can explore the arrangement of agency. From there I will build a theoretical framework connecting actor network theory (Latour, 1996, 1999) and material semiotics (Law 2009) with the concept of liveness in (digital) performance. Following Auslander (2012) I will conclude the second chapter arguing that digital liveness "results from our conscious act of grasping virtual entities as live" (2012, 10).

In the third chapter I will broaden the theoretical framework discussing digital materiality and the "nature" of digital code. I will follow Manovich (2013) in "close reading" the Max MSP visual programming interface though I will do so recognizing that the computer and the software interface are extremely marked by metaphors (Van den Boomen, 2014). I will start from the notion of the "object as a data structure" and the general approach of adopting terms and concepts from computational sciences as proposed by Manovich (2001, 39). Second, I will discuss

the "object as a metaphor" emphasizing crucial role of the metaphor (Van den Boomen, 2014) and real-world analogies (Bolter & Grusin, 2000) in the perception of the computer interface. Close reading the Max MSP interface will reveal that the metaphors used in visual programming reflect the materiality of the computer hardware itself. Finally it will be argued that the computer can best be understood as "a metamedium, whose content would be a wide range of already-existing and not-yet-invented media" (Kay & Goldberg, 1977, 40; Manovich, 2013, 105).

Finally, I will situate visual programming in DIY and Maker culture arguing that visual programming can best be understood as maker practice. That does not mean that I wish to suggest that artistic practices and DIY or Maker culture are the same thing. It merely implies that the artist and the maker share an interest in exploring the computer interface and digital code as a medium for their own expression. Still it seems to me that the maker perspective on computer technology is of vital importance. I will argue that both artists and makers playfully explore digital technology as "design fiction" (Bleecker, 2009) suggesting that it is the perception of technology rather than the technology itself that is at stake. I will conclude that visual programming and creative code allows the artist or maker to playfully re-imagine the "history of the future".

Max MSP vs. Pure Data

My analysis of the visual programming interface is based upon the commercial Max MSP software, but that does not mean that the scope is limited to Max MSP; it also applies to Pure Data, the open source alternative to Max MSP. Of course there are other visual programming tools such as Scratch[2], the excellent educational package developed by the MIT Media Lab. However, as both Max MSP and Pure Data are specifically aimed at live performance, these applications are of primary concern here. Obviously there is an important difference between open source and commercial or branded software. The pricing of the commercial Max MSP package is

[2] Scratch (https://scratch.mit.edu/) is freely available as a browser-based online application and as a standalone application

steep, especially when considering the availability of Pure Data, the free alternative developed by Miller Smith Puckette, the very same person that also developed Max MSP. The fundamental and deeply political differences between open source and commercial software may invoke principal discussions within artist and maker communities as well as in scientific discourse. However, this complex and highly politicized debate is beyond the scope of this thesis. For now it seems plausible to assume that both Max MSP and its open source equivalent Pure Data appeal to a more or less similar audience of artists and makers. One may be willing to buy the Max MSP software while the other may principally reject the use commercial software thus preferring Pure Data. Still others will avoid purchasing software at all using illegal versions.

The complex mathematical foundations of electronic music production and visual mapping techniques in live performance - in Max MSP or Pure Data - will also be avoided, first because my aim here is much broader than the mathematical details that underpin live performance and music production, and second because the theoretical side to these issues - tone generation, modulation and sound processing to name a few - have already been described in detail (Puckette, 2007).

Cultural interfaces

Software studies

In the introduction to *Software studies: A lexicon* (2013) Fuller notes that "while applied computer science and related disciplines such as those working on computer-human interface have now accreted around half a century of work on this domain, software is often a blind spot in the wider, broadly cultural theorization and study of computational and networked digital media" (2013, 3). Thus "software is seen as a tool, something that you do something with. It is neutral, grey, or optimistically blue" (2013, 3). Over the last decade the interest in software within the Humanities has steadily grown, and the field of software studies has emerged as a vital part of the "Digital Humanities". Lev Manovich, perhaps the most prominent scholar in software studies, recognizes that "outside of certain cultural areas such as crafts and fine art, software has replaced a diverse array of physical, mechanical, and electronic technologies used before the twenty-first century to create, store, distribute and access cultural artifacts" (Manovich, 2013, 2). Manovich also point out that we are not merely interfacing with a computer system. That is to say, we use the computer to interact with cultural data. The human-computer interface has become a human-computer-culture interface, or in short, a cultural interface (2001, 80).

Thus the role of software and those whom produce software interfaces should not be underestimated. Pold (2008) writes: "the relations between the software's senders and receiver(s) or user(s) are defined, most often within very strict limits. Normally, it is only possible to change certain things and change them the way the senders have prefigured" (2008, 219). Initially most attention has gone out to widespread 'popular' or 'mainstream' productivity software such as the software bundled in the Adobe Creative Suite[3] (Manovich, 2013). At first glance it seems not

[3] The Adobe Creative Suite (http://www.adobe.com/products/cs6/faq.html) bundles all Adobe productivity software for printmedia, AV production and webdesign. In 2015 the standalone applications in the Creative Suite (version 6) have been replaced with the Adobe Creative Cloud versions that offers a subscription model.

that difficult to trace the pre-digital practices that most common media productivity software such as Adobe Photoshop[4] or Apple's Final Cut[5] remediate. The Adobe software is building upon existing practices and relies on metaphors firmly rooted in the pre-digital era. As a consequence "many software techniques that simulate physical tools share a fundamental property with these tools: they require a user to control them "manually." The user has to micro-manage the tool, so to speak, directing it step-by-step to produce the desired effect" (Manovich, 2013, 128). Still, and as Manovich also points out, below the surface of the user interface the analogies between software and real world practices are complex and deceitful (2013, 59).

Thus over the past decade the insight in "mainstream" and "productivity" software has steadily grown and software design has been recognized as of vital importance in the complex interplay between culture and technology. Or, as Rieder & Schäfer (2008) suggest, software may provide a bridge between culture and technology. Their suggestion is build around two intertwining observations; the first recognizes that "software plays an increasingly important role in our everyday lives, accentuating culture as a hybrid of technology and discourse" (2008, 168); the second adds "software production flourishes outside of the classical institutions and methodology of engineering" (2008, 168).

Designer, artist and maker

Rieder & Schäfer also recognize a shift from engineering towards design, as "every age seems to have an epitomical figure of technical creation: the craftsman for the Middle Ages, the inventor in the industrial Revolution, and the engineer in the 20th century: the *designer* as the toolmaker of the information age" (2008, 159). Thus the (software) designer has emerged as the new central or iconic figure in a software driven society; replacing the engineer as the paradigmatic figurine. Still one should remain cautious when confronted with the protagonist in the heroic narratives of

[4] Adobe Photoshop (www.adobe.com/Photoshop); the hallmark application in the Adobe software family
[5] Final Cut Pro (http://www.apple.com/final-cut-pro/) video editing software, only available for the Apple platform

innovation. The term “designer” can be confusing as it may include a wide variety of practices. Whereas Rieder & Schäfer (2008) seem to understand the designer as a *software* designer creating new digital tools, for Manovich (2001, 2013) the designer is to be understood as a *media* designer, a professional producing (new) media. Obviously there is a crucial difference between those using existing tools and those creating new ones. The average media designer working with mainstream productivity software, using brushes, selection tools, and the paint bucket to earn her monthly salary might be far closer to the traditional *craftsman* then to the iconic image of the toolmaker of the information age.

In contrast to the Adobe Creative Suite the Max MSP interface does not offer manually controlled brushes, selection tools or paint buckets. Max MSP draws on the electronic component and (analogue) circuitry as its primary interface metaphor. Max MSP thus draws from *engineering* rather than craftsmanship. Furthermore, Max MSP allows the user to create (visual) code. As such, the artist working with Max MSP might be closer to the software designer than the (new) media designer.

Restricted vs. mass production

In this thesis a distinct is being made between artists and designers, as the first is more precise, and the second is more inclusive. Here the artist is understood as a (new) media artist whom not only makes use of tools but also actively engages in (tool) making practices. Till a certain end the artist may also stand for the “limited” or “artistic” production where the designer should be read as an umbrella-term that includes a wide variety of practices in cultural production. That does not mean that I wish to affirm the “high culture” versus “low culture” dichotomy, it merely implies that the distinct between restricted and mass production still operates as an important organizing principle in cultural production. It is also important to recognize that “certain forms of digital art are starting to enter the mainstream both commercially and institutionally” (Anderson & Pold, 2011, 8). Indeed, the realm of the arts does not always or necessarily operate outside of commerciality, and the aesthetics of contemporary interface culture may call for a re-examination of the relation between the (digital) arts and design (2011, 8). In a more general sense one

could add that “there is now a huge amount of cultural production taking place on the boundaries between sub-fields of mass and restricted production; or, perhaps better still, that restricted production has become introduced *into* the field of mass production” (Hesmondhalgh, 2006, 222). The important observation that “software production flourishes outside of the classical institutions and methodology of engineering” (Rieder & Schäfer, 2008, 168) might also suggest that the archetypical distinct between ‘the engineer’, ‘the designer” and ‘the artist’ may need reconfiguration. The rise of software may call for a new kind of typology for those involved in cultural making practices.

Visual language

The 20th century avant-garde artists explored the machinery of mass production as an artistic tool, rejecting the traditional craftsmanship of the pre-industrial era. In the 21th century the attention is shifting towards software interfaces, the algorithm and (creative) code. The tools most commonly associated with, and often regarded as essential to the creative practice - the software bundled in the Adobe Creative Suite - have been thoroughly examined by Manovich (2013). Manovich concludes: “before they adopted software tools in the 1990s, filmmakers, graphic designers, and animators used completely different technologies. Therefore, as much as they were influenced by each other or shared the same aesthetic sensibilities, they inevitably created differently looking images” (2013, 297). Manovich argues that these distinct artistic languages - both in terms of form and content – have been replaced by “the same software-generated iconography . . . found across all types of media, all scales, and all kinds of projects” (2013, 303). The common sense explanation provided by Manovich is that the palette of software used by contemporary designers is small; they make use of the same set of software to design everything (2013, 300).

This may indeed hold for the mainstream of commercial media production and a large part of the vast body of commercially produced media that circulates the Internet. Indeed, the Adobe software family, including Photoshop, Illustrator and Indesign dominates the broad mainstream of media production and media design.

But of course this mainstream culture does not represent human culture as a whole. One might add that it seems hardly surprising that the global mainstream of commercial media production is marked by its homogeneity.

Compared to the Adobe Creativity Suite – the software bundle often considered the industry standard for the "creative industries" - the Max MSP user base may be small, and the Pure Data community may be even smaller. Still the artist working with Max MSP and Pure Data do reach a substantial and significant audience. Recalling his visit to electronic musician, writer and artist DJ Spooky Manovich (2013) briefly mentions Max MSP:

> The only "instrument" Paul Miller (aka DJ Spooky That Subliminal Kid) owned was his 15-inch PowerBook laptop, made by Apple. This was his "Dynabook": a "self-contained knowledge manipulator in a portable package the size and shape of an ordinary notebook." Although this "Dynabook" did not have Smalltalk, it ran another programming environment which was powerful, fast and allowed for visual programming—MAX, the language of choice worldwide for tens of thousands of electronic musicians, VJs, dancers, theatre performers and others working with different forms of real-time performance (333).

Indeed, (electronic) musicians and performing artists have developed their own practices building their own collection of software tools. Besides Max MSP and Pure Data - the visual programming software discussed here - the most notable tools are Ableton Live[6], Propellerheads Reason[7], Native Instruments Traktor[8], Avid Protools[9] and exclusively for the Apple users Logic Pro[10]. It is important to add that most performing artists will also carry one or several physical devices to provide the necessary knobs, sliders and buttons as the mouse and keyboard are practically

[6] Ableton Live (https://www.ableton.com/) combines electronic music production an live performance.
[7] Reason (https://www.propellerheads.se/reason) electronic music production studio
[8] Traktor (http://www.native-instruments.com/en/products/traktor/) soft- and hardware for DJ's
[9] Protools (http://www.avid.com/us/products/family/pro-tools) for professional studio recording
[10] Logic Pro (http://www.apple.com/logic-pro/

useless in the context of live performance. In other words, in live performance the physical human–computer interface matters. What is unique to Max MSP – and its open source counterpart Pure Data - is that the artist can create her own (physical) user interface. Furthermore, Max MSP allows the artist to control other applications through the MIDI protocol. The artist may thus create a unique custom interface combining physical hardware and software applications through (visual) code. A data stream can be transcoded into sound patterns, a slider might be replaced by a physical gesture and moving towards specific positions could trigger the projection of visual material.

Max MSP

The first version of Max, released in 1988, was limited to MIDI control signals. The computers available in the 1980's were not fast enough to deal with real time audio. The rapid increase in processing speed allowed for the release of the MSP [11] extension thus making the computer a musical instrument capable of live performance (Puckette, 2007, ix). In 2003 Max was further extended with Jitter, a code library that allows for real-time video processing. Still initially Max MSP was conceptualized as a musical instrument, "played" or "operated" by a human performer. Puckette (1991) explains:

> I am against trying to set the computer up as a musical performer. Large software systems which try to instill "musical intelligence" in the computer, while interesting as research projects, are not likely to be musically useful . . . The computer is better used as an instrument. A unique one, to be sure: no violin has a programmable user interface. The computer instrument widens the possibilities of musical expression - human musical expression - in ways which we are only beginning to explore (1991, 2)

Since it's first release in 1988 Max – now often referred to as Max MSP, a convention that I will follow - has gradually been extended well beyond the realm of music

[11] MSP is short for Max Sound Processing. It also refers to the initials of Miller Smith Puckette

production. According to the official Max MSP website[12] "Max is built on the idea of connecting things together to make something new. Connect plugins, media players, and custom DIY effects, or build something completely from scratch . . . Max provides everything from the most basic nuts and bolts to advanced effects modules that are ready to use. Mix, match, and tinker without limits". The project pages in the Max community hosted by Cycling'74 showcase a wide variety of projects, loosely organized using tags[13]. Even though a substantial amount of projects lacks tagging, the list of the 49 most frequently used tags gives an impression of how Max MSP is used in practice. The top ten tags are: Interactive (127), Performance (125), Max-For-Live (116), Installation (116), Real-Time (103), Ableton (89), Audio (89), Sound (83), Midi (83) and Video (81). The most frequently used tags suggest that Max MSP is still being used as an instrument – or as a tool to create instruments - in music production and in live performance. Arguably the most well known artists that are known to make use of Max MSP include Autechre, Aphex Twin, Daft Punk, Jamie Lidell and Radiohead. However, the use is most certainly not limited to music production; Max MSP is also widely used in interactive installations and in a broad range of other types of performance. It is hard, if not impossible to narrow down the wide variety of different projects into a small set of subcategories. Perhaps the one key characteristic that most if not all projects share is the use of a variety of hardware devices.

The MIDI protocol

In its first releases - before the Max Sequencing Protocol (MSP) modules were added - Max was designed as a software interface for analogue hardware synthesizers using the Musical Instrument Digital Interface (MIDI) protocol. The MIDI protocol was standardized in the early 1980's and was set up as a message system between analogue hardware devices to control a wide range of musical parameters such as tempo, note and pitch. The development of the MIDI protocol was important because it allowed musicians to combine controllers and instruments from different

[12] https://cycling74.com
[13] https://cycling74.com/community/?q=project

manufacturers into a single setup. The MIDI protocol was adapted by the upcoming personal computer industry halfway the 1980's and has since remained the industry standard for both music software and the music hardware industry. A stored MIDI file contains instructions only and does not carry sound signals, thus a MIDI file requires a synthesizer (or a software emulation of a synthesizer, a soft-synth) to become audible. The only significant change in the MIDI standard up till now has been the introduction of the USB (Universal Serial Bus) interface replacing the five-pin DIN connector. Still older MIDI devices using the five-pin connector can easily be connected to a modern personal computer or laptop using a cheap USB to MIDI hardware interface.

Physical Interaction and hardware devices

The hallmark feature of Max MSP thus is its capability of connecting and controlling hardware devices. The Cycling'74 homepage explicitly mentions: "(1) the Arduino to connect electronic sensors, motors, and other components with this programmable board; (2) controllers to use any MIDI controller with knobs, sliders, buttons, or keyboards; (3) synthesizers to combine hardware synths with Max to create custom editors or drive a multi-hardware performance; (4) DMX lightning to add audio-responsive and interactive elements to a show system; (5) projectors with interactive support for multiple screens, OpenGL hardware graphics, and video playback; and (6) live inputs to connect live instruments and sound sources to Max" (Cycling'74, 2015, June 29). Furthermore, the tools section of the user community offers a wide variety of user-contributed patches for connecting and controlling hardware. In short, the cultural artefacts produced through Max MSP challenge the popular notion of the "desktop" computer operated using a mouse, keyboard and screen interface.

Shifting attention to the physicality of interaction should not go at dispense of the complex cognitive processes involved. Manovich (2001) strongly opposes using the term "interactive media" precisely for this reason: "when we use the concept of "interactive media" exclusively in relation to computer-based media, there is danger that we interpret "interaction" literally, equating it with physical interaction

between a user and a media object (pressing a button, choosing a link, moving the body), at the sake of psychological interaction" (2001, 71). Manovich points at the principle of hyperlinking to illustrate how association, problem solving, recall and reflection are externalized. These mental processes central to human thinking are objectified and reduced to following a path of hyperlinks (2001, 74). As a result, "the interactive media asks us to identify with somebody's else mental structure . . . a computer user is asked to follow the mental trajectory of a new media designer" (2001, 74).

This may indeed count as a serious problem in the mainstream of digital media and in the mainstream of productivity software as both follow the same relentless doctrine of "user-friendliness" where the "new media designer" does the thinking and the "user" is restricted to clicking. But the context of live performance is very different. Electronic musicians, VJs, dancers and performers do not merely make use of digital computer technology to produce media. In digital performance the relations between the artist and the environment in which the code lives are complex, but most of all they are physically embodied. One might say that in the context of (artistic) live performance the physical human-computer interface created through Max MSP is part of the artwork. Perhaps that should not come as a surprise as the interactive arts in general and digital performance in specific have build a tradition in exploring the embodied physicality of interaction, questioning the traditional notion of the interface as standing in-between or separating the "user" from the "content". However, that does not mean that there is only physical devices and programmable hardware to recon with, or that "there is no software" as Kittler famously posed (1995, 3). In contrast, it is through – and in - software code that the digital artwork comes into being.

Distributing agency

The Eunoia project

The Eunoia project introduced earlier well illustrates the notion of the artwork as a human-computer interface running on software code. From a "neutral" engineering perspective the Eunoia project can easily be summarized (Figure 1): The meditating performer (1) uses the commercially produced Emotiv EEG headset (2) that can detect a limited range of emotions like excitement, engagement, meditation, frustration, boredom. The captured data is transcoded into the Open Sound Control (OSC) format and send to Max MSP (3). The data stream is then used to manipulate the volume, panning, and playback speed of a pre-recorded set of sound samples (4) stored on the computer's hard drive (5). The digital audio data is converted into analogue signals using a Digital to Audio (DAC) interface capable of handling multiple audio streams (6). The modified speakers (7) receive the analogue signal, now audible for the audience (8). The sound waves become visible as the speaker's oscillation disturbs the calm water (7). The reaction of the audience (8) and the sound produced by the speakers (7) affect the emotional state of the performing artist thus creating a feedback loop.

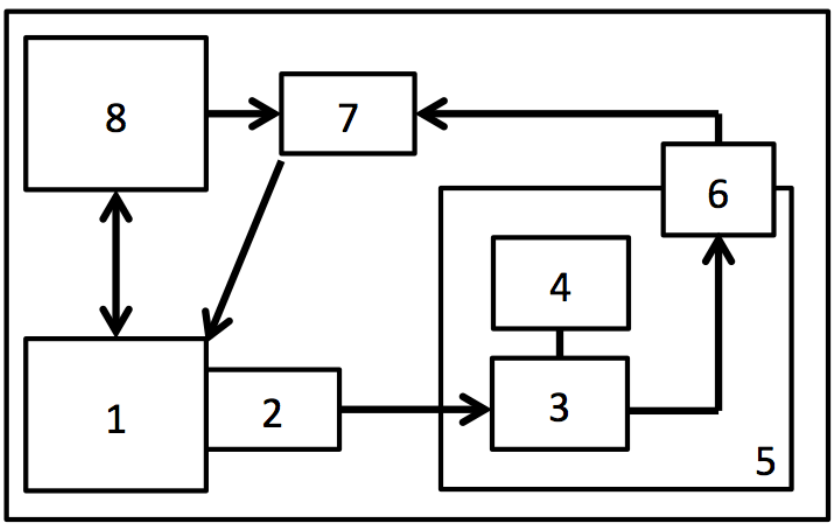

Figure 1: The Eunoia Project

Of course this plain and descriptive summary of the Eunoia project falls short in many ways. It does not address the subjective complexity of the artwork nor does it fully capture the complex relation between human performer, code, physical hardware and the audience in digital performance. And this description ignores the crucial role of software code. It excludes the complex algorithms processing the EEG data captured by the Emotiv EEG device[14] and the artistic interpretation of the EEG data through visual programming. What is of importance here is the subjective artistic interpretation of the data rather than the data produced by the EEG device. That is to say, the Eunoia project should not merely be understood as an engineering project and its goal is not to optimize the performance of hardware; it is an act of creative coding that attempts to make the hardware performative.

Creative code

The move towards creative coding of which Eunoia is just one example marks a shift in paradigm as "creative coding wants to fashion a 'new reality', a hybrid of the good old familiar phenomenological 'reality' and new 'virtual realities', new experiences of existence in a hybrid real/virtual dimension" (Saphiro, 2014). Or, as Saphiro writes: "creative coding where a line of code is an aesthetic artefact and not only an instruction to the machine, where a new software layer opens up as a performance space for music, poetry, storytelling, dance and philosophy" (Saphiro, 2014). As such, creative coding seems a logical progression in the artistic tradition of collage. If the collage restructures the spatial organization of images on a flat canvas and the remix restructures the temporal organization of audio-visual media then creative coding reconfigures the relations between actors. Creative coding thus allows the artist to explore the distribution of agency. If the principal concern in contemporary artistic practice is the composition of relations through operations, then perhaps indeed "the greatest interactive work is the interactive human-computer interface itself" (Manovich, 2003, 15).

[14] Exactly how the Emotiv EEG device (http://www.emotiv.com) and the devices produced by rivaling commercial companies transcode the captured brain signals into digital code is not publicly available. The company only provides detailed instructions on how to handle the data output.

Actor Networks and material semiotics

The Eunoia project could thus be described as a complex network of ideas, physical artefacts and interactions bound together in software code. As such it seems only a small step to the Actor-Network-Theory (ANT) which "describes the enactment of materially and discursively heterogeneous relations that produce and reshuffle all kinds of actors including objects, subjects, human beings, machines, animals, "nature," ideas, organizations, inequalities, scale and sizes, and geographical arrangements" (Law, 2009, 141). Though this brief description may effectively summarize ANT, a few clarifications need to be made. Both Latour (1996, 1999) and Law (2009) express their discomfort with the term "Actor Network Theory" as it may easily lead to misunderstandings. Law argues that "it is better to talk of "material semiotics" rather than "actor network theory." This better catches the openness, uncertainty, revisability, and diversity of the most interesting work" (2009, 142). Latour playfully suggests adding hyphen thus writing 'actor' 'network' 'theory' to underline that each element may easily be misinterpreted (1999, 24).

Let me briefly address the three terms combined in ANT. First, Latour (1996) stresses that "an "actor" in AT is a semiotic definition -an actant-, that is, something that acts or to which activity is granted by others. It implies *no* special motivation of *human individual* actors, nor of humans in general. An actant can literally be anything provided it is granted to be the source of an action" (1996, 7). Thus agency is in flux; it is distributed and activated through a network rather than bound to the human agent. Second, the word 'network' may suggest a technological network in the sense of engineering, or in the sense of the computer network. However, as mentioned earlier, the network in ANT can include anything that is related to action. It is important to emphasize that the networks that are of interest here are not "locked" in the (software) interface; they may includes the human performer(s), the physical hardware, the audiences, the communities of makers that engage in producing digital artefacts, ideas in (artistic) discourse and so on. Third, ANT is an approach, not a theory (Latour, 1999, 19; Law, 2009, 141). Law explicitly underlines the importance of stories in material-semiotics (2009, 143) stressing that the narrative is

performative, not innocent (2009, 155). The Eunoia Project well illustrates how different traditions may construct very different networks of relations, thus creating stories that by no means merely describe the "real", they also produce the realities that they aim to depict. The ways in which the signs, symbols, metaphors and stories that mark technology are being produced and simultaneously are being productive should not be neglected. Software in general, and the syntax of digital code in specific are frequently portrayed in terms of neutral abstractions. Material-semiotics reminds us of the crucial role of the metaphor in digital technology, not only in representation and discourse but also in the very thingness of digital objects themselves (Van den Boomen, 2014, 188).

Liveness in performance

Even though "the general point that agency is distributed between humans and nonhumans has been well established in many different fields" (Mackenzie, 2006, 9) in the context of artistic performance some difficulties may arise. In *Performance studies: An introduction* (2013) Schechner states: "whatever is being studied is regarded as practices, events, and behaviors, not as "objects" or "things". This quality of "liveness" – even when dealing with media or archival materials – is at the heart of performance studies" (Schnechner, 2013, 2). Though the subject of interest here is software as an artistic tool rather than the artistic expression itself, and even though this thesis should be situated within software studies rather than in performance studies still Schechner points at an important issue: the virtue of "liveness" in performance.

I wish to avoid the complex discussion on what exactly may count as live, still "liveness is not an ontologically defined condition but a historically variable effect of mediatization. It was the development of recording technologies that made it both possible and necessary to perceive existing representations as "live"" (Auslander, 2012, 3). As such the use of software may radically alter the shape of live performance, but that does not necessarily mean that the use of technology limits

human expression. Puckette (2007) explains that when realizing both Pure Data and Max MSP[15] the capability of live performance was, and presumably still is an explicit goal in realizing these software tools (Puckette, 2007, ix). The emphasis on live performance in these software interfaces seems to underscore that "the phrases *live broadcast* and *live recording* suggest that the definition of what counts as live has expanded well beyond its initial scope as the concept of liveness has been articulated to emergent technologies. And the process continues, still in relation to technological development" (Auslander, 2012, 5-6). My point here is that the use of software allows for artistic arrangements and live composition that supersede a distinct between "live" versus "recorded" or "analogue" versus "digital". Auslander (2012) concludes: "digital liveness emerges as a specifc *relation* between self and other, a particular way of "being involved with something." The experience of liveness results from our conscious act of grasping virtual entities as live in response to the claims they make on us" (Auslander, 2012, 10). It seems fair to assume that these "virtual entities" are similar – or at least not structurally different from – the actor or actant in ANT. Analysing the Max MSP software used in digital performance and interactive installations will provide an insight how these 'virtual entities' can be arranged trough and in creative code.

[15] Both Pure Data (https://www.puredata.info/) and Max MSP (https://cycling74.com/) are visual programming tools primarily used by electronic musicians, VJs, dancers, theatre performers and others working with different forms of real-time performance (Manovich, 2013, 333).

Digital Material

New media objects

In *The Language of New Media* (2001) Manovich refers to his method of enquiry as "digital materialism" (2001, 35) explaining that "rather than imposing some *a priori* theory from above, I build a theory of new media from the ground up. I scrutinize the principles of computer hardware and software, and the operations involved in creating cultural objects on a computer, in order to uncover a new cultural logic at work" (2001, 35). Manovich analyses a wide variety of existing (new) media technologies and the content of a broad selection of what he refers to as "new media objects". These "objects" can be as small as a single pixel and as large as the entire World Wide Web. In this approach the "object" functions as a key concept as it serve as point of departure in exploring software as a cultural phenomenon.

Manovich identifies five hierarchically structured principles that "should be considered not as some absolute laws but rather as general tendencies of a culture undergoing computerization" (2001, 49). The five key principles are: (1) Numerical representation; (2) Modularity; (3) Automation; (4) Variability; and (5) Transcoding. For Manovich it is the last principle that describes "the most substantial consequence of media's computerization" (2001, 63). The fifth key principle thus seems more complex and more powerful than the first four principles. Manovich explains:

> The ways in which computer models the world, represents data and allows us to operate on it; the key operations behind all computer programs (such as search, match, sort, filter); the conventions of HCI — in short, what can be called computer's ontology, epistemology and pragmatics — influence the cultural layer of new media: its organization, its emerging genres, its contents (2001, 65).

Obviously, the "object" should be understood as a direct reference to computational science as Manovich advocates adopting "the terms and paradigms of computer

science for a theory of computerized culture" (2001, 39). Indeed, one can hardly overestimate the influence of digitalization on contemporary culture, and indeed the terms and paradigms of computer science can hardly be ignored. Still the computer sciences terminology has its limitations. Or, and as will be argued here, real-world analogies and cultural logic are of crucial importance when analysing the material "nature" of digital code.

Avant-garde

That does not mean that Manovich ignores the cultural layer as a whole. Besides the obvious links to computational sciences and object orientated programming for Manovich the "object" also invokes the ideas of the early twentieth century avant-garde movement as "the word pointed toward the model of industrial mass production rather than the traditional artist's studio, and it implied the ideals of rational organization of labor and engineering efficiency which artists wanted to bring into their own work" (2001, 39). Thus for Manovich the "new media object" should be understood in the light of the early twentieth century modernist ideology as "one general effect of the digital revolution is that avant-garde aesthetic strategies came to be embedded in the commands and interface metaphors of computer software. In short, the avant-garde became materialized in a computer" (2001, 306-307).

Linking new media to the early 20th century avant-garde and the modernist movement is most certainly useful. Still it seems to me that the modernist ideology of "rational organization and engineering efficiency" only captures part of contemporary techno-culture. That is to say, producing new media is not limited to the professional media designer (Jenkins, 2012) and software production is not bound to the institutions and methodology of engineering (Rieder & Schäfer, 2008, 168). I will come back to the issue of cultural logics in the final chapter of this thesis, my aim here is to analyse the material "nature" of the digital code used in visual programming. I will follow Manovich (2013) in "close reading" the Max MSP software interface though I will do so recognizing that the computer and the software interface are extremely marked by metaphor and analogy (Van den

Boomen, 2014).

The Object as a Data Structure

In more recent publications Manovich seems to have abandoned the "new media object", now promoting "data structure" as "the term will keep reminding us that what we experience as "media," "content" or "cultural artifact" is technically a set of data organized in a particular way" (2013, 201). Or, as Manovich summarizes:

> Software simulation substitutes a variety of distinct materials and the tools used to inscribe information (i.e., make marks) on these materials with a new hybrid medium defined by a common data structure. Because of this common structure, multiple techniques that were previously unique to different media can now be used together (Manovich, 2013, 203).

For Manovich "the crucial factor is not the tools themselves but the workflow process, enabled by "import" and "export" operations and related methods ("place," "insert object," "subscribe," "smart object," etc.), that ensure coordination between these tools" (2013, 300). From there Manovich concludes that at a more fundamental level the same goes for visual techniques and design strategies: "the same software-enabled design strategies, the same software-based techniques, and the same software-generated iconography are now found across all types of media, all scales, and all kinds of projects" (2013, 303).

Conceptualizing "media", "content" and "cultural artifacts" as objects defined by a common data structure is useful as it well describes the general effect of computation in contemporary culture. But it is also useful because creating data structures and controlling data flow is at the heart of visual programming. In Max MSP "media", "content" and "cultural artifacts" are indeed represented as objects that can be organized in a visual data structure. For example: a movie clip is displayed using the movie object (see figure 1) that can be controlled using the inlet(s) and the outlets of the visual object. The inlet(s) can receive messages, the outlets send out messages. But the movie object does not display the movie clip

content by itself; it is merely a placeholder waiting for instructions. A movie clip has to be loaded into the object by adding an argument – the filename and path to the location of the movie clip - or by sending a "read" message to the movie object inlet (see figure 1). The movie clip content is not displayed within the max MSP workspace; it is opened in a separate output window after receiving a "start" message. Finally, the outlets can be used to send messages to other objects. Thus (media) objects can be connected and the parameters of the one can be used to influence the other. Music notes may be used to generate visuals; an excel sheet containing financial data might be interpreted as a musical composition and EEG data could be used to trigger sounds and the motion of water.

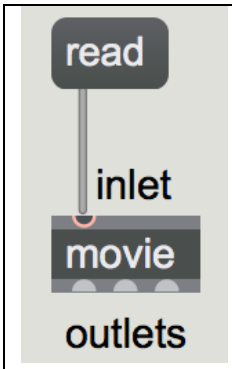

Figure 1 The movie object in Max MSP

In most of the mainstream productivity software "import", "export" and related operations can only be accessed through drop down menu's and pop-up dialog windows. The data structure of the object itself is hidden from the user. In Max MSP these operations and methods allow the artist or maker to combine, remix, reconfigure and repurpose the parameters of objects through (visual) programming code. Max MSP thus allows artists and makers to question conventions in mainstream culture and existing cultural forms. That does not mean that it is easy to do so, it requires creativity and ingenuity to combine, remix, reconfigure or repurpose "media," "content" and "cultural artifacts" through creative code.

Processing

Of course visual programming in Max MSP is just only one of the many approaches towards creative coding. One example of a programming environment popular amongst artist and makers is Processing[16]. The official processing website reads: "Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology". Though

[16] Processing (https://www.processing.org) is fully compatible with Java, Python and Ruby and is available as a free download

aimed at the creation of visuals and the visual arts Processing relies on the same coding conventions the same code syntax as other programming languages. Playing a movie clip thus requires a short script – or *sketch* - to be executed:

```
import processing.video.*;
Movie myMovie;

void setup() {
myMovie = new Movie(this, "/path/to/filename.mov");
  myMovie.loop();
}

void draw() {
image(myMovie, 0, 0);
}

void movieEvent(Movie m) {
  m.read();
}
```

The “Import” command in the first line includes the code library for handling movie clip data. The second line creates an empty variable that will be used to hold the movie clip. In the setup the location of the movie clip is specified, and finally the movie clip is displayed in a separate window using the “draw” and “movieEvent” functions.

The *Sketch* (Processing) and the *Patch* (Max MSP) do the same thing in terms of functionality. A variable is created as a placeholder for the movie clip. The movie clip object is then loaded into the variable and displayed in a separate output window. Still there is a great difference between the *patch* and the *sketch.* Obviously the code is represented in a very different way, as a visual mapping versus linguistic syntax. But the differences between *Patch* and *Sketch* are not merely differences in style; they also shape the perception of the computer itself. The syntax based coding style used in Processing emphasises the abstract “nature” of software code; the visual programming style accentuates the “thingness” of the computer hardware.

The Object as a Metaphor

Both Max MSP and Processing are explicitly intended as artistic tools, but that does not necessarily make them similar tools. The quite obvious metaphors of *Patching* and *Sketching* illustrate the specific character of these tools. Max MSP is aimed at building custom interfaces for (live) performances; Processing is aimed at creating visual effects using software code. That does not mean that Processing cannot be used in live performance or that Max MSP is unsuitable for exploring visuals. In fact they are often combined or used together as both have their specific strengths and weaknesses. Still the differences between *Patch* and *Sketch* illustrate that metaphors and analogies do matter when analysing software as an artistic tool. Or, as Van der Boomen points out: "software is inextricably connected to and formatted by metaphors and analogies" (2014, 156). Van der Boomen stresses that in the context of digital materiality the metaphor should not merely be understood as a conceptual metaphor. The notion of the metaphor should be extended because in digital praxis the metaphor is embodied in discourse, but also in the objects themselves:

> Digital computer technology is extremely marked by metaphors. Here, metaphors nestle themselves not only in the representations of the technology and the discourse on its use and functions, but also in the technological objects themselves: the very thingness of digital objects consists of metaphors made material and operational. (Van den Boomen, 2014, 188).

Thus the metaphor is not merely a way of conceptualizing digital material, it also shapes the very "nature" of the digital object. In other words, the metaphor is "inscribed" into the object. One of the crucial problems in understanding the material "nature" of software code then is that "the complexity of digital code is necessarily black boxed in user-friendly interfaces, and this makes assumptions of mysterious immateriality hard to exorcize" (Van der Boomen et Al., 2009, 9).

Remediation

The concept of remediation (Bolter & Grusin, 2000) has played a vital role tracing existing media formats, metaphors and real-world analogies in digital technology. Bolter and Grusin explain: “what is new about digital media lies in their particular strategies for remediating television, film, photography, and painting. Repurposing as remediation is both what is "unique to digital worlds" and what denies the possibility of that uniqueness” (2000, 50). Indeed, “if we limit ourselves to looking at the media surfaces, the remediation argument accurately describes much of computational media” (Manovich, 2013, 59). Still one should not merely think of the computer as a “remediation-machine” as “computer simulations of physical media can add many exciting new properties to the media being simulated” (Manovich, 2013, 86). Thus “while on the level of appearance computational media indeed often remediate (i.e. represent) previous media, the software environment in which this media “lives” is very different” (Manovich, 2013, 86). Or, as Manovich concludes from “close reading” the “Wave Filter” in Adobe Photoshop: “many algorithms only simulate the effects of physical tools and machines, materials or physical world phenomena when used with particular parameter settings; when these settings are changed, they no longer function as simulations” (2013, 136).

Manovich makes an important point by stressing that software should not merely be understood as a simulation of already existing media or real-world practices. Indeed, the parameter settings of the tools and filters in Adobe Photoshop can be abused and exploited. Simultaneously the Adobe Photoshop interface also lacks the directness and the physicality of the real-world process of painting on a canvas or drawing on a sheet of paper. Still the “user” is severely limited by the conceptual space provided by the Adobe Photoshop canvas. Still, and even though some exploits may be available, the Adobe Photoshop interface is designed as a specific kind of productivity tool, optimized for the production of digital images. But the metaphor does not merely touch the visual surface and the conceptual notion of the computer interface. The crucial point is that in max MSP “media”, “content” or “cultural artifact” is *performed* rather than produced.

The visual programming interface

Thus the metaphor also operate on a deeper - and often hidden – layer within the software code (Van der Boomen, 2014). The Max MSP (visual) programming interface reveals how the metaphor is "inscribed" in the software code: the "inlets" and "outlets" of an object define how the object interacts with its environment. The "movie object" discussed earlier can only send and receive a specific subset of messages that are defined by the movie clip metaphor. The same principle applies to other programming languages. An object is always bound by a specific subset of operators, attributes and libraries that allow for interactions with other objects. In other words, the metaphor structures the relations between the "user" and the software interface as well as the relations between objects in software code. Let me provide two very basic examples to illustrate my point. The first is a simple calculation (figure 2), the second is a very basic single tone generation patch (figure 3). Both examples were made using Max MSP version 7.

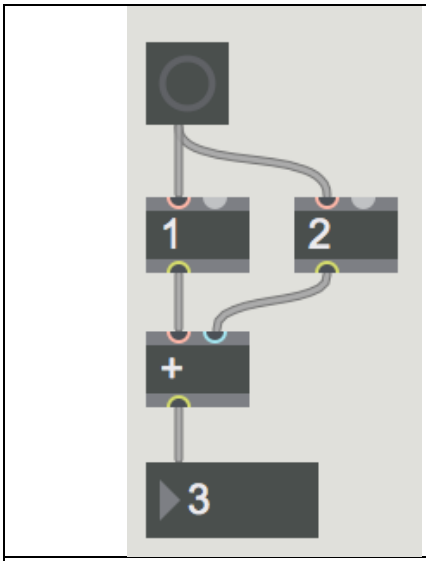

Figure 2 Basic Calculations Patch in Max MSP

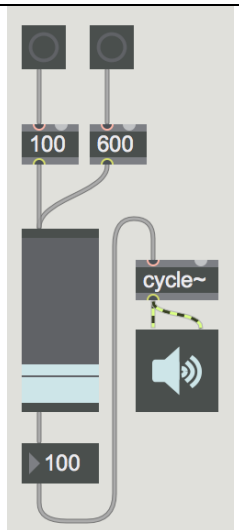

Figure 3 Tone Generation Patch in Max MSP

To create a simple calculation in Max MSP several objects have to be combined or "patched" together. The first object in this diagram (figure 2) is a button that initiates the calculation by sending a "bang" through its bottom outlet. The "bang" is

received through the left inlets of the number boxes that display and output a number, in this case the values of “1” and “2”. The math operator receives the numbers and adds them together. The result, in this case “3” is displayed in the number box that completes the diagram, or *patch*. Up till this point the term “data structure” seems to adequately describe the Max MSP interface, as indeed the interface is a visual representation of data flowing through a system. Thus the basic idea behind visual programming is “boxes and arrows”, where boxes represent objects and arrows represent relations between these objects. One might also notice that for the simple calculation in this example the visual representation seems quite inefficient. In most common syntax a single line of code could perform the same operation. Still the ridiculously simple operation of adding 1 and 2 as described here is not possible within most of the standard media productivity software. One could say that brushes, paint buckets and selection tools are not very suitable tools for basic maths.

The second example (figure 3) shows a slightly more complicated patch used for generating and manipulating a single tone. The values in the two numerical boxes represent the minimum and maximum value of the slider object they connect to. Here the slider can be set to a minimum of 100 and a maximum of 800 using the buttons, or it can manually dragged to any value in between. These values are not fixed or predefined; they are merely used as an example. The number box below the slider displays its current value and sends passes it to the cycle~ object that operates as a standard oscillator generating a tone based on the value it receives. The tone is then passed to the ezdac~ object at the end of this chain, the recognizable audio icon (the user interface version of the dac~ object) that serves as an output for sounds. The Max MSP reference entry for the slider reads: “the slider is a user interface that resembles a sliding potentiometer”. Thus the slider resembles an electric component, a voltage divider used to measure electric potential. The potentiometer is widely applied in electronic devices; one of its many appliances is controlling oscillators, the electric components used to generate sound. Of course this patch is far less complex in terms of wiring compared to its analogue counterpart, and the slider is not controlled by an actual potentiometer somewhere

inside the machine. In contrast, the dac~ object does refer to a very real component found every digital device capable of producing sound. Every mp3 player, iPhone, iPod, laptop computer, DVD player or smart watch needs a Digital-To-Analog-Converter. The Max MSP reference explains: "The dac~ object is the Digital-To-Analog-Converter through which you will route all signals from MSP out to your computer speakers or audio hardware to be audible to the human ear. It also gives you access to the Audio Status window which controls your audio settings and hardware". This very basic example clearly shows how the physical hardware and the electronic circuitry of the computer itself are part of the metaphors underpinning the visual programming interface.

Inscription

The metaphors used in mainstream software may refer to the craftsmanship of the 19th century, the 20th century avant-garde agenda or the modern office environment. Still the computer *itself* and its hardware infrastructure seem to be avoided. The Max MSP patch shows that where "data structure" should remind us "that what we experience as "media," "content" or "cultural artifact" is technically a set of data organized in a particular way" (Manovich, 2013, 201); as a counterpart the "object" should remind us that these data structures are inscribed into the physical "stuff" that computers are build off. One could also say that the slider, cycle~ and dac~ objects simulate the computer itself, reshaping its circuitry into creative material. The visual programming interface reconnects software code with the hardware environment in which the code "lives". Still the notion of material inscription also holds for software development in general. Or, as Van der Boomen (2014) argues:

> For software developers the inscription metaphor, with all its layers and ambiguities, is a material metaphor that profoundly organizes their daily work: inscribing code, editing, modifying, running it, revising, testing, putting it in a version control system, and so on. But that material metaphor is not supposed to travel outside the programmers' den. After all, the imperative

> for user-friendly software is precisely to make the large chain of translations invisible for ordinary users" (van der Boomen, 2014, 101).

The Processing platform explicitly aims at demystifying software code, and in MIT's Scratch software the visual programming approach is used in an attempt to increase code literacy amongst children and teenagers. The Max MSP community seems less explicit in its concern with these affords in increasing code literacy. Of course there is a broad range of tutorials available online, starting from the most basic level, and code fragments and patches are actively shared. Still the primary concern within the Max MSP community is the exploration of the computer as an artistic and performative platform. One might add that in the specific context of artistic performance the somewhat mythical quality of software code might even be useful as illusion and enchantment are powerful tools in the artistic repertoire.

The Computer as a Meta-medium

For Manovich "the greatest interactive work is the interactive human-computer interface itself" (2003, 15); and "the greatest avant-garde film is software such as Final Cut Pro or After Effects which contains the possibilities of combining together thousands of separate tracks into a single movie" (2003, 15). Dixon (2007) firmly rejects the suggestion of artistic quality in mainstream software or the standardized computer interface. Dixon points out that this proposition would be the same as "to propose the theatre building as the greatest piece of theater, since that is where the finest performance *can* – or may, at some time, perhaps, be staged" (Dixon, 2007). Where Manovich celebrates the notion of the computer as the "universal machine" the most common configuration of the computer interface can also be described as an anachronism:

> An anachronistic dinosaur of a machine that places file-cabinet icons borrowed from the nineteenth century offices onto a TV screen monitor design originated in the 1930's, above a QWERTY keyboard that, even when it was launched as a typewriter in 1878, was shown to have the worst possible letter pattern configuration" (Dixon, 2007, 6).

These positions represent two very different narratives that create a dichotomy between the conceptual model of what a computer could be and the very specific reality of the computer in the urban office and in everyday life. My point here is that these stories can be countered by artistic explorations of what the computer *might* be. Manovich adopts the description of Kay & Goldberg (1977) whom think of the computer as *"*a metamedium, whose content would be a wide range of already-existing and not-yet-invented media*"* (1977, 40). The notion of the computer as "a metamedium" is useful as it is inclusive; it combines the conceptual ideal of the "universal machine" with the reality of everyday practice. If the computer is indeed to be understood as a "metamedium" then it is of vital importance to permanently re-imagine its configuration. Max MSP invites the artist to explore the computer as a meta-medium and creative coding in Max MSP allows for the playful re-imagination and reconfiguration of the computer interface.

In the previous chapter it has been argued that creative coding has opened a new kind of performative space (Saphiro, 2014). Following Auslander (2012) It has been suggested that the "experience of liveness results from our conscious act of grasping virtual entities as live in response to the claims they make on us" (2012, 10). We may now add that these "virtual entities" are far from "virtual". The "virtual entity" materializes in a data structure, and it materializes in the composition of objects shaped by metaphor and analogy. It has also been argued that - in line with the Actor Network approach (Latour, 1996, 1999; Law, 2009) - these "virtual entities" can best be understood as "actors" or "actants" in a complex network that combines human actors, technologies, ideas and cultural practices. Creative coding in Max MSP allows the artist or maker to visualise and orchestrate actions in actor networks, exploring the computer as a "meta medium".

Cultural logic

DIY and Maker Culture

The artists and makers that make use of Max MSP do not strive towards "the rational organization of labor" or "engineering efficiency". Still Manovich's

observation that “the avant-garde became materialized in a computer” (2001, 306-307) is crucial because it signifies that our understanding of the computer interface is grounded in cultural logics. If the (new) media designer is to be understood as a professional working within the “creative industries” then Manovich most certainly has a point. And perhaps indeed the entire field of the arts has been infused with the suggestion that the “production” of human culture should be understood as an industrial process (Adorno & Horkheimer, 2007, 34 -43).

But simultaneously “software production flourishes outside of the classical institutions and methodology of engineering” (Rieder & Schäfer, 2008, 168). Rieder & Schäfer emphasize the importance of the open-source scene that “distinguishes itself from traditional engineering in social norms and general mindset” (2008, 166). This “general mindset” of the open-source scene, though diverse on its own, seems part of an even broader and more diverse cultural phenomenon: *Do-it-Yourself (DIY) or Maker Culture*. DIY or Maker Culture holds its own specific values, and the way that specific concepts are understood may radically differ from the global business environment. These differences are far from semantic; they express values deeply anchored in distinct cultural positions. A key example is “creativity”, a concept formerly associated with the arts and humanities that over the past decade has gradually been incorporated in management-lingo. From a business perspective creativity should be applied and exploited as a valuable resource, from the maker perspective creativity is to be understood as a virtue on its own. Or, as Kuznetsov & Paulos (2012) conclude: “DIY communities and projects are driven by creativity. The vast majority of our respondents contribute to DIY communities not to gain employment, money or online fame, but to express themselves and be inspired by new ideas” (8). Still maker practices also rely on the infrastructure provided by commercial enterprise and the formal institutions. Even though maker practice may flourish outside the walls of the institution, still “some of the most successful platforms in the DIY world have emerged from universities” (Tanenbaum et Al., 2013, 2610). In other words, the university can play a vital role in connecting creative communities.

Amateur Operators

It may be tempting to understand contemporary maker practices and DIY culture as an effect of digitalization and networked society but in many ways the ideas, ideals and practices hold within contemporary DIY or Maker Culture are far from new:

> In the 1980's, the low-cost MIDI equipment enabled people without formal training to record electronic music, evolving into the rave culture of the 1990's. During this time, computer hobbyists also formed communities to create, explore and exploit software systems, resulting in the Hacker culture. (Kuznetsov & Paulos, 2012, 1).

Modern desktop or laptop computers may be traced to the fundamental work done by epic figurines such as Vannevar Bush, Alan Key or Alan Turing; still local, "amateur" computer clubs such as the *Homebrew Computer Club* in the vicinity of Palo Alto also played a vital role in popularizing the personal computer during the 1970's and 1980's. Contemporary maker culture may also be linked to the birth of the radio and the era of the amateur operator: "one of the earliest "modern era" DIY communities formed among amateur radio hobbyists in the 1920's. These hobbyists relied on amateur handbooks, which stressed "imagination and an open mind" nearly as much as the technical aspects of radio communication" (Kuznetsov & Paulos, 2012, 1). Indeed, "the amateurs didn't just adopt this new technology; they built it, experimented with it, modified it, and sought to extend its range and performance. They made radio their own medium of expression" (Douglas, 1986, 44). But the enthusiasm for the radio was ideological as much as it was technological; it also served as a vessel for utopian dreams:

> The properties of radio seemed to perfectly encapsulate the recurrent dream of universal and direct communication . . . It was not just telegraphy or telephony without wires; everyone with a receiver could tune in and feel connected to a virtual community. Because of the messianic character of live broadcasting, the popular idea took hold that radio could be a tool to

> establish social cohesion and world peace, bringing direct democracy and global harmony to the people (De Vries, 2012, 114).

Of course there are many fundamental differences between the era of the amateur operator and contemporary DIY culture and of course one should be weary not to impose the present onto the past. Still there is a striking similarity between the ideals expressed in early 20th century discourse surrounding amateur operators and contemporary DIY culture. A similar notion of "democratized technological practices" can be found in the contemporary discourse on DIY: "The rise of maker culture is creating new values around technological practices" (Tanenbaum et Al., 2013, 2605).

In contrast to the amateur operators in the early days of the radio contemporary maker culture does hold such high expectations towards technological innovation itself. In contrast, maker culture celebrates a somewhat nostalgic or romantic interest in "out-dated" (media) technology rather than a utopian belief in salvation through innovation. On the other hand, contemporary DIY and maker culture most certainly "use artifacts to explore a new and present future of design, fabrication, and consumption. These designs often embody cultural imaginations, engaging in the relatively recently articulated concept of *design fiction*" (Tanenbaum et Al., 2013, 2606).

Design fiction

The concept of "design fiction" introduced by Bleecker (2009) can be summarized as "a conflation of design, science fact, and science fiction. It is an amalgamation of practices that together bends the expectations as to what each does on its own and ties them together into something new. It is a way of materializing ideas and speculations without the pragmatic curtailing that often happens when dead weights are fastened to the imagination" (Bleecker, 2009, 6). Bleecker clarifies his position as he writes: "I think of design as a kind of creative, imaginative authoring practice — a way of describing and materializing ideas that are still looking for the right place to live. A designed object can connect an idea to its expression as a made, crafted, instantiated object. These are like props or conversation pieces that help speculate,

reflect and imagine, even without words" (2009, 6). Thus design fiction and science fiction can be closely related though the first explicitly includes maker practices:

> Design fiction aims at creating conversational pieces that aid discussions on how this imagined future would look, feel, and be lived in; it is a technique for reflecting on what technology we should, or should not, design (Tanenbaum et Al., 2013, 2606).

However, design fiction should not overhasty be embraced as a vehicle for imagining the future. In an extensive analysis of science fiction novels Jameson (2005, 286) argues that it is not so much the future as well as the present that is under negotiation. In other words, science fiction restructures our experience of the present. Jameson concludes:

> We no longer entertain such visions of wonderworking, properly "science-fictional" futures of technological automation. These visions are themselves historical and dated – streamlined cities of the future on peeling murals – while our lived experience of our greatest metropolises is one of urban decay and blight" (Jameson, 2005, 286).

Still the "streamlined cities of the future on peeling murals" provide a rich repository for repurposing and remixing the history of the future. One might say that the makers in maker culture explore the dreams of the past rather than facing the uncertainties of an unpredictable future. One might also say that their works are introspective rather than extravert. "Collectively, subculture members create a common vision of what that subculture is, and how the practice of making fits in" (Tanenbaum et Al., 2013, 2607). Maker culture creates its own multiplicity of alternative modernity's favouring playful re-imagination and creative exploration over practical utility and rigid usability. Interpretations are playful, creative, artistic, poetic, romantic or even nostalgic but by no means historic or predictive in a scholarly sense. They are build by creators that make use of cultural artefacts and technological practices from past days as well as the present, creating a diverse

imaginary, a remix of the many histories of already past futures. History and future convolute in contemporary maker culture.

Conclusions

Three overarching themes – materiality, agency and sociality (Mackenzie, 2006, 171) - have been set out as an approach to answering the initial question: How can visual programming be understood as a creative tool? How then can visual programming be understood, in terms of agency, materiality, and sociality? Who does what, what counts as the basic stuff that exists and how do we belong together? And how these complex concepts intertwine? A first remark needs to be made concerning the question of agency, that is, following Actor Network Theory the question should actually read: Who *or what* does what? The question of *who, or what* and the complex layers of materiality will serve to conclude this thesis.

Materiality

Though easily recognizable in the visual programming environment, the principles of new media (Manovich, 2001) only touch the surface of the complex layers of digital materiality. Understanding the metaphors that shape digital material is crucial in unwrapping digital materiality (Van der Boomen, 2014). As such, these principles function as powerful metaphors too; they act as confirmation of the popular yet problematic virtual-real dichotomy. The visual programming interface illustrates that software cannot be separated from the hardware environment in which the code "lives".

In artistic practice the "desktop-metaphor" and the mouse-keyboard-screen interface seem problematic. The hallmark feature of both Max MSP and its open source counterpart Pure Data is the ability to create and control hardware devices to create music instruments, live performances, interactive installations, and so on. A useful yet complex solution then is the inscription metaphor; though the question then is: what exactly is it that is being inscribed? Most obviously there is data, and

the objects combined in data structures. But that far from answers the question. The question of *what* has to be redirected to the question of who-*or what* does what, as inscription invokes agency.

Agency

My suggestion – perhaps a provocative one – is that, at the least in the context of artistic digital media performance, agency itself has become the material of the 21th century artist. The *distribution* of agency is inscribed in the physicality of hardware. Thus another shift is made; back to layers of materiality. That is to say, the metaphors underpinning visual programming remediate the materiality of computer hardware itself. The Max MSP interface can be read as a visual representation of the components inside the "black box". Visual programming thus reveals rather than hides. Visual programming allows the artist to create an assemblage of relations between objects; where the object represents action. As an artistic tool, visual programming may thus function as a tool for exploring actor-networks in the sense of actor network theory. Still visual programming should not be positioned in engineering; it allows for expression in and through code. Its goal is not to optimize the performance of hardware; it is an act of *creative coding* that attempts to make the hardware performative.

Sociality

Though visual programming may be positioned outside of the institutions of engineering; that does not imply that it should be placed within the institutions of the arts. The craftsman, inventor, engineer, artist and designer, perhaps these are themselves historical and dated, as much as the "streamlined cities of the future on peeling murals" of science fiction (Jameson, 2005). Still these "murals" do not merely portray the incapacity to imagine technological advance, they are also infused with hopes and dreams that reoccur through history (Huhtamo, 1997; De Vries, 2012). As a tool, visual programming may be associated with the "dream of democratizing technologies" held within contemporary maker culture. As a tool, visual programming allows for playful re-imagination of the history of the future. And as a

tool, it allows the artist, maker or creator to infuse machinery with subjective imagination and the narratives of design fiction.

Methodological reflection & further research

Perhaps one might hold that – at least till a certain extend – the perception of the computer interface evolves around *affordances*. Tracing affordances in human-computer interactions has been of great influence in "user-centred" software design and in the on-going agenda of realizing usable and useful interfaces; where "useful" should be understood as rigidly goal-orientated (McGrenere & Ho, 2000, 6). Or, as put by Norman (1990): "an interface is an obstacle: it stands between the person and the system being used. How can anything be optimal if it is in the way, if it stands between the person and what needs to be done?" (1990, 216) Simultaneously the concept of affordances has proven to be a valuable instrument in questioning what exactly it is that needs to be done and who is to determine what should count as useful. As such, the agenda of creating "useful" interfaces has been criticized as the construct of the "user" may reduce the human subject to a predictable and controllable object at the mercy of the designer, the engineer or marketer (Almquist &Lupton, 2010, 3). The concept of affordances has been avoided precisely for that reason: what is of concern here is the nature of (visual) code as an artistic material rather than the complex political relations instilled in the mainstream of productivity software.

The term "affordance" was first coined by Gibson (1977) and popularized by Norman (2013). For Gibson "the affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill" (1977, 127). Norman (2013) adds a crucial detail by stressing the importance of what is *perceived* as possible; based on knowledge and the perception of things (2013, 219). This thesis aimed at shifting the attention towards the use of visual programming in artistic maker practice. The argumentation presented here focussed on the visual programming interface as well as the "cultural logic" of DIY and maker culture; stressing the importance of the metaphor at the expense of affordances.

Still, and even though some critical remarks have been made concerning the understanding of the human subject as a "user" in the field of Human Computer Interaction, the concept of affordances is of vital importance in software studies. A thorough understanding of the differences between the visual representation of code and the far more common linguistic or syntax-based style of programming could help in further developing the understanding of code. Simultaneously software studies should not lock itself within the black box of code, the algorithm or the software interface. I have positioned the use of software in a specific context of DIY and maker culture, arguing that the understanding of software and the computer interface may radically differ between distinct fields and subcultures in (post) modern society. As such, material semantics is of importance as it helps in recognizing how the interface, affordances and metaphors are being understood. Of course (sub) cultures are complex, diverse and hard to grasp. If indeed the artist, designer and the engineer represent their own specific cultural logic, then it is important to expand the insight in how (sub) cultures understand software and digital code. The "nature" of digital code and digital materiality cannot fully be understood without a thorough understanding of distinct real-world cultural praxis.

References

Adorno, Theodor and Horkheimer, Max. (2007). The culture industry: enlightenment as mass deception. In S. Redmond & S. Holmes *Stardom and celebrity: A reader* (pp. 34-43). London: SAGE Publications Ltd. doi: 10.4135/9781446269534.n4

Almquist, Julka, and Julia Lupton. 2010. "Affording Meaning: Design-Oriented Research from the Humanities and Social Sciences." *Design Issues* 26, no. 1 3–14. doi:10.1162/desi.2010.26.1.3

Andersen, Christian Ulrik, and Søren Pold. 2011. Interface criticism: aesthetics beyond the buttons. Aarhus [Denmark]: Aarhus University Press.

Auslander, Philip. 2012. "Digital Liveness: A Historico-Philosophical Perspective". *PAJ: A Journal of Performance and Art.* 34 (3): 3-11. doi:10.1162/pajj_a_00106.

Bleecker, J. (2009). Design Fiction: A short essay on design, science, fact and fiction. *Near Future Laboratory*, *29*.

Bolter, Jay David, and Richard Arthur Grusin. 2000. Remediation: understanding new media. Cambridge (Mass.): MIT Press.

Cramer, F. (2002, March). Concepts, notations, software, art. In *Seminar for Allegmeine und Vergleischende Literaturwissenschaft*.

Cramer, Florian and & Ulrike Gabriel. 2001. "Software Art and Writing." In *American Book Review 22 (6), issue "Codeworks"* edited by Alan Sondheim.

De Vries, Imar. O. 2005. "Mobile Telephony: Realising the Dream of Ideal Communication?" In *Mobile world past, present, and future* edited by Hamill, Lynne, and Amparo Lasen. 11–28. New York, N.Y.: Springer.

De Vries, Imar O. 2012. Tantalisingly close an archaeology of communication desires in discourses of mobile wireless media. Amsterdam: Amsterdam University Press.

Dixon, Steve. 2007. A Digital Performance: History of New Media in Theater, Dance, Performance Art, and Installation. Cambridge, Mass: MIT Press.

Douglas, Susan. 1986. "Amateur operators and American broadcasting: shaping the future of radio." In *Imagining tomorrow: history, technology, and the American future,* edited by Joseph J. Corn. 34-57. Cambridge, Mass: MIT Press.

Fuller, Matthew, ed. 2008. Software studies a lexicon. Cambridge, Mass: MIT Press.

Gibson, James. J. 1977. The theory of affordances. *Hilldale, USA*.

Hesmondhalgh, David. 2006. Bourdieu, the media and cultural production. *Media, culture & society*, *28*(2), 211-231. doi:10.1177/0163443706061682.

Jameson, Fredric. 2005. Archaeologies of the future: the desire called utopia and other science fictions. New York: Verso.

Jenkins, Henry. 2012. Textual poachers: Television fans and participatory culture. New York: Routledge.

Kay, A., & Goldberg, A. 1977. Personal dynamic media. *Computer*, *10*(3), 31-41. doi:10.1109/c-m.1977.217672

Kittler, Friedrich. A. 1995. There is no software. *ctheory*, *10*(18), 1995.

Kuznetsov, S., & Paulos, E. (2010, October). Rise of the expert amateur: DIY projects, communities, and cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries.* 295-304. New York, NY: ACM

Latour, Bruno. (1996). “On actor-network theory: a few clarifications plus more than a few complications.” Soziale welt, 369-381.

Latour, Bruno. 1999 “On Recalling ANT.” *The Sociological Review* 47 (1). 15–25.doi:10.1111/j.1467-954x.1999.tb03480.x.

Law, John. 2009. "Actor Network Theory and Material Semiotics". In *The new Blackwell companion to social theory,* edited by Bryan S Turner. 141-158. Chichester, West Sussex, United Kingdom: Wiley-Blackwell.

Mackenzie, Adrian. 2006. Cutting code: software and sociality. New York, N.Y.: Peter Lang.

Manovich, Lev. 2001. The Language of New Media. Cambridge, Mass: MIT Press,

Manovich, L. 2003. “New media from Borges to HTML.” Introduction to *The New Media Reader*, edited by Noah Wardrip-Fruin and Nick Montfort. Cambridge, MA: The MIT Press, 2003, 13-25.

Manovich, Lev. 2013. Software Takes Command. New York: Bloomsbury, 2013. Print.

McGrenere, Joanna and Wayne Ho. 2000. “Affordances: Clarifying and Evolving a Concept”. In: *Proceedings of Graphics Interface 2000* 179-186. May 15-17, 2000, Montreal, Quebec, Canada.

Norman, Donald A. 1990. “Why interfaces don’t work”. In *The Art of human-computer interface design,* edited by Laurel, Brenda, and S. Joy Mountford.. Reading, Mass: Addison-Wesley Pub. Co.

Norman, Donald A. 2013. “The design of everyday things.” New York, NY: Basic Books.

Park, Lisa. 2015, March 2. “An Interview with Lisa Park.” Retrieved from https://cycling74.com/2015/03/02/an-interview-with-lisa-park/

Pold, Søren. 2008. "Preferences/settings/options/control panels." In *Software studies: a lexicon,* edited by Matthew Fuller. 218-224. Cambridge, Mass: MIT Press

Puckette, Miller. 1991. "Something Digital". Computer Music Journal. 15 (4): 65-69.

doi:10.2307/3681075.

Puckette, Miller. 2007. The theory and technique of electronic music. Hackensack, N.J.: World Scientific Publishing Co.

Rieder, Bernhard, and Schäfer, Mirko Tobias. 2008. "Beyond Engineering: Software Design as Bridge over the Culture/Technology Dichotomy". In *Philosophy and Design: From Engineering to Architecture,* edited by Pieter E. Vermaas. 159-171. Dordrecht: Springer Netherlands.

Saphiro, Alan. N. 2014, may 17. “Code as Expanded Narration.” retrieved from: http://www.alan-shapiro.com/software-code-as-expanded-narration-by-alan-n-shapiro/

Schechner, Richard. 2013. Performance studies: an introduction. London: Routledge.

Stinson, Liz. 2014, “Watch An Artist Control Pools of Water With Her Brainwaves”. Wired Magazine, November 24. Accessed July 6, 2015. http://www.wired.com/2014/11/watch-artist-control-pools-water-brainwaves/

Tanenbaum J.G., Desjardins A., Tanenbaum K., and Williams A.M. 2013. "Democratizing technology: Pleasure, utility and expressiveness in DIY and Maker practice". Conference on Human Factors in Computing Systems - Proceedings. 2603-2612.

Van den Boomen, Marianne. 2014. Transcoding the digital: how metaphors matter in new media. *Theory on demand*, *14*. Amsterdam: Institute of Network Cultures, 2014.

Van den Boomen, Marianne, ed. 2009. Digital Material: Tracing New Media in Everyday Life and Technology. Vol. 2. Amsterdam: Amsterdam University Press.