

University of Wollongong
Research Online

Faculty of Creative Arts - Papers (Archive)

Faculty of Arts, Social Sciences & Humanities


1-1-2007

Instrumental relations: software as art, art as software

Brogan S. Bunt

University of Wollongong, brogan@uow.edu.au

Follow this and additional works at: <https://ro.uow.edu.au/creartspapers>

 Part of the [Arts and Humanities Commons](#), and the [Social and Behavioral Sciences Commons](#)

Recommended Citation

Bunt, Brogan S.: Instrumental relations: software as art, art as software 2007, 78-87.
<https://ro.uow.edu.au/creartspapers/233>

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Instrumental Relations: Software as Art, Art as Software

Brogan Bunt
University of Wollongong
Wollongong, Australia
brogan@uow.edu.au

ABSTRACT

Software art is characterised by a close concern with the culture of software and the medium of programming. This inevitably demands an engagement with the terrain of the instrumental; software is a sphere of tool-making and programming is governed by conceptions of functional (and generic) utility. Yet where does this leave art? If, in Kantian terms, art is defined by its uselessness (by its lack of any externally grounded necessity) and if, in classical critical theoretical terms, this alienation from function opens up a space of critique, then how can art explore and participate within the instrumental without abandoning its fragile critical autonomy? This paper addresses this question, drawing upon Heidegger's conception of technology and Plato's conception of *poesis* to argue that critical software art can not simply oppose the instrumental character of software; instead it must acknowledge its own complicity in the operations of hiding and unreflective functioning that characterize the instrumental once the latter is re-conceived apart from the simplicity of human agency and humanly determinable ends. I examine one of my own software projects as a means of clarifying the dilemmas of critical aesthetic purchase that emerge as a result of this engagement with the instrumental dimension of software.

Keywords

Software art, aesthetics, instrumental, engines, tools

1. INTRODUCTION

Contemporary software art achieved its first notable recognition in 2001 when a prize for 'artistic software' was awarded at the Berlin *Transmediale* media arts festival. Subsequent key events included the 2002 Read_Me 1.2 Software Art/Software Art Games festival (Moscow) and the Whitney Museum's 2002 CODEeDOC exhibition (New York). The jury for the *Transmediale* festival define software art in terms of its difference from new media. Software art, they argue, shifts the focus from the visible surface of digital art to the constitutive space of code. Unlike traditional media, code, in their view, is not a passive intermediary; it does something, it is executable, it performs actions [23]. Programming represents a new condition of writing, in which the terrain of written abstraction obtains powers of curious literal agency. On this basis, the jury rejects the conventional notion of software as a tool. They argue that "digital code is virulent" [23], that it can only appear as a tool by disguising its actual operations. Software art has the potential - and crucial aesthetic responsibility - to expose the machinations of code, to make code visible. It represents an effort to reassert human, critical aesthetic agency and to counter the motions of hiding and disguise that are characteristic of code processes. In this manner, it imagines a direct opposition between software art and instrumental software - projecting an aesthetic alternative of manifest and critically reflective code.

My aim in this paper is question the viability of this approach, to suggest that the issues are more complex and uncertain. This issue takes shape for me in relation to the uncertainty of one of my own works. I describe it as a software art work, but with some hesitation. The work lacks an adequate aesthetic manifestation - either as code or as visible interface. It is a set of tools and an engine. It is concerned with the representation of time and the pragmatics of enabling a temporal display. The title of the work is *Cropper/Propper/Gridder*. If the work is of any interest, it is because it pursues a poetic idea through instrumental means - or better, it struggles to discover a potential for poetry in the aesthetic estrangement of software. While software art conventionally resists the instrumental character of software - struggling to make software aesthetically, reflectively appear - *Cropper/Propper/Gridder* deliberately engages with the aesthetic blindness of instrumental functioning.

This paper begins by addressing the general issue of the relation between the aesthetic and the instrumental - considering how the self-definition of critical software art adheres to a very conventional aesthetic scheme and how, in contrast, Heidegger's notion of technology [13] and Plato's notion of *poesis* [20] suggest an alternative relation that is characterized by dimensions of affinity, resemblance and undecidable difference. I then discuss *Cropper/Propper/Gridder* as a specific instance of the risking of the aesthetic within the terrain of non-identity and displacement that the instrumental represents.

2. RETHINKING THE INSTRUMENTAL

Within the tradition of critical theory, the notion of the instrumental is associated with a specifically modern mode of rationality that is oriented towards the purposive accomplishment of tasks, in the process deliberately bracketing questions of human value. Instrumental rationality addresses issues of efficiency and running, ignoring wider ethical, political and cultural concerns. The sociologist Max Weber argues that this mode of reason takes characteristic form in the mechanisms of modern bureaucratic administration and industrial capitalism [25]. This broadly social conception of the instrumental is predicated on a more fundamental notion of the nature of an instrument. An instrument is a device that moves but lacks free being. It produces results but without any awareness of cause or result. It functions unreflectively. It proceeds blindly. In this sense, despite its status as a technical contrivance, an instrument - in its motion, in its running - comes to resemble the deterministic processes of nature. At the very outset of his discussion of art in his 1790 *Critique of Judgement*, Kant explains that "Art is distinguished from nature as making (*facere*) is from acting or operating in general (*agere*); and the product of the result of the former is distinguished from the latter as work (*opus*) from operation (*effectus*)" ([16] p.523). In the same manner, an instrument can be regarded as performing operations which produce effects rather than performing actions

which shape (aesthetic) works. This indicates the obvious dilemmas that confront any attempt to chart an association between the instrumental and the aesthetic. Conceived as intermediary, mechanical and unreflective, the instrumental appears directly opposed to the finality, freedom and reflective nature of art.

2.1 Art and Engineering

In 1920 the Russian Constructivist artist, Vladimir Tatlin, produced a proposal for the Monument to the Third International. He envisaged a 400 metre high steel and glass tower that incorporated a dynamic spiral structure and rotating internal rooms. It was utopian art adopting the guise of an architectural plan and was criticized for its impracticality by revolutionary artists and politicians alike. Another Constructivist artist, Gabo, cautioned Tatlin to “either create functional houses and bridges or create pure art, not both” [26]. The work was condemned for confusing two distinct languages and modes of making: art and engineering. Furthermore it transgressed the conventional boundaries between the aesthetic end-in-itself and the sphere of useful things. These flaws are also the basis for its lasting significance as an icon of avant-garde art. The monument posed the essential problem concerning art’s relation to modern forms of making and, more generally, art’s relation to industrial modernity.

Two broad historical strategies emerge in relation to this challenge. On the one side there is the model supplied by Dada or incorporating technology and technological forms of making as a means of waging a multi-pronged assault on autonomous art, bourgeois humanism and instrumental rationality. This approach takes archetypal form in Marcel Duchamp’s *Large Glass – The Bride Stripped Bare by Her Bachelors, Even* (1915-23), which provides an ironic take on our pleasant fictions of love, free will and organic, human difference by representing human courtship and erotic coupling in mechanical terms. In a parodic reference to the history of industrial machinery, the processes proceed upwards from steam, to internal combustion engine to electricity. This is not, of course, a working machine. It is a playful, subversive, metaphorical apparatus. It functions as a piece of critical commentary rather than as a literal instrumental device. A crucial distance then is maintained between art and engineering so that art, however fractured, however affected by industrial modernity, can shape a properly aesthetic space of critique.

The other strategy, evident especially in Constructivism and the Bauhaus, strives towards a unity of art and industry. It projects an integration of the aesthetic (as mode of formal appearance) and the instrumental (as sphere of functional, mass-produced products). Art abandons its reflective autonomy to enter into the texture of practical things. While crucial as a critique both of art and alienated labour [5] ([14] p.12), this strategy runs the risk of providing an aesthetic sheen for forces that actually undermine the potential of art to suggest alternative social and imaginative possibilities. Furthermore, this effort to draw a close association between the aesthetic and the instrumental is much easier to manage with simple, everyday things – coffee cups, tables, light fittings, etc. - in which form and function share a common immanent material being. Software programming is harder to conceive in these terms because it institutes a separation between the domain of instrumental instructions and the visible interface. The former indicates a space of symbolic abstraction and functioning that is hidden from view - that is not instantly coextensive with the terrain of user interaction. Although authors

such as Donald Knuth [18] portray programming as a practical art which can be regarded aesthetically in terms of values such as economy and elegance, this has the unfortunate consequence of making the aesthetics of code only accessible to programmers and represents a return, as Cramer argues, to a very traditional neo-classical aesthetic space ([7] p.10).

In short, neither critical nor integrative strategies genuinely engage with the instrumental in its non-aesthetic distance. Critical avant-garde art resists literal instrumental functioning while modernist design works to aestheticize the functional. Neither provides an adequate means of conceiving the field of software programming, which refuses to adopt a conventional aesthetic form, which is directed elsewhere, which shapes instructions rather than an easily critical or conciliatory work. If there is anything unique about the situation of software art it lies precisely in this search for an aesthetic rationale without the possibility of any recourse to the non-instrumental or the consolation of immanent form.

2.2 A Problematic Definition

The jury for the 2002 Moscow Read_Me 1.2 festival offer an influential definition of software art:

“We consider software art to be art whose material is algorithmic instruction code and/or which addresses cultural concepts of software.” [22]

Although intended to be inclusive, this definition works to obscure the key issue of the relation to the instrumental. Instead it focuses on distinguishing two strands of software art practice - formally oriented code-based experimentation and culturally oriented software critique. The formalist option is expressed in terms that recall the language of high modernism; the focus is upon defining the material essence of the software medium, which here takes the form of “algorithmic instruction code” [22]. In this manner, a complex cultural assemblage – a language and a field of discourse – is reduced to the status of a simple material, like paint or clay. This reduction of code to the simplicity of an aesthetically malleable material is what enables formalist software art to be represented as a purely conceptual meditation on aspects of system without any integral concern with dimensions of culture. However, a close engagement with the medium of code can have other implications. It can have a cultural dimension. It can represent an engagement with a specifically culturally determined discursive space. More particularly, it can represent an interrogation of the instrumental language and strategies of conventional software. But unfortunately, by positioning code as a base aesthetic matter, formalism loses sight of this possibility. It is left to the other side of the definition to engage with software as a cultural phenomenon.

But correspondingly, although the culturalist option “addresses cultural concepts of software” [22] it seems to lack a specific point of discursive purchase. How is the nature of this mode of address to be described? Is this critique spoken in the language of code as actual functioning software or is it expressed in other terms? There is a need to explain how critical software art relates to the layer of instrumental, non-reflective language that provides the basis for its operations. There is a need to think through the engagement with the material language of code. In this sense, the cultural critique of software cannot be conceived apart from the apparently formalist option. The distinction between formalist and cultural tendencies obscures this vital issue.

If this bifurcated notion of software art is ultimately disabling, working to impoverish both formal experimentation and cultural critique, it is because it misconceives the field in terms of a tension between contrasting aesthetic tendencies rather than in terms of a more constitutive tension between art and the instrumental dimension of software.

2.3 Software Becoming Art

The notion of software art appears at one level as a transgression of ordinary aesthetic proprieties. In a traditional avant-garde spirit, it seems to unsettle the complacent autonomy of art, insisting that art engage with a space of non-art - a realm of engineering and technical implementation. Yet at another level it proceeds in an opposite fashion. Rather than genuinely risking a relation to the alterity of another cultural and discursive space, it conceives software in terms of art. It dialectically subsumes those aspects of software that are aesthetically useful and digestible, while discarding everything else. This is evident in as much as the specific characteristics of software art correspond to a very conventional aesthetic scheme. It is worth briefly outlining the contours of this scheme in terms of Kant's classical model of aesthetics and fine art.

According to Kant (1980), aesthetics denotes a realm of non-instrumental engagement with things. It is a sensuously enabled mode of reflective judgement that rises above the dimension of sense to enter into dialogue with the *apriori* space of conceptual understanding [9] ([16] p.484). The experience of beauty, for instance, relates to the recognition of order in the symmetrical forms of nature – mineral and organic forms that are not themselves conceptual but that nonetheless reveal a systematic, formal logic (pattern, unity and harmony); an order that is apprehended through the senses but that instantly summons an awareness of the universal and the metaphysical ([16] p.493). Fine art, as a specific experience of the beautiful, manifests a purposiveness without purpose, a disinterested, non-utilitarian demonstration of the felt rightness of the conceptual ([16] pp.524-525). It strips real objects of their ordinary reality, their contextual significance as objects that are practically desired, manipulated and used. Art objects suspend the dimension of conventional instrumental utility in order to attain a higher conceptual utility as signs of an ultimate reconciliation of human faculties. Their lack of instrumental utility takes the form of an organic finality, a dimension of formal coherence without goal. The production of art depends upon genius; an “innate mental aptitude (ingenium) through which nature gives the rule to art” ([16] p.525). Unlike instrumental craft, which is the product of practical, formulaic labour, fine art is conceived as a generative expression of the soul as a protean ‘second nature’ ([16] p.528). Kant's aesthetic scheme is representative of an Enlightenment conception of art as non-instrumental, final, reflective and the product of genius.

Now, without trying to suggest that contemporary conceptions of software art are strictly-speaking Kantian, there are curious affinities linked to how issues of instrumental function, reflection and artistic subjectivity are conceived.

2.3.1 Function

We have seen that software art characteristically resists the notion of software as a tool. There is a strong preference for work that undermines utility and suspends ordinary functioning. Adrien Ward's *Signwave Auto-Illustrator* (2001) provides an iconic

example, although it is a work that represents, in my view, an ambivalent relation to the instrumental. In its adherence to the interface conventions of commercial creative software, *Auto-Illustrator* at once deconstructs and delights in the notion of software as tool. While the deconstructive orientation is emphasized, the manner in which the work draws inspiration from the conventional language of tool-based software escapes explicit attention.

Software art's suspicion of tools connects to the classical aesthetic bracketing of the instrumental, although clearly the aim is less to determine a pure space of disinterested perception than to critically respond to the dominant models of commercial application software. Yet it seems to me that the rejection of the notion of the tool – as well as the rejection of the tool's effort to disguise itself – creates fundamental problems for software art. Even if a piece of software is not ostensibly a tool, it must speak the language of tools. It is devised as a system, an apparatus. It functions. As languages and discursive forms, programming languages bear the necessary imprint of the industrial forces that have shaped them. The concept of a tool is implicit within programming structure – in the notion of an algorithm that processes data, an object that performs a specific (encapsulated) task and a procedure that runs more or less efficiently. In bracketing all of this, in trying to think algorithm and procedure beyond the instrumental space of tools and tool functioning, software abandons a crucial point of aesthetic purchase. The goal in my view is not to resist the notion of the tool, but to engage with issues of abstraction, disguise and efficiency, to somehow risk and re-imagine the aesthetic in an alien terrain.

There are already models from within software – works that may not be primarily aesthetically constituted but that have aesthetic, poetic implications, revealing the potential for an instrumental imaginary. Just to briefly mention three: Ivan Sutherland's 1962 *Sketchpad*, which was not only the first graphic drawing program but which, more particularly, as Allen Kay argues [17], re-invents drawing in terms of the conceptual structures of object-oriented programming; Richard Stallman's *Emacs* (1975) which is a bizarre jalopy-style software, defiantly resisting task specialisation and ordinary boundaries between work and play; and finally even the modern integrated development environment, *Eclipse* (2004), which is utterly generically conceived – which can be radically reconfigured to accomplish different programming tasks and which appears as a kind of meta-tool, a tool for creating tools. In my view these software tools are as much a source of inspiration as is work that is specifically (safely, neatly, clearly) positioned as software art.

2.3.2 Reflection

The primary motivation of software art is to encourage reflection upon underlying programmatic software processes. This notion of reflection is hardly the affirmative, grandly reconciling reflection of Kantian aesthetics – it is often, for instance, critical and deconstructive, but it nonetheless privileges code that does more than simply operate - that somehow finds the means to reflect upon its own operations. Without wishing to altogether question this orientation towards reflection, the issue, as I have suggested, is more complex. Programming entails relations that extend beyond the fantasy of visibility and self-collected reflection.

This is evident at the very outset of modern computer science in Alan Turing's model of computation [24] [10]. If Turing chooses to compute, it is because computation is mechanical, it proceeds

stupidly step by step. Computer programs may represent brilliant efforts of reflective analysis, abstraction and design, but program operations at the atomic level of specific digital events are utterly simple and unambiguous. Reflection constitutes a problem for underlying digital processes, a quandary that suspends their functioning. It is worth examining the play of reflection and machine unconsciousness in Turing's famous halting problem. Turing reflects upon the mechanism of computation, upon its procedural logic. He sets computation a reflective trap. The universal machine is programmed to halt if it is stuck and proceed if it is not. Then, in a crucial reflective step, Turing makes the computer process its own code. Now, it seems, it must halt if it proceeds and proceed if it halts. Unable to decide whether to proceed or to halt, the mechanism comes undone precisely through a motion of reflection.

The jury for the *Transmediale.01* festival suggests that the fascination of computer programming depends precisely upon code's capacity to function, the passage it makes from a reflective conceptual state to one of actual machine processing:

"Perhaps the most fascinating aspect of computing is that code – whether displayed as text or as binary numbers – can be machine executable, that an innocuous piece of writing may upset, reprogram, crash the system." [23]

Turing's example suggests that this necessitates a relation of reflection to something other than reflection – to a space of blind motion that functions only on condition that it does not reflect. Programming demands a close engagement with this other space. It opens up a vital relation to the blindness of machine processing. The aesthetics of code is as much about the unseen, the hidden and the disguised as it is about the reflective and the visible. In this context, strategies of abstraction and encapsulation are also relevant – indeed as are all of the strategies that structure programming as a work of inscribing layers and guises above an unreflective foundation.

Abstraction is not only the positive representation of something in a symbolic form; it is also indicates a motion of leaving behind. That which is abstracted no longer itself appears. It is replaced by the abstraction. While this work certainly has a reflective aspect, its consequences are to make reflection itself more difficult. The many layers of computational process work precisely to make lower layers disappear. The principle of encapsulation denotes a particular form of this disappearance in which specific internal features of an object are deliberately hidden from view in order to protect them from unwarranted interference and to enable the simplicity of a general public interface. Encapsulation works both to protect the integrity of individual objects and to enable them to be treated as simple building blocks in more complex structures. A work of hiding then is implicit within the linguistic structure of contemporary programming. Object-oriented programming involves choreographing a play of hiding and manifestation. So while software art expresses a fascination with the executable character of code, it withdraws from the thinking of this space to the extent that it insists upon a purely reflective conception of software art.

In an attempt to realize this reflective conception of software art the 2002 CODEDOC exhibition adopted the strategy of literally displaying code. The first thing the user encountered was the code and only then obtained a link to what would ordinarily be described as the visible work. While appropriate within the

context of drawing attention to the normally neglected space of programming, this can hardly serve as a general strategy. For a start, code is simply not meaningful to non-programmers. Visible code asserts that code is significant, but beyond that it serves as little more than a connotative surface – indeed it can quickly work to mystify software art, to suggest some realm of arcane, abstract power that bears little relation to actual, practically-directed programming. Even programmers have difficulty simply reading code. Even the person who actually wrote the code can have trouble making sense of it (especially after a few days or weeks away from it). Code is most legible as it is being written, especially in the alternation between writing and execution. It resists entirely contemplative visibility (the traditional form of the aesthetic). Code is engrossing within the overall event space of writing, performance and debugging. In this sense, the notion of software art can be interpreted as the artist-programmer's fantasy that coding may somehow take literal, exhibitable shape for an audience. But this is not really possible. Programming is essentially participatory rather than something to be seen (an aesthetic spectacle). In order to become visible it has to persist with abstraction. It has to hide and shape disguises. It has to render the dimension of code metaphorically apparent. Cramer and Gabriel acknowledge this point when describing the *Web Stalker* (IOD, 1997) alternative browser:

"The code of the Web Stalker may dismantle the code of the Web, but does so by formatting it into just another display, a display which just pretends to "be" the code itself." ([6] p.2)

In this sense, the notion of rendering code visible entails something other than a puritanical resistance to code's processes of disguise and layering. Revelation is itself a staging, a manifestation, a motion away from origin.

2.3.3 Expression

Software art associates the aesthetic character of code with a dimension of personal inflection. Florian Cramer and Ulrike Gabriel argue that:

"[C]oding is a highly personal activity. Code can be diaries, poetic, obscure, ironic or disruptive, defunct or impossible, it can simulate and disguise, it has rhetoric and style, it can be an attitude." ([6] p.3)

This is hardly the concept of aesthetic genius (which is actually much more ambiguous, which actually deeply problematizes issues of agency) but it places a similar emphasis on the expressive potential of code. Code that is impersonal and formulaic appears less aesthetic. In my view, however, code is inevitably formulaic. There are all kinds of standard idioms, patterns and stylistic conventions. It is less by resisting these and affirming some notion of personal, differentiated expression that code becomes aesthetic, but by pursuing the formulaic closely and intimately. Rather than asserting subjectivity, it is a matter of finding it elsewhere, of re-inscribing it at a distance. Code is only personally inflected within the texture and through the agency of impersonal formula.

There is a vital need then to consider the instrumental character of software beyond the conventional framework of Enlightenment aesthetics. As Derrida argues the tool is never a mere subservient vessel but always appears as a force that intimately affects and undermines the notion of human agency. Writing appears as an aid to human memory but actually destabilizes human memory

and renders it in other, alien terms [8]. Re-conceiving the instrumental character of the software tool depends upon considering the nature of a tool more closely, rather than turning away with a sense of traditional aesthetic disdain.

2.4 Heidegger – Technological Revealing

In his famous 1953 article, “On the Question Concerning Technology”, Heidegger begins by suggesting that “the essence of technology is nothing technological” ([13] p.287). He is determined to reinterpret technology, to discover within it another meaning. Heidegger questions the common sense view of technology as a neutral means to an end and as an expression of human agency. He describes this view as “the instrumental and anthropological definition of technology” ([13] p.288). This would seem to represent a similar rejection of human instrumental agency that we find within software art, yet the notion of the instrumental makes a strange return as the argument proceeds – a return in which the notion of a subservient means is thought apart from the necessity of original agency or determined end.

Re-examining the nature of technological making as traditionally conceived (in the Aristotelian conception of *techne* [2]), Heidegger finds that it involves a motion of “bringing-forth” that is aligned with *poesis* ([13] p.293). It also summons a more complex sense of causality which eludes the modern sense of means end rationality and engages with processes of revealing - the manifestation of truth ([13] p.294). A classic instance is evident, perhaps, in Michelangelo’s conception of uncovering figures in marble; he less makes the figure (*ex nihilo*) than releases and reveals the inherent potential of the figure from within the marble. In this sense the artist lacks absolute agency, appearing instead as a mechanism for an overall process of revealing (he is caught up in the mystery of Being).

Heidegger argues that while this model appears applicable to traditional handicraft, modern technology radically changes things. Rather than adapting to implicit nature – tending it and gently bringing it forth, modern technology exploits materials; it extracts from them and transforms them. Materials become bare functional resources that are never revealed as such but that are instead stored up, ordered and operationalised ([13] p.298). Traditional processes permit the object its distinct appearance, autonomy and finality, whereas modern modes of technological manufacturing enable no space of rest or of contemplative existence:

“Unlocking, transforming, storing, distributing, and switching about are ways of revealing. But the revealing never simply comes to an end.” ([13] p.298)

This operational system affects not only natural materials and the technological devices but also the human beings that “run” them. All elements become regulated components within an overall mechanical constellation; none of them can ever be revealed in themselves – instead they constantly point elsewhere and only gain meaning in their systematic (differential) functioning.

Surprisingly, rather than altogether rejecting this prospect of systemic displacement and human alienation, Heidegger discovers within it a sense of strange hope. This hope is linked precisely to the instrumental character of modern technology; specifically to the ambiguous relation it opens up between revealing and hiding. Rather than directly, un-problematically, displaying Being in a natural and organic fashion (as evident in the model of traditional handicraft), modern technology shapes a blindness, a layering, a

system of guises. For Heidegger this has the potential to provide access to a deeper layer of revealing – the truth, precisely, that truth can never appear as such, that it is inevitably in disguise - dissembling and adopting the form of copy, metaphor and sign. Heidegger argues that humanity “keeps watch the unconcealment – and with it, from the first, the concealment – of all coming to being on this earth” ([13] p.313). If “the essence of technology is nothing technological”, it is because it is actually about the ultimate mystery of being and revealing:

“The question concerning technology is the question concerning the constellation in which revealing and concealing, in which the coming to presence of truth comes to pass.” ([13] p.315)

If this alternative thinking of technology appears dangerous it is because it risks becoming lost in the tissue of concealment – truth is no longer co-extensive with direct, lucid appearance. It passes away from itself and beyond the control of self-collected, critical, human consciousness. The anthropocentric dream of human control and mastery is abandoned in order to conceive technology in radical instrumental terms as an opening and a displacement. Hence, for Heidegger, the importance of art as both a species of *techne* and as a means of maintaining a human, reflective element:

“[E]ssential reflection on technology and decisive confrontation with it must happen in a realm that is, on the one hand, akin to the essence of technology and, on the other, fundamentally different from it.” ([13] p.317)

However, art can only perform this task of maintaining reflection within the space of semblance and loss if it takes technology seriously, if it “does not shut its eyes to the constellation of truth concerning which we are questioning” ([13] p.317). The catch, however, is that this also necessitates a questioning of the nature of art as critique:

“Yet the more questioningly we ponder the essence of technology, the more mysterious the essence of art becomes.” ([13] p.317)

This mystery takes shape precisely as the risking of critique – art itself appears as a passage away from truth as simple revealing. This is not a consequence of its inevitable opposition to technology (the conventional romantic aesthetic attitude that contrasts irrational, sensible-material art to the rational abstraction of technology), but instead arises from a fundamental engagement with the problem of technology. The mystery of art lies in its participation within the problematic of the instrumental.

Heidegger’s conception of technology has clear relevance to the nature of software which is characterized by an enframed writing, a motion of functioning without human agency and by endless processes of structural hiding (abstraction and encapsulation). Art cannot resist these processes by simply projecting a naïve opposite. There is instead a need to insert itself within software, to partake of its processes, to follow its complex system of layering and dissembling.

2.5 Plato – Inspiration and Mimesis

Heidegger’s perspective emerges as a creative response to a specifically modern concern, yet there are also ancient models for this view. It is easy to imagine an ancient unity of *techne* and *poesis* that is split apart within modernity, yet this sense of division is also apparent within the ancient world. Plato, for instance, writing around the same time as Aristotle, is adamant

that *techne* and *poesis* are fundamentally opposed [21]. Whereas Aristotle, in his *On the Art of Poetry* [3], positions (dramatic) *poesis* as a domain of conceptually-guided skill, Plato, in his dialogue *Ion*, casts *poesis* as form of sympathetic magic, of intoxication. The discussion between Socrates and the Homeric rhapsodist Ion sets out to establish that the latter rhapsodises not through the mechanism of clear aesthetic precepts and skills but through the agency of divine inspiration:

“For all good poets, epic as well as lyric, compose their beautiful poems not by art, but because they are inspired and possessed. And as the Corybantic revelers when they dance are not in their right mind, so the lyric poets are not in their mind when they are composing their beautiful strains: but when falling under the power of music and metre they are inspired and possessed.” ([21] p.5)

For my purposes, what is interesting here is that inspiration renders the artist an instrument. They are no longer in control, they can no longer entirely reflect upon, or claim essential priority for, the processes in which they are involved. They are caught up in operations that exceed them. Plato describes inspiration in terms of the metaphor of a magnet:

“This stone not only attracts iron rings, but also imparts to them a similar power of attracting other rings; and sometimes you may see a number of pieces of iron and rings suspended from one another so as to form quite a long chain: and all of them derive their powers of suspension from the original stone. In like manner the Muse first of all inspires men herself; and from these inspired persons a chain of other persons is suspended, who take the inspiration.” ([21] p.5)

The metaphor suggests a chain of inspiration that takes shape as a set of mediated relations. The “original stone” can not itself be seen – it passes away from itself in order to manifest its attractive force. Each iron ring – Homer, the Homeric rhapsodist Ion and the audience – is linked together instrumentally as an ordered sequence and as a chain of unconscious attraction. However the chain also gives rise to apparitions, because inspiration becomes manifest through appearances, through mimetic guises. If Plato ultimately expels the poets from his ideal republic [21], it is not only because they encourage dimensions of irrational and emotional excess but because they produce beguiling appearances that are a “third remove” ([21] p.425) from truth. Genuine truth has its home in the sphere of abstract, mathematical form, whereas human beings live in the world of appearances (dark and shadowy and yet visible), and artists create appearances of appearances. The fundamental paradox is that inspiration has its basis in the revelatory experience of music and metre (which traces intimate links to the realm of ideal truth), but instead of producing truth it gives rise to falsehoods. Just like technology (conceived in Heidegger’s terms) mimetic art renders revealing as concealing.

We find then that Plato’s rejection of the mechanism of art (*techne*) only enables its more thorough grounding within art – not as conceptually guided, skill-based practice, but as the instrumental character of *techne* which here informs the nature of poetic inspiration and mimetic form; taking shape as the suspension of self-collected human agency and in systems of dissembling that chart an undecidable relation between the revealing and concealing of truth.

How can Plato’s scheme, in which the aesthetic and the instrumental discover a surprising space of association, contribute to a re-evaluation of the status of the instrumental within software art? The layers of abstraction that characterize code operations are certainly not mimetic, but they obey the fundamental form of mimesis inasmuch as they involve a motion away from self-present origin. Similarly, although Plato’s conception of poetic intoxication may seem very distant from rational software processes, the notion of involuntary *poesis* - in its automatism and blind pull - summons a sense of Turing’s concern with the universal machine’s dumb mechanical functioning. Within this context it is worth recalling that Adorno and Horkheimer conceive instrumental rationality precisely in terms of a limit point of reason in which rationality and irrationality coincide ([1] p.172). If instrumental rationality reveals an irrational dimension, it is not only in terms of the division it opens up between *episteme* (knowledge of invariable principles) and *phronesis* (morally guided practical action), but also in terms of its suspension of human agency, its orientation towards an automatism that inevitably comes to resemble intoxication.

3. CROPPER/PROPPER/GRIDDER

Overall then the genuine aesthetic potential of software lies in engaging with everything within software that seems most intrinsically inimical to the aesthetic – dimensions of instrumental function, non-reflective process and formulaic expression. Rather than struggling to find means of lifting up software to the status of art, there is a need to delve into the instrumental character of software, to genuinely engage with this space of risk and aesthetic alienation. This is what *Cropper/Propper/Gridder* attempts. The work provides an example of an effort to conceive the relation to the instrumental differently. If it does not take adequate shape as either a piece of software art or genuinely useful tool, then it is because it is concerned to explore a space of tension and awkward possibility.

3.1 An Awkward Possibility

The name itself is awkward. *Cropper/Propper/Gridder* refers to three separate pieces of software that together form an apparatus for decomposing video and playing it back in discrete, grid-based portions. When it was exhibited, however, the work had a different name. It was called *Ice Time*, which related to a specific instance of the work which focused on video sequences from the Ross Sea region in the Antarctic. This suggests another dimension of awkwardness; that of the distinction between the visible interface with its specific instances and the generic character of the work as an engine, as a mechanism of decomposition, choreography and display. Which of these demands attention? Which of these has a properly aesthetic character? Or is it both? If so, how are they to appear simultaneously? What would this mean? The work raises these kinds of awkward questions. Prior to considering its uncertain status as a piece of software art, there is a need to provide more detail about the work itself, considering the underlying concept, the technical system and the exhibition context.

3.1.1 Concept

The project had its basis in the philosopher Henri Bergson’s condemnation of the cinematographic representation of time [4]. According to Bergson, time as duration represents a qualitatively whole motion that cannot be subdivided without fundamentally altering its character:

“All is obscure, all is contradictory when we try, with states, to build up a transition.” ([4] p.313)

Film, as a technology for cutting up time into frames and reassembling it for illusory temporal display, appears as a metaphor for the modern alienation from the genuine experience of duration. In response to this, I wondered, perversely, whether time was not better experienced through a mechanism; not as a predictable linear sequence, but as a work of setting time astray, of manufacturing, emphasizing and exacerbating its obscurity. I was thinking of projectors that run too slowly, in which the individual frames are visible, in which a sense of time emerges precisely through the disengagement of actual continuous time, in which time is manifest not as a single flow but as a set of flickering instants which serve both as an alienated reminder of some other time and as an immediate, yet dislocated, perception of current duration.

This interest, this thematic space, is clearly not unique. It charts relations to long-standing aesthetic concerns within avant-garde film and video art, from the exploration of aspects of temporal sequence in Dziga Vertov’s 1929 *Man with a Movie Camera*, to Chris Marker’s concern with the invisible time of the black film leader in *Sans Soleil* (1983), to the Australian artists, Rodney Glick and Lynette Voevodin’s display of columns of hours from a single day in *24Hr Panoramas* (1999-2006). It also connects to the tradition of experimental new media which explores issues of time in terms of the re-combinatory possibilities of computation [15]. Some influential works include Joachim Sauter’s and Dirk Lusebruk’s *Invisible Shape of Things Past* (1995) which reconstitutes time slices as peculiarly non-temporal, sculptural entities, Martin Reinhart’s and Virgil Widrich’s *tx-transform* (1992-2002) which swaps the axes of temporal and spatial representation, and most relevantly Camille Utterbach *Liquid Time* (2001-2) which enables portions of the video frame to play at different speeds and in different directions. My work explores similar possibilities. It decomposes the video frame into rows and columns of independently playing image sequences - in an effort to stage both the deconstruction of ordinary time and a summoning of temporal alterity.

It is at this point that the conventional aesthetic idea necessarily engages with a technical imaginary. There is a need to consider how the various aspects of the system can be implemented. There is a need to devise systems, tools, engines. There is a temptation to disregard this as a work of subsidiary technical implementation, but for me it indicates the vital process in which the aesthetic concept takes practical and poetic shape as an instrumental apparatus.

3.1.2 Technical System

Figure 1 displays a diagram of the display system. The screen is composed of any number of squares which may or may not be arranged in a grid based manner. Each square is composed of a set of sequences of still images. Each sequence may be played independently and in various ways (in terms of speed, direction, etc.). Sequences may have associated sound files which may loop or play a specific number of times. Finally aspects of playback may be choreographed in advance or enable live interactive control.

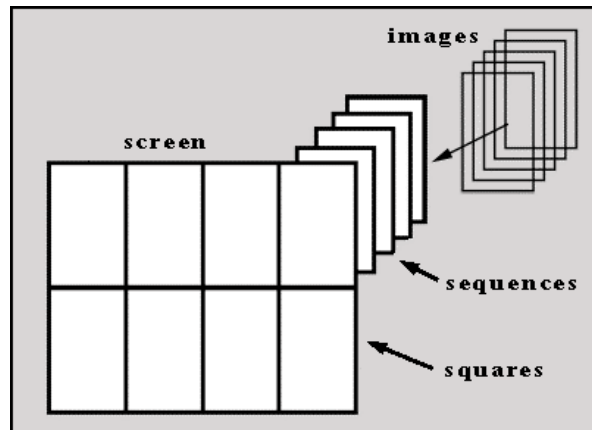


Fig. 1: Brogan Bunt, *Cropper/Propper/Gridder* design concept

The basic technical challenge involved finding means to decompose a video sequence into a set of independently playing image sections. The neatest and most logical approach was to employ a single video file and dynamically decompose and reassemble frames from the cube of spatio-temporal data. For long sequences, however, this was likely to demand retaining a very large number of frames within RAM and a constant process of multi-frame analysis to constitute any specific display frame. This is probably technically feasible but seemed beyond my means. Another obvious approach was to cut up the video in advance and play back any number of independent video streams. This proved unworkable due to the considerable overhead that each stream of video imposed on the overall system. It was not possible to play back more than a couple of video files at once. My only other option was entirely simple, even anachronistic. It involved conceiving the video sequences as game-style sprites. Video sequences were decomposed into sets of video stills and then decomposed again in to sets of cropped images. Represented as sprite arrays, these sets of cropped images could be played back in conventional sequential order, randomly or in any number of specific algorithmic ways. This was the approach I adopted and miraculously it seemed to work even for a finely articulated grid (60 or so sequences running simultaneously), but it had one major drawback. Instead of a single video file or a relatively small number of cropped video files, I had multiple directories filled with innumerable tiny image files. In this sense, it was a plainly awkward and inefficient solution. Moreover, in its literal complexity, in its fragmentation of data, it opened up the necessity for a set of specific tools to handle aspects of decomposition, choreography and display.

3.1.2.1 Cropper

Cropper is a small and unassuming utility program that handles the process of first cutting up sequences of video stills into rows and columns of cropped images and then saving them within an appropriate directory structure. It obscures the major part of its underlying functioning, merely displaying dynamic information concerning the percentage of images processed.

3.1.2.2 Propper

Propper is a much more ambitious program. The role of this tool is to produce the underlying score that choreographs aspects of display. It builds XML (Extensible Markup Language) description files that the display engine, *Gridder*, reads in order to know what media to load when and where. XML makes

dimensions of logical structure visible, legible and easily accessible (within text editors, browsers and so on), however it can be slow to prepare manually. *Propper* provides a rapid, visual means of writing these files.

3.1.2.3 *Gridder*

Gridder is the display engine. A dialogue opens requesting that the user point to a relevant project directory. *Gridder* reads the project xml description file and commences media display. Additionally, the software enables interactive control of the playback parameters of image sections. *Gridder* displays in a screen window with a standard title bar. This is intended to remind the viewer/user that the work is a piece of software, not a piece of linear, pre-constructed video.

Figure 2 displays a screen shot without the title bar to provide a sense of the visible output of a more complex piece (a 9X3 grid with multiple 'video' sequences).

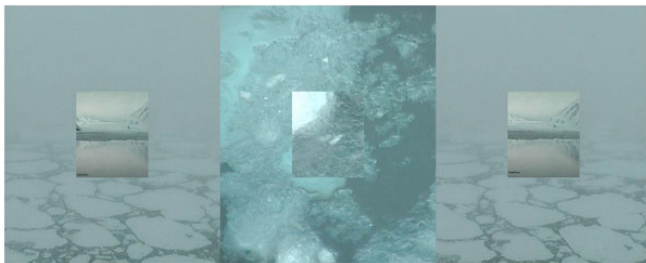


Fig. 2: Brogan Bunt, *Gridder*, *Ice Time* exhibition (2005)

3.1.3 *Exhibition*

The emphasis was upon the display of fragmented video sequences of the Antarctic. The choice of footage was deliberate. Antarctica is generally perceived as a space of pure glacial duration, yet we have recently become aware of its extreme temporal fragility; the Antarctic is entering another time – a time of division, of breaking up, even of imminent catastrophe. Without pursuing this point in an explicit political manner, the conjunction of the software apparatus (as a work of temporal decomposition and the flickering re-summoning of time) and the samples of an entirely fragile realm of duration suggests a dimension of temporal uncertainty that has political implications.

Leaving aside the specific thematic issues addressed in this exhibition, the issue that mainly concerned me was the near invisibility of the *Cropper/Cropper/Gridder* apparatus in the exhibition display. Although I had spent close to two months producing the software and perceived it as the vital context in which the aesthetic concept took generic shape, there seemed no satisfactory way of acknowledging the apparatus, of making it appear aesthetically. I was very aware that the project may appear as a work of video compositing rather than software art. It is this sense of uncertainty (and frustration) concerning how to adequately exhibit the work that has prompted this specific reflection on the instrumental.

3.2 Software Art?

The vital problem that the project raises for me is in identifying the properly aesthetic character of the work. A conventional view would distinguish between the aesthetically significant exhibited work and the aesthetically inconsequential background technical infrastructure. The contemporary notion of software art seems to provide a corrective to this view but ends up insisting upon a non-instrumental model of software as a form of abstract formal

enquiry and/or self-reflexive software critique that has the unfortunate consequence, once again, of positioning the instrumental component of *Cropper/Propper/Gridder* as a work of mere technical implementation. At the same time the software art status of the exhibited interface is questionable because it is less about reflectively revealing the dimension of code than about setting code into relation with the particularity of specific temporal samples. On what basis then do I regard the overall project as a work of software art?

The project represents a meditation on issues of the coded, discontinuous character of represented time that is conducted through the medium – the linguistic and discursive forms – of software. In this sense, it represents an example of what Mathew Fuller describes as “speculative software” ([11] p.29). Although the boundaries blur, Fuller distinguishes speculative software from “critical software” ([11] p.22) in that the former is oriented less towards deconstruction than making; it engages with “the havoc of invention” ([11] p.32). *Cropper/Cropper/Gridder* may be regarded as a speculative apparatus; it takes shape as a perverse media player, one in which the dimension of time is disarticulated and re-composed.

The work gains aesthetic coherence as an overall system that includes an element of generic operation and specific instantiation. In terms of the generic character of the work, the underlying poetic idea is realized as a linked system of functional tool-based operations which together form an abstract machine, an engine. *Cropper* represents the motion of decomposition, *Propper* the work of reassembling, and *Gridder* the rendering of an unnatural spatio-temporal logic in actual time. In its operation, the engine inevitably structures a moment of instantiation. The aesthetic dimension of the latter emerges in the friction between coded time and the particularity of sampled actual time. Without an awareness of the background software, this sense of friction is lost. The dimensions of interface and implementation are integrally aesthetically related.

The instrumental orientation represents an important aesthetic choice. The strangeness and technically anachronistic character of the project is heightened precisely by pursuing it through the agency of the instrumental, by discovering means to realize a perverse, absurd, idiosyncratic idea as efficiently as possible. Accordingly, at a stylistic level the software resists adopting a conventional aesthetic guise; it is deliberately blankly ordinary. The Java Swing style interface elements – menus, tabbed panes, hierarchical lists, radio buttons, etc. – interested me particularly in their anonymity and their embeddedness in the logic of instrumental software production and use. If the explicit conceptual theme is the alienation of time via mechanical division then the choice of a blank instrumental style works in my view to heighten the sense of alienation.

Of course, the problem is that none of this was seen by the exhibition audience. This is written then as a critique of the work’s original mode of exhibition. The work needed to demonstrate both the engine and the interface in order to properly address the conceptual theme of the coded representation of time, as well as the equally important theme of the relation between an instrumental apparatus and an aesthetic concept.

4. CONCLUSION

My overall argument is that rather than positioning the instrumental, tool-based character of software as some grim fact

that must be rigorously resisted there is vital need to work through the instrumental, to explore its possibilities. This entails a risk – the risk of facilitating software functioning, of engaging with its work of abstraction, encapsulation and disguise. It projects a space of uncertain creation that cannot altogether shake off a relation to the blindness of mechanical process - that must find means to reflect amidst a work of operational making. The clear difficulty is in finding ways to conceive a work of critical reflection within the texture of instrumental relations when the self-consciousness of critical awareness is precisely what is put at risk. In my view there is no easy solution to this dilemma. Instead there is a constant work of negotiation - of engagement and distancing with whatever it is that an instrumental device and an aesthetic work represents. This would seem to demand a re-examination of the relation between “software culturalism” and “software formalism” ([7] p.10). It may be that it is precisely at the level of form (regarded as a material discursive fact and experimental space) that the most profoundly cultural questions are raised. Of course, how these questions are to be articulated - how they are to take constitutive shape as processes, engines and interfaces - remains an open question.

6. REFERENCES

- [1] Adorno, T. & Horkheimer, M. (2000) 'The Concept of Enlightenment' in O'Connor, B. *The Adorno Reader*, London, Blackwell Publishing, pp. 156-173. First published 1944.
- [2] Aristotle (350 BC c.) *Nicomachean Ethics*, <http://classics.mit.edu/Aristotle/nicomachean.html>, accessed 4th January 2007.
- [3] Aristotle (1965) *On the Art of Poetry* in Dorsch, T.S. (ed.) *Classical Literary Criticism*, Great Britain, Penguin Books, pp. 31-75, First published: undated.
- [4] Bergson, H. (1911). *Creative Evolution*. New York, Henry Holt and Company.
- [5] Burger, P. (1984). *Theory of the Avant-Garde*. Minneapolis, Minnesota, University of Minnesota Press
- [6] Cramer, F. and Gabriel, U. (2001). 'Software Art'. Transmediale.01 Arts Festival, Berlin.
- [7] Cramer, F. (2002). 'Concepts, Notations, Software, Art'. Seminar for Allgemeine und Vergleichende Literaturwissenschaft.
- [8] Derrida, J. (1976). *Of Grammatology*. Baltimore & London, The Johns Hopkins University Press
- [9] Eagleton, T. (1990). *The Ideology of the Aesthetic*. Oxford and Cambridge, Massachusetts, Basil Blackwell.
- [10] Feynman, R. P. (1996). *Lectures on Computation*. Edited by Hey, A. J. G. & Allen, R. W. London, Penguin Books.
- [11] Fuller, M. (2003). *Behind the Blip: Essays on the Culture of Software*. Brooklyn, New York, Autonomedia.
- [12] Gere, C. (2002). *Digital Culture*. London, Reaktion Books.
- [13] Heidegger, M. (1978) 'The Question Concerning Technology' in Krell, D.F. (ed.) *Martin Heidegger: Basic Writings*, London, Thames & Hudson, pp. 284-317. First published 1953.
- [14] Huyssen, A. (1986) *After the Great Divide: Modernism, Mass Culture, Postmodernism*. Bloomington and Indianapolis, Indiana University Press
- [15] Jaschko, S. (2003) 'Space-Time Correlations Focused in Film Objects and Interactive Video' in Shaw, J. & Weibel, P. (eds.) *Future Cinema: The Cinematic Imaginary After Film*. Cambridge, Massachusetts, ZKM and The MIT Press.
- [16] Kant, I. (1980). *Kant*. 'Great Books of the Western World' Series, edited by Adler, J. and Brockway, W. Chicago, University of Chicago Press (Encyclopedia Britannica, Inc.). First published 1790.
- [17] Kay, A. (2003). from video 'The History of the Personal Workstation' (1986), in Wardrip-Fruin, N. and Montfort, N. (eds). *New Media Reader*. Cambridge, Massachusetts, The MIT Press.
- [18] Knuth, D. (1973 -1998). *The Art of Computer Programming*. Reading, Massachusetts, Addison-Wesley.
- [19] Manovich, L. (2005). 'Remixability & Modularity.' <http://www.manovich.net/>
- [20] Plato (380 BC c.) *Ion*, <http://classics.mit.edu/Plato/ion.html>, accessed 6th January 2007.
- [21] Plato (1955) *The Republic*, London, Penguin Books, translated by Lee, D.P. First published: 360 BC c.
- [22] Read_Me 1.2 festival jury (2002). 'Read_Me 1.2 Software Art/Software Art Games festival jury statement'. http://www.macros-center.ru/read_me/adden.htm.
- [23] Transmediale.01 Media Arts festival jury (2001). 'Transmediale.01 Media Arts festival jury statement.' Retrieved 15 December, 2006, from http://transmediale.de/01/de/s_juryStatement.htm.
- [24] Turing, A. (1995). 'On Computable Numbers, with an Application to the Entscheidungsproblem' (extract), in Norman, M. (ed). *From Gutenberg to the Internet: A Sourcebook on the History of Information Technology*. Novato, California, historyofscience.com. First published in 1936.
- [25] Weber, M. (1946). *Max Weber: Essays in Sociology*. Edited by Gerth, H. H. & Mills, C. W. New York, Oxford University Press.
- [26] Wikipedia, entry on Constructivism, Retrieved December 23, 2006, [http://en.wikipedia.org/wiki/Constructivism_\(art\)](http://en.wikipedia.org/wiki/Constructivism_(art)).

7. WORKS MENTIONED

- [1] Duchamp, M. (1915-23). *Large Glass - The Bride Stripped Bare by Her Bachelors, Even*.
- [2] Eclipse Foundation (2004). *Eclipse 3.0*. [computer program]
- [3] Glick, R. and Voevodin, L. (1999-2006). *24Hr Panoramas*. [video art project]
- [4] IOD (1997). *Web Stalker*. London. [software art work]
- [5] Marker, C. (1983). *Sans Soleil*. [film]
- [6] Reinhart, M. and Widrich, V. (1992-2002). tx-transform, in Shaw, J. & Weibel, P. (eds.) (2003) *Future Cinema: The Cinematic Imaginary After Film*. Cambridge, Massachusetts, ZKM and The MIT Press, pp. 442-443. [new media art work]
- [7] Sauter, J. and Lusebruk, D. 'Invisible Shape of Things Past' in Shaw, J. & Weibel, P. (eds.) (2003) *Future Cinema: The Cinematic Imaginary After Film*. Cambridge, Massachusetts, ZKM and The MIT Press, pp. 436-439 [new media art work]
- [8] Stallman, R. (1975). *Emacs*. [computer program]
- [9] Sutherland, I. (1962). *Sketchpad*. [computer program]
- [10] Tatlin, V. (1920). *Monument to the Third International*.
- [11] Utterbach (2001-2). *Liquid Time*. in Shaw, J. & Weibel, P. (eds.) (2003) *Future Cinema: The Cinematic Imaginary After Film*. Cambridge, Massachusetts, ZKM and The MIT Press, p.434 [new media art work]
- [12] Vertov, D. (1929). *Man with a Movie Camera*. [film]
- [13] Ward, A. *Signwave Auto-Illustrator*. [software art work]

