Public CulturalProduction Art(Software){

Christiane Paul

The focus on notions of software as art has increased in recent years, which to some extent can be seen as a logical and inevitable consequence of the nature of the digital medium and the power of its structures and rules. Software is a driving force of the digital medium—a creative tool that is culturally and politically "encoded" and embedded in a commercial system.

Software is generally defined as formal instructions that can be executed by a computer. However, there is no digital art that doesn't have a layer of code or algorithms, a procedure of formal instructions that accomplish a "result" in a finite number of steps. Even if the physical and visual manifestations of digital art distract from the layer of data and code, any "digital image" has ultimately been produced by instructions and the software that was used to create or manipulate it. It is precisely this layer of "code" and instructions that constitutes a conceptual level which connects to previous artistic work such as Fluxus' and Dada's experiments with formal variations and the conceptual pieces by Duchamp, Cage and Sol LeWitt that are based on the execution of instructions.

However, it is important to distinguish data constructs such as digitized images or texts from algorithmic code that enables generative processes. One has to make a major distinction between art that uses digital technologies as a tool in the creation process and results in a "traditional" art object (print, photograph, painting, sculpture), and art that employs the technologies as a medium—that is, art that has been created and is stored and presented by means of them. It is only the latter that can potentially exhibit generative processes in real time. Code also cannot be understood as separable from its overall structure. As Adrian Ward, Alex McLean, and Geoff Cox pointed out in *The Aesthetics of Generative Code,*[1] the written form of code—as "a notation of an internal structure that the computer is executing, expressing ideas, logic, and decisions"—is "a computer-readable notation of logic" and not what the computer really executes. The execution takes place through various layers of interpreting and compiling.

Software as art has been discussed in the wider context of generative art (for example at *generative.net)*[2] and has more recently been explored in the context of festivals such as Read_me[3] (explicitly devoted to this art form) and the software art award at Transmediale,[4] or the runme software art repository,[5] an open, moderated database that launched in January 2003. The introduction to the latter site describes software art as a crossover between two seemingly unrelated realms, software and art: while software culture is considered a "living substance" that to a large extent evolves on the Internet and stems from and permeates various cultural realms, art is traditionally presented in exhibitions in galleries and museums or at festivals.[6] The "software art" fusion consequently would introduce software culture into the art world and at the same time expand art beyond its institutional boundaries.

The consideration of software as an art form evidently raises a number of questions: can any software be considered art and, if not, where do we draw the line between software as art and software as a "mere" commercial product? To what extent is the identity of so-called "new media art" or digital art defined by its nature as art based on code? What

are the aesthetics of software art and how can they be assessed by traditional art-immanent criteria (or should they be at all)?

Software = Art / (Product + (Formalism / Culturalism))

The Read_me 1.2 jury broadly defined software art as art based on code as formal instructions, or art offering a cultural reflection of software, definitions that cover a broad territory. If one takes a look at the subcategories listed on the runme repository's site, one encounters a landscape that may be fairly confusing in its topography but nevertheless makes important distinctions and can still be roughly summed up under the above mentioned definitions. Labels such as algorithmic appreciation, generative art, code poetry, data transformation, as well as digital folk and artisanship (e.g. ascii art and screen savers) arguably seem to put an emphasis on the aesthetics of formal instructions. On the other hand, classifications such as existing software manipulations (cracks and patches or plug-ins) or political and activist software (e.g. cease-and-desist-ware and software resistance) point to the role of software art as critical reflection of software's cultural status its encoded political or commercial agenda. Games, artistic tools, and conceptual software can fall into either of these two groups, depending on the execution of the respective project and the weight it places on formal aspects or critical reflection.

It would be difficult to argue that software—from Adobe Photoshop to Maya—can in and of itself be considered art. The "art aspect" manifests itself in artist-written software (concept, "writing style," results of execution etc.) or the re-writing / re-engineering of existing software as an act that examines the underbelly, inscribed aesthetics, and agenda of the original construct and thus opens it up to discussion. The inherent hope and promise here is that software production can be seen in the broader context of cultural production or, as Pit Schultz has put it, "that writing code has more meaning than making a program run or crash or sell."[7]

If one rephrases the above classification of software art—as either focused on code as formal instructions or on cultural reflection—as a manifestation of formalism vs. culturalism, one has to pose the question if these are opposite ends of a spectrum and if software art overall can be distinguished according to these categories. In "Concepts, Notations, Software, Art," Florian Cramer outlines that much of contemporary software art takes two opposite approaches to software art and software criticism: either "software as first of all a cultural, politically coded construct" or a focus on "the formal poetics and aesthetics of software code and individual subjectivity expressed in algorithms."[8] The inherent dangers that Cramer identifies for each of these approaches are indeed important to note: as he puts it, a reduction of software art to the first one could make it "a critical footnote to Microsoft desktop computing" that neglects the potential of formal explorations; the second approach could result in "a neo-classicist understanding of software art as beautiful and elegant code." The latter is exemplified in the criteria for evaluation established by computer scientist Donald Knuth, who has been talking about "computer programming as an art" since the 1970s[9]: among these are correctness, maintainability, lucidity, grace of interaction with users, and readability (which would make the Obfuscated C Code Contest a failure in the art of programming).

Cramer aligns the "software as cultural construct" school with artist-programmers such as Matthew Fuller and the group I/O/D or Graham Harwood and the group Mongrel. I/O/D single-handedly "established" alternative browsers as art form with their *Web Stalker*[11] [Fig. 1], an application that allows users to draw "frames" in a blank window and select information they would like to display in them – for example, a graphical map of the site that presents all its individual pages and the links between them; the text from a URL

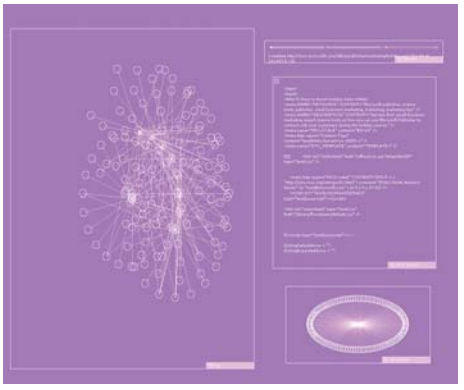

Fig.1: I/O/D: *Web Stalker*

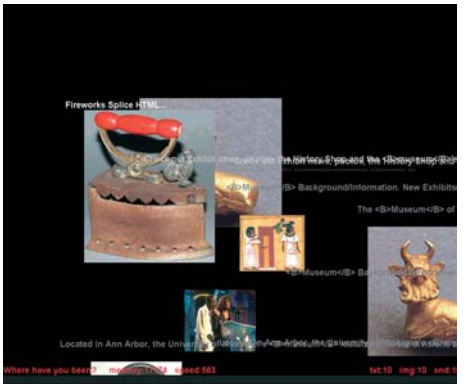

Fig. 2: Maciej Wisniewski: *netomat ™*

and the source code of the HTML page; a "stash" of URLs users would like to save. Although the *WebStalker* didn't display graphics, it expanded the functionality of existing browsers in a way that questions the paradigms of the conventional information display and Internet "architecture." While different in its approach, Maciej Wisniewski's *netomat ™* [12] [Fig. 2], which abandons the page format of traditional browsers and treats the Internet as one large database of files, would fall in the same category. Using an audiovisual language designed specifically to explore the unexplored Internet, *netomat ™* reveals how the ever-expanding network interprets and reinterprets cultural concepts and themes and takes visitors for a ride into the Internet's "subconscious."

The approach of the formalism camp is, according to Cramer, exemplified by the groups revolving around Adrian Ward and Alex McLean and participants in the mailing list "eugene." Adrian Ward's *Auto-Illustrator*[13] (winner of the Transmediale.01 software art prize) is a graphic design application that allows users to play with a variety of procedural techniques in the production of their own graphic designs while Alex McLean's *forkbomb.pl*[14] (winner of the Transmediale.02 software art prize) [Fig. 3] is a script written in Perl that creates an artistic impression of the user's computer system under pressure (by repeatedly creating new processes at such a speed that the system comes to a halt).

While the emphasis of the above mentioned projects (or their "creators'" intent) may lean towards one side of the formalism / culturalism spectrum, it would be problematic to miss the more subtle pointers to the other side of the scale in each of these works. The *Web Stalker* may engage notions of the browser as culturally coded construct, but one cannot neglect its distinct aesthetics and their art-historical references. In his essay *Visceral Facades: taking Matta-Clark's crowbar to software*[15], I/O/D's Matthew Fuller establishes a connection between the *WebStalker's* approach to information architecture and American artist Gordon Matta-Clark's technique of literally "splitting" the existing architecture of buildings, an application of formal procedures that would result in a revelation of structural properties. Matta-Clark's as well as the *WebStalker's* "deconstructionism" and "anarchitecture" are as much statements against certain social conditions as they are aesthetic acts oscillating between reconstructions of the destroyed and destructions of closure. Ward's *Auto-Illustrator* may be an application that explores the beauty and elegance of graphic design but at the same time makes a statement about the conventions and standardization of commercial graphic design applications. Taking a closer look at an artist's body of work, it is often hard to align them with one camp or the other. Mark Napier's *FEED*[16] [Fig. 4], which deconstructs webpages into a stream of pixels that are graphed and plotted in nine different displays, can almost be seen as an automization of

aesthetic "strategies" from abstract expressionism to minimalism. Napier's *Riot*[17], on the other hand, is an alternative browser that mixes text, images, and links from the three recent URLs that *Riot* users worldwide have accessed into one browser window and collapses territorial conventions like domains, sites, and pages. Illustrating how the Net resists traditional notions of territory, ownership, and authority, it questions the politics of encoding information. If software in general is not neutral but culturally encoded, there always is an interplay between formal and cultural aspects, which obviously varies depending on the emphasis of a specific project.

Aesthetics of Perception / Poetics of Construction

The categorization of artist-written software seems to undergo shifts depending on the manifestation the artwork takes. As mentioned above, any work of digital art incorporates a layer of code. It is noteworthy that digital art installations—even if they are ultimately driven by artist-written software—are seldom considered software art. Although the movements and reactions of robotic devices and objects (or the responses produced by sensors) may be driven or processed by artist-written software, little attention is commonly paid to the conceptual aspects, cultural impact, or "elegance" of the software itself, which remains a hidden force that isn't foregrounded and often induces such complex interactions that its "writing process" simply isn't as accessible as that of a piece of code poetry. Understanding at least the basic nature and language of digital art and its foundation in code-driven or algorithmic processes is an important element in establishing its identity.

What is commonly accepted as "software art" today varies greatly in its focus and manifestation. Software art pieces can present themselves as anything ranging from visuals (driven by a largely hidden layer of artist-written code) or as the written code itself. Code poetry such as Graham Harwood's *London.pl* (by William Blake)[18] would be an example of the latter category. Programming as artistic practice and expression remains largely undervalued and underappreciated. As Florian Cramer puts it, the focus on the purely perceptual aesthetics of art "is a straight continuation of romanticist philosophy and its privileging of aisthesis (perception) over poeisis (construction), cheapened into a restrained concept of art as only that which is tactile, audible, and visible."[19]

What distinguishes software art from other artistic practices, is that, unlike any form of visual art, it requires the artist to write a purely verbal description of their work (which

```
my $strength = $ARGV[0] + 1;

while (not fork) {
  exit unless --$strength;
  print 0;
  twist: while (fork) {
    exit unless --$strength;
    print 1;
  }
}
goto 'twist' if --$strength;
```

Fig. 3: Alex McLean: *forkbomb.pl*

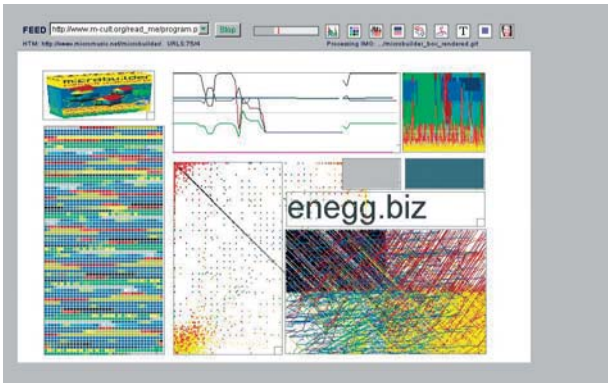

Fig. 4: Mark Napier: *FEED*

then often remains hidden behind the actions resulting from it). In most traditional art forms, the "signature" and "voice" of an artist manifests itself in aesthetics of visuals and execution. The aesthetics of artists who write their own source reveal themselves both in the poetics of the code and its visual results as actions derived from it. Artist John F. Simon, Jr. has repeatedly talked about code as a form of creative writing, where anything from the choice of story to the language of narration and the "story line" embody the artist's voice. Although Cramer sees Ward, McLean and Cox as largely privileging execution, they also emphasize that a separation of code from its resultant actions would result in a limitation of the aesthetic experience[20] and that both ends need to be considered. The study and criticism of software art has to be equally literate in the aesthetics of the back end's construction and the front end's (multi-sensory) perception. The crucial dilemma of software art may very well be that the study of its "backend" will always remain a fringe culture that won't be integrated into the mainstream of (perception-oriented) art criticism. The interconnectedness of written code and the actions it produces also begs the question how transparent the relationship between these two forms can or should be, and whether this might be a criterion for evaluating the art. Meaning, is software art more successful if one can "see" the algorithms at work in the unfolding of visuals / sound and can establish direct connections between the front end and the code driving it? Art that allows this connection to be made will certainly be accessible to a wider audience but it remains questionable whether the transparency of cause-and-effect relationships is a criterion for the quality of art. The issue here seems to be one of reference and is embedded in a larger discussion surrounding the status of representation in digital art. If one defines representation as a "likeness" or image of an external referent (an "object" or "scene" in the broadest sense), digital art in general ultimately represents data—be it an external data set or the data source of its own construction. While this applies to digital art in general, software art is more concerned with the generative construction process of data representation. One could draw the conclusion that we are facing a major transformation of the status of representation itself, which becomes a process that constitutes a convergence of language and mathematics, which in turn has the potential to drive a multi-sensory "display."

This transformation of representation also implies that software art would be more context-dependent in that its data source is always embedded in a specific context. In software art that is focused on data visualization—the creation of visual models for data sets—the issue of context becomes particularly important. For any given set of data, there are multiple possibilities for giving it a visual form, which in turn lend themselves to reconfiguration. This again leads to contextual shifts: the context provided for creating meaning of any given set of data is to a large extent determined by the dynamics of the interface. In this case, an external referent, a set of data, becomes part of its own representation. Context also becomes key when it comes to the success of the visualization process itself. While there needs to be a certain focus on and reflection of the data changes at any point, the visual model tends to turn into a form of wallpaper (no matter how beautiful it is) if it loses the larger context of the data sets.

What complicates matters further is that software art by no means exists in an art-historical and cultural vacuum and constitutes a clearly separable realm in and of itself. The label "software art" may be just one distinguishing characteristic of an artwork that is embedded in multiple contexts. John F. Simon, Jr's color panel series (consisting of custom hardware and software), for example, has strong art-historical references. *Color Panel v 1.0*[21] is a time-based study of color in which Simon's software explores possibilities and "rules" for color as proposed by Bauhaus artists such as Klee and Kandinsky—an investigation of color theory in its relation to motion and time. It would be problematic

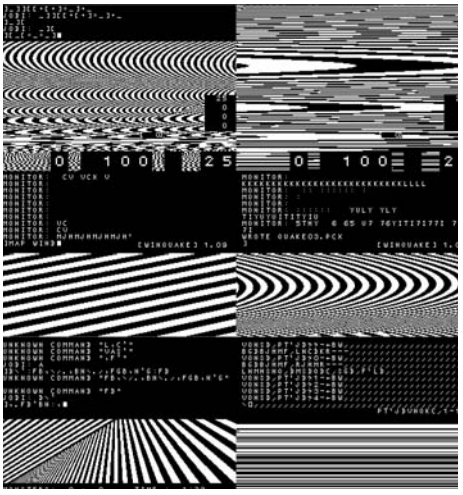

Fig. 5: Jodi: *Untitled Game*

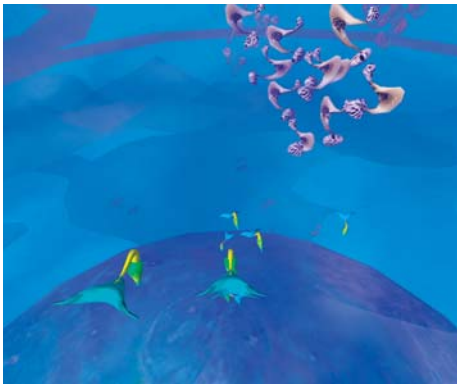

Fig. 6: John Klima: *Ecosystem*

to understand the project as a continuation of Bauhaus tradition without paying attention to the consecutive actions of the code that transcend a single moment in time and space and lead to continuous contextual changes.

There is a whole body of software art that references games and gaming culture—either by writing original games or rewriting and re-engineering existing ones—and has to be understood as an interplay of its contextual framework (games), the code, and its actions. Jodi's "deconstructions" of the original Wolfenstein in their *SOD*[22] or of Quake I in their *Untitled Game*[23] [Fig. 5] cannot be "read" without being literate in the essential characteristics of video games (aesthetics, architecture, user vs. system control etc.). *SOD* replaces the representational elements of the original game with black, white, and grey geometrical forms and creates a new architecture that challenges both orientation and navigation. *Untitled Game* strips Quake of its original architecture, re-engineers its structure and interactivity, and uses the original game engine as a tool for the creation of abstract art. Cory Arcangel created several works (among them *I Shot Andy Warhol* and *Landscape Study #4* [24]) that are based on reverse-engineered cartridges of the Nintendo game *Super Mario Brothers;* the original chips are melted off a Super Mario cartridge and then replaced with his self-manufactured chips. One could assume that the appreciation of these pieces at least to some extent relies on an awareness of the "retro" aesthetics of their original context. Landscape Study #4 fuses traditional landscape photography with gaming aesthetics, creating a scenery that effectively transcends the media it borrows from and seems to evolve into a new manifestation of a pop art genre. The software works of John Klima—among them *glasbead, Go, Fish,* and *ecosystm*[25] [Fig. 6], which represents global currency data in a 3D environmental simulation, where the population and behavior of flocks of insect-like "birds" (representing countries' currencies), are determined, respectively, by the currency's value against the dollar and its daily/yearly volatility—are all deeply influenced by gaming paradigms or aesthetics. While all the artists above seem to work in the field of software art and gaming, their approach to programming and aesthetics is distinctly different and each of these artists' body of work requires a different contextual frame.

The interest in developing criteria for the study and criticism of software art has been growing, but the question remains what impact this art will have both in the field contem-

porary art and culture at large. One could hope that a growing awareness of the art-historical lineage between conceptual art and software art would lead to more acceptance of media art in the larger field of contemporary art. As Matthew Fuller has pointed out, contemporary art has already been engaging with networks and computation by exploring some of their characteristics, such as a "relational aesthetic," but mostly without actually addressing specific digital technologies.[26] The question is not only what software art could do for the art world and its institutions but what these institutions could do for software artists. The nurturing of software and programming literacy is also essential when it comes to expanding the role of software in the broader context of cultural production. Initiatives such as *Processing,* an open project by Ben Fry and Casey Reas[27] (cf. p. 206) that creates an environment for learning the fundamentals of computer programming and is meant as an electronic sketchbook for developing ideas, are a step in that direction. At this point in time, there still needs to be a much broader appreciation of software as art and cultural expression in order to reach a level where software is more than an off-the-shelf product that is judged mostly by its efficiency.
}

1 Adrian Ward, Alex McLean, and Geoff Cox, "The Aesthetics of Generative Code," *http://generative.net/papers/aesthetics/*
2 *www.generative.net/*
"Generative art is a term given to work which stems from concentrating on the processes involved in producing an artwork, usually (although not strictly) automated by the use of a machine or computer, or by using mathematic or pragmatic instructions to define the rules by which such artworks are executed." (Adrian Ward)
"Generative art refers to any art practice where the artist creates a process, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is then set into motion with some degree of autonomy contributing to or resulting in a completed work of art." (Philip Galanter)
3 *www.m-cult.org/read_me/*
4 *www.transmediale.de*
5 *www.runme.org;* developed by Amy Alexander, Florian Cramer, Matthew Fuller, Olga Goriunova, Thomax Kaulmann, Alex McLean, Pit Schultz, Alexei Shulgin, and The Yes Men
6 *www.runme.org/about.tt2*
7 "QuickView on Software Art," *runme.org/project/+quickview*
8 Florian Cramer, "Concepts, Notations, Software, Art" (2002), *http://userpage.fu-berlin.de/~cantsin/homepage/writings/software_art/concept_notations//concepts_notations_software_art.html*
9 Donald E. Knuth, "Literate Programming," *CSLI Lecture Notes.* Number 27. Center for the Study of Language and Information. Stanford, CA, 1992
10 *www.ioccc.org/*
11 I/O/D, *WebStalker, www.backspace.org/iod/*
12 Maciej Wisniewski, *etomat ™, www.netomat.net*
13 Adrian Ward, *Auto-Illustrator, www.auto-illustrator.com*
14 Alex McLean, *forkbomb.pl, www.slab.org*
15 Matthew Fuller, "Visceral Facades: taking Matta-Clark's crowbar to software," *www.backspace.org/iod/Visceral.html*
16 Mark Napier, *FEED, www.potatoland.org/feed/*
17 Mark Napier, *Riot, www.potatoland.org/riot/*
18 Graham Harwood, *London.pl* by William Blake, *www.runme.org/project/+londonpl/*
19 Ibid. [8]
20 Ibid. [1]
21 John F. Simon, Jr., *Color Panel v 1., www.numeral.com/panels/colorpanelv1.0.html*
22 Jodi, *SOD, http://sod.jodi.org/*
23 Jodi, *Untitled Game, www.untitled-game.org*
24 Cory Arcangel, *I Shot Andy Warhol,* Landscape Studies, *http://beigerecords.com/cory*
25 *www.cityarts.com*
26 Ibid. [7]
27 *http://proce55ing.net/*