$Id: software_decontextualizaton.tex,v 1.3 2002/07/10 16:21:00 paragram Exp $

CONTEXTUALIZING SOFTWARE ART

FLORIAN CRAMER

CONTEXTUALIZATIONS OF ART AND SOFTWARE

Who does (de)contextualize which? I think the title of this panel, "software decontextualization", is very good because it tells a lot about software art and also about the problems of creating such a term or category. Of course, "software art" has much to do with contextualization: contextualizating (or maybe decontextualizing) art as software and computer software as art. This contextualization can be observed on two separate levels: In the work of artists themselves who use software as a material or reference, or in the work of juries and curators who decide whether a piece of software is interesting as art or not. In some cases, artists and curators can be one and the same person; for the Moscow festival "read_me", the well-known Net.artist and jury member Alexei Shulgin secretly entered an anonymous piece of cracker software which visually disturbed the Microsoft Windows desktop interface into the software art competition. In the end, that entry actually received an honorary mention of the jury, thus putting up interesting questions of what is included by a category like "software art" and what is excluded.

I would like to begin, and make these questions less abstract, with two examples of contemporary digital art, each of them dealing with software in a very individual way. The first example is Jodi's work "Surgery/havoc" `http://wwwwwwwww.jodi.org/100cc/havoc/`, a part of jodi's now-classical Net.art website, the second is "Screen Saver" by Eldar Karhalev and Ivan Khimin which we gave one of three equal prizes in Moscow.

Jodi, "Surgery/havoc". At first glance, Jodi's piece looks like a monochrome jump'n'run computer game similar to Nintendo's "Super Mario Brothers". There is jump'n'run action on the screen and the possibility to enter several zoom views or game levels with a few mouseclicks. It is a work which almost everyone would easily identify as a omputer (respectively: game console) program. The point however is that it's not. If we look at the sourcecode of the pages, we see that the piece chiefly consists

Date: July 13rd, 2002.

of simple animated graphics files (demonstration). The HTML sourcecode of the web page multiplies one minimal GIF animation file over the screen; the seeming "zoom function" is simply a link to another web page with enlarged images. If we define as computer software algorithmic programs written in a Turing-complete programming language like C, Java or Basic, then Jodi's "Surgery/havoc" is not software because it doesn't contain any algorithms, but only graphical mock-ups of algorithms. — I will explain better how an algorithm and programming works in a few minutes. — So its a piece that de- or miscontextualizes it as software. Since it is, as a trompe l'oeil simulation, with no doubt an aesthetic and cultural reflection of software. It thus is a very good example why software art, through its own contextualization, doesn't necessarily have to be real software.

Eldar Karhalev and Ivan Khimin, "Screen Saver". (Demonstration.) While Jodi's piece seems to be, but is not really software, Eldar Karhalev's and Ivan Khimin could be described just the other way around. It is simply a step-by-step instruction to configure the Windows screensaver in such a way that it turns into a monolithic display of a giant square that slowly slides from left to right and black, slightly modulating its color in the process. Of course, with this piece coming from Russia, one is immediately reminded of Kasimir Malevich's "black square", but it doesn't intuitively make sense to call this a piece of software, respectively software art. Like in the jodi example, no hardcore computer programming was used to create it.

Nevertheless, it could be seen as software in two ways: Firstly, it could be called post- or meta-software, since manipulates existing software instead of coding from scratch, managing to turn it upside down even without complex technical skills; it reprograms Windows without using program code. Secondly, its plain English instruction for misconfiguring the software is a formal instruction code itself, an algorithm. If we define software as algorithmic code, then this piece surely is software.

WHAT IS SOFTWARE ART, OR: WHAT COULD IT BE?

The question after looking into the software-ness of jodi's "Havoc" and Karhalev/Khimin's "Screen Saver" is of course: Why at all insistent on software, and why use the term "software art"? To date, critics and scholars in the arts and humanities have considered computers primarily as storage and display media, as something which transmits and reformats images, sound and typography. But reflection of the as such invisible layer of software is rare. Likewise, the whole 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. Given that software code *is* conceptual writing, this is not without its own irony. Much of the problem lies, I think, in the problem that digital art is being called [new] "media art" and thus lumped together with unrelated technologies such as analog video. If one defines as a medium something that it is between a sender and a receiver, then computers are not only media, but also senders and receivers which themselves are capable of writing and reading, interpreting and composing messages within the limitations of the software rule sets inscribed into them. The computer programs for example which calculate the credit line of checking accounts or control medical instruments in an emergency station can't be meaningfully called "media". I personally think that its worth siding with a term like "software art" even if only for the sake of getting rid of the meaningless "media" word in arts and criticism.

A Crash Course in Programming. If, as said before, a piece of software is a set of formal instructions, or, algorithms, then it could also be called a logical score put down in a code. It doesn't matter at all which particular sign system is used as long as it is a code, whether digital zeros and ones, the Latin alphabet, Morse code or, like in a processor chip, an exactly defined set of registers controlling discrete currents of electricity.

Thus defined, software instructions don't necessarily have to run on computers. In 1924, the Dadaist Tristan Tzara's wrote a generic instruction for writing Dada poems by shuffling the words of a newspaper article[1]:

> To make a Dadaist poem:
> Take a newspaper.
> Take a pair of scissors.
> Choose an article as long as you are planning to make your poem. Cut out the article.
> Then cut out each of the words that make up this article and put them in a bag.
> Shake it gently.
> Then take out the scraps one after the other in the order in which they left the bag.
> Copy conscientiously.
> The poem will be like you.
> And here you are a writer, infinitely original and endowed

[1][Tza75]

with a sensibility that is charming though beyond the understanding of the vulgar.

This poem is effectively an algorithm, a piece of software which may as well be written as a computer program.[2]. We could note it more formally like this:

(1) Count the number of pages of your newspaper.
(2) Choose a random number between 1 and the number of newspaper pages.
(3) Pick the page that bears the random number you chose.
(4) Count the number of articles on the page you picked.
(5) Choose a random number between 1 and the number of articles on the page.
(6) Pick the article whose position relative to the others corresponds to the random number you chose.
(7) Cut out all single words of the article you picked.
(8) Count the number of words you cut out.
(9) Take blank piece of paper.
(10) Repeat the following until no words are left:
    (a) Choose a random number between 1 and the number of words you counted
    (b) Pick the word whose position in the article corresponds to the random number you chose.
    (c) Write down the word you picked onto the piece of paper.
    (d) Reduce the number of words you counted by one.
(11) The poem is finished.

These English instructions could be transcribed almost line by line into a computer program. This is what they would look like in the programming language Perl (which I chose for the sake of simplicity):

This is, if not a complex, yet still a very typical program sourcecode, using loops and conditional statements. In case you didn't know about computer programming yet, you have just made a crash course in it and know all the basics. Our transcription of Tzara's poem from English to Perl was not a transcription of something into software, but a transcription of a non-machine software into machine software. The instructions only have to

[2]My own Perl CGI adaption is available under `http://userpage.fu-berlin.de/{~}cantsin/permutations/tzara/poeme{\protect\T1\textunderscore}dadaiste.cgi`

meet the requirement of being executable by a human being as well as by a machine. Of course the opposite is also true: A machine is not necessary to execute even the Perl program; we can execute the code as well in our minds. Theoretically, this is true for any computer program, and programming handbooks, whose example code is rarely ever run on machines, are a striking practical example.

So my argument is quite contrary to Friedrich Kittler's media theory according to which there is either no software at all or at least no software without the hardware it runs on:[3] If any algorithm can be executed mentally, as it was common before computers were invented, then of course software can exist and run without hardware.

What is interesting about programming? Contrary to conventional data like digitized images, sound and text documents, algorithmic instruction code allows a generative process. It uses computers for computation, not only as storage and transmission media. And this precisely distinguishes program code from non-algorithmic digital code, describing for example the difference between algorithmic composition on the one hand and audio CDs/mp3 files on the other, between algorithmically generated text and "hypertext" (a random access database model which as such doesn't require algorithmic computation at all), or between a graphical computer "demo" and a video tape. Although one can of course use computers without programming them, it is impossible not to use programs at all. The question only is who programs. There is, after all, no such thing as data without programs, and hence no digital arts without the software layers they either take for granted, or design or manipulate themselves. To discuss "software art" simply means to not take software for granted, but pay attention to how and by whom programs were written.

SOFTWARE ART

Executable Code in Art.

Concept Art and Software Art. The question of what software is and how it relates to non-electronic contemporary art is at least thirty-two years old. In 1970, the art critic and theorist Jack Burnham curated an exhibition called "Software" at the Jewish Museum of New York which today is believed to be first show of concept art. It featured installations of US-American concept artists next installations of computer software Burnham found interesting, such as the first prototype of Ted Nelson's hypertext system "Xanadu". Concept art as an art "of which the material is 'concepts,' as the material of

[3][Kit91]

for ex. music is sound" (Henry Flynt's definition from 1961[4]) 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".[5]

If concepts become, to quote Flynt again, artistic"material", then concept art differs from other art in that it actually exposes concepts, putting their notations up front as the artwork proper. In analogy, software art in particular differs from software-based art in general in that it exposes its instructions and codedness.

My favorite example of both concept art in Flynt's sense and non-computer software art is La Monte Young's "Composition 1961", a piece of paper containing the written instruction "Draw a straight line and follow it". The instruction is unambiguous enough to be executed by a machine. At the same time, a thorough execution is physically impossible. So the reality of piece is mental, conceptual.

The same duplicity of concept notation and executable code exists in Sol LeWitt's 1971 "Plan for a Concept Art Book", a series of book pages giving the reader exact instructions to draw lines on them or strike out specific letters.[6] A writing which writes itself, LeWitt's "Plan" could also be seen in a historical continuity of combinatory language speculations: From the permutational algorithms in the Sefer Jezirah and ecstatic Kabbalah to the medieval "ars" of Raimundus Lullus to 17th century permutational poetry and Mallarmé's "Livre". The combinatory most complex known permutation poem, Quirinus Kuhlmann's 1771 sonnet "Vom Wechsel menschlicher Sachen" consists of $13 * 12$ nouns can be arbitrarily shuffled so that they result in 10^{114} permutations of the text.[7] Kuhlmann's and La Monte Young's software arts meet in their aesthetic extremism; in an afterword, Kuhlmann claims that there are more permutations of his poem than grains of sand on the earth.[8] If such implications lurk in code, a formal analysis is not enough. Concept art potentially means terror of the concept, software art terror of the

[4][Fly61]
[5]ibid.
[6][Hon71], p. 132-140
[7][Kuh71]
[8]ibid.

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 its simultaneous reflection in the medium of prose.[9] The popularity of spamming and denial-of-service code in the contemporary digital arts is another practical proof of the perverse double-bind between software minimalism and self-inflation; the software art pieces awarded at the transmediale.02 festival, "tracenoizer" and "forkbomb.pl" also belong to this category.

If jodi's art, "Screen Saver", "tracenoizer" and "forkbomb.pl" have one thing in common, then it probably is that they take a hacker-approach to software, in the case of Jodi and Khimin/Karhalev even with the attempt to hack software itself as a concept. "forkbombs" like the one awarded at transmediale.02 have been well-known and written in countless implementations years before by Unix hackers. So is "Software Art" just an arty repackaging of hacking, perhaps just even a lowbrow version of what is written with much more technical skill in hacker and free software communities?

The answer is yes and no. It is indeed a question of contextualization, but one that contextualization is never "just" contextualization, but what it's all about.

Two years before Burnham's "Software" exhibition, the computer scientist Donald E. Knuth published the first volume of his famous textbook on computer programming, "The Art of Computer Programming".[10] Knuth's wording has adopted in what Steven Levy calls the hacker credo that "you can create art and beauty with computers".[11] Even if Levy's account is idealized, it is telling and — from my point of view — true 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 and rewrite it into a concept of digital art as inner beauty and elegance of code. Such aesthetic conservativism is widespread in engineering and hard-science cultures; fractal graphics are just one example of Neo-Pythagorean digital kitsch they promote.

[9] As Abraham M. Moles noticed already in 1971, [Mol71], p. 124

[10] knuth:art

[11] according Steven Levy [Lev84]; among those who explicitly subscribe to this is the German Chaos Computer Club with its annual "art and beauty workshop".

But as a contemporary art, the aesthetics of software art includes ugliness and monstrosity just as much as beauty, not to mention plain dysfunctionality, pretension and political incorrectness.[12] Above all, software art today no longer writes its programs out of nothing, but works within an abundance of available software code. This makes it distinct from works like Tzara's Dada poem which, all the while it addresses an abundance of mass media information, contaminates only the data, not its algorithm; the words become a collage, but the process remains a synthetic clean-room construct. One thus could say that contemporary software art operates in a postmodern condition in which it takes pre-existing software as material — reflecting, manipulating and recontextualizing it. This observation applies, by the way, to all the software art works I mentioned.

Free Software and Software Art. But exactly this observation shows that the relationship between software art and software non-art is not easily dismissed. The first subculture which understood software as something intertextual was the Free Software and Open Source movement, i.e. the movement which produced software like GNU and Linux, from which Steven Levy's notion of the "hacker" was largely derived. It is true that the notions of craft and art as beauty are generally conservative in Free Software, hacker and engineering cultures; nevertheless, especially GNU/Linux provides many examples of software which would be wildly successful as software art if it only were properly advertised or, if you prefer, contextualized.

A good reference is the website `http://www.sweetcode.org` which indexes and links to unusual, original and sometimes mind-challenging software, the vast majority of it under free/Open Source licenses. But I would like to give you a more classical example of a software that is contained in almost all typical GNU/Linux distributions: The software library "aalib" which transforms computer graphics real-time into typogrammatic text code. Since many GNU/Linux programs work with this library, you can view image files, watch TV, watch video files and DVD movies and even play "Doom" and "Quake" in ASCII; and since it is text code, this also works over simple Telnet or ssh terminal connections in the Internet. (Demonstration, bb)

This project is strikingly similar to the well-known Net.art "ASCII Art Ensemble" which produced, among others, an ASCII Art version of the porn movie "Deep Throat". In fact, the AAlib software is, in technical terms,

[12]which is why I think would be wrong to (a) restrict software art to only properly running code and (b) exclude, for political reasons, proprietary and other questionably licensed software from software art presentations.

vastly superior to anything the ASCII Art Ensemble has made. It was written by a group of very young Czech hackers; I have no doubt they would be "media art" stars today if they had contextualized their work in art channels by entering it to festivals and competitions.

Conclusion. Software Art abolishes barriers, but creates new ones; contextualization = disciplining (in the double sense of the word). Software programming and digital art: two highly developed net.cultures; conflict is potentially productive and welcome. Danger: Premature canonization, sparse input to competitions and festivals, quick hypes.

REFERENCES

[Fly61] Henry Flynt. Concept art. In La Monte Young and Jackson MacLow, editors, *An Anthology*. Young and MacLow, New York, 1963 (1961). 6

[Hon71] Klaus Honnef, editor. *Concept Art*. Phaidon, Köln, 1971). 6

[Kit91] Friedrich Kittler. There is no software, 1991. `http://textz.gnutenberg.net/textz/kittler_friedrich_there_is_no_software.txt`. 5

[Kuh71] Quirinus Kuhlmann. *Himmlische Libes=küsse*. ?, Jena, 1671. 6

[Lev84] Steven Levy. *Hackers*. Project Gutenberg, Champaign, IL, 1986 (1984). 7

[Mol71] Abraham A. Moles. *Kunst und Computer*. DuMont, Köln, 1973 (1971). 7

[Tza75] Tristan Tzara. Pour fair une poème dadaïste. In *Oeuvres complètes*. Gallimard, Paris, 1975. 3

C/O FREIE UNIVERSITÄT BERLIN, SEMINAR FÜR ALLGEMEINE UND VERGLEICHENDE LITERATURWISSENSCHAFT, HÜTTENWEG 9, D-14195 BERLIN, CANTSIN@ZEDAT.FU-BERLIN.DE, HTTP://USERPAGE.FU-BERLIN.DE/~CANTSIN