Generator: The Value of Software Art

Geoff Cox

Practices that combine the fields of art and technology employ a contested range of terms. The term 'software art' has become popular to describe the contemporary artistic preoccupation with software production. Certainly 'media arts' is far too broad a description and one that would focus attention too heavily on the 'medium' and 'mediation' of software rather than emphasize its dynamic properties, processes and metaphors. Software art is clearly not just media art, as it expresses more complex processes than simply something mediated between sender, apparatus and receiver.

Software refers to a computer program and the resources related to it that act upon the hardware of the physical machine components and machine. In more detail, this means software includes not only the instructions written in a particular language as the program, but also the other materials required for it to run, that are usually combined for distribution. Hardware is worked upon, and software performs the work. This link to performance also clarifies something about the use of the term 'software art', in describing not merely software used to produce art, but the software itself as the artwork. In other words, the programmers put the pre-existing hardware to work, in a similar way to artists producing concepts and manipulating materials in more traditional forms.

There is little new in placing emphasis on process rather than end product in this way, but the assertion of this essay is that software art exemplifies process-orientated practice in a way that lends itself to critical work appropriate to contemporary conditions. Clearly there is a history to this, and there have been many previous examples of artists generating creative work in an algorithmic manner, using instructions and constraints, whether using computers or not. The older term 'generative art' is generally used in this connection, as well as 'computer arts' with reference to practices

of the 1960s and 70s. A general view has emerged that older definitions associated with generative art stress the formal rule-based and syntactical properties of software, and thus do not place sufficient emphasis on semantic concerns and social context. Although, in general, this may be the case, formal concerns are essential to understand the more cultural aspects and the generative or transformative aspects of software. The chapter argues that taken together, the terms generative art and software art emphasize productive contradictions – inherent to both, and between the two.

Generator

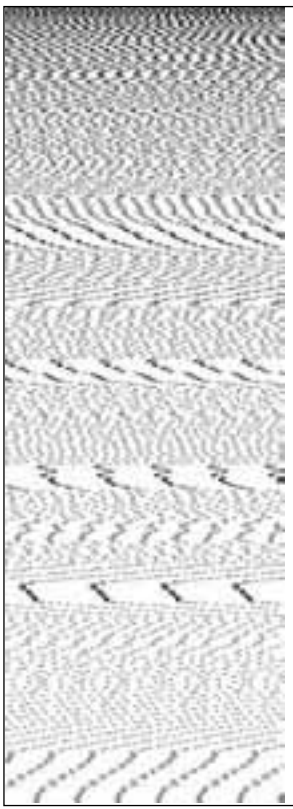
There is broad agreement that generative art is a term applied to artwork that is automated by the use of instructions or rules by which the artwork is executed. The outcome of this process is unpredictable, and can be described as being integral to the apparatus or situation, rather than a direct consequence of the artist's intentions. But importantly, the description recognizes that other agencies are at work, including human agency as an integral part of the production process in setting the rules. It is this line of thinking that informed the curation of the touring exhibition *Generator* (2002/3),[1] presenting a series of self-generating projects, incorporating digital media, instruction

Figure 1. (detail, video still): Yoko Ono's *Mend Peace for the World* (2001).

and participation pieces, experimental literature and music technologies. The work of artist-programmers was combined with artists from a conceptual tradition who employ rules and instructions in their practice. For instance, the work of Alex McLean, Joanna Walsh and Adrian Ward were presented in parallel to Stuart Brisley, Sol LeWitt and Yoko Ono, amongst others (http: //www.generative.net/generator). All work was considered performative in the sense that the artwork was generated through a real-time process.

To stress the point about agency, two specific examples are offered: Ono's *Mend Peace for the World* (Figure 1) consisted of broken dishes from around the world and materials to mend them. The instructions, to be executed by those visiting the exhibition, were: 'Keep adding more crockery as it gets fixed. Keep wishing while you mend.'

In contrast, McLean's *forkbomb.pl* was a program script designed to take a computer to its operational limit. The program script creates new processes repeatedly using the fork system call, until the process table gets filled up and the system crashes. A computer system under such high load causes unpredictable results that pattern differently depending partly on the operating system it runs upon, in this case visualized as binary data.



```
#!/usr/bin/perl -w

use strict;

die "Please do not run this script without reading the
documentation"
    if not @ARGV;

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;
```

Figure 2. Output and program script from Alex McLean's *forkbomb.pl* (2001).

Both examples – one tending towards reparation or reconstruction, the other towards destruction – emphasize a rejection of what one might refer to as 'software-determinism'. They demonstrate how the producer can concede control to some extent – and this is an important qualification – over the production of the work but that human intervention is paramount to (software) production. In other words, the artwork is necessarily programmed – with or without the aid of a computer. Whether the artist was involved in the writing of the software or not is beside the point. Someone was.

Generative art

In contrast to what has been said about these examples from *Generator*, much of the work in the field of generative art stresses issues of unpredictability and autonomy rather differently. In seeking to clarify what constitutes generative art, Philip Galanter's definition is much cited and positions generative art as broadly rule-based:

> Generative Art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art. (Galanter 2003)

Also defining generative art, John McCormack adds the influence of biology and emergent behaviour and, in particular, the terms 'genotype' and 'phenotype'. He argues that software can be seen in terms of 'genotypes' (DNA in cells) as machine code, and 'phenotypes' (the higher level form of behaviour) as what happens when it runs. The programmer would set the parameters that defined the fitness, and the software would evolve 'autonomously'. Put simply, McCormack generalizes that the authoring process is directed towards a genotype as the specification of a process, and when this process is executed it generates the phenotype as the 'experience of the artwork' (in Brown 2003). It is worth noting the elevated position of the artist in this description as responsible for the DNA of the artwork in the perpetuation of a 'creationist' myth. Clearly, other external factors are at work in creative production in art and life.

In his essay 'What is Generative Art? Complexity Theory as a Context for Art Theory' (Galanter 2003), Galanter supplements his earlier definition by referring to generative systems as also displaying emergent behaviour. His definition of generative art is an eclectic one, contributing to wider discussions around cultural practices (and not just art) that allow for the inclusion of practices that do not necessarily involve computers at all. But to Inke Arns, this is part of the problem as the definition is far too inclusive, applied across many fields of practice that focus attention on the end product of a process. She quotes Tilman Baumgärtel's article 'Experimental Software' (from 2001) to stress the distinction between earlier work using computers and software art, where the latter is:

> ...not art that has been created with the help of a computer, but art that happens in the computer, software is not programmed by artists in order to produce autonomous artworks, but the software itself is the artwork. What is crucial here is

> not the result but the process triggered in the computer by the program code. (Arns 2004)

Both Galanter and McCormack's statements do appear to verify an emphasis on end product as opposed to Baumgartel's emphasis on process. There is a danger of emphasizing the formal and syntactic aspects in using the term generative art. But in the case of software, it is not simply a choice of process or product but of the interaction between source code and its executed form. An example of this is McLean's *forkbomb.pl* (2001, described earlier) that demonstrates both the aesthetic appreciation of source code in parallel to a visualization of the process when run. This is, in fact, how it was exhibited as part of the *Generator* show, with the source code displayed as an integral part of the work. Nevertheless, the criticism Arns is making is that the privileging of execution, even if in combination with source code, avoids some of the contemporary practices associated with software art. She is thinking of programs that are not necessarily executable, or executable only on a conceptual level (often referred to as 'codework'). Perhaps it is simply a case of generative art requiring improved description to shift emphasis from the object generated to the process of generation. In this way, and quite literally, the term generator stands for: that which generates.

To generate something accounts for most creative activity in a very general sense. A more specific use in relation to arts practice can be traced to a lecture, 'Generative Art Forms' (presented at the Queen's University, Belfast Festival), in 1972, by the Romanian sculptor Neagu (who also founded a Generative Art Group). But a more common reference is Noam Chomsky's *Syntactic Structures* (1972), first published in 1957, often cited as the source of the concept 'generative grammar' (sometimes referred to as 'transformational grammar'). Chomsky assumes that somehow grammar is given in advance ('hard-wired') and, therefore, human consciousness contains innate grammatical competence that is pre-social (Chomsky 1972). This explains his interest in 'syntactic structures' by which sentences are constructed in particular languages to understand the properties that underlie successful grammars (ibid.). These concerns have also been the inspiration for much artistic experimentation using computers, as it lends itself to the procedural qualities of programming as an expression of transformative grammar.

Often cited in this connection is the 'Ouvroir de Littérature Potentielle' (OuLiPo), a group of writers and mathematicians founded in 1960 by Raymond Queneau and François Le Lionnais. Their concerns were syntactic rather than semantic, concerned with constraints 'brought to bear on the formal aspects of literature' (Le Lionnais, in Motte 1998). Rather than a chance operation (such as in the work of John Cage), Oulipean texts are generated through the use of constraints or rules, wherein any ideas associated with freedom of expression is undermined. An example that lends itself to computation is Queneau's *Cent Mille Milliards de Poèmes* [one hundred thousand billion poems] (1961) in which ten sonnets can be arranged according to formal rules. To each of the ten first lines, the reader can add any of ten different second lines, and

so on. The sonnet has fourteen lines, so the possibilities are of the order of 10 to the power of 14, or one hundred trillion sonnets. Le Lionnais makes a claim for the significance of this in terms of technical superiority: 'the work you are holding in your hands represents, itself alone, a quantity of text far greater than everything man has written since the invention of writing' (Motte 1998).

Potential writing in this sense implies the impossibility of its potential reading – and both are exponentially bound. The full potential of this work lies unrealized for practical reasons, perpetually in a suspended state of its further reading. In an experiment to exploit the potential of the computer, Paul Braffort was commissioned to program some of the OuLiPo works, such as Queneau's *Cent Mille Milliards de Poèmes*. In describing this enterprise as 'algorithmic literature', Paul Fournel argues that the machine allows the author to dominate the existing relations of computer, work and reader in new ways (Motte 1998: 140–2). Inspired by such examples, for *Generator*, Joanna Walsh's *Oulibot* operated as a member of an active 'irc' (chat room) community (Figure 3). The program (bot) learnt from the channel and produced plausible utterances based on constraints, and other oulipean transformations of text; such as Jean Lescure's 'S+7' method in

Figure 3. (detail, video still): Joanna Walsh's *Oulibot* (2002).

which a text is taken and each word ('s' for substantive) is replaced by the seventh following it in a dictionary.

Originality is clearly not the point in this work. Rather, creative endeavour is seen to be programmable and is considered in terms of its execution. But far from a deferral of authorship, the computer offers new potentialities in this way. What Le Lionnais calls a 'combinatory literature', is expanded greatly by the computer and its systematic compositional structure. The use of executable formal instructions makes explicit the idea of software as potential literature (or art), whether running on a computer or not. Indeed all conventions of writing and reading, of both text and code, have in common that they are part of a set of abstract (coded) systems of input and output.

Software art

In the many comparisons between contemporary software art and the older practices associated with generative art, McCormack explains one key difference was that in the 1960s and 1970s artists simply had to write (or ask someone to write) their own software in order to generate the outcomes (McCormack in Brown 2003). The now wide availability of authoring software has changed the conditions for the production of software art by the artist-programmer. It is with some of these issues in mind that Richard Wright traces the 'divergence between programmers and program users', based around the issue of whether a computer is considered a medium or a tool (Wright 2004). In a hierarchy of programming languages, Wright points out that not all programming practices are equal. He is thinking of the predominance of scripting languages such as Flash Actionscript (but also Lingo, Perl, MAX, JavaScript, Java, C++, as well as other programming and scripting languages) that use libraries of functions and a certain shared, if not prescribed, vocabulary of styles.

For Wright, this changes the terms of the discussion from a general issue of artistic programming to one of what kind of programming is being used. He cites the historical shift in Harold Cohen's practice from a painter to developing software to automate his artwork, through the use of what Cohen refers to as 'autonomous machine (art making) intelligence' (http: //crca.ucsd.edu/~hcohen/). Developed from 1973 onwards, the *AARON* program represents to Wright the historical transition towards contemporary culture, where the use of computers has become pervasive. As a result, the terms of practice have fundamentally changed for the artist-programmer. His argument is that 'Programming is not only the material of artistic creation, it is the context of artistic creation. Programming has become software.' (Wright 2004)

This refers to earlier practices that were characterized by artists working at the meta-level of programming. Despite this, programming in Cohen's work operates in a rather ambiguous relation to the overall artwork. Clearly in a general sense it is part of the artistic output but more in terms of a representation of his skills and technique, rather than as a constituent part of the artwork as such. In the tradition of generative art, the

emphasis tends towards the completed work of art rather than the program or programming being a work in itself.

In contrast to Cohen's work, a more contemporary reference that situates software art overtly in terms of programming is the exhibition *CODeDOC* first for the Whitney Museum of American Art's 'artport' website (2002) and later at Ars Electronica (2003) (http: //artport.whitney.org/commissions/codedoc/index.shtml). The curator, Christiane Paul, set the invited artist-programmers an instruction to 'connect and move three points in space' in a language of their choice (Java, C, Visual Basic, Lingo, Perl) and to exchange the code with the other artists for comments. The viewers of all the works in *CODeDOC* were invited to first read the written code and then see the executed work. This raised some controversy on mail lists at the time, for deliberately obfuscating or aestheticizing code to non-programmers, rather than demystifying the creative process. Yet the significance is that code is taken to be part of the work and not simply meant to assist interpretation. The curatorial statement contains a number of useful comments

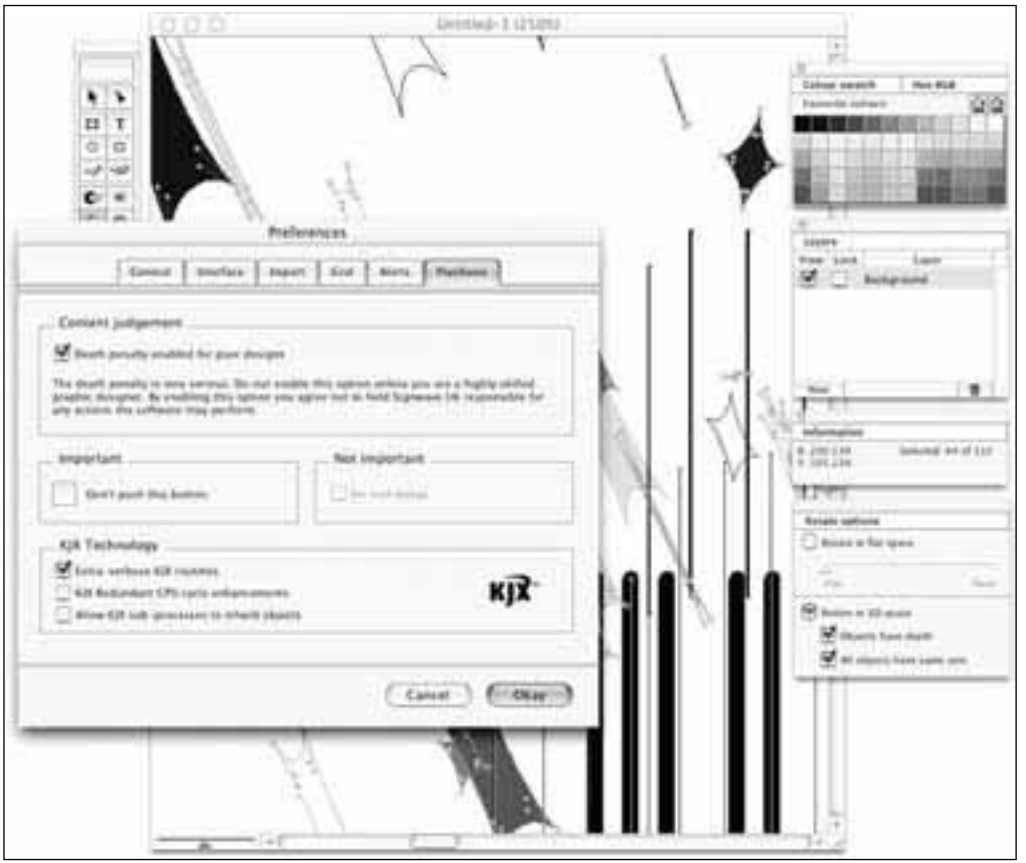


Figure 4. (screen grab): Adrian Ward's *Auto-Illustrator* (2000).

on the intentions of the experiment and reiterates the potential of software itself as artwork:

> In software art, the 'materiality' of the written instructions mostly remains hidden. In addition, these instructions and notations can be instantaneously activated; they contain and – further layers of processing aside – *are* the artwork itself. While one might claim that the same holds true for a work of conceptual art that consists of written instructions, this work would still have to be activated as a mental or physical event by the viewer and cannot instantaneously transform, transcend, and generate its own materiality. (Paul 2003)

This approach parallels some of the curatorial decisions for the *Generator* show, in particular McLean's *forkbomb.pl* where the source code was exhibited as an integral part of the artwork. Whereas formerly artists had to engage with programming in early computer arts practice, the lack of necessity now allows for other critical issues to be engaged (just as previously the invention of photography perhaps freed painting from figurative representation).

An example of critical software is Signwave's *Auto-Illustrator* that defies user expectation as a parody of the vector graphics design software Adobe *Illustrator*. It looks like and indeed works like conventional commercial software, but carries some extra auto-generative functionality that renders designs outside of the direct control or creativity of the user. Cheekily included in early releases was a license agreement that indicated that any designs were necessarily co-authored by the company Signwave who supply the software (aka Adrian Ward). Here, the parody operated particularly effectively, as some users were outraged that a company would insist on such a clause in a direct assault on their creative and intellectual rights. It highlights the issue that full authorship is rarely acknowledged in making art using software, as is the labour of all those involved in the process. The software was released as a boxed version for the exhibition *Generator* with a 'User's Manual' that contained both technical detail and critical essays. In this way, the commercial packaging added a further layer to its ironic critique of the commodification of art and software as art.

In linguistic terms, artist-programmers appear to have shifted their attention from an engagement with the syntax of programming to semantic concerns. This is, indeed, how Cramer makes the distinction between generative art and software art, by associating the former with syntax and the latter with semantics (2003). But this is not simply a shift from one to the other. Syntax, although not concerned with meaning in itself, certainly has implications for semantics, and both are required to inform an overall theory of language. Yet what Cramer is trying to emphasize is a shift in software art from 'pure syntax' to 'something semantic, something that is aesthetically, culturally and politically charged' (2003). It is not a choice of one or the other but a change of emphasis.

The apparent dualism between generative art and software art is also something that Mitchell Whitelaw disputes in questioning the binary relation of formalism (associated with generative art) and culturalism (associated with software art). Rather than seeing this as an impasse, Whitelaw suggests a 'complementarity' of positions that leads to alternative modes of being and relation (Whitelaw 2005). He calls this 'critical generativity' to stress the emergent and transformative properties that reflect social complexity and software's latent cultural agency (ibid.). This chapter also argues for new critical forms, but rather than seeking Whitelaw's complementarity or fusion, argues for a contradictory relation. That said, the competing definitions matter little in themselves but only in as much as they operate in terms of an overall contribution to a critical discourse around the practice of software art and culture. Software necessarily includes, if only on a conceptual level, a generative process in which something is always ready to come into being, however latent. It is for convenience only that this is referred to as software art.

The generative properties of software can be seen in parallel to the argument that art should now deal with the central issue of transformation rather than representation (whether software is used directly or not). On the surface this sounds like a very contemporary position, supported by the curator Nicolas Bourriaud's claim that the image is now defined by its 'generative power', and that art can be seen to be a program(me) for the generation of forms and situations (Bourriaud 2002). His term 'relational aesthetics' describes a practice that involves human interactions, social context and the new aesthetic and cultural concerns that arise from this. He is referring to artwork that is a programme to be followed, a model to be reproduced, or an encouragement to do something – and points to the parallel activities of artists engaging in ideas of interaction and sociability, set against the hype of interactive computer systems. To Bourriaud, artwork not using the computer has as much potential to make work about its effects. This may well be the case (as with many of the works selected for *Generator*), but this position is simply based on systems thinking, and the statement is consistent with Gregory Bateson's 1971 position on art (from *Steps to an Ecology of Mind*) that focuses attention not on the message but the code (Bateson 2000). Bateson considers the production of art, and art as product, in terms of behaviours or rules that are embodied in the machinery that then generates transformations (Bateson 2000). Software appears to characterize arts practices that privilege the idea, code, process, system and its transformational qualities. Whether using a computer or not, art has become ever more like software.

Software as cultural metaphor

Clearly there is a history to software art, and a canon appears to have emerged. Andreas Broegger is one researcher amongst many who situates the contemporary term software art in the historical context of the *Radical Software* journal published by the Raindance collective (launched in 1970, http: //www.radicalsoftware.org/) and Jack Burnham's exhibition *Software, Information Technology: Its Meaning for Art* at the Jewish Museum, New York (also 1970). Broegger describes the ways the term software was

used as a metaphor for arts practice at that time, to stand for the transmission of information using available communications technologies, in contrast to the 'hardware' of object-based art (Broegger 2003). Although any discussion of software requires an understanding of its relationship to hardware (even if it is accepted that software can exist without hardware), it is clear that the term software is being used in a rather different sense in the 1970s. In the field of art at least, the description runs in parallel to Conceptualism and its associated shift away from the end product at that time. A contemporary use of the term software reflects an emphasis on process, which has become the orthodoxy in contemporary cultural practices. Software is more than just art and expands an understanding of art's possibilities to engage wider social issues.

A statement from the first issue of the *Radical Software* journal gives a clear indication of its agenda: 'Power is no longer measured in land, labour or capital, but by access to information and the means to disseminate it' (Ross 2003). On a technical level, it was the widespread availability of the video portapak that inspired the belief that this could contribute to social transformation, through people gaining increased access to the means of production and becoming producers. In the context of its publication in the United States, the position of the journal was influenced by the rise of the civil rights movement, a general mistrust of the communications media on offer, requiring more independent and alternative media and cultural practices, combined with ecological concerns (according to Ross 2003). Those associated with this project 'imagined a world in which the contest of ideas and values could take place freely and openly' outside of the existing institutional and ideological frameworks of commercial telecommunications. They proposed 'a new information order in which the very idea of hierarchical power structure might be transformed or even eliminated' (Ross 2003). In this sense, what is radical about software is that it acts upon hardware. It operates as a metaphor for an emphasis on social processes that involve an engagement with relations of production and 'radical' transformation.

In parallel to the *Radical Software* journal, 'software as a metaphor for art' was explored in Burnham's *Software* exhibition. The show can be seen as a product of its times with its overt structuralist and conceptualist concerns, and its aim to focus attention on the technical apparatus. It corresponds to what has since become commonplace in looking to the 'dematerialization' associated with the conceptual arts tradition and the 'immaterialization' of information and communications technology. In his essay 'The House that Jack Built' (1998), Shanken traces Burnham's concerns with particular reference to his book *Beyond Modern Sculpture: The Effects of Science and Technology on the Sculpture of this Century* (1968) that ends with an account of 'systems aesthetics'. By an aesthetics of systems, Burnham refers to non-object–based art and time-based practices such as performance, interactive and conceptual art, but also public interaction that breaks down the false distinction between the operating systems of art and non-art. This is software metaphorically speaking, the abstract 'internal logic' of a program receiving feedback from human subjects. In summary, the exhibition *Software* was an attempt to reveal some of the contradictions between

object and non-object, art and non-art, artist and non-artist, evident in art's organizational and systemic logic.

Conceptualism has been particularly influential in attempts to draw software art into an art historical register. Referring to Lucy Lippard's portrayal of dematerialization, software art is clearly both concerned with art as idea and action, which both on a conceptual and technical level describes source code and its execution. The generative approach of conceptual artist Sol LeWitt is evocative of software in this connection: 'The idea becomes a machine that makes the art' (in Lippard 1997). In LeWitt's work, instructions are provided for the production of artworks that are then executed by other people. For instance, in *Chicago*, a publication produced for *Generator*, a serial variation using nine found postcards was produced using a simple algorithm as follows:

1
2
1 2
3
1 2 3
4
1 2 3 4
5
1 2 3 4 5
6
1 2 3 4 5 6
7
1 2 3 4 5 6 7
8
1 2 3 4 5 6 7 8
9
1 2 3 4 5 6 7 8 9

The comparison of software art to earlier art movements such as Conceptualism, but also the avant-garde activities of the 1920s in Russia and Germany, provides an historical understanding of radical forms and strategies. But in the contemporary situation, it appears that many of the claims of the historical avant-garde have become:

> ...embedded in the commands and interface metaphors of computer software. In short, the avant-garde vision became materialized in a computer. All the strategies developed to awaken audiences from a dream-existence of bourgeois society [...] now define the basic routine of a post-industrial society: the interaction with a computer. (Manovich 1999)

The once radical technique of montage has become commonplace. On the surface, it seems that what was once a radical aesthetic vision to reveal the social structure behind

Figure 5. (video still): final page of Sol LeWitt's *Chicago* (2002), produced in collaboration with Alec Finlay of MorningStar/Pocketbooks.

the visible surfaces has become a standardized form through the use of computer technology. Lev Manovich discusses these perceptions of change and the ways in which ideology naturalizes these changes. This reflects contemporary culture's reliance on appropriation, wherein recycling, re-working, and re-combining media are the standard techniques. He concludes that 'the avant-garde becomes software' and that it continues to introduce revolutionary techniques but the terms are different:

> 'software does not simply adopt avant-garde techniques without changing them; on the contrary, these techniques are further developed, formalized in algorithms, codified in software, made more efficient and effective' (Manovich 1999).

If art holds radical potential at all in these post-political times, the issue remains how to produce art that resists its seemingly inevitable commodification and how to reconcile the apparent failure of the avant-garde to deliver its promises (evident in Conceptualism

Figure 6. (detail, video still): Stuart Brisley and Adrian Ward's *Ordure: : real-time* (2002), courtesy of the UK Museum of Ordure.

and Dada). If, as Manovich thinks, software has naturalized montage techniques, how can software be further developed as a radical project in revealing the ideological processes at work?

Exhibited as part of *Generator* in the form of a large projection, Stuart Brisley and Adrian Ward's *Ordure: : real-time* is both a representation and a process of detritus that slowly 'corrupts' – pixel by pixel. The corruption is triggered by viewing the image. The more people that view the image, the more prevalent its decay. When the image is left alone, it rewrites itself anew. On one level, the resultant image with pixels moving incrementally out of order is the same as the first image where the pixels are in the correct order. The data is consistent, the pixels merely rearranged. The work demonstrates the dialectical play between two interconnected states of order and disorder, between generation and corruption, suggesting the potential for change built into any system. Indeed, complexity theory verifies that systems are not closed but can be seen to be also sensitive to small changes.

The challenge for a critical practice in software art is to maintain contradiction in the process of transformation, for this is where politics is evident and where re-invention takes place. In terms of the legacy of previous radical arts practice and some of the examples cited here, the lessons of art history exemplify the point that Lippard makes: that in a contemporary situation where conceptual strategies have become the orthodoxy of contemporary art and effectively recuperated, radical art can be found in social energies not yet recognized as art (Lippard 1997). Perhaps software art and culture represents such an instance – for now at least.

Note

1. *Generator* was a SPACEX touring exhibition, curated by Geoff Cox and Tom Trevor, with support from the National Touring Programme of the Arts Council of England. It was shown at Spacex, Exeter (2002), then toured to the Liverpool Biennial (2002) and Firstsite, Colchester (2003). Commissioned artists included emerging computer artist-programmers, as well as more established figures from a conceptual art tradition, all of who work with generative forms and ideas: Mark Bowden, Stuart Brisley, Angus Fairhurst, Alec Finlay, Tim Head, Jeff Instone, Zoë Irvine, Sol LeWitt, limbomedia, Alex McLean, Guy Moreton, Netochka Nezvanova, Yoko Ono, Organogenesis Inc., Jon Pettigrew, Colin Sackett, Sulawesi Crested Macaques from Paignton Zoo, Joanna Walsh, and Adrian Ward. http: //www.generative.net/generator/.

References

Arns, I., 'Read_me, Run_me, Execute_me: Software and Its Discontents, or: It's the Performativity of Code, Stupid!', in Goriunova, O., & A Shulgin, A., (eds.) *Read_Me: Software Art & Cultures – Edition 2004*, Ärhus: Digital Aesthetics Research Centre, University of Ärhus, 2004, pp. 176–193.

Bateson, G., *Steps to an Ecology of Mind*, Chicago/London: University of Chicago Press, 2000 [1971].

Bourriaud, N., *Relational Aesthetics*, trans. Simon Pleasance & Fronza Woods, Dijon-Quetigny: Les Presses de Réel, 2002.

Broegger, A., 'Gigliotti on "(radical) software"', as part of software art thread, *Rhizome* (via Andreas Broeckmann) 10 Oct 2003, http: //www.rhizome.org/.

Brown, P., (ed.) 'Generative computation and the arts', in *Digital Creativity*, vol. 14, no. 1, Lisse: Swets & Zeitlinger, 2003.

Burnham, J., *Beyond Modern Sculpture: the Effects of Science and Technology on the Sculpture of this Century*, New York: George Braziller, 1968.

Chomsky, N., *Syntactic Structures*, The Hague: Mouton, 1972 [1957].

Cramer, F.,'Exe.cut[up]able statements: the Insistence of Code', in Gerfried Stocker & Christine Schöpf (eds.), *Code – The Language of Our Time*, Ars Electronica, Linz: Hatje Cantz, 2003, pp. 98–103.

Galanter, P., 'What is Generative Art? Complexity Theory as a Context for Art Theory', *Generative Art 03*, international conference, Politecnico di Milano, Italy, 2003, http: //www.generativeart.net.

Lippard, L., (ed.) *Six Years: the dematerialization of the art object from 1966 to 1972*, London:

University of California Press, 1997.
Manovich, L., 'Avant-garde as Software', 1999, http: //www.manovich.net/TEXTS_04.HTM.
Motte, W. F. Jr (ed.), *Oulipo: a Primer of Potential Literature*, trans. Warren F. Motte, Illinois: Dalkey Archive Press, 1998.
Paul, C., 'Public Cultural Production Art(Software)' in Gerfried Stocker & Christine Schöpf (eds.), *Code – The Language of Our Time*, Ars Electronica, Linz: Hatje Cantz, 2003, pp. 129–135.
Ross, D. A., 'Radical Software Redux', 2003, http: //www.radicalsoftware.org/e/ross.html.
Shanken, E. A., 'The House That Jack Built: Jack Burnham's Concept of "Software" as a Metaphor for Art', in *Leonardo Electronic Almanac*, 6: 10, November 1998, http: //mitpress.mit.edu/e-journals/LEA/ARTICLES/jack.html.
Shanken, E. A., 'From Cybernetics to Telematics: The Art, Pedagogy, and Theory of Roy Ascott', in Roy Ascott, ed., *Telematic Embrace: Visionary Theories of Art, Technology, and Consciousness*, Berkeley: University of California Press, 2003, pp. 1–94.
Whitelaw, M., 'System Stories and Model Worlds: A Critical Approach to Generative Art', in Olga Goriunova (ed.), *Readme 100: Temporary Software Art Factory*, Dortmund: Hartware Medien Kunst Verein, 2005, pp. 134–153.
Wright, R., 'Software Art After Programming', *Metamute*, July 2004, http: //www.metamute.org/en/node/414.