Super-Abstract: Software Art and a Redefinition of Abstraction

Brad Borevitz

June 5, 2004

http://www.onetwothree.net
brad at onetwothree.net

Super-Abstract: Software Art and a Redefinition of Abstraction

By Brad Borevitz

The word abstract denotes, most literally, separateness, a meaning directly correlative to its Latin root *abstractus*, drawn from, separated. Abstract art might be abstract in that it contemplates an art object or an art practice as itself, separate from an instrumentality that would situate art in the context of a representational imperative. In that sense, abstract art is unencumbered by a relationship to a pre-existing exterior world. At the same time, abstract art is abstract in that it becomes a meditation on the properties of art or media considered separately from each other and potentially from histories, practices, traditions or representations. This is at least the idealized and depoliticized notion of abstraction that we receive from Greenberg's interpretation of modernist practice, which has become definitive of the modern. While perhaps an apt interpretation of some variants of abstract practice including constructivist, expressionist and minimalist productions, it cannot necessarily account for the perspectival experiments of cubism or Kandinsky's theoretical insistence on a language of abstract form.

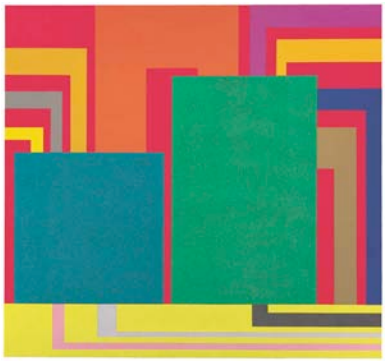
In the domain of computation, abstraction is less controversial. The ever-increasing degrees of generalization that are achieved through strategies like object-oriented programming (OOP) organize the growing complexities of computational possibility, and are built on the abstract principals of parameterization and data typing that define the programmatic.

For art, however, abstraction has been at the center of a contentious and unending debate about the definition of the modern that stretches over the last hundred years. In every era, the tensions between abstract and representational strategies took on the political entailments of the historical context, charging what might appear as the merely stylistic affinities of artists with particular significance in regard to the character of the political climate, the nature of perception, the role of art and the artist, and the limits of human possibility.

In a postmodern era, where a junk heap of past styles seem emptied of particular significance and always available for re-appropriation, the implications of pursuing a strategy of abstraction are less clear. An inheritance of suspicions inevitably haunts a modality with a history, so contemporary abstractionists labor under the specters of derivativeness, exhaustion and irrelevance. To accept uncritically an essentially modernist method, is to embrace a discredited faith in progress, humanism and capital. Still, artists continue to produce abstract work and critics manage to assign contemporary relevance to it.

The hard-edged geometries of Peter Halley's work, dubbed neo-geometric conceptualism, for example, are read as metaphors for an atomized but networked social landscape. They refer in their visual language to both the layout of the integrated circuit and the art historical precedents of geometric abstraction from Mondrian to minimalism. The re-imagining of the square as the carceral cell parodies the high-modernist valuation of geometry as the apotheosis of formalist purity and sufficiency. The cell, within the

contemporary imaginary, is a partial object which desires promiscuous connection within a network exactly because it is incomplete in itself.



Peter Halley, *Objective,* 2000, Acrylic, Day-Glo, pearlescent and metallic acrylic and Roll-a-Tex on canvas 73 x 78 in.

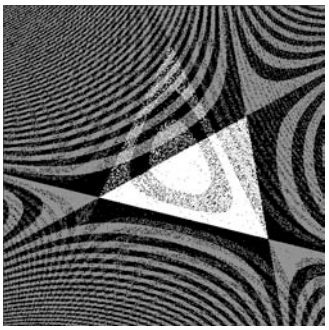
As the field of software art is gradually defined through exhibitions and conferences, the identification of a strain of work with a tendency towards abstraction is undeniable and similarly requires an explanation that makes sense of its relation to both art history and the current moment. The 2003 Abstraction Now exhibition hosted by Vienna's Künstlerhaus explicitly identified a contemporary trend in abstraction that spans traditional and digital media and is linked to constructivist and minimalist precedents. As in other media, abstraction in software art remains one tendency among many. Judging from recent prize winners in both technically oriented shows like the Transmediale or Ars Electronica, and mainstream venues like the Whitney Biennial, the trend is to valorize digital works which focus on social, political and conceptual issues as opposed to formal ones, and which use visual strategies other than abstraction. That said, the exhibition spaces of the 2003 Ars Electronica, which took "Code" as its theme, was full of computational abstractions, from the work of pioneering algorist Roman Verotsko to that of MIT Media Lab alumni Casey Reas and Golin Levin. Half of the works included in the Whitney's 2002 CODeDOC show were abstract, as were half of those in the 2003 CODeDOC at Ars Electronica.

The persistence of an abstract software art practice, even as it may be shunted to the periphery by the mechanisms of institutionalization, indicates that it is a trend deserving of critical attention. The frequent closeness of the term "code" to instances of abstract practice suggest a link. The computer, in its idleness, flourishes into the abstract: that interstitial moment called the "screen saver." As software contemplates itself, it tends towards the abstract. The deinstrumentalized machine conducts experiments exploring its own limits and possibilities. The mechanisms of computation are repurposed to make work rather than do it.

The CODeDOC show, focused on the relationship between software's code and software's effects, is a promising place to look to begin developing techniques for reading computational abstraction. Several pieces from the show are representative of the strain of abstraction in software art that visually resembles modernist traditions of abstraction, yet has a basis in the logic of its own particular modes of production. These pieces are written in the OOP language Java, which introduces its own characteristic strategies of abstraction: being based on the concept of an object, an abstract data structure operated on by methods and organized in a hierarchy of classes and instances. The works address the rather abstract commission of CODeDOC which states that, "The code should move and connect three points in space. [This could obviously be interpreted in a visual or more abstract way]."[1] The relationship between the code and the work of digital art is confused here since the instructions refer to what the code should do and then only

parenthetically to its potential visual presentation. On the other hand, this split is one of the basic assumptions of the show, as demonstrated in the language of the commission, which also lays out CODeDOC's aesthetic concerns:

> Digital Art is not a purely visual medium but always consist of a mostly invisible back end–source code or scripting languages–and a front end, the results created by "computer language." … … the aesthetics of artists who write their own source code manifest themselves both in the code itself and its visual results. … … "CODeDOC" takes a 'reverse' look at artists' projects by focusing on and comparing the back end of the code.



Martin Wattenberg, *Connect the Dots,* 2002, Java

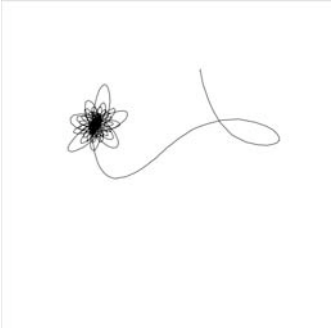
Wattenberg's code for *Connect the Dots* is compact and makes little use of OOP techniques, emphasizing instead a procedural simplicity. The logic of the work is easily read from the source. A loop iterates over a stochastic algorithm 5000 times. The value of a mathematical function at each random point in the image is evaluated and compared to determine what color it will be. The mathematical function relates the given random point to the three others, which are changed based on the user's mouse movements. The behavior of the mathematical function is difficult for the inexpert to predict. The changes will fade in gradually and randomly, point-by-point each time the mouse moves and changes the main parameters.

Looking at the running applet, a pattern emerges in random dots. A white triangle is surrounded by alternating grey and black forms. Clicking on the screen causes a new pattern to fade in with the same kind of random dot fill. Clicking and dragging repeatedly, one begins to recognize the structures of the visual system that the piece creates. There are clearly many variations but they all conform to a pattern: a central triangle is inscribed by black and white curved bands which hug the triangle's borders; grey and black contour lines radiate out from the central triangle. The fades between patterns create complicated moiré effects where the new and old patterns overlap. Playing with the mouse creates various textural effects and ghostly superimpositions. The user has a sense of accomplishment in having figured out how the thing works. There is a pleasure in the way one can predict the effects. But when the surprise of its variations are exhausted, the user loses interest in play.

The path of a user's experience follows a narrative trajectory: from puzzlement, to discovery, to understanding, and finally exhaustion. The pleasures of this passage involve the sensual, empathetic experience of the algorithms of the software. The phenomenological description of the applet is not that different in character from the conjectures based on a reading of the code, but it is embodied and engaged. Moreover, it is not dependent on a reading of the code. There is a way in which the basic

programmatic logic of the work is as clearly evident in its visual presentation as it is in the code itself.

Reading Wattenberg's source, for the programmer, and perhaps for the uninitiated as well, there is a certain awe of the way that the code actually produces the visual display. There is also an appreciation of its economy and elegance. Without being compelled by CODeDOC, it is unlikely that one would take the time to carefully parse the code before looking at the piece. Most people would be interested in the code afterwards–if at all–in order to see how it achieves its effects, or as a confirmation of one's intuitions of the program's logic. Instead, the user is invited to experience and explore the piece's logic by entering the system as an index of its parameterizations. The outcome of a user's manipulations are manifest over time and dot by dot in such a way that the user comes to experience the time of the algorithm: iterative time, discrete time. Commands are executed sequentially and their repetitions accumulate effects, which develop into its outcome. The user is inserted into to that loop and experiences the piece in that time.



Scott Snibbe, *Tripolar,* 2002, Java

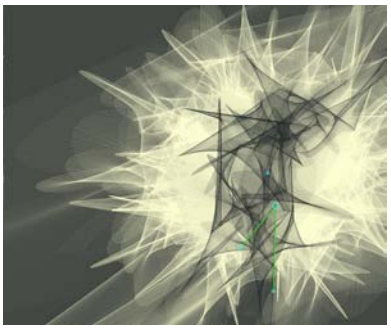
Snibbe's *Tripolar* is based on a simulation of a classic problem in chaos theory where a metal pendulum is released over a set of magnets. Its effects depend on the meta-chaos of variables he has fixed in the code and a mechanism for interpolating between pixels to show the details of the chaotic system. His code makes use of the OOP structure of java more extensively than Wattenberg's. He develops a complete set of methods and properties for the *Tripolar* class. Some handle the mouse movements and some create the elements of the simulation. Another function controls the simulation itself.

The program takes as input an x/y pair and calculates and plots for that point a trajectory over time. A function iterates over a "while" loop so that calculations continue as long as the velocity parameter remains over a threshold and the iterations do not exceed 10,000. Each subsequent point on the path is calculated as the sum of the last point and a series of separately computed forces: gravity, magnetic attraction, friction and inertia. The tendency of each force and its calculations are simple to understand individually, but their complex interactions over time are unpredictable by definition**.**

Again, parsing the code deepens the viewer's understanding of the software system, but it is not a prerequisite to apprehending its logic in some fashion. Time, effort and attention offer similar insights into the logic of the work (and external documentation might more efficiently supplement understanding as it often does conveniently for art in other media).

Each click brings a rendering in tiny line segments of a Lissajou daisy or eccentric spiral at the end of a curvaceous stalk. The terminus approaches the cursor in a few steps. The dense portions of the line coalesce around one of three points. The stalk is often short when one clicks near any of them. Sometimes the stalk wanders around these points of

attraction. If the user holds down the mouse or drags, the line animates and approaches the cursor. The rest of the line dances between the three poles and the tight knots of its end flutter about and tighten and loosen like springs. In some spots the sinusoidal curve is sent careening and blinking among the poles. In other regions, the line's terminus bounces gently, while the stalk gracefully bends this way and that. The system is exhilaratingly wild in its variations, but it is also peculiarly constrained. Moves are matched and answered by a gesture that is both surprising and fitting–unpredictable and yet structurally and gesturally consistent with every other signature curve produced by the applet.



Mark Napier, *3 Dots,* 2002, Java

As in Wattenberg's piece, Snibbe's *Tripolar* reveals its logic to the user through the presentation of successive instantiations of its system. The user is inserted into that system through mouse play. In contrast to *Connect the Dots*, in *Tripolar* the system has no visual memory of itself–it leaves no traces of its history to merge or compare with its present. *Tripolar* rather demands that the user's memory index its history to discover its systematicity. Other strategies of reception are also possible. In Napier's *3 Dots*, for example, the user is presented not with a series of instantiations from which they may infer the rules of the system, but rather a continuous development of a single instance from which the user can derive the logic of the system by thoughtful observation over time. The basic mechanism of viewership, however, is not really different. The fascination of these works lies in the sensual apprehension of their procedural logics.

The front-end/back-end model taken as paradigmatic of digital art by CODeDOC is what makes the show's reversal, its focus on the code, make sense. Software Art is created in the act of coding–by recording in a formal computer language the instructions for executing some set of operations; but the execution is separate and deferred. It makes sense to compare this with Sol LeWitt's oft quoted description of his conceptual practice:

> "In conceptual art the idea or concept is the most important aspect of the work. When an artist uses a conceptual form of art, it means that all of the planning and decisions are made beforehand and the execution is a perfunctory affair. The idea becomes a machine that makes the art."[2]

Each time software is run, it creates an instance of its doing. Each instance, though not necessarily identical, is true to the instructions of its software. Its variations are not incidental–as might be variations in the execution of LeWitt's instructions, as the result of human factors: mistakes, differences in style or interpretation–but are implicitly encoded by the software. Software produces contingent, emergent, and randomized effects programmatically, in response to interactivity, or to the sensed or networked environment. An instance of software's running might be likened to the execution of conceptual instructions. In either case though, an experiential encounter with the art as

product should occur. For software, the experience of the code's productivity is essentially tied to the variability and the unpredictability of its product. Iterative processes have the ability to produce startling behavior as is evidenced in the vagaries of artificial life and fractal geometries. The running of code reveals not just the explicit logic of the program, but the latent and emergent logics of iteration and interaction as well.

Code extends beyond the intent of its author and is mediated in relation to the reception of the user. Code when running is in a continual state of becoming, in that the values of its parameters are changed as a result of its execution, creating a multitude of possible outcomes. This is why the authors of "The Aesthetics of Generative Code" assert that, "the aesthetic value of the code lies in its execution, not simply its written form."[3] Code has a writerly aspect which tempts us to see it as the art itself: "Code is intricately crafted, and expressed in multitudinous and idiosyncratic ways."[4] However, it is necessary to experience the code execute to fully grasp its logic, its possibilities, and its effects.

Software is an abstraction that is experienced through its instantiations during runtime or as a result of them. And software always exists as part of a system of dependent relations organized as a continuum of layers of abstract objects that extend through to the user at an interface and down to the binary code on the hardware. This stack may be imagined to have a terminus at the front for the user; but, the stack is a stack of interfaces, and each layer is transmitting and translating from and to contiguous layers. Given a programmer-user at the console, it might just as easily feedback into itself as end.

What is at stake in emphasizing the position of the code in relation to the interface is really human agency. Software problematizes authorial intent in a new way. Putting a human agent back in front of the machine is to recuperate symbolically the control that an author relinquishes to the code or to the machine. In reading the code for clues to the author's intention, we repeat this exercise. But code is a message for the machine, not for a human. Code animates the machine. Code is read by the machine operationally, and not culturally. Kittler draws our attention to the way that the mass migration of knowledge (as data) and control (as procedure) makes vulnerable a central tenant of modernity, the understanding of rational thought as the determinant attribute of human subjectivity.[5]

Computation, which begins as a machinic reduction of human intelligence, continues to trouble the relationship between human and machine by presenting the procreativity of its logics. The modernist program in art delivered abstractions as the pinnacle of rational achievement, but as the clean geometries of this era issue from the monitor, praise for the rational diminishes. Yet, the empathy present in the situation of reception that helped give the modern its human character, is also a part of what attains in the perception of programmatic logic within computational abstractions.

For programming, abstraction means parameterization: making something a function of something else. Parameterization is rooted within and dependent on a larger abstract process, namely modeling, the process of producing a schematic description of a system, real or imagined. The heritage of Norbert Wiener's cybernetics lends its characteristics to the contemporary practice, so distinctions between human and machine become

irrelevant. Cybernetics is an epistemological scheme that defines its world in terms of abstract systems of control and communication. In programming practice, this type of systems thinking finds it apotheosis in the OOP technique which has come to dominate the field and is abstract in that its structure is governed by the practice of modeling.

Programming exists under a regime defined by multiple levels of abstraction: modeling, object-orientation, and parameterization. However, it is not necessarily abstract in the sense of being divorced from the consideration of particular entities. In most contexts, programming is subservient to an instrumentalism that refers it to real world situations and demands from its systems solutions to business, scientific or military problems. In this way, programming may be considered representational; in the case of its use for simulation, it is perhaps literally so. And yet, simulations such as Snibbe's may refer only generally to real-world physics, since they borrow the formulations of Newtonian rule merely to abstract them and play them according to the demands of an aesthetic production rather than in the service of modeling an actual or potential physical systemThis split is not a perfect reflection of the historical art world divide between representational strategies and abstract ones; the line of distinction is drawn, rather, between an instrumentalizing use of the technology and a deinstrumentalizing one. To set software free to play, opens it to the possibilities of critique and art. Abstraction, though certainly not the only path in this direction is a sure and direct one. Greenberg insists that abstraction is not the negation of representation, rather, abstract painting comes to have an immediacy, or what he terms an at-onceness: "Painting and sculpture can become more completely nothing but what they do; like functional architecture and the machine, they *look* what they *do*."[6]

For Greenberg, the embrace of abstraction is an outgrowth of a Kantian immanent critique and a progress towards a kind of truth: "The essence of modernism lies," he writes, "in the use of the characteristic methods of a discipline to criticize the discipline itself–not in order to subvert it, but to entrench it more firmly in the area of its competence... "[7] It is not necessary to associate abstraction with a suspect truth claim when there is value in the act of divorcing software from its habitual uses and examining its character separate from the instrumental realm.

There is a risk, however, in aestheticising computation, which should be obvious given the historical lessons that tie futurist enthusiasm for a machine aesthetic to fascist politics. It is far too easy to slip from what appears to be a critical exploration of the aesthetic possibilities of computation to the capitulation to, if not a celebration of, the mechanisms of domination practiced by global capital: the massive and rapid transfer and manipulation of data as capital and capital as data by digital means. In the end, the risk is unavoidable since a reflection on software is crucial exactly to the degree it serves as a tool for domination. Software must have its politics, its aesthetics, its poetics, and its criticism.

If there is a chance that software will contribute significantly to a new politically relevant aesthetics, it lies in the way software shows us a way out of order, in and through order. It engages the tensions between possibility and constraint. Software gives us not objects, but instances–occasions for experience. We see our own embeddedness in networks of

abstraction, structuration and system making, and in seeing, find ways of inhabiting this situation of constraint as if it were possibility. Software can create systems of production that present us with the generation of endless variation within programmatic limitations. When freed from its instrumentalist telos, it is possible for software to exist solely on its own terms: it stages its own abstraction and serves nothing save it's own play, display, and critique, that of abstraction itself. If it is possible for software to exist solely on its own terms then it may become Super-Abstract.

---

[1] *CODeDOC*, http://www.whitney.org/artport/commissions/CODeDOC/CODeDOC.html
[2] LeWitt, Sol, "Paragraphs on Conceptual Art," in Alberro, et al., eds., *Conceptual Art: A Critical Anthology,* Cambridge, MIT Press, 1999, pg. 12
[3] Geoff Cox, Alex McLean, Adrian Ward, "The Aesthetics of Generative Code", http://www.generative.net/papers/aesthetics/
[4] Ibid.
[5] Kittler, Friedrich, "On the Implementation of Knowledge-Toward a Theory of Hardware," in *Nettime*, February 6, 1999, http://www.nettime.org/nettime.w3archive/199902/msg00038.html
[6] Greenberg, Clement, "Towards a Newer Laocoon," *The Collected Essays and Criticism*, Volume 1, Chicago and London, The University of Chicago Press, 1993, pg. 34
[7] Greenberg, Clement, "Modernist Painting," *The Collected Essays and Criticism*, Volume 4, Chicago and London, The University of Chicago Press, 85-6.