Digital Code and Literary Text

By Florian Cramer

No. 20 – 22.10.2001

Abstract

Digital code as text. Can notions of text that were developed without electronic texts in mind be applied to digital code, and how does literature come into play here?
My talk is based on the general (yet disputable) assumption that the theoretical debate of literature in digital networks has shifted, just as the poetic practices it is shaped after, from perceiving computer technology solely as an extension of conventional textuality (as manifest in such notions as 'hypertext', 'hyperfiction', 'hyper-/ multimedia') towards paying attention to the very codedness of digital systems themselves. Several phenomena may serve as empirical evidence:

- The early focus of conceptualist Net.art on the aesthetics and politics of code;
- in turn, the impact of Net.art aesthetics on experimental literature / poetry in the Internet;
- the close affinity of Net.art with political activism in the Internet;
- which itself increasingly affiliates itself with an older, technical 'hacker' culture (of Chaos Computer Club, 2600, etc.);
- the strong interest for (a) Free/Open Source Software and (b) network protocol standardization in all these camps;
- the fact that hacker aesthetics, Net.art aesthetics, code aesthetics and network protocol aesthetics have a tremendous impact on contemporary writing in the Internet. (See the work of mez, Alan Sondheim, Talan Memmott, Ted Warnell and others.)

I wll discuss how "Codeworks" (Alan Sondheim) fit notions of text that were crafted without digital code (and most importantly: machine-executable digital code) in mind and vice versa. Is it a coincidence for example that, reflecting the low-level codes of the Internet aesthetically, codeworks ended up resembling concrete poetry? And, apart from aesthetic resemblances, how do computer programs relate to literature? Is that what is currently being discussed as "Software Art" a literary genre?
Since many of these positions remain debatable, I would like to put up questions in my presentation rather than give answers.

Code

In his abstract for this conference, John Cayley takes a position which to some extent seems to be the opposite to mine. It's very exciting for me that we will have a debate at this conference, and so I would like to make my point and clear up my basic assumptions about the term ``code''.

Since computers, the internet and all digital technologies are based on zeros and ones, they are based on code. Zeros and ones are an alphabet which can be translated forth and back between other alphabets without information loss. The internet and computers run on alphabetic code, whereas, for examples, images and sound can only be digitally stored when translating them into code, which - unlike the translation of conventional text into digital bits - is a lossy, that is, not fully reversible and symmetric translation. In digital systems, literature is a privileged symbolic forms for this very reason. We may automatically search a collection of text files for all occurences of the word "bird", but doing the same with birds in a collection of image files or bird songs in a collection of audio files is incomparably tricky, nasty and error-prone - and relies after all on nothing else but artificial intelligence algorithms which, by pattern recognition, translate image pixels and sound waves back into text patterns.

The reverse is also true: We can perfectly translate digital data and algorithms into non-digital media like print books, as long as we translate them into alphabetic signs. This is exactly what is done, for example, in programming handbooks or in technical specification manuals for Internet standards. Meanwhile, there are two famous examples of a forth-and-back translation between print and computers:

1. The sourcecode of Phil Zimmerman's cryptography program ``Pretty Good Privacy'' (PGP). The PGP algorithms were legally considered a weapon and therefore became subject to U.S. export restrictions. To circumvent this ban, Zimmerman published the PGP sourcecode in a book. Unlike algorithms, literature is covered by the U.S. First Amendment of free speech. So the book could be exported outside the United States and, by scanning and retyping, translated back into an executable program.
2. The sourcecode of DeCSS, a small program which breaks the cryptography scheme of DVD movies. Since U.S. jurisdiction declared DeCSS an ``illegal circumvention device'' according to the new Digital Millennium Copyright Act (DMCA), the ban equally affected booklets, flyposters and t-shirts on whom the DeCSS sourcecode was printed.

So it is in fact terminological sloppiness if we speak of ``digital media''. Strictly speaking, there is no such thing as digital media, but only digital information.

Today, an average personal computer uses magnetic disks (floppy and hard disks), optical disks (CD-ROM and DVD-ROM) and chip memory (RAM) as its storage media, and electricity as its transmission media. Theoretically, one could build a computer with a printer and a scanner which uses books and alphabetic text as its storage media.[1] Punchcards in 1970s computing were actually similar to this, the computer museum in Boston even features a mechanical computer built entirely from wood.

Juxtapositions of ``the book'' and ``the computer'' are quite misleading, because they confuse the storage media (paper vs. a variety of optical, magnetical and electronical technologies) with the information (alphabetical text vs. binary code). It also ignores, by the way, the richness of storage and transmission media in traditional literature which, aside from the book, include oral transmission and mental storage, audio records and tapes, the radio, to name only a few.

If there is, strictly speaking, no such thing as digital media, there also is, strictly speaking, no such thing as digital images or digital sound. What we refer to as a ``digital image'', for example, is actually a piece of code which contains the machine instructions to produce the flow of electricity with which an analog screen or an analog printer displays an image.[2] Of course it is important whether a sequence of zeros and ones translates, into, say, an image because that defines its interpretation and semantics. The point of my formalistic argumentation is not to deny this, but to clarify that

1. when we speak of ``multimedia'' or ``intermedia'' in conjunction with computers and digital art and literature, we actually don't speak of digital systems in themselves, but about translations of digital information into analog output and vice versa;
2. text and literature highly are privileged symbolic systems in these translation processes because (a) their alphabetical signifiers can be infinitely translated forth and back without information loss and (b) computers factually run on them.

Literature and computers meet where alphabets and code, human language and machine language intersect, rather than in the interfacing of analog devices through digital control code we call multimedia. The computer does not extend literary media in any way, because all those media - electricity, electrical sound and image transmission etc. - existed before and without computers and digital information.

So I have to correct myself and the position I presented last year at this conference: If we speak of digital poetry, or of computer network poetry, we don't have to speak of certain media, and he don't have to speak of certain machines. If computers can be built from broomsticks and if any digital data, including executable algorithms, can be printed in books, there is no reason why computer network poetry couldn't or shouldn't be printed as well in books.

Perhaps the term of digital ``multimedia'' - or better: ``intermedia'' - would be more helpful if we redefine it as the *the possibility to losslessly translate information from one sign system to the other, forth and back, so that the visible, audible or tacticle representation of the information becomes purely arbitrary*. This can't be done unless the information isn't coded in some kind of alphabet, whether it's alphanumerical, binary, hexadecimal or Morse code.

Literature

Synthesis: putting things together

When we observe the textual codedness of digital systems, there is of course the danger to generalize and project one's observations of digital code onto literature as a whole. Computers operate on machine language, which is syntactically far less complex than human everyday language. The alphabet of both machine and human language is interchangeable, so that ``text'' - if defined as a conglomeration of alphabetical signs - remains a valid descriptor for both machine code sequences and human writing. In syntax and semantics however, machine code and human writing are not interchangeable. Computer algorithms are, like logical statements, a formal language and thus only a restrained subset of language as a whole.

However, it is a common mistake in my opinion to claim that (a) machine language would be only readable to machines and hence irrelevant for human art literature and (b) that, vice versa, literature and art would be unrelated to formal languages.

One should not forget that computer code, and computer programs, are not machine creations and machines talking to themselves, but written by humans.[3] The programmer-artist Adrian Ward suggests that we put the assumption of the machine controlling the language upside down:

> ``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.''[4]

Perhaps one also could call it composing scores, and it does not seem accidental to me that musical artists have picked up and grasped computers much more thoroughly than literary writers. Western music is an outstandig example of an art which relies upon written formal instruction code. Self-reflexive injokes such as ``B-

A-C-H" in Johann Sebastian Bach's music, the visual figurations in the score of Erik Satie's ``Sports et divertissements" and finally the experimental score drawings of John Cage shows that, beyond a merely serving the artwork, formal instruction code has an aesthetic quality and complexity of its own. In many works, musical composers have shifted instruction code from classical score notation to natural human language. A seminal piece, in my opinion, is La Monte Youngs ``Composition No.1 1961" which simply consists of the instruction ``Draw a straight line and follow it."[5] Most Fluxus performance pieces were written in the same notation style. Later in 1969, the American composer Alvin Lucier wrote his famous ``I am sitting in a room" as a brief spoken instruction which very precisely tells to perform the piece by playing itself back and modulating the speech through the room echoes.

In literature, formal instructions is the necessary prerequisite of all permutational and combinatory poetry, which I spoke about last year. Kabbalah and magical spells are important examples as well. But even in a conventional narrative, there is an implict formal instruction of how - i.e. in which sequence - to read the text (which maybe or followed or not, as opposed to hypertext which offers alternative sequence on the one hand, but enforces its implicit instruction on the other). Grammar itself is an implicit, and very pervasive formal instruction code.

Although formal instruction code is, as I said, only a subset of language, it is nevertheless at work in all speech and writing.

But what seems remarkable about computing to me is that the namespace of executable instruction code and nonexecutable code is flat. One cannot tell from a snippet of digital code whether it is executable or not. This property does not stand out in the alphabet of zeros and ones, but is solely dependent on how another piece of code - a compiler, a runtime interpreter or the embedded logic of a microprocessor - processes it. Computer code therefore is highly recursive and highly architectural, building upon layers of layers of code.

Analysis: taking things apart

The fact that one cannot tell from any piece of code whether it is machine-executable or not after all is the principle of all E-Mail viruses on the one hand and of the net poetry of jodi, antiorp/Netochka Nezvanova, mez, Ted Warnell, Alan Sondheim, Kenji Siratori and others that pretends to be viral machine code on the other.

I would not attempt to make a theoretical point for the digital poetry as code poetry here if it wasn't backed up by others' artistic practices and my own aesthetic preferences in net poetry.

I think the ``codeworks" (to borrow from Alan Sondheim) of these writers and programmer-artists are prime examples for a digital poetry which reflects the intrisic textuality of the computer. But it does so not by writing, to quote Alan Turing via Raymond Queneau, computer poetry to be read by computers[6], but by playing with the confusions and thresholds of machine language and human language, and by reflecting the cultural implications of these overlaps. The ``mezangelle'' poetry of mez/Mary Ann Breeze, which mixes programming/network protocol code and non-computer language to a portmanteau-word hybrid, is an outstanding example of such a poetics.

In comparison with the poetics of formal instruction like in La Monte Young's composition 1961, in Fluxus pieces or permutational poetry, there is an important difference: The Internet code poets do not construct or synthesize code, but they use code they found and take it apart. I agree with Friedrich Block and the theses he wrote for this conference that digital poetry must be seen in context of experimental poetry in general. A poetics of synthesis was characteristic of combinatory and instruction-based poetry, a poetics of analysis characterized Dada and its later followers. But one hardly finds poetry with analytical approach to formal instruction code in the classical 20th century avant-garde.[7] Internet code poetry is being written in a new - if you like, ``postmodern'' - condition of machine code abundance and overload.

I said that there is no such thing as digital media and that digital code may be stored in any medium; it doesn't surprise me that the codework poetry is an excellent example to verify this thesis. Unlike hard-coded hypertext and multimedia poetry, most of the artists I mentioned prefer to write plain ASCII text. This also reveals the critical implication of its poetics and aesthetics. The poetics of hyperfiction and multimedia poetry ran more or less parallel to the establishment of the World Wide Web; hyperfiction authors rightfully saw themselves as its pioneers and, in the course of nineties, continued to push the technical limits of both the Internet and multimedia computer technology. Much digital art and literature became testbed applications for new commercial browser features and multimedia plugins like QuickTime, ShockWave and Flash, but by this locked itself into industry-controlled closed code formats, thereby assuming an uncritical, after all affirmative role in the proprietary reformatting of the Internet.

Shifting the focus of the reader back from slick multimedia interfaces to raw code, code poetry appears to have strong aesthetical and political affinities to hacker cultures. While hacker cultures are far more diverse than the singular term ``hacker'' suggests[8], hackers could as well be distinguished between those who put things together (like Free Software and demo programmers) and those who take things apart (like crackers of serial numbers and communication network hackers like the Chaos Computer Club). Code poets have factually adopted many poetical forms that were originally developed by various hacker subcultures from the 1970s to the early 1990s, including ASCII Art, code slang (like ``7331 wAr3z d00d'' for ``leet [=elite] wares dood'') and poetry in programming languages (such as Perl poetry), or they even belong to both the ``hacker'' and the ``art'' camp, like my fellow speaker Walter van der Cruijsen.

Conceptualist Net.art was, from its beginning on, engaged in a critical politics of the Internet and its code, being closely affiliated with critical discourse on net politics in such forums as the ``Nettime'' mailing list. In its aesthetics, poetics and politics, codework poetry clearly departs from Net.art, not from hyperfiction and its Brown University roots.

To resolve the title of my paper ``Digital Code and Literary Text'', I would like to strongly argue in favor of considering both to be related and intertwined. Given that literary text, and not digital code, is the reference measure, one can subscribe to this without, as John Cayley seems to suggest, having to subscribe to Friedrich Kittler's techno-determinist media theory; a theory which I consider a prime example of the metaphysical trap Derrida describes in ``Écriture et différence'': Having replaced a metaphysicial center (in Kittler's case that of ``Geist'' - spirit -, ``Geistesgeschichte'' - intellectual history - and ``Geisteswissenschaft'' - humanities -) with a different one (i.e.: technology, history of technology and technological discourse analysis). Wrongly believing to have rid itself from metaphysics, it proceeds to write it under a different label.

The subtitle of this text addresses an open question: ``Can notions of text which were developed without electronic texts in mind be applied to digital code, and how does literature come into play here?'' At the moment, I can answer this question at best provisionally: While all literature should teach us to read and deal with the textuality of computers and digital poetry, computers and digital poetry might teach us to pay more attention to codes and control structures coded into language in general. My list of musical compositions and literary forms is fragmentary in this respect at best. For the generally thinking about language and text, program code appears to amalgamate in itself two concepts which are traditionally juxtaposed and unresolved in modern linguistics: the structure, as conceived of in formalism and structuralism, and the performative, as developed by speech act theory.

References

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

[MB98] Harry Mathews and Alastair Brotchie, editors. *Oulipo Compendium*. Atlas Press, London, 1998.

[Que61] Raymond Queneau. *Cent mille milliards de poèmes*. Gallimard, Paris, 1961.

Notes

1. Such a machine would operate slower than with magnetical or optical media, but on the other hand provide more robust and durable information storage
2. Normally, this code is divided into three pieces, one - the so-called sound or image file - containing the machine-independent and program-independent abstract information, the second - the so-called display program - containing the instructions to mediate the abstracted information in a machine-independent, yet not program-independent format to the operating system, the third - the so-called operating system -, mediating the program output to the output machine, whether a screen or a printer. But these three code layers are nothing but arbitrary conventions. Theoretically, the ``digital image'' file could in itself contain all the code necessary to make itself display on analog end devices
3. No computer can reprogram itself; self-programming is only possible within a limited framework of game rules written by a human programmer. A machine can behave differently than expected, because the rules didn't foresee all situations they could create, but no machine can overwrite its own rules by itself.
4. quoted from an E-Mail message to the ``Rhizome'' mailing list, May 7, 2001
5. [hun90], no page numbering
6. [Que61], p.3
7. An exception being the the ALGOL computer programming language poetry written by the Oulipo poets François le Lionnais and Noël Arnaud in the early 1970s, see [MB98], p.47
8. Boris Gröndahl's (German) Telepolis article ``The Script Kiddies Are Not Alright'' gives an excellent overview of the multiple camps associated with the term ``hacker'', "http://www.heise.de/tp/deutsch/html/result.xhtml?url=/tp/deutsch/inhalt/te/9266/1.html"