SOFTWARE ART

FLORIAN CRAMER AND ULRIKE GABRIEL

What is *software art*? How can "software" be generally defined? We had to answer these questions at least provisionally when we were asked to be with the artist-programmer John Simon jr. in the jury of the "artistic software" award for the *transmediale.01* art festival in Berlin, Germany.

Since more than a decade, festivals, awards, exhibitions and publications exist for various forms of computer art: computer music, computer graphics, electronic literature, Net Art and computer-controlled interactive installations, to name only a few, each of them with its own institutions and discourse. Classifications like the above show that attention is usually being paid to how, i.e. in which medium, digital artworks present themselves to the audience, externally. They also show that digital art is traditionally considered to be a part of "[new] media art," a term which covers analog and digital media alike and is historically rooted in video art. But isn't it a false assumption that digital art – i.e. art that consists of zeros and ones – was derived from video art, only because computer data is conventionally visualized on screens?

By calling digital art "[new] media art," public perception has focused the zeros and ones as formatted into particular visual, acoustic and tactile media, rather than structures of programming. This view is reinforced by the fact that the algorithms employed to generate and manipulate computer music, computer graphics, digital text are frequently if not in most cases invisible, unknown to the audience and the artist alike. While the history of computer art still is short, it is rich with works whose programming resides in black boxes or is considered to be just a preparatory behind-the-scenes process for a finished (and finite) work on CD, in a book, in the Internet or in a "realtime interactive" environment. The distribution of John Cage's algorithmically generated sound play "Roarotorio," for example, includes a book, a CD and excerpts of the score, but not even a fragment of the computer program which was employed to compute the score.

While software, i.e. algorithmic programming code, is inevitably at work in all art that is digitally produced and reproduced, it has a long history of being overlooked as artistic material and as a factor in the concept and aesthetics of a work. This history runs parallel to the evolution of computing from systems that could only be used by programmers to systems like the Macintosh and Windows which, by their graphical user interface, camouflaged the mere fact that they are running on program code, in their operation as well as in their aesthetics. Despite this history, we were surprised that the 2001 *transmediale* award for software art was not only the first of its kind at this particular art festival, but as it seems the first of its kind at all.

When the London-based digital arts project I/O/D released an experimental World Wide Web browser, the *Web Stalker* `http://www.backspace.org/iod/`, in 1997, the work was perceived to be a piece of Net Art. Instead of rendering Web sites as smoothly

Date: August 15, 2001.

formatted pages, the *Web Stalker* displayed their internal control codes and visualized their link structure. By making the Web unreadable in conventional terms, the program made it readable in its underlying code. It made its users aware that digital signs are structural hybrids of internal code and an external display that arbitrarily depends on algorithmic formatting. What's more, these displays are generated by other code: The code of the *Web Stalker* may dismantle the code of the Web, but does so by formatting it into just another display, a display which just pretends to "be" the code itself. The *Web Stalker* can be read as a piece of Net Art which critically examines its medium. But it's also a reflection of how reality is shaped by software, by the way code processes code. If complex systems and their generative processors themselves become language, formulation becomes the creation of a frame within which the system will behave, and of the control of this behaviour. The joint operation of these processes creates its own aesthetics which manifests itself no longer by application-restricted assignments, but in the free composition of this system as a whole. (Which simply is what developing software is all about.)

Since software is machine control code, it follows that digital media are, literally, written. Electronic literature therefore is not simply text, or hybrids of text and other media, circulating in computer networks. If "literature" can be defined as something that is made up by letters, the program code, software protocols and file formats of computer networks constitute a literature whose underlying alphabet is zeros and ones. By running code on itself, this code gets constantly transformed into higher-level, human-readable alphabets of alphanumeric letters, graphic pixels and other signifiers. These signifiers flow forth and back from one aggregation and format to another. Computer programs are written in a highly elaborate syntax of multiple, mutually interdependent layers of code. This writing does not only rely on computer systems as transport media, but actively manipulates them when it is machine instructions. The difference is obvious when comparing a conventional E-Mail message with an E-Mail virus: Although both are short pieces of text whose alphabets are the same, the virus contains machine control syntax, code that interferes with the (coded) system it gets sent to.

Software art means a shift of the artist's view from displays to the creation of systems and processes themselves; this is not covered by the concept of "media." "Multimedia", as an umbrella term for formatting and displaying data, doesn't imply by definition that the data is digital and that the formatting is algorithmic. Nevertheless, the "Web Stalker" shows that multimedia and terms like Net Art on the one hand and software art on the other are by no means exclusive categories. They could be seen as different perspectives, the one focussing distribution and display, the other one the systemics.

But is generative code exclusive to computer programming? The question has been answered by mathematics proper and the many historical employments of algorithmic structures in the arts. A comparatively recent classical example is the *Composition 1961 No. I, January I* by the contemporary composer and former Fluxus artist La Monte Young, which is at once considered to be one of the first pieces of minimal music and one of the first Fluxus performance scores:

> "Draw a straight line and follow it."[1]

This piece can be called a seminal piece of software art because its instruction is formal. At the same time, it is extremist in its aesthetic consequence, in the implication of infinite space and time to be traversed. Unlike in most notational music and written theatre plays,

[1] facsimile reprint included in [hun90], no page numbering

its score is not aesthetically detached from its performance. The line to be drawn could be even considered a second-layer instruction for the act of following it. But as it is practically impossible to perform the score physically, it becomes meta-physical, conceptual, epistemological. As such the piece could serve as a paradigm for Henry Flynt's 1961 definition of Concept Art as "art of which the material is 'concepts,' as the material of for ex. music is sound."[2] Tracing concept art to artistic formalisms like twelve-tone music, Flynt argues that the structure or concept of those artworks is, taken for itself, aesthetically more interesting than the product of their physical execution. In analogy, we would like to define software art as art of which the material is software.

Flynt's Concept Art integrates mathematics as well, on the acognitive grounds of "de-emphasiz[ing]" its attribution to scientific discovery.[3] With this claim, Flynt coincides, if oddly, with the most influential contemporary computer scientist, Donald E. Knuth. Knuth considers the applied mathematics of programming an art; his famous compendium of algorithms is duely titled "The Art of Computer Programming."[4]

Should the transmediale software art jury therefore have consisted of mathematicians and computer scientists who would have judged the entries by the beauty of their code?

What is known as Concept Art today is less rigorous in its immaterialism than the art Flynt had in mind. It is noteworthy, however, that the first major exhibition of this kind of conceptual art was named "Software" and confronted art objects actually with computer software installations.[5]. Curated in 1970 by the art critic and systems theorist Jack Burnham at the New York Jewish Museum, the show was, as Edward A. Shanken suggests, "predicated on the idea of software as a *metaphor* for art [my emphasis],"[6]. It therefore stressed the cybernetical, social dimension of programmed systems rather than, as Flynt, pure structure.

Thirty years later, after personal computing became ubiquituous, cultural stereotypes of what software is have solidified. Although the expectation that software is, unlike other writing, not an aesthetic, but a "functional tool" itself is an aesthetic expectation, software art nevertheless has become less likely to emerge as conceptualist clean-room constructs than reacting to these stereotypes. The "Web Stalker" again might be referred to as such a piece. In a similar fashion, the two works picked for the transmediale award, Adrian Ward's "Signwave Auto-Illustrator" and Netochka Nezvanova's "Nebula M.81," are PC user software which acts up against its conventional codification, either by mapping internal functions against their corresponding signifiers on the user interface (Auto-Illustrator) or by mapping the signifiers of program output against human readability (Nebula M.81).

The range of works entered for the transmediale.01 software art award shows that coding is a highly personal activity. Code can be diaries, poetic, obscure, ironic or disruptive, defunct or impossible, it can simulate and disguise, it has rhetoric and style, it can be an attitude. Such attributes might seem to contradict the fact that artistic control over generative iterations of machine code is limited, whether or not the code was self-written. But unlike

---

[2] Henry Flynt, *Concept Art* [Fly61] "Since 'concepts' are closely bound up with language," Flynt writes, "concept art is a kind of art of which the material is language."

[3] ibid.

[4] [Knu98]

[5] Among them Ted Nelson's hypertext system in its first public display, according to Edward A. Shanken, *The House that Jack Built: Jack Burnham's Concept of "Software" as a Metaphor for Art*, [Sha]

[6] ibid.

the Cagean artists of the 1960s, 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.[7]

REFERENCES

[Fly61] Henry Flynt. Concept art. In La Monte Young and Jackson MacLow, editors, *An Anthology*. Young and MacLow, New York, 1963 (1961).

[hun90] George Maciunas und Fluxus-Editionen, 1990.

[Knu98] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, 1973-1998.

[Sha] Edward A. Shanken. The house that jack built: Jack burnham's concept of 'software' as a metaphor of art. *Leonardo Electronic Almanach*, 6(10). `http://www.duke.edu/~giftwrap/House.html`.

[7] Or, as Adrian Ward puts it: "I would rather suggest we should be thinking about embedding our own creative subjectivity into automated systems, rather than naively trying to get a robot to have its 'own' creative agenda. A lot of us do this day in, day out. We call it programming." (quoted from an E-Mail message to the "Rhizome" mailing list, May 7, 2001)