MISE EN ABYME IN SOFTWARE ART: A COMMENT TO FLORIAN CRAMER

TROELS DEGN JOHANSSON

Draw a straight line and follow it.[1]

1. The "No Future" of Software Art

The concept of *mise en abyme* seems particularly pertinent if we are to capture a dominant wing in software art—and in some respect software art in general; pertinent that is not only in terms of analyzing the formal level of singular art works but also when considering their underlying artistic strategies.

In comparative literature, the use of the literary trope of *mise en abyme* has tended towards the Symbolist tradition of experimental fiction, where the limits of language are tested in an extreme self-reflexivity closed off from the reference function of language. In the study of narrative and visual representation in general, the concept has thus become emblematic for the instability or fragility of representation, for example by thematizing the perishable or transitory character of the material of expression (e.g. the paper of a book, writing in the sand, etc.), an erratic structure of enunciation (e.g. an insane narrator, as in Poe, or a narrator who turns the levels of fiction and reality reference upside down, as in Borges), or that the epic depiction of a human being which eventually turns out to be staged before the audience as if in a game (e.g. Peter Weirs film "The Truman Show, USA, 1998.)

The use of the literary trope of *mise en abyme* thus characterizes art's exploration of representational problems of one's means of expression or of language in general; in software art the exploration of representational problems in computer media. What seems important to at least one wing in software art is to thematize that the *computer is a tool and not a medium*; that the idea of the computer

medium being a transparent vehicle for the expression of other, culturally established media ("the book" as in word processing software or text displaying software, "cinema" as in media players, etc.) is based on a representational "repression" of the level of software, program code, which of course is necessary for the computer medium to function at all. Accordingly, software art has concentrated on the software itself as a tool and not primarily as a means of expression of something else. Hence the fundamental self-reflexivity in software art; a self-reflexivity which echoes the heritage of deconstructivism and the criticism of ideology in philosophy (Derrida[2], Althusser) and media theory (Baudry, Ulmer, etc.), in which the medium is similarly scrutinized as for the technological foundation of presence, transparency, and "reality effects" (Barthes), and where the figure of the mise en abyme, in Greg Ulmer's terms, is referred to as a 'reflexive structuration, by means of which a text shows what it is telling, does what it says, displays its own making, reflects its own action.' (1991) In other words, a rhetorical figure which characterizes a critical discourse where the criticism is manifest as a theme as well as an immanent principle of the rhetorical strategy.

In his contribution to the Read_Me Festival 2002, Florian Cramer made a plausible distinction between two wings in software art, namely "software formalism" and "software culturalism". Whereas software formalism consists in exploring the formal beauty of software as an artistic material in its own rights, "software culturalism" approaches software as a "cultural, politically coded construct"; an otherwise transparent construct that the artist should make visible, manipulate/tweak, destroy, and/or apply for other purposes than originally intended in order to make manifest its importance to the medium and to culture. To Cramer, the two wings are represented by two "semi-coherent London-based groups" respectively; the culturalists around Matthew Fuller, Graham Harwood, I/O/D, and Mongrel ("Web-Stalker"), and the formalists around artists such as Adrian Ward ("Auto-Illustrator"), Alex Mclean

("forkbomb.pl"), the mailing-list based community "eu-gene", and the DorkBot "salon" but also predecessors such as Donald E. Knuth with his "The Art of Computer Programming" and Stephen Levy with his so-called hacker-credo that "You can create art and beauty with computers".

The self-reflexivity of software art referred to above may be said to characterize both wings in as much as they are both preoccupied primarily with the program code itself rather than with the effects that it may bring about in a graphic user interface when executed on a computer. However, as I shall seek to demonstrate below, the figure of the mise en abyme seems especially characteristic of the software culturalism since this wing concentrates on the emergence of code in otherwise transparent graphic user-interfaces meant for entertainment, the solving of tasks, etc. In this manner, one may say that software culturalism has a general problem with representation in the sense that it seeks to make manifest the underlying code that a rendered graphic user interface is based on. Still, as I will also argue, one may find a similar pattern in the formalist camp; a finding that may indicate that the two wings are not absolutely torn on this matter.

However, as a headline for the strategical dimension of software art, the application of the concept of "mise en abyme" may also give an indication of software art's current state of affairs; that software art in a certain sense has come to a dead end, a cul de sac, an abyss. Having made his distinction between the two wings of software art, Cramer also ended up by pointing at what we may designate as *software art's general aporia*: That none of the two wings seem particularly promising as separate projects, that is without reference to the other, and that even together it is not obvious whether there is any hope for further progress… Contrary to a general trend in contemporary art, in which art is dedicated to the exploration of opportunities and strategies, Cramer rather seems to announce a "no future" of software art; except that is for the histories of software art which 'still remain to be written.' (p.7)

For Cramer, the software formalism is important since it sets off from the recognition that 'software art could be generally defined as an art of which the material is formal instruction code.' However, in the same moment, Cramer admits that if software art were simply to be reduced to this project, "one would risk ending up with a neo-classicist understanding of software art as beautiful and elegant code". For Cramer, it is thus 'telling that hackers, otherwise an avant-garde of a broad cultural understanding of digital technology, rehash a late-18TH century classicist notion of art as beauty, rewriting it into a concept of digital art as inner beauty and elegance of code.' (p. 6) Software formalism may thus be said to be "beautiful", but "not interesting" when seen as a wing of contemporary art, "not interesting" that is in the Kierkegaardean sense since it does not give rise to critical questions, reflections, and enlightenment ouside of its own field in the fashion that modern art is supposed to do, incl. the major formalist and minimalist traditions. To Cramer, software formalism represents an "aesthetic conservatism" which is 'widespread in engineering and hard-science cultures; fractal graphics are just one example of Neo-Pythagorean kitch they promote.'[3]

On the other hand, reduced to "software culturalism" software art 'could end up being a critical footnote to Microsoft desktop computing, potentially overlooking its speculative potential at formal experimentation.' Software culturalism may thus be said to be "interesting" in the sense evoked just above and hence in a more profound sense avant-gardistic but as such lacking what one may call a "lasting perspective", in that its main difficulty is to take things further than to comment on the established formats in computer mediated communication. Software culturalism would have taken things further if it sought to made out an alternative to these formats, as one would expect from a true avant-garde project—if we by this term understand a project whose purpose it is "to make Art serve Life" in the "Name of Art" but in the same moment by annihilating art as a separate institution.[4]

2. The "Nature" of Software Art: Abysmal Concepts

In his essay, Florian Cramer presents a convincing argument for the nature and necessity of software art. Software art, Cramer argues, is dedicated to the exploration of the algorithmic program codes, not the code's transparent expression in a computer-mediated presentation; say a graphic user interface, a media player, etc.

Cramer's paper may be said to be an essentialist defence of software art in the sense that it is its ambition to demonstrate that software art exists in terms of being software. On the other hand, however, his essentialist approach also tends to undermine his defence of software art as an artistic project since implicitly he points at something which both may be characterized as its ultimate completion and its finalization, its end.

As stated previously, software art to Cramer is art that brings software itself to the fore rather than the effect that it may produce. As Cramer notes initially, this project is much overlooked in art and digital aesthetics, and Cramer is probably right when he states that the

> *history of the digital and computer-aided arts could be told as a history of ignorance against programming and programmers. Computer programs get locked into black boxes, and programmers are frequently considered to be mere factota, coding slaves who execute other artist's concepts.*

Cramer's thesis is echoed in what Mary Flanagan has heralded as the "bible of electronic art", namely Margot Lovejoy's *Digital Currents: Art in the Electronic Age*; a work which is only marginally interested in experiments on the level of program code and which rarely credits the programmers who have been involved in the development of the generous number of art works that Lovejoy includes in her presentation. We may thus take Lovejoy as an example of what Cramer refers to as the "romanticist philosophy" that is underlying much treatment of electronic art; a treatment which may be characterised by its 'privileging of aesthesis (perception) over poiesis (construction), cheapened into a restrained concept of art as only that what is tactile, audible and visible."[5]

To Cramer,

> *software art exists and operates on an immaterial, entirely conceptual level, that is, what he sees as the level of software as such. Cramer defends this position firstly by arguing that software may be read and executed entirely as a mental act ('as it was common before computers were invented'…), and that software thus exists without a computer. Doing so, he successfully counterargues Friedrich Kittler's claim that software does not exist without the hardware that it runs on, that is, that software and hardware are inconceivable without each other.6*

Secondly, Cramer points at what he takes for software art's essential affiliation to minimalist concept art. To Cramer,

Concept art as an art "of which the material is 'concepts,' as the material of for ex. Music is sound" (Henry Flynt's definition from 1961)[7] and software art as an art whose material is formal instruction code seem to have at least two things in common:

1. the collapsing of concept notation and execution into one piece;
2. the use of language; instructions in software art, concepts in concept art. Flynt observes: "Since 'concepts' are closely bound up with language, concept art is a kind of art of which the material is language"

The collapse of the differentiation between concept notation and execution is an important theme in the tradition of concept art that Cramer refers to in his 2002 Read_Me paper. His "favourite example", i.e. La Monte Young's piece "Composition 1961", which is made out by a piece of paper with the instruction, 'Draw a straight line and follow it', demonstrates this collapse in a beautiful fashion by instructing the reader to initiate the execution of an open-ended program and thus emphasizing ironically the impossility of performing the instruction physically while one may actually imagine oneself doing it.

Cramer's example not only clarifies what he takes for the "collapse of concept notation and execution"; it also captures the abysmal character of software art in his conception. La Monte Young's piece is a fine example of mise en abyme. Whereas it is true that in it self the text may not "show what it is telling, or do what it says", etc., as Ulmer has it. However, it does assume this character by its

"computation", that is when one initiate the execution of its instruction—and obviously when one realizes its cool irony. The imagined, eternal execution of the open-ended instruction program thus perfectly simulates the visual realization of the mise en abyme, say by the feed-back loops of a monitor that present the filmed live image of itself, or by watching the halo of oneself's mirror reflexions of one's mirror image when placed between two large parallel mirrors. The particular use of language becoming a common denominator of concept and software art precisely makes out experiments with the "limits of language" and closes off the ordinary reference function of language in favour of self-reflexivity.

La Monte Young's piece becomes emblematic for what we may designate as Cramer's abysmal concept of software art. Cramer not only sees software art as being subordinated concept art: 'Since formal instructions are a subset of conceptual notations, software art is, formally, a subset of conceptual art'; he also happens to point at a distinct piece of concept art in order to demonstrate the ideal realization and thus ultimately the finalization of software art as an artistic program. In other words, by instructing the software artist to execute mentally La Monte Young instruction program he makes the clearest of demonstrations of what software art is, but he also seems to exhaust the possibilities of software art as an artistic program. Cramer himself concludes that

Concept art potentially means terror of the concept, software art terror of the algorithm; a terror grounded in the simultaneity of minimalist concept notation and totalitarian execution, helped by the fact that software collapses the concept notation and execution in the single medium of instruction code.—Sade's "120 days of Sodom" could be read as a recursive programming of excess and ts simultaneous reflection in the medium of prose. The popularity of spamming and denial-of-service code in the contemporary digital arts is another proof of the perverse double-bind between between software minimalism and self-inflation; the software art pieces awarded at the transmediale.02 festival, "trancenoizer" and "forkbomb.pl" also belong to this category.

Obviously potentiality here does not mean risk but but little more than realizing the ultimate consequences of the argumentation and indeed the nature of the material that software art is devoted to. Whereas "120 Days of Sodom" may be could be read as a recursive programming of excess, software art could inversely be read as a programmed excessive reiteration of transgression; that is as a charicature of "120 Days …" After Cramer, software art seems to have little to do but to repeat the point; to follow the straight line that they have drawn

.

[1] La Monte Young, "Composition 1961", quoted from Cramer 2002.

[2] Derrida who has frequently made use of and referred to the rhetorical figure of the mise en abyme in his own philosophical writing.

[3] I think that Cramer is too general in his characterization of the formalist wing at this point: The artists of the "software formalism" camp captures in my view a much broader range of approaches to art than the ones described.

[4] Hence Peter Bürger's definition of the historical, never fully realized avant-garde. (Bürger 1974: 73 ff.)

[5] Hence Lovejoy's understanding of "electronic image-production", which to her is 'immaterial, existing only as an image structure or accumulation of data, without physical substance. It does not lead necessarily to the production of a material art object unless the artist makes a conscious decision to translate it into one that can maintain a physical presence with a particular dimensional level within a perceptual field.' (Lovejoy: 159)

[6] Cramer refers here to Kittler (1991).

[7] Cramer refers here to Flynt (1963).

Literature:

1. Baudry, Jean-Louis. 1986 (1970). "Ideological Effects of the Basic Cinematographic Apparatus." In Philip Rosen (Ed.) *Narrative, Apparatus, Ideology*. New York: Columbia University Press.
2. Bürger, Peter. 1974. *Theorie der Avantgarde.* Berlin: Suhrkamp XXX
3. Cramer, Florian. 2002. 'Concepts, Notations, Software, Art.' Paper for the Read_Me Festival, 2002.
4. Flynt, Henry. 1963 (1961). 'Concept Art.' In La Monte Young and Jackson MacLow, Eds.: An Anthology: Young and MacLow.
5. Kittler, Friedrich. 1991. 'There Is No Software.' Observed at http://textz.gnutenberg.net/textz/kittler_friedrich_there_is_no_software.txt on May 25th, 2004.
6. Lovejoy, Margot. 2004 (1989). Digital Currents: Art in the Elelctronic Age. London and New York: Routledge/Taylor & Francis.
7. Ulmer, Greg. 1991. 'Grammatology Hypermedia.' *Postmodern Culture,* Vol. 1. No. 2. Observed on http://jefferson.village.virginia.edu/pmc/text-only/issue.191/ulmer.191