The idea becomes a machine that makes the art – Sol LeWitt[1]
A smart artist makes the machine do all the work – Cornelia Solfrank[2]

then its characteristics must adapt to accommodate this subtle but seismic shift in media.

Some of the most renowned online artists seem to have taken to heart Cornelia Solfrank's dictum that a smart artist makes the machine do all the work. Why make art when you can have a machine do it for you? Of course, to trust a machine to make art is already an enormous departure from the direct gesture of hand-brushing oil paint on canvas. In the case of Internet art, the remove is even greater, since the machine in question is not the artist's but one owned by a viewer who might be halfway around the world. To get someone else's PC to draw lines, flash colors, or spin words across the screen, Internet artists rely on instructions which can be stored on a disk or squeezed through a modem line. These instructions take the form of code.

Why would artists forego the directness of squeezing out tubes of cadmium red or chiseling away chunks of marble in favor of disembodied commands? For many practitioners, code is not simply a means to an end; on the contrary, they revel in the intricacies of **document.write**; they chisel lines of **Perl** or **Java** instead of marble, creating elegant solutions to artistic problems. Code is their muse.

But professional computer programmers also pride themselves on elegant solutions, even though the solutions in question may be to mathematical conundrums or e-commerce applications. If programming is an art, is any programmer with high standards an artist? No. As we shall see, software artists misuse code in a way that employs craft for social provocation or stands it on its head. As hinted in the introduction, this perverse practice — reminiscent of the immune system's polymorphous antibody production — lends art a quirky and prophetic vision that is unlikely to emerge from a purely utilitarian approach.

[[subA1]] How Can Art Compete with the Mandelbrot Set?

Like many self-proclaimed examples of software art, Every Icon embodies the **procedural**[15] aesthetic Judson Rosebush espoused in 1989. In Rosebush's manifesto, the goal of computer art is to create the greatest richness and meaning from the fewest lines of code. This criterion fits Simon's work but also resonates with such aesthetic traditions as the 19th-century poet Samuel Taylor Coleridge's definition of beauty as the marriage of 'multeity and unity'. But if code art's yardstick for aesthetic success is 'the biggest bang for the buck', how do we judge the merit of Proceduralist art against such remarkable achievements of the scientific community as the Mandelbrot Set, whose single-line formula generates an astounding wealth of complex iterable shapes? One clue lies in the irony of Simon's failed promise to deliver every conceivable image, which provokes the question of whether software must be devious to qualify as art.

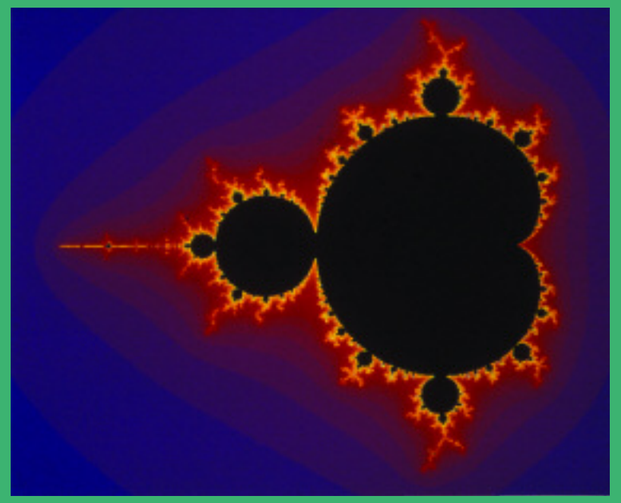
As we'll see, the answer helps resolve a debate that has raged in the past decade among creative coders who champion formal, as opposed to social, values. The invention of software in the 20th century produced a new medium for artistic expression. Like literature or painting, software can be appreciated from the inside out (basing an interpretation on a program's syntax, or internal architecture) or from the outside in (reading the function of a software application against a broader social context). The key to understanding the connection between these two readings will turn out to be a feature that sets software apart from these earlier media: its capacity for utility. Performative speech is rare outside a courtroom, but almost all computer programs are performative in some way. If our definition of art is to adapt to a 21st-century context,

[[prompt head]]
Benoit Mandelbrot, Mandelbrot Set
[[subAprompt]]

TO SEE THE WORLD IN A GRAIN OF SAND

It is not much of an exaggeration to say that Benoit Mandelbrot, creator of the Mandelbrot Set, devised a universe from a line of code. Mandelbrot was one of the first mathematicians to recognize the power of computers to generate forms that had never before been seen or described. That power is based on iteration, a method of solving an equation by repeatedly adding together increments toward the answer until the sum is close enough to suggest the correct result.

Interestingly, many equations that require iteration are extremely simple. The formula Mandelbrot focused on, for example, consisted of a mere six characters: $z = z^2 + c$. It turns out[14] that this formula yields two kinds of results, **convergent** and **nonconvergent**, depending on which number Mandelbrot chose to 'seed' his equation. More surprising is the fact that it proved impossible for Mandelbrot to predict in advance which seed numbers would produce which of the two results.



Plugging in numbers by hand was tedious, but the advent of computers made such calculations simply a matter of waiting for the dumb machine to spit out its results. Even better, computer **plotter**s made it easy to see the results mapped out on paper — by assigning nonconvergent seeds a white **pixel** and convergent seeds a black one, for example. Allowing z to be a special kind of two-valued number, called a complex number, permitted the result to be mapped out on a plane. When Mandelbrot and his colleagues saw the result of this **recursive** formula in their plotter output, they were astounded at the complexity of the result: a fantastically intricate 'bug' bristling with innumerable tendrils and appendages, any detail of which seemed as complex as the original gnarled shape.

Scientists have applied Mandelbrot's intricate geometries, known as **fractal**s, to systems in a wide range of scales and contexts, from the cycle of booms and busts in national economies and antelope populations to the geography of coastlines and the frequencies of dripping faucets. What unites all of these natural phenomena is nonlinear feedback: a force of nature or culture that amplifies small effects or diminishes large ones.

Feedback leads to a strange mix of the periodic and the unpredictable, and it is perhaps because of this familiar tension that the Mandelbrot Set retains interest as a mathematical and visual construct apart from its real-world applications. While fractals are by no means native to the Internet, they have inspired enthusiasts to create numerous resources for exploring their infinite diversity, from posters to screensavers to online viewers such as the *Fractal Microscope* (www.shodor.org/master/fractal/software/mandy/index.html).

When college students pin up posters of the Mandelbrot Set next to their posters of Monet's water lilies, are they correct in placing these disparate forms of cultural expression in the same category?

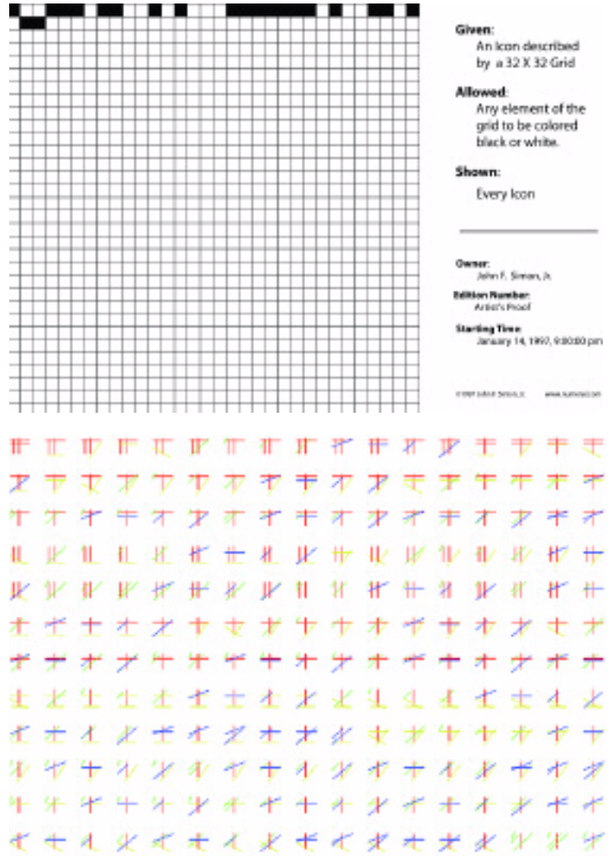
[[counterprompt head]]
John F. Simon, Jr., Every Icon
[[subAcounterprompt]]

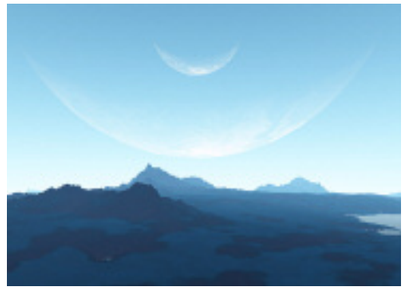
BINARY CODE UNLEASHED

Like the Mandelbrot Set, John F. Simon Jr.'s *Every Icon* (www.numeral.com/eicon) reveals the power of computers to create a world from a minuscule amount of code. But Simon's world deliberately promises more and delivers less than its fractal counterpart. His description of his program is simplicity itself:

> [[list]] Given: A 32 x 32 Grid
> Allowed: Any element of the grid to be black or white
> Shown: Every Icon

The word *shown* is somewhat misleading. Once triggered by the user, Simon's **applet** will in good faith begin to display every possible combination of black and white elements; yet even at a typical desktop computer's rate of a hundred new icons per second, it would take over 10298 years to draw every icon. Like Mandelbrot's creepy-crawly apparition, Simon's library of images requires a computer to generate its diversity. However, if Mandelbrot's order steadily emerges from his fractals' relentless recursion, in Simon's case the moments of order are few and far between; even on a supercomputer of unimaginable speed, recognizable images would melt in and out of focus, meandering across the space of all possible images rather than gradually converging on a definitive shape. *Every Icon* holds out the potential of displaying a meaningfulimage — indeed, *every* meaningful image — but in practical terms the user is likely to be exhausted long before the icons are. In fact, Simon estimates that the first recognizable image wouldn't appear for several hundred trillion years. (What would it be? A 'no smoking' icon? A **bitmap**ped *Mona Lisa*?) The result is an apt emblem for the Web, where needles await in an electronic haystack of global proportions.



[[subB1]] **Proceduralism**

Although most of Mandelbrot's disciples publish in technical journals devoted to the mathematics or computer-graphics communities, one researcher who has candidly championed the use of fractal algorithms in art-making is Ken Musgrave, who for six years worked with Mandelbrot in Yale University's computer-science department. While Musgrave's fractal **algorithms** are cutting-edge computer graphics — his special effects enhanced the films Titanic and Apollo 11 — he uses them to generate that most ancient of artistic genres, the landscape. Although landscapes have hitherto been the province of painters trained in two-point perspective, Musgrave has cited Mandelbrot to underscore the inadequacy of Euclidean geometry for describing nature: 'Clouds are not spheres, mountains are not cones, coastlines are not circles.…'[16] What shapes can accommodate Nature's wispy, rugged, meandering profile? Fractals, of course, because they marry repetition and randomness. Musgrave writes recursive algorithms which perform the mathematical equivalent of Genesis, building mountains crag by crag and trees limb by limb, tracing the paths of virtual light rays to cast shadows of virtual peaks and to paint reflected sky in virtual water.

While Musgrave's pictorial process is highly original, the subject matter and style of the images are not. The photorealistic landscapes that result from his application of fractal algorithms to landscape generation — windswept deserts dominated by spiky mountain ranges, cratered moons reflected in alien lakes — resemble the clichéd cover art of science-fiction paperbacks more than contemporary abstract or representational painting. Not surprisingly, Musgrave justifies his aesthetic not by positing any relevance to contemporary aesthetics but by invoking Rosebush's notion of Proceduralism to explain the algorithmic purity behind his working method:

> [[extract]] If I dislike the resulting hue in a particular highlight (a local effect) I may change the color of the light source accordingly, but this changes tones everywhere that light falls in the scene. Similarly, if I dislike the shape or location of a given wave in the water or mountain peak in the terrain, I may change it, but this change will also affect all other waves or peaks and valleys. The randomness at the heart of the fractal models I use grants both enormous flexibility and expressive power, but it also entails complete abdication of control over specific details in relation to their global context.[17]

Despite this deterministic process, Musgrave argues vehemently that the computer 'is in no way the creator, the author of the product',[18] emphasizing instead the role of


intuition and serendipity in his process. Falling back on mathematical jargon to describe his art, Musgrave searches '*n-space for local* **maxima** *of an aesthetic* **gradient**…the task of the artist then is first to create these *n* **parameters** (*n* being usually around two to five hundred in my own images)…then to 'tweak' the values of these parameters to obtain a satisfactory result or image'.[19] Tellingly, Musgrave spends a lot of time defining the parameter space he manipulates but very little time explaining the aesthetic gradient on which it acts — except to say that it's dependent on the artist's subjectivity and mood.

Except for a passing mention of Sol LeWitt, the only explicit aesthetic yardstick Musgrave invokes is 19th-century landscape painting, and he is frank about fractal geometry's limited success according to this benchmark: 'The fact is, the computer artwork has not yet been produced which could stand a side by side comparison with, say, a great van Gogh painting. My own best image would pale, stood beside a Bierstadt.'[20] Rather than a difference of tone or aim, however, Musgrave explains this lack as a deficiency of the output medium, citing oil painting's continuously gradated surface, abundance of perceptual information, and variety of scales (from small-scale brushwork to large-scale composition).

Perhaps Musgrave isn't nervous about the relevance of his artistic product because he believes the integrity of the process — i.e. the code — will vindicate his work. He cites Rosebush's claim that Proceduralist computer art 'will increasingly be appreciated as a major art movement by this and future generations', adding, 'If Mr. Rosebush and I are correct, we may be witnessing one of the truly definitive events in the history of Art.'[21]

While instruction-based creativity has been around since the 1960s,[22] Proceduralism has recently found a more aesthetically sophisticated, if less grandiloquent, expression in the eu-gene email list and the software-art communities around the transmediale and read_me new-media festivals. More hip to contemporary art and criticism than the fractal calendar-makers, this group has decided to resolve Musgrave's understandable anxiety about aesthetic output by means of a Gordian-knot approach. These makers of 'generative art' tolerate, or even celebrate, output that is unruly or ugly – though to be sure it is often ugly in an arty kind of way — and to focus on the code itself as the object of interest. Although the terms of discourse and standards of achievement may differ, the definitions proposed by this group are a direct echo of Proceduralism: 'Generative art refers to any art practice where the artist creates a process, such as a set

of natural language rules, a computer program, a machine, or other mechanism, which is then set into motion with some degree of autonomy contributing to or resulting in a completed work of art.'23

[[subB1]] **Social Software**

Opposed to this formal exploration of code as an art in itself is a group of Internet artists, notably the London-based collaboratives I/O/D and Mongrel, known for projects with covert or overt political agendas. I/O/D member Matthew Fuller, who has written cultural criticisms of quotidian programs such as Microsoft Word,[24] has challenged formal explorations of code to go beyond inbred aestheticization, calling for an examination that provokes a greater awareness of code's social and political contexts. Internet artist Amy Alexander expresses the same point of view in an interview with the jury for the read_me software-art competition: 'Most non-art software pretends to be neutral and objective technology — devoid of human influence. Software art opens itself up to examination of its human-created biases and its human-experienced influences — so it helps us understand how these factors operate in "normal" (non-art) software as well.'[25] The German code historian and free-software activist Florian Cramer calls this divergence 'software formalism vs. software culturalism', lamenting:

> [[extract]] If Software Art would be reduced to only the first, one would risk ending up with a neo-classicist understanding of software art as beautiful and elegant code.... Reduced on the other hand to only the cultural aspect, Software Art could end up being a critical footnote to Microsoft desktop computing, potentially overlooking its speculative potential as formal experimentation.[26]

To evaluate the Mandelbrot Set and *Every Icon* side by side is to sit uncomfortably on the horns of Cramer's dilemma. Should they be evaluated according to code (form) or result (function)?

[[subB1]] **A Third Alternative**

The biological body copes with this dilemma by using form to produce function. Genetic jumblers manipulate the code (form) of an antibody in order to produce the wide variety of antibodies that can then detect an invading virus (the function). This misuse of genetic code — which might prove lethal elsewhere in the body — allows the immune system to anticipate shapes it has never encountered. Thus, random and playful misuse turns out to be central to the body's main strategy for coping with foreign invasion. Artists employ a similarly wrongheaded strategy, playfully misusing software and other technological codes to help us cope with technology's invasion of the social body.

If being creative with technology means thinking 'outside the box', then there is no better example than the artwork *Magnet TV* (1965), in which the Korean-born artist Nam June Paik planted a hefty magnet on top of an ordinary television set. Paik's magnet distorted whatever soap opera or car commercial happened to be airing at the moment into an exquisite variety of geometric patterns. In this case, the box in question is a television — the technological device that has contained human potential more efficiently than any other in recorded history. Before Paik,27 this box was an inviolable mainstay of every American living room; the image flickering on its screen was broadcast from a distance, out of the control of the end user. That is, after all, how it had been designed.

What made Paik's intervention creative was not so much his use of technology as his deliberate misuse of it. As the artist himself put it, 'Television has attacked us for a lifetime — now we strike back.' In lugging a magnet onto a TV, Paik violated not just the printed instructions that came with the set but also the political assumptions underlying its widespread use. Artistic misuse exploits a technology's hidden potential in an intelligent and revelatory way. The 1960s were a heyday of such activity, with artists like Paik, Jean Tinguely, and Robert Rauschenberg misusing everything from roller-skates to internal-combustion engines.[28]

There are examples of artistic misuse of technology prior to the 1960s, such as the effects Charlie Chaplin created by cranking film the wrong way through a camera. But the proliferation of digital media in the 1980s and burgeoning Internet access in the 1990s created an explosion in opportunities to conspire in, and observe the perversion of, technologies. Code artists deviate from the potential utility inherent in software by perverting the original use of the code to a speculative or critical end. This playful perversity distinguishes the artistic use of code from the merely technical one.[29]

The perversity central to the artistic misuse of code has nothing to do with 'perverse' subject matter. There's little perversity lurking in the code behind Bestiality.com or Sexy.com. Unlike works of artistic misuse, successful porn sites are clean under the hood (or is that under the sheets?), for they rely on a transparent, fully functional click-and-pay interface; glitches in the site would only remind porn hunters that they were paying to ogle pixilated curves rather than flesh-and-blood ones.

By contrast, the works we'll examine in this chapter retain some of the virtuosity of Proceduralism as well as the responsibility of social software, but to that they add a dose of perversity as they misuse code as syntax, tool, or experience. In these works, code may serve as a formal language worth studying for its own sake, and as a language for building alternative applications to prepackaged, especially corporate, software. But in the extent to which it veers from technique to art, code art will also work as a language for creating idiosyncratic, sometimes ephemeral acts of perception that typically arise out of playful misuse.

[[subA1]] **Transparency and Artistic Misuse**

In works that pervert code as syntax, tool, or experience, a certain legibility is necessary for such misuse to have artistic value. Whether the work is a conceptual installation by Sol LeWitt, a quantum-mechanics diagram-turned-archival print by Eric Heller, or a popup window gone haywire at jodi.org, the perversion of code is ineffectual as an artistic strategy if such misuse is concealed to all but the technological cognoscenti.

One way to encourage transparency is the 'open-source' approach typified by the 'CODeDOC' exhibition, an online show curated by Christiane Paul of the Whitney Museum of American Art in New York. This exhibition presents software art in such a

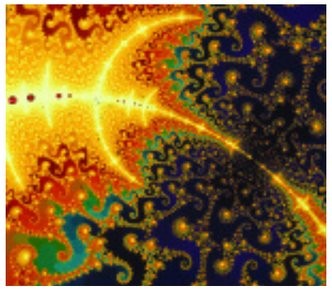
way that its underlying **if-then**s and **do-while**s are readily viewable by the user. Unfortunately, there are other occasions when viewing the code directly is impossible or unhelpful; still, what's key is that viewers understand how a rich visual or conceptual experience can be generated by the misuse of a particular technology. Nam June Paik hauled his electromagnet into plain sight on top of *Magnet TV*'s casing instead of hiding it inside, and he chose a store-bought magnet instead of custom-making a new one that would not have been recognizable to the average viewer.

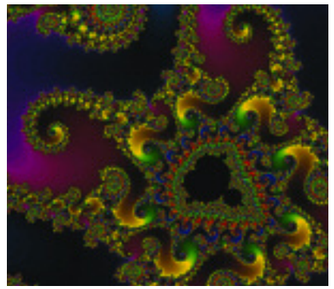
Sometimes, paraphrasing code can make it even more legible than displaying it. John Simon does not reveal the Java behind his *Every Icon*, but he's gone to the trouble of summarizing it in three short phrases. If you can think like a programmer, you can reconstruct the binary formula behind his marching pixels, but Simon's paraphrase is so guileless that lay viewers can get a feeling for the dynamic as well.

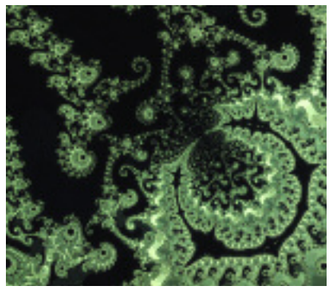
Provided the concept behind a work is simple enough, a visual paraphrase of the underlying code can be just as enlightening as a verbal one. The initial screen of Simon's 1995 Java applet *Combinations* (www.interport.net/~gering/combo.html) is a square with four colored lines which constitute the algorithm's graphic vocabulary. Once the user has dragged the endpoints so as to choose the angles and placement of the four segments — in effect, defined the algorithm's parameters – the applet produces a grid of every possible combination of those four lines on the computer screen or in a plotter printout. A Sol LeWitt drawing based on such combinations can take a team of draughtspeople weeks to complete; Simon's turbocharged drawing machine spits out his entire grid of permutations in milliseconds. While Combinations' patterning is seductive on a purely visual level, it produces this embarrassment of riches with an unnerving suddenness, throwing down an impertinent gauntlet to its Conceptual forerunners from the 1960s.[30]

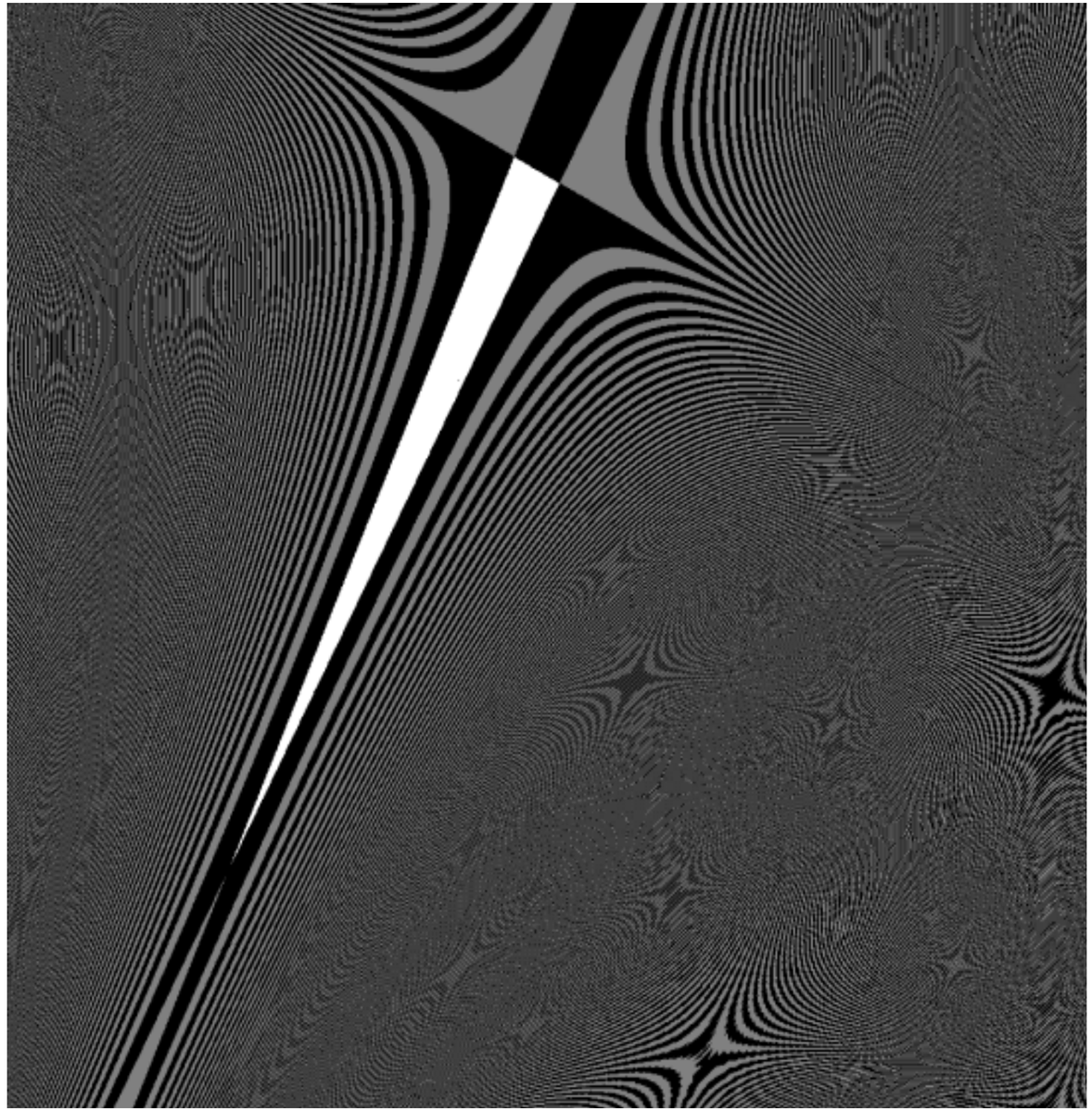
If the dynamic behind Simon's early works[31] is simple enough to paraphrase verbally or visually, the fractals and ray-tracing algorithms behind Ken Musgrave's alien landscapes can be incomprehensible to all but the computer-graphics specialist, even though his photorealistic sci-fi imagery is more accessible to the popular imagination. In recent years, however, Musgrave has hit upon a way to let ordinary users in on the process by which his fractal planets are created. His Pandromeda Web site (http://www.pandromeda.com/) offers freeware tools for navigating 3-D fractal planets (the 'Transporter') as well as 'pro' tools for building the planets themselves ('MojoWorld Generator'). In an essay that defined his earlier Proceduralist stance,

'Formal Logic and Self Expression', Musgrave cites the American photographer Ansel Adams's analogy between the photographic negative as a musical score and its print as a performance. Applying this analogy to Proceduralist fractal image-making, Musgrave writes: 'We currently have the capability in this new process to produce the negative or the score, almos.....









Comparison Studies

[[divider]] **Code as Syntax: Use and Misuse**

A look at the deepest level of code — the do loops and document.writes — shows how deliberate misuse of code helps to winnow the art from the craft. To pervert syntax means breaking the conventions that govern small-scale composition: creative painters commonly upset pictorial conventions, while creative poets commonly perturb grammar and word order. But if a creative programmer violates the rules that govern Java or Perl, the program won't work. So how can artists misuse the syntax of computer code to make art?

[[head2]]
Donald Knuth, 'The Art of Programming'
[[subA2]]
CRAFT WITHOUT CRAFTINESS

The series of books called 'The Art of Programming' by the Stanford University computer scientist Donald Knuth is about as distant a context for art as you can get from the Louvre or Sotheby's. Yet programmers compelled by a sense of beauty in code mention Knuth's name with reverence,[33] and his own writing supports the assertion that programs can be works of art:

> [[extract]] My feeling is that when we prepare a program, the experience can be just like composing poetry or music.... when we read other people's programs, we can recognize some of them as genuine works of art.... The possibility of writing beautiful programs, even in assembly language, is what got me hooked on programming in the first place.

Even if Knuth at times dismisses utility as the sole reason for studying code, his many volumes of algorithms and performance analyses are nevertheless directed at producing *good* code — code which is optimized for practical applications like sorting and searching. Software artists who work at the syntactic level, on the other hand, tend to be interested in *bad* code — code that doesn't or shouldn't work. While the craft Knuth espouses is useful background for their work, Florian Cramer has pointed out that craft is not usually the focus of their artistic activity:

> [[extract]] I thus would never limit software art to craftsmanship of programming (i.e. software art as a Donald Knuth-style 'art of programming'), but consciously take speculative, unclean, or even non-computer-related approaches into account, from certain forms of poetic play and conceptual art to the use of machine code fragments as private languages in artistic 'codeworks'.[34]

Elsewhere Cramer identifies Knuth's emphasis on craft with the Proceduralist art of Musgrave and his fellow computer scientists:

> [[extract]] Knuth's wording has adopted what Steven Levy calls the **hacker** credo that 'you can create art and beauty with computers'. It is telling that hackers, otherwise an avant-garde of a broad cultural understanding of digital technology, rehash a late-18th century classicist notion of art as beauty, rewriting it into a concept of digital art as inner beauty and elegance of code. But such aesthetic conservativism is widespread in engineering and hard-science cultures; fractal graphics are just one example of Neo-Pythagorean digital kitsch they promote. As a contemporary art, the aesthetics of software art includes ugliness and monstrosity just as much as beauty, not to mention plain dysfunctionality, pretension and political incorrectness.[35]

For a better look at Cramer's 'ugly' code, let's turn to some programmers who deviate from the Olympian precepts of Knuth to explore the purposeful perversion of code syntax.

[[head2]]
Mez, mezangelle
[[subA2]]
CODE-INFECTED WRITING

One strategy is to write code that deliberately doesn't work on a computer and is instead targeted for the 'literary **compiler**' in our own brains. E-poet Alan Sondheim has coined the term ***codework*** to represent a writing style influenced by — or on occasion directly modified by — computer code. Numerous online artists have experimented with codework in their contributions to art-related email lists, including Netochka Nezvanova, jodi, and Sondheim himself. One of the most elaborated of these digital dialects belongs to the Australian artist mez (Mary Anne Breeze), whose 'wurk' consists of texts emailed to discussion lists written in her own lingo, called *mezangelle*.

If pidgin scripting languages like Lingo attempt

```
"Then i sawe a C0mmandE @ the d0llah
pr0mpte:
holdinge in itz teXt the keys of M: I: L: E:
N: & U. 1t seiZed sm
0v the lettaHs: & K0dEd them up in2 a
Th0usand N-crypti0nz. The
C0mmandE sent the N-crypti0nz
staticscreaminge t0 the Binne.
The zer0Hs & w0nnes then flick.erred with
joy, N the
DataHK0de was b0rnE with the re:maining
lettaHz, the I: the
E: N 0therz B-sides, the P: & H: & O: all
linked N clicking
straighte in2 the Inph0ennium..."

"0h[& 1] wash the teXt with pixels 0v yr
spacE
N weep c0de in2 yr boardes and chr0me-hic
dreamz:
The Ange1 & the godde-S d0 dance with-N
the Prayer
pr0Grammez
N lectr0split & text-N d0llah-flippe in2 the
screenz
N inph0spherez:
so greate iz yr teXt: my KodE: yr p0wer to
d0wn10ad."

"& i will smite thEE with mye vEYErus,
N seeke 2 break the millennium
corrupt K0dE
that binds u in itz thymiC
0verfl0w..."

" N we shell l00k 2 the screen, N on those
screens we shell f1nd the line--__--____
and the d0t:: . . , and the d0t shell
speax. N the d0t will teXt:

$take up yr keys and type
$N click yr modemz :0N:
$N scroll the pages thru
$N delete @b0minations
$N n0.s N linkz that breakE the KoDE
$N thenne merge the stringz
$N u shell B phree."
```

to make programming syntax look more like English ('put this in it'), *mezangelle* attempts to make English look more like a programming language. mez's texts are peppered with characters more commonly found in code such as /, [, and (. Programmers use these delimiters to associate numbers or **string**s with a common function, but mez uses them to give syllable clusters multiple meanings through double or triple entendres.

Clearly, *mezangelle* represents the increasing influence of programming code in our daily lives, although its variorum of natural-language semantics makes it impossible to write a compiler that could interpret it. The read_me jury recognized in *mezangelle* historical precedents such as Oulipo, Futurist, concrete, and Perl poetry, though a better analogue might be the innovative writing style of hackers, which wittily incorporates programming syntax into email and other electronic communication. As Eric Raymond notes in his linguistic analysis of 'hackish',[36] true hacker banter is an informal yet syntactically precise argot characterized by multivalent wordplay. For all its pretension as e-poetry, mezangelle lacks the precision that characterizes hackish.

On the other hand, hackish is a discursive practice without a single author or self-contained 'works'; it would be much easier, for example, to cite a *mezangelle* manifesto in a poetry anthology than to excerpt an email exchange among hackers, if only because it would be impossible to know where to begin or end the latter. Yet the similarity of these practices suggests that a truly expansive definition of art should be able to accommodate artistry that does not fit into well-defined genres, a point to which we shall return.

µ[[head2]]
Tom Duff, Duff's Device
[[subA2]]
DISGUSTING FUNCTIONALITY

More perverse than private dialects like *mezangelle* are scripts based on improper syntax or puzzling recursion that can nevertheless be executed. As an example of a program that both does its job and embodies the 'monstrosity' Cramer champions in software art, Internet artist Amy Alexander cites *Duff's Device*, 'a remarkable algorithm that somehow marries two **C** structures in a surprising and efficient way. The result is both ugly and beautiful; disgusting and elegant'.[25]

At the time he wrote *Duff's Device*, Tom Duff was a programmer at Lucasfilm trying to animate a series of video shorts into a longer sequence. The conventional method of automating such sequencing — what programmers call a loop — was producing delays because of the bottleneck produced by the action of the processor looping back and forth. Duff's original code looked like this:

```
[[list]] send(to, from, count)
register short *to, *from;
register count;
{
do
*to = *from++;
while(—count>0);
}
```

To speed up the processing, he found an extremely unconventional way to short-circuit the do loop:

```
[[list]] send(to, from, count)
register short *to, *from;
register count;
{
register n=(count+7)/8;
switch(count%8){
case 0: do{        *to = *from++;
case 7:            *to = *from++;
case 6:            *to = *from++;
case 5:            *to = *from++;
case 4:            *to = *from++;
case 3:            *to = *from++;
case 2:            *to = *from++;
case 1:            *to = *from++;
}while(—n>0);
}
}
```

While Duff's hack circumvented the bottleneck he was encountering in animating a sequence of shorts without interruption, in the process he committed a number of serious sins against what Knuth would call 'literate' code. For example, the purpose of a **switch statement** is to be able to distinguish one case from another. This vexing program, however, exploits a **bug** in the language itself – the fact that C doesn't automatically break between cases — to permit the program to fall through from one case to another, executing all the lines along the way. Duff also begins his do loop in *case 0* and ends it in *case 1* — another violation of regular syntax. The result is a form of animation that shouldn't be possible syntactically.

Duff himself was ambivalent about his discovery: 'Disgusting, no? But it compiles and runs just fine. I feel a combination of pride and revulsion at this discovery. If no one's thought of it before, I think I'll name it after myself.'[37]

It is remarkable that this hack works at all — and even more amazing that this solution to a practical problem actually works *better* than the syntactically correct method by which Duff originally tried to solve the problem. As Alexander pointed out, the output that **Duff's Device** generates so efficiently is completely mundane, but the aesthetics of Duff's algorithm are sufficiently interesting to transcend their original purpose — so much so that this perverse routine created for an immediate utilitarian need has since become a puzzle used to confuse students and experienced programmers alike, an M. C. Escher image written in C.

[[head2]]
jaromil, Forkbomb
[[subA2]]
LETHAL GRAFFITI

A forkbomb is a program that, once initiated, churns out of control until it has taken over all **system resources** and crashed the computer. In 2002 a forkbomb by Perl programmer Alex McLean was awarded the software art prize of the transmediale festival in Berlin, but in the category of 'bang for the buck' it's hard to beat the minimal forkbomb written by jaromil, Italian hacker and free software author.

jaromil's forkbomb only contains 13 characters: :(){ :|:& };: When typed on the keyboard of certain UNIX computers, this simple instruction starts one process (symbolized by the colon), then replaces it with two processes, then three, four, five, and so on *ad infinitum*. As Florian Cramer has pointed out, this Molotov cocktail of code most resembles **emoticon**s such as :(and ;(, which represent crude emotional states in chat and email messages.[27] jaromil's message, on the other hand, is performative as well as representational — an example of the executable culture we'll examine in more detail in Chapter 04.

According to Cramer, jaromil believes software should be beautifully crafted. Clearly, however, jaromil's sense of beauty is antithetical to Knuth's. If Knuth represents legibility and control, jaromil represents illegibility and explosiveness; the latter's emoticon graffiti set into motion a chain reaction that — out of control — feeds on itself, as though merely typing Einstein's $E = mc^2$ into the right typewriter could set off a nuclear explosion.

[[divider]] **Code as tool: Use and Misuse**

The examples we've examined up to now have all used or misused code at the level of syntax, yet creators can also misuse code at the level of utility. Code happens to be the basis of contemporary tools with which artists can manipulate images and sounds, from graphics packages like Adobe Illustrator to video editors like Apple's Final Cut. For digital artists of the formalist persuasion, to tear one of these CDs out of its shrink-wrap is to unlock a new toolkit brimming with unexplored techniques and effects. To those of the culturalist persuasion, however, such applications are subliminal straitjackets to creativity, so that artistic genres become defined less by an intent common to a group of artists than by a particular set of Photoshop filters or default Flash sounds.

A number of creative self-starters have taken it upon themselves to code their own applications from scratch, creating new tools with fresh ideologies not governed by corporate interests. Much virtual ink has been spilled on email lists over whether these artist-made tools are themselves art. Sometimes artists custom-make a tool simply to advance their own work, like the long sticks the bedridden Henri Matisse devised to pin his colorful cutouts on a wall. In other cases, innovative programmers make and distribute their tools to realize the aesthetic visions of others. These tools by artists for artists provoke the question of whether such software production might be an artistic practice in itself.

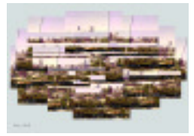
This doesn't mean that an art-making tool can never itself be art. But more is required of such tools than that their makers claim to be artists. If programmers of perverse syntax abuse the conventions of code, programmers of perverse tools use perfectly proper syntax to abuse their users' expectations about how digital tools are meant to function. Perverse tools aren't slaves to some predefined purpose, but instead offer wrongheaded, often counterproductive approaches to the task their users expect them to fulfill.

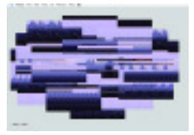
[[head2]]

Grahame Weinbren, Paul Weinbren, and Stephen Bannasch, *Limosine* (1982-85)[3] [28]

[[subA2]]

GRINDING YOUR OWN VIRTUAL PAINT





In 1982, Grahame Weinbren and Roberta Friedman began work on *The Erl King*, one of the first interactive video installations. Weinbren hoped to create an 'interactive cutaway' in which touching a screen while one video was playing would switch to another video without interrupting the soundtrack. Building on his experience as a video editor and interactive designer for the 1982 World's Fair, he soon discovered that there was no way, given the relatively crude and inexpensive equipment then available, to create the effect he was after. So, he and his brother set out to program a video-editing **suite** in **PASCAL** to do just that. The result, Limosine, ran on the minimal **CP/M operating system** on a primitive Zilog Z-80 personal computer. Despite this humble provenance, Limosine was capable of tracking video interactions over time to create a more complex and unpredictable cinematic experience than canned multimedia packages like Macromedia Director would offer decades later. Weinbren originally intended to distribute Limosine freely to other artists who wanted to experiment with interactive video, but without the Internet as a distribution tool his efforts were doomed to failure. In today's world, where a new software release can be downloaded at the click of a mouse, the artist-made tool has blossomed into an artistic genre in its own right. Perhaps the best known of these applications is Nato, a project by the slippery online presence known variously as Netochka Nezvanova, nn, antiorp, and integer. Based on the venerable Max audio-editing software, Nato is a programmable interface for manipulating and mixing multiple audio and video feeds into a live networked performance. Nato has enabled new forms of artistic expression premised on networking together far-flung video images in real time. For example, the artist Meta used Nato to create his work *Panorama* (www.meta.am/process/panorama/), a rare example of video-based Internet art. *Panorama* knits together Webcam images from around the world into a panoply of simultaneous video streams. Tools comparable to Nato have since been released by David Rokeby (softVNS), Jennifer and Kevin McCoy (Live Interactive Multiuser Mixer), and others.

But why stop here? Why not characterize as art any small-scale independent production of a tool for artists? Max, for example, the graphical-programming environment for music and multimedia on which Nato and softVNS are based, has enabled the creation of innumerable innovative audiovisual works by artists since its release in the mid-1980s. The trouble is that Max's inventor, Miller Puckette, was trained as a mathematician and doesn't exhibit in galleries or museums (as Grahame Weinbren does) or hang out on art-related email lists (as Netochka Nezvanova does). To qualify Limosine or Nato but not Max would be to perpetuate the misinterpretation of Marcel Duchamp and his work that this book is trying to replace. Although they have been the basis of wonderful artworks like *The Erl King* and *Panorama*, Limosine and Nato are not software art any more than a hammer and glue gun are installation art. And tool-making artists like Jennifer McCoy often admit the aesthetic limitations imposed by utility: 'As artists, we're not interested in revisiting the code. We just care if it allows us to create the art we set out to make.'[29]

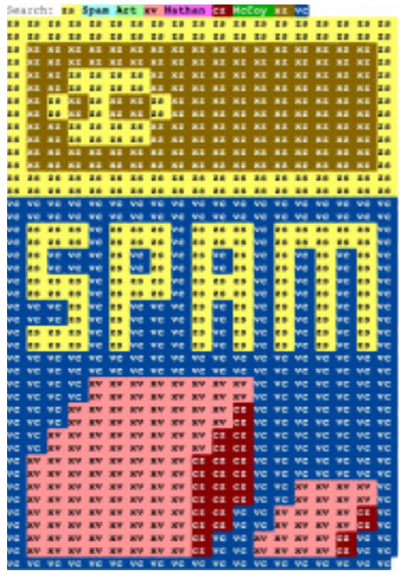
[[head2]]

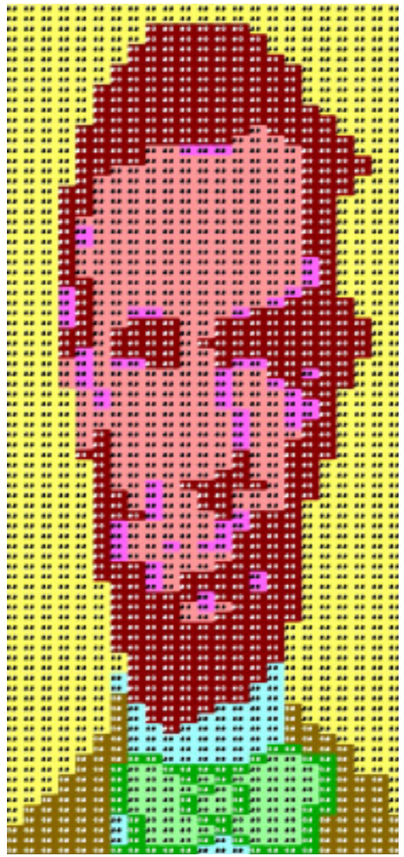
Tim Flaherty, Stuart Langridge, et al.,

Google Groups Art

[[subA2]]

PAINT BY LETTERS





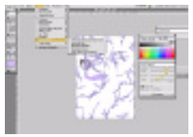
Adobe and Macromedia trumpet their new tools for artistic expression by offering ever-higher resolutions, smoother transitions, or fancier brush shapes. **ASCII** art, in contrast, offers a perversely low-tech way to make images. To make a picture in ASCII is simply to render a **grayscale** image (such as a portrait or a landscape) in the character set most common to the Internet -- from &s, @s, and #s rather than from pixels of red, green, or blue. **Translators** from image to text have been around since the 1960s, but only in the past few years have ASCII artists realized that the popular search engine Google represented an ASCII translator lurking right under their noses. Google color-highlights search terms within the **Usenet posts** it displays; to hijack this feature for pictorial purposes, users post contrived messages with carefully chosen strings of text ('aaaaabbbbaaaaa'), then run a Google query on the letter combinations from which their images are assembled. The result is the programming equivalent of painting by numbers. Stuart Langridge even scripted an automated tool to help would-be ASCII Leonardos reverse-engineer their images. What's engaging about this project is not the images themselves, which are as kitschy as most ASCII art, but the way in which a simple utility can let anyone hijack a dominant technology like Google for aesthetic ends.

[[head2]]

Adrian Ward, Auto-Illustrator

[[subA2]]

GRAPHIC DESIGN GONE HAYWIRE



Auto-Illustrator (www.auto-illustrator.com/) is a fully functioning perversion of Adobe's popular commercial software, Illustrator. Like its eponymous progenitor, Auto-Illustrator is a vector-graphics program; Illustrator provides a visual interface that helps artists (mostly graphic designers, actually) generate images defined by points and formulas rather than pixels. Such images have the advantage of being scalable and, to some extent, programmable. In a single keystroke, for example, an Illustrator user can select three lines in a picture and rotate each of them 20 degrees without changing their positions.

Produced by software artist Adrian Ward, Auto-Illustrator takes this programmability to an almost absurd extreme. Users can select tools that 'Bauhaus' a graphic into a particular style of geometric abstraction, or that 'degenerate' a line using a slider than runs from 'stupid' to 'pointless'.

Unlike Adobe Illustrator, whose interface is designed to be as straightforward as possible, Auto-Illustrator lends itself to a haphazard method of visual composition based on random discovery rather than premeditated planning. It is this oblique relation between intention and result that distinguishes

[[head2]]
I/O/D, Web Stalker; Bill Cheswick, Internet Mapping
[[subA2]]

THE WEB WITHOUT PAGES

Perhaps the most prolific genre of do-it-yourself tools is the alternative **browser**. The proliferation of Web crawlers, spiders, and mappers at the turn of the millennium reflected the increased importance of this graphic medium, which since its introduction in the early 1990s has usurped the function of earlier text-based protocols like Telnet and Gopher to become the primary lens through which netizens viewed cyberspace. An additional motivation was the opportunity to demonstrate the technical and conceptual limitations resulting from the monopoly of a single such lens, Microsoft's Internet Explorer, which began to dominate the market at the same time that these alternative browsers emerged.

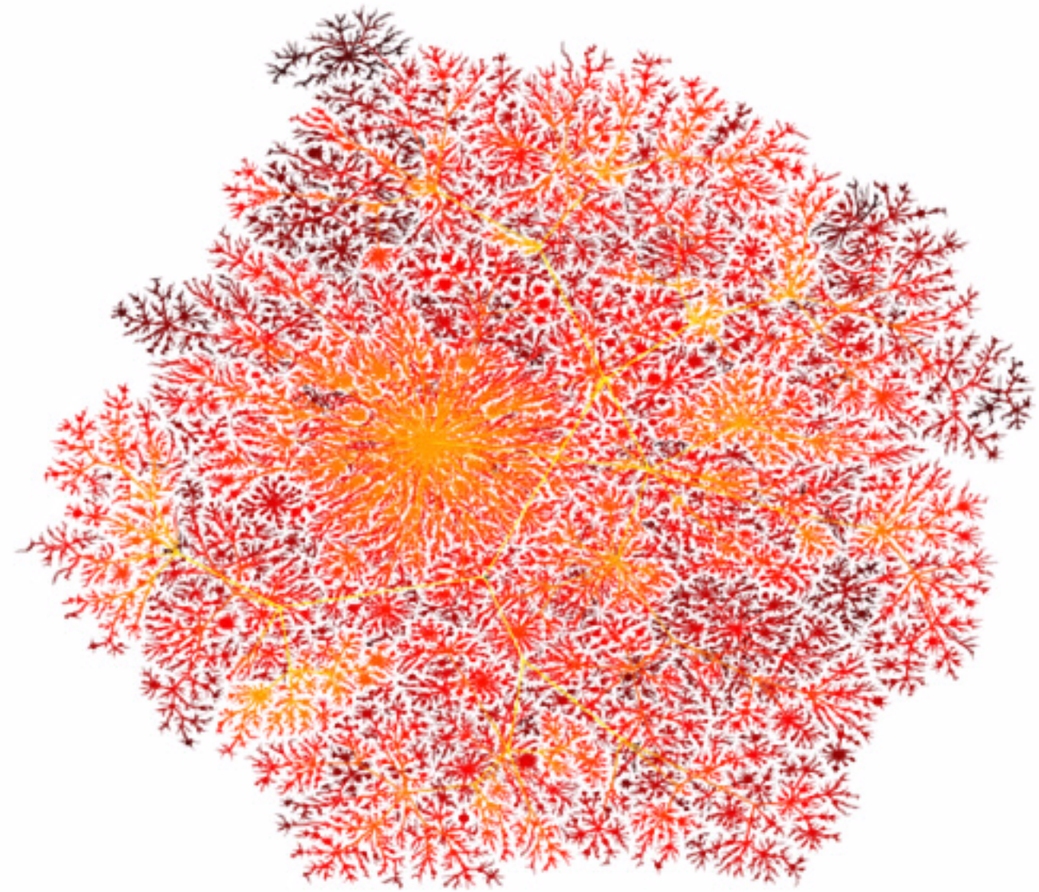
The challenge of picturing cyberspace has attracted programmers from various disciplines. One of the earliest and best-known alternative browsers is Web Stalker by the London-based artist collaborative I/O/D (Matthew Fuller, Colin Green, and Simon Pope; www.backspace.org/iod; bak.spc.org/iod/). While commercial browsers such as Netscape and Internet Explorer represent the Web as a series of print-inspired pages, Web Stalker offers surfers a glimpse of the Web as a network. In place of a single window onto the Web, Web Stalker offers multiple windows with suggestive names like 'Dismantle', 'Stash', and 'Extract'. It is the 'Map' window, however, that offers the most compelling vision of a pageless Web. Upon loading a particular **url**, Web Stalker draws a circle in the middle of this window, then draws spokes connecting to new circles, each of which is a page linked to the original Web page. Left to its own devices, Web Stalker will continue to spider new nodes and connections, gradually filling the screen with a visual network of increasing complexity and beauty.

If I/O/D's Web Stalker is a subversive critique of browser monopolies, researcher Bill Cheswick of the industry think-tank Bell Labs has a more scientific motivation for mapping the Internet. Cheswick (cm.bell-labs.com/who/ches/map/gallery/index.html) aims to envision the Net's technical structure by tracing the routes information **packet**s take along the frequently traveled paths of the information superhighway, from the Sprint and AT&T interstates down to the 'country roads' of regional phone systems. Studying the resulting diagrams, which plot these interconnected interstates as skeins of different colors weaving through a tangled knot of forking paths, can yield concrete, scientific conclusions. For example, a series of real-time maps of the Yugoslavian Internet during the1999 Balkan conflict demonstrates how a network crippled by bombing can re-organize itself to route around damage.

Is there any reason to consider I/O/D's map art and Cheswick's not? I/O/D's palette is more restrained, and the work is regularly cited in online exhibitions and art-based email lists, while Cheswick's colorful design appeals more to popular taste yet is little known outside a narrow community of telecommunications researchers. Like all good art, both kinds of maps do not merely reflect the world as it exists but rather construct the worlds — geographical, religious, political — we inhabit. Is the difference, then, just the company Cheswick keeps? If Cheswick's work appeared in the Duchampian context of a gallery or new-media festival, would it suddenly be recognized alongside Web Stalker on art-related email lists like Rhizome and nettime?

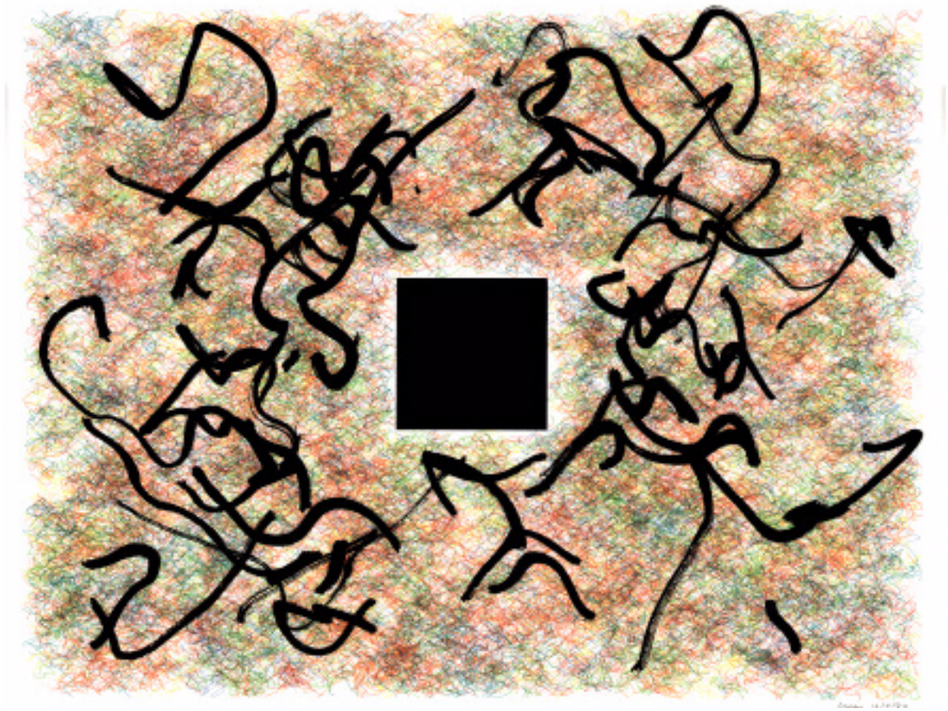
The question is ironic, because Cheswick's pretty pictures are actually more amenable to traditional frames than I/O/D's constantly changing interface. In fact, Cheswick is all too happy to offer 'framable' posters of his diagrams for sale on his Web site. His choice to exhibit computer-made images isn't exactly a radical intervention in the art world; deterministic illustrations have been produced by computers for some time now, and, as we'll see in the next section, they often are prized for their adherence to established aesthetic expectations rather than their departure from them.

By contrast, I/O/D's dynamic, screen-based application enlightens its viewers not through representation but through participation. And though I/O/D doesn't market matted prints of Web Stalker **screenshot**s, they have deliberately 'framed' their work in the browser genre, precisely in order to call attention to the limitations imposed on that genre by page-based browsers like Explorer and Netscape. In effect, Web Stalker puts quotation marks around the 'browser' concept, drawing attention to its limitations by jamming round nodes into this square window. By comparison, there is nothing particularly perverse about Cheswick's diagrams; his use of **traceroute algorithm**s, while it may be inventive, is perfectly in the spirit of their original purpose.



[[divider]] **Code as Experience: The Proceduralist Product**

The use of computers to generate creative experience, as opposed to creative syntax or tools, has been explored for several decades, yet the pioneers of this approach have achieved little recognition among mainstream museums and art critics. The first exhibition of computer art dates back to 1965, when three mathematicians — Frieder Nake, A. Michael Noll, and George Nees — installed photographic enlargements of plotter drawings at the Technische Hochschule in Stuttgart and the Howard Wise Gallery in New York.[32] Yet none of these mathematicians-turned-artists is well known outside a small circle of computer-art cognoscenti. The fault lies partly in the art world's limited purview but also in the obsession with product over process displayed by those who followed the first generation of Proceduralists.



[[head2]]

A. Michael Noll, Harold Cohen and Roman Verosko

[[subA2]]

PROCESS OR PRODUCT?

The work of A. Michael Noll exemplifies a tendency present in computer art from the very beginning, namely that regardless of any Proceduralist justification for the use of computers to create radically new imagery, most of the output of computer art ended up emulating traditional compositions. Noll, a mathematician at Bell Labs, used equations to generate geometric structures like his *Gaussian Quadratics*. The technique was entirely new, yet the resulting image bore a resemblance to Russian Constructivist sculpture from four decades earlier. That same year – 1963 — Nam June Paik first exhibited his perverted television signals, which echoed Noll's geometric forms yet were created by means of an entirely unstudied, unsystematic, and perverse approach. In a conscious, if not necessarily self-critical, acknowledgement of this drift toward the pictorial, Noll later designed programs for the express purpose of printing out variations on painterly compositions such as Piet Mondrian's 1917 *Composition with Lines*.[33]

Painter-turned-computer artist Harold Cohen has also explored the conscious imitation of aesthetic output. His automaton of choice is not a computer plotter but AARON, a turtle-like robot on wheels that he has armed with pen and brush. If Noll considers his computer to be more of a partner than a tool,[34] Cohen treats AARON as a child, instructing it in a series of 'aesthetic upgrades' to draw lines, then closed shapes, then colored forms, and, finally, entire compositions on paper taped to the floor. By encoding aesthetic sensibility into repeatable instructions, both Noll and Cohen aim to produce art-like images — but the novelty of automatic, familiar-looking drawings is their means, not their end. Noll and Cohen are trying to learn more about the process by which any artist creates, whether built of flesh and blood or of gears and servos.

The same cannot be said of the heirs to Noll and Cohen, many of whom justify interest in their process by invoking the algorithmic purity of Proceduralism yet present and 'frame' their output in an art context that overwhelmingly favors product over process. Roman Verostko, for example, painted all-over canvases by conventional means until he discovered that a computer program could do it for him. His custom-designed software, Hodos, emulates not the style of children's drawings or Modernist paintings but the atmospheric abstraction this mature programmer-artist had already explored in pen-and-ink. Verostko has described as 'uncanny' the resemblance between his pre-computer, handmade drawings and those scribbled by the fourteen-pen-plotter hooked up to his personal computer. Perhaps part of that uncanniness stems from the ease with which a 'machine that makes the art' has usurped the role of its progenitor and aesthetic role model, seemingly without introducing any aesthetic surprises or detours of its own.

[[head2]]
Eric Heller
[[subA2]]
PRODUCT OVER PROCESS

In perhaps the most extreme example of the use of code to create artworks whose authenticity is vouchsafed by traditional aesthetics, the Harvard chemist and physicist Eric Heller writes code that attunes his work not to his own taste but to that of his audience. To be sure, Heller also claims allegiance to a form of algorithmic purity; in fact, he claims more relevance for his algorithms than for the ungrounded mathematics of his peers:

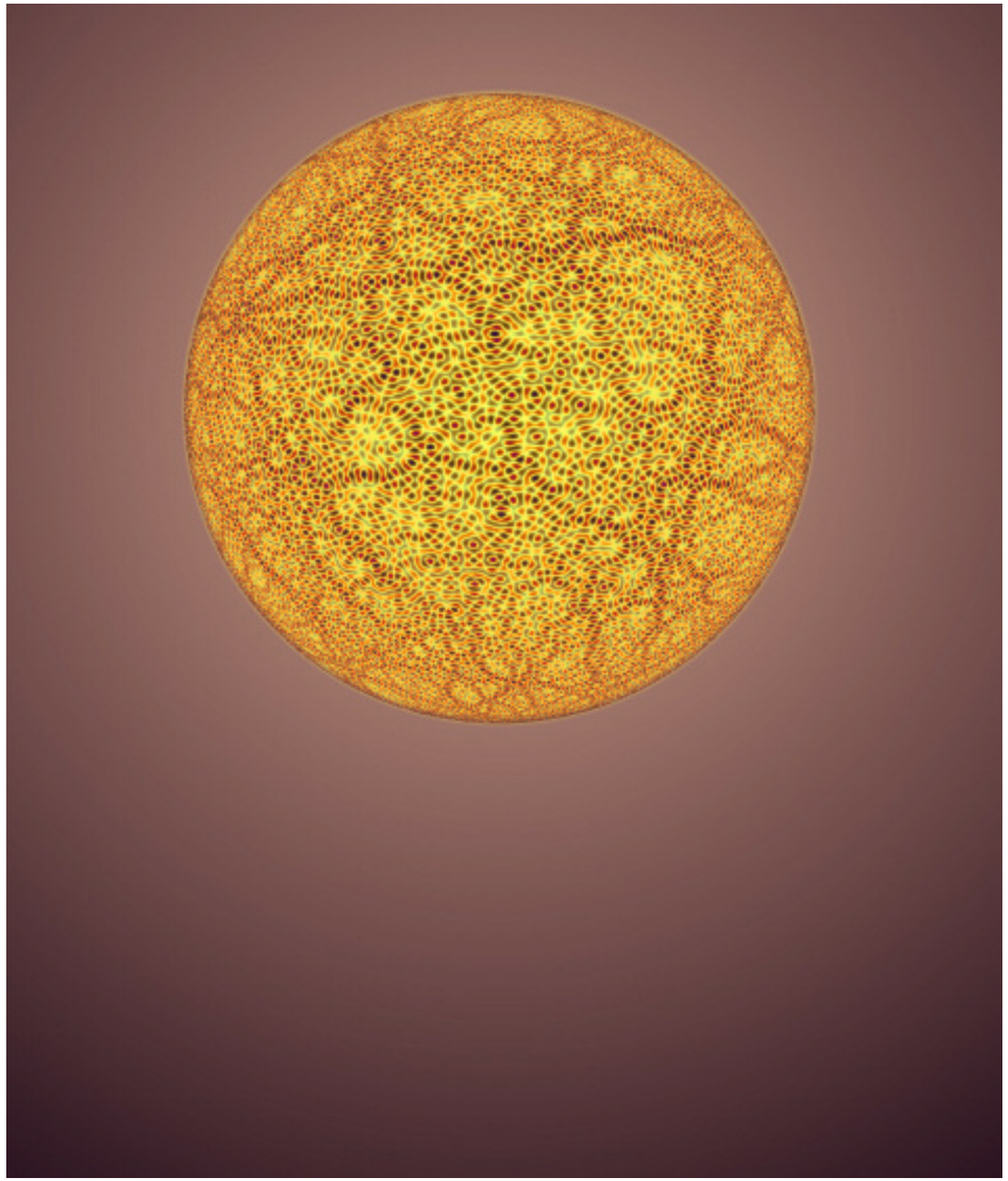
> [[extract]] Digital artists need no longer emulate traditional media only! The computer allows us to create new media, with new rules, more naturally suited to the new tool. But such rules are best when they too follow physical phenomena, instead of arbitrary mathematical constructs. I have learned to paint with electrons moving over a potential landscape, quantum waves trapped between walls, chaotic dynamics, and with colliding molecules.... You could say that I'm using physics as my brush.[35]

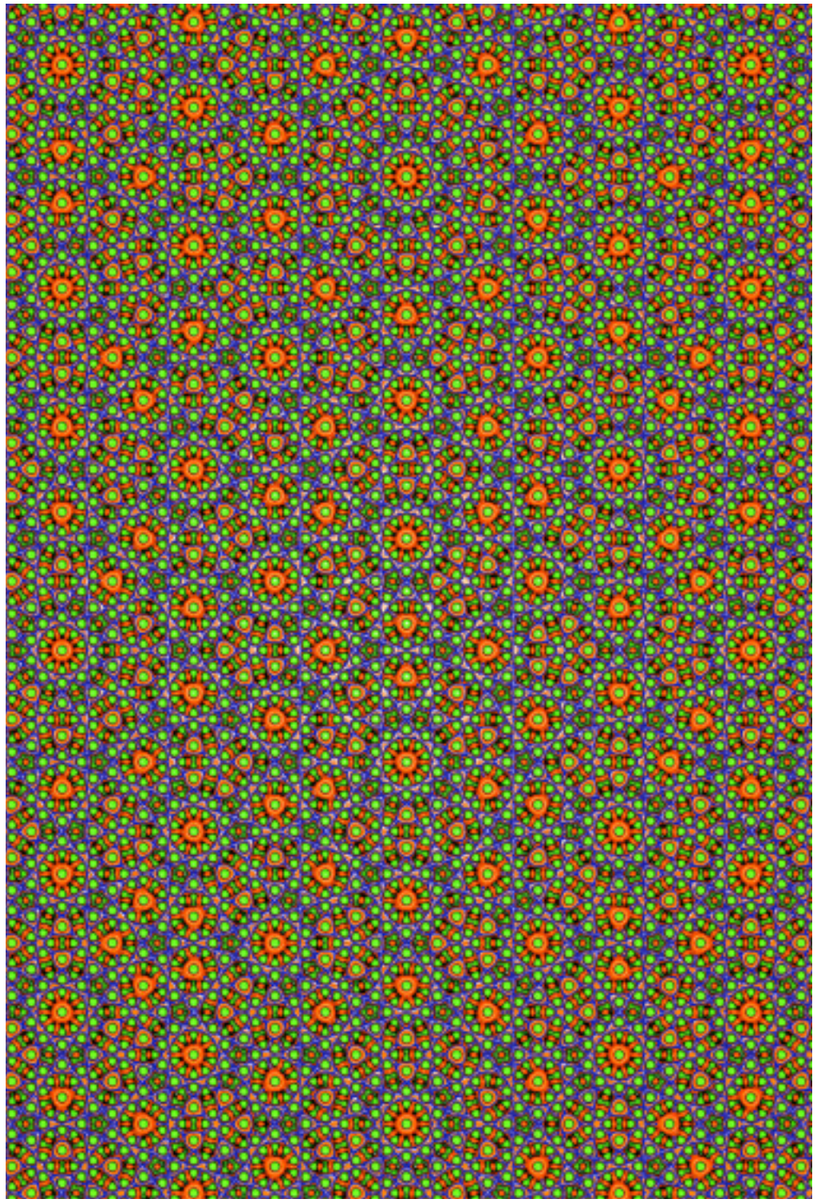
Heller seems to espouse a sort of reverse-Proceduralism when he argues that 'art can lead to science,' a claim he justifies by noting that chaos theory 'would never have taken off without some gorgeous images — like fractals — that came out of computers'.[36] But he wants to have it both ways; although the scientific Heller demands that his code be based squarely on the rigorous application of physical models, the artistic Heller manipulates the resulting colors using Photoshop to create 5-by-8-foot digital printouts.

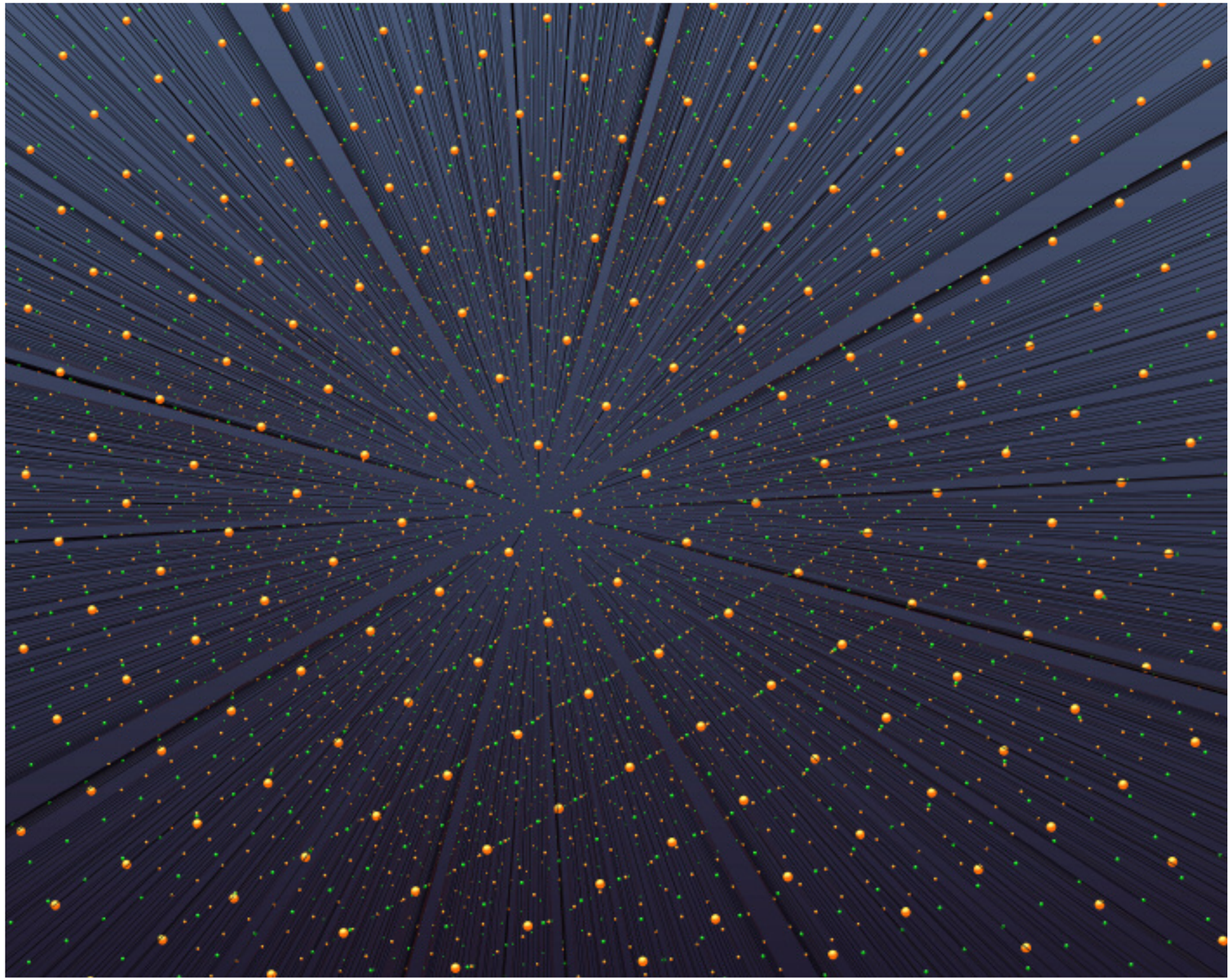
Although a staunch Proceduralist might object to retouching computer output rather than resetting parameters in the original code, manipulating colors may not seem as impeachable to a mathematician or physicist used to dealing with the invisible, for whom color is an arbitrary attribute added after the fact. For these scientists — including Mandelbrot — color is less a referent to a hue in nature than a means to distinguish areas on a plane or altitudes on a surface.[37] Heller, however, goes further in his cavalier treatment of color, offering to 'color-shift' certain of his 'museum-quality' prints to suit a buyer's preference.[38]

As though to stave off the horrified reaction this pronouncement would provoke in the artist trained in the Modernist tradition of art-for-art's sake, Heller has cited Sol LeWitt as a role model.[39] LeWitt's analog wall drawings are generated by assistants following the predetermined set of instructions in their titles, such as *Ten thousand lines about 10 inches long, covering the wall evenly*. Yet Heller's willingness to bend colors to suit his viewer's taste shows that he doesn't understand the Proceduralist strategy behind Conceptual art, which was to use a combination of logic and randomness to avoid tasteful adjustments which were not the direct product of the work's governing dynamics.

The value of LeWitt's work — and of good Conceptual art in general — lies in the tension between the simplicity of the rules and the visual complexity of the result. But this tension is only palpable if the artist has made the relationship between rule and result absolutely clear; although ten thousand lines on a wall create a surprisingly dazzling composition, any viewer who reads the instructions in the work's title can immediately grasp how they gave rise to the jazzy texture on the wall. By contrast, the same viewer would have to pour through *A Quantum Mechanics Primer* or *Mathematical Models of Semiconductor Physics* to fathom how Heller's spiraling corkscrews and garish splashes emerged from the study of our physical world. Artists like LeWitt deliberately kept the concepts underlying their work as simple as possible, precisely so that viewers could connect the dots between process and product. Similarly, Proceduralism will only achieve the experiential clarity of Conceptual art if its code is visible, simple, and devoid of any artificially introduced noise — such as manipulating color to suite the audience's taste[40] — that could obfuscate the transition from rules to result.







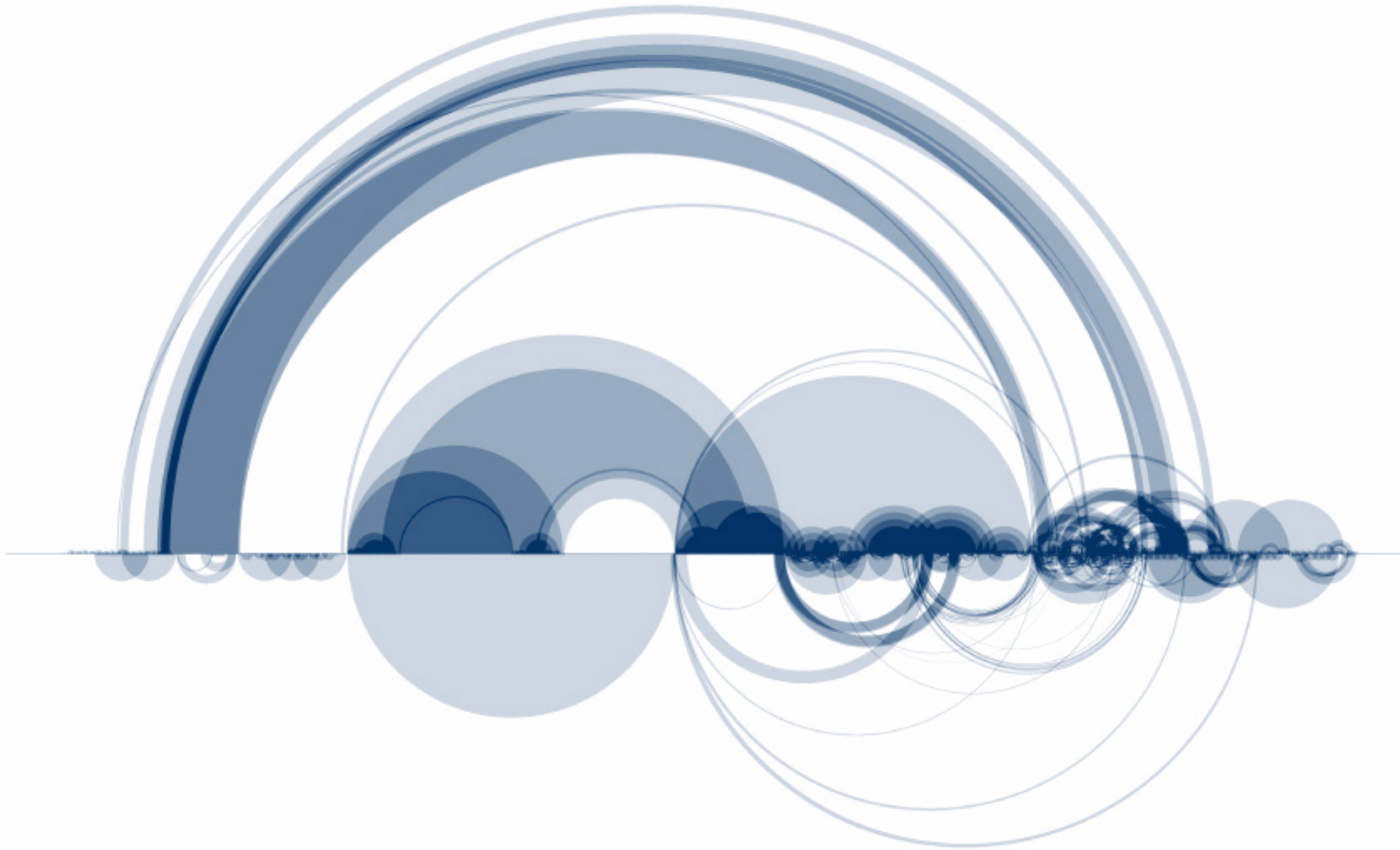
[[head2]]
Martin Wattenberg, The Shape of Song
[[subA2]]
PROCESS OVER PRODUCT

The Shape of Song (http://www.turbulence.org/Works/song) is one of the few projects based on mathematically complex algorithms where readers without an MIT PhD can still see the code in the results. That's partly because the work's creator, Martin Wattenberg, is an expert interface designer, but it's also because he's not out to make arty images with code but images that shed light back onto code itself.

The Shape of Song is an algorithm Wattenberg wrote to analyze **MIDI** versions of musical scores, i.e. encoded instructions about duration and pitch. By drawing arcs to connect all sets of notes that recur in a composition — whether on the scale of measures or of movements — Wattenberg's program reveals patterns of symmetry and asymmetry across a score as a whole. The shape of John Lennon's *Imagine* is a regular pattern of translucent semicircles equidistantly spaced over the score, while Bach's Brandenberg Concerto No. 2 is an intricate tapestry of blue skeins arching across arpeggios, chord progressions, and refrains.

In a perverse but serendipitous act of introversion, Wattenberg also applied this software to analyze the code in *other* computer programs. The resemblances between his diagrams of software and his diagrams of Lennon or Bach speak of the profound similarities between code and music louder than any text on 'The Art of Programming'.

As these examples indicate, scientific analysis frequently generates images at least as compelling as aesthetic output. Sometimes those with scientific training are too eager to bend their output into a frame defined by art-world norms and don't recognize that the most artistically illuminating option may be to leave the work in the state of a scientific diagram.[4]



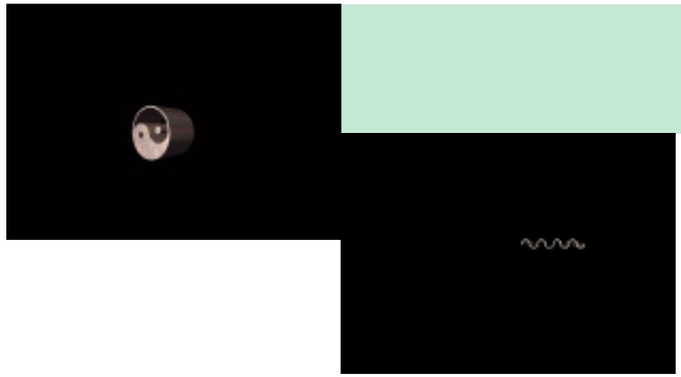
[[divider]] **Code as Experience: Screen**

So far, we've argued that software must be perverse yet intelligible to succeed as art. Requiring the misuse of code to be legible puts software artists in a difficult position, for perversity can get in the way of a viewer trying to figure out the connection between code and output. Fortunately, instruction-based process art of the 1960s, which rejected the undue fixation on product that ill-fated the work of their computer-artist contemporaries, offers code artists several strategies to wed perversity with intelligibility. LeWitt's wall drawing Ten thousand lines… demonstrates the strategy of logical excess: pursue a trivial notion — in this case drawing a line on a wall — doggedly to its extreme. The result is an artwork in which the relationship between instruction and experience is both perverse and lucid.

Besides excess, another tactic explored by Conceptual artists is impertinence. Florian Cramer has suggested an example of a nondigital artist whose simple yet impertinent instructions provoke 'improper' thoughts or experiences.[42] In 1960, the avant-garde artist and musician La Monte Young published this cheeky single-line instruction in an anthology of Conceptual and performance-art writings: 'Draw a line and follow it.' Young's directive, entitled Composition 1960 #10 (to Bob Morris), might seem inconsequential, but it's freighted with philosophical and political consequences. Anyone who tried to follow his command would soon run figuratively or literally up against a brick wall, because few legal systems permit unrestricted trespassing.

If Tom Duff perverted programming by writing an unreasonable script for a reasonable purpose, La Monte Young perverted performance by writing a reasonable script for an unreasonable purpose. Unlike Duff's improper case statements, there is nothing 'disgusting' about Young's syntax, which is grammatically correct and easy to understand. Yet, while Duff's device is functional despite its incorrectness, Young's instruction is dysfunctional despite its correctness. Like most Conceptual art, Composition 1960 #10…) breaks rules in practice, not in theory.

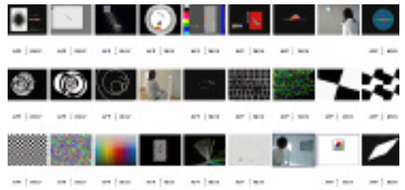
Software artists of the culturalist persuasion can take a cue from Young and employ a perfectly ordinary piece of code in an impertinent way that chafes social norms and expectations. Whether it bumps into brick walls or firewalls, impertinent code can reveal the hidden ideological constraints that mark real-world processes.

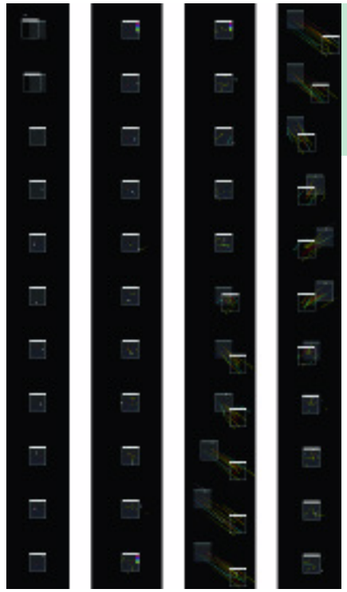


[[head2]]
Eldar Karhalev and Ivan Khimin, Screen Saver
[[subA2]]
FOLLOW THE BOUNCING SQUARE

Software art is generally confined to a window the user can show or hide at will; one exception is the screensaver, which allows programmers to use the entire screen as their canvas. Art-world celebrities from painter Peter Halley to architect Greg Lynn have made screensavers,[43] but none are as impertinent as Eldar Karhalev and Ivan Khimin's eponymous Screen Saver (www.karhalev.net/desoft). To 'run' this program is simply to change a few settings in the default screensaver that comes with pre-2001 versions of Microsoft Windows. A user who follows Karhalev and Khimin's instructions will convert Windows' default screensaver into a rectangle of fluctuating color sliding back and forth across the screen. In a strange way, Screen Saver is an update of Young's *Composition 1960 #10…* for virtual space, in which a square tries to continue in a straight line but ends up bouncing off the walls constraining it.

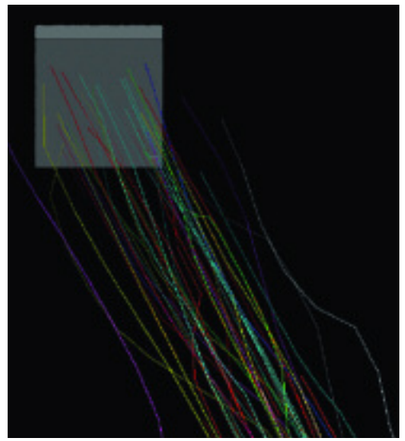
Like Google Groups Art, Screen Saver wonderfully illustrates how software art can be created without even learning a programming language.[44] Ultimately more important than knowledge of Java or ActionScript is an impertinent impulse to bend computers to purposes different from — and sometimes better than — those intended by their creators.[45]





[[head2]]
John Maeda
[[subA2]]
SOFTWARE FROM ANOTHER PLANET[46]

Taking a cue from A. Michael Noll, graphic-design maven John Maeda of the MIT Media Lab encourages designers to think of computers as impertinent collaborators rather than slavish layout tools. The graphic applications Maeda builds are even more brazen than Auto-Illustrator; while the latter could be used for practical design work, Maeda's 'reactive graphics' (www.maedastudio.com) feel more like software for denizens of another planet. The stroke of a virtual pencil in Time Paint drifts off the screen like smoke from a smokestack, while to paint with Inverse Paint, the user moves the window — the pencil stays still. While mastering drawing with the wind or a frame requires some dexterity with a mouse, Maeda also wanted to program an instrument his three young daughters could play without a steep learning curve. The result was Tap, Type, Write, a perversion of the ordinary computer keyboard that lets loose a torrent of letter rotations, magnifications, and general typographical nuttiness in response to keystrokes.



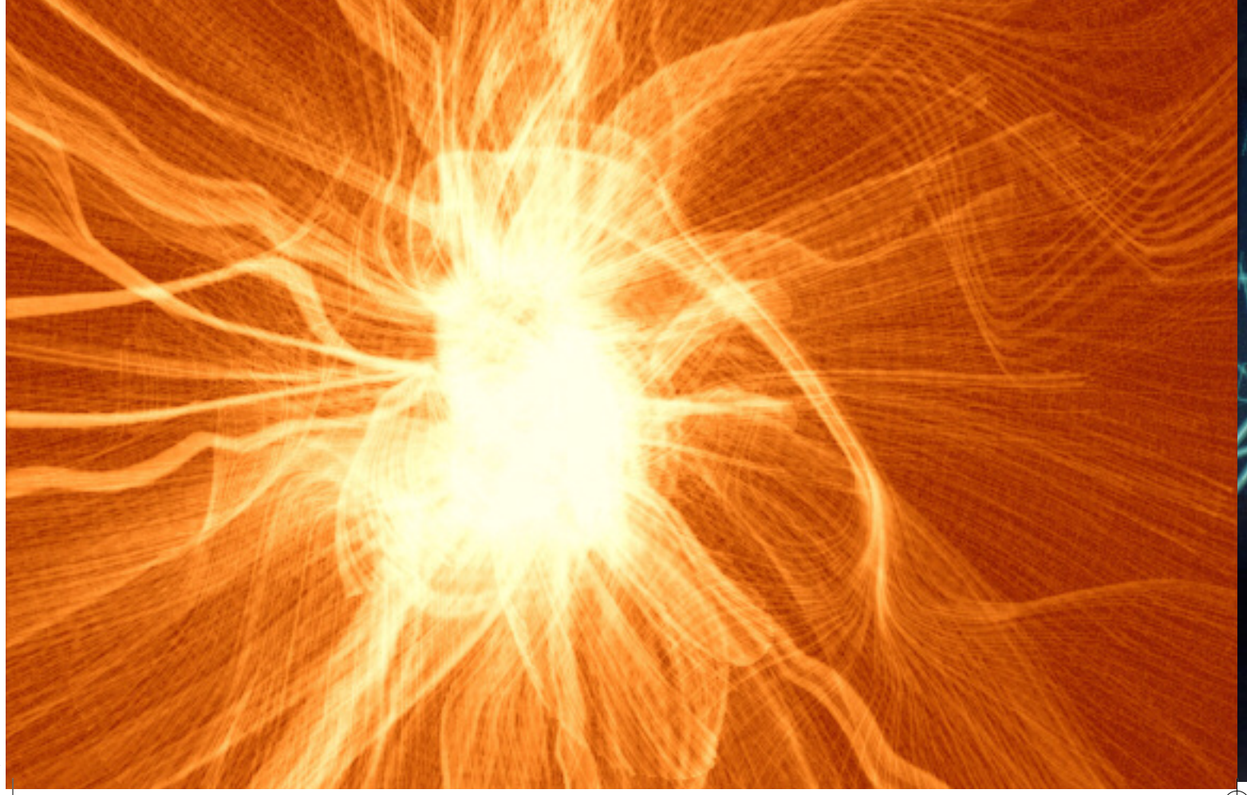
[[divider]] **Code as Experience: Peripherals**

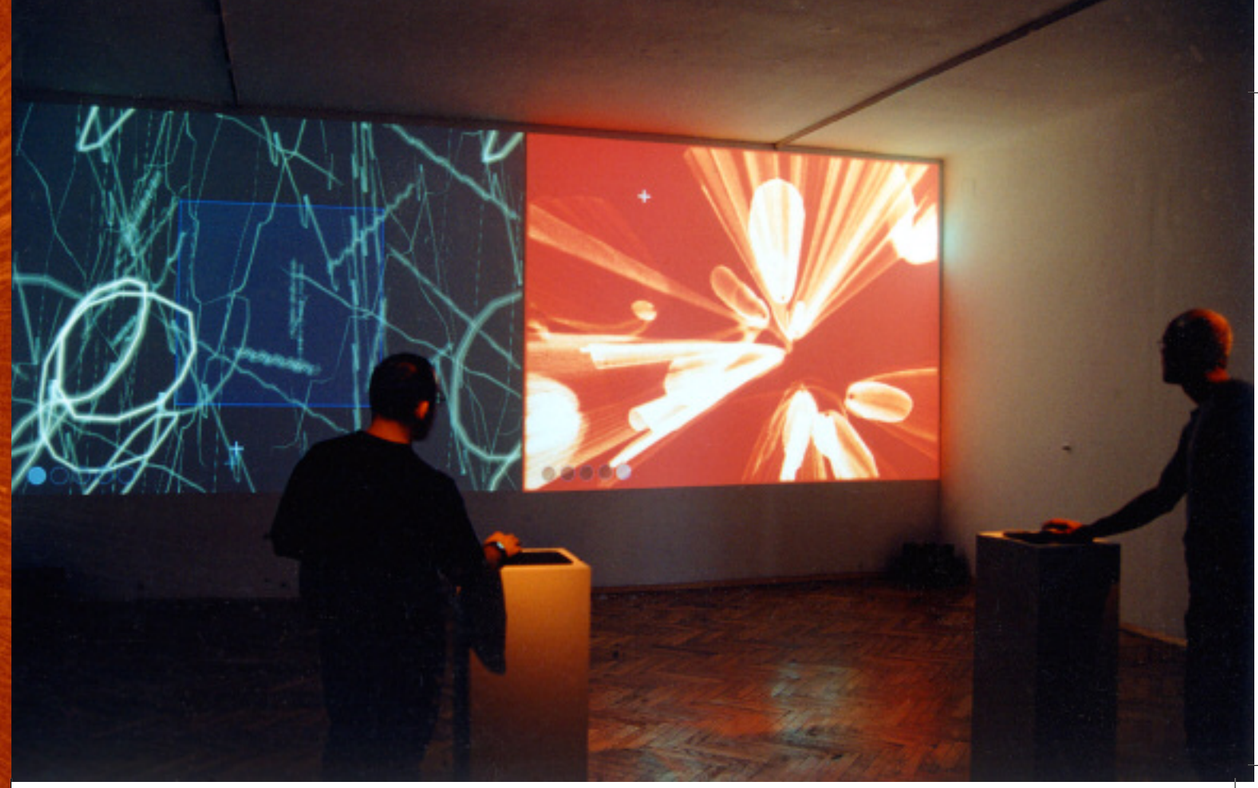
With mainstream browsers now supporting DHTML, Flash, and Java, numerous digital designers have experimented with new ways to click, drop, and drag your way around a screen. A few, however, have pushed the Graphical User Interface to its extreme; in the right hands — literally and figuratively — even the lowly keyboard can be misused to generate aesthetic excess. The result places the mouse, keyboard, and other computer peripherals front and center as expressive instruments in their own right.

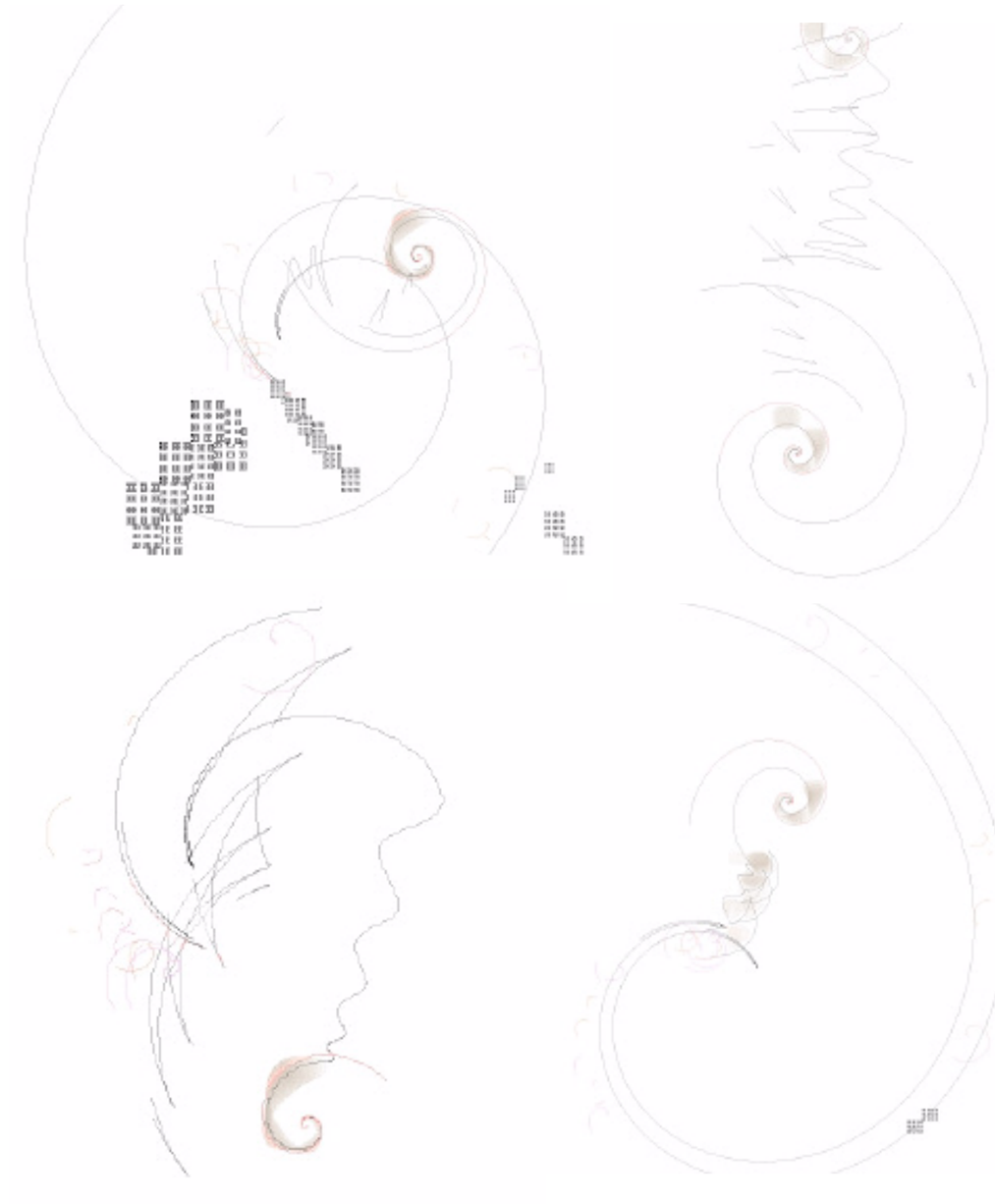
[[head2]]
Golan Levin, Dakka-Dakka and AVES
[[subA2]]
THE MEDIUM IS THE MASSAGE

MIT Media Lab graduate Golan Levin (flong.com) inherited his mentor John Maeda's penchant for perverting the keyboard. By translating the placement and cadence of keystrokes into percussive patterns, Levin's Dakka-Dakka reminds us that whenever we are typing, we are also drumming. But Levin parts company with Maeda stylistically, especially in his later works based on **click-and-drag**. Maeda is tight, Levin is loose; where Maeda's geometries tend to be Euclidean, Levin's are organic tendrils and nebulous clouds.

Like Maeda's reactive graphics, Levin's interfaces are more instrument than tool, leaving their user with memories of fleeting sensations rather than 'museum-quality' prints. In no work is this more evident that his Audiovisual Environment Suite (AVES), a set of five interfaces for producing visual gestures and sounds animated in real time. Each instrument allows its player to deposit a different inexhaustible 'substance' across the screen by clicking and dragging. In *Aurora* this substance appears to be a shimmering cloud that disperses with time; in *Floo* it's soft-edged, growing tendrils; in *Yellowtail* it's brushlike strokes whose placement on the screen becomes a visual score for synthesized music.







[[head2]]

Dextro and Lia, Turux

[[subA2]]

GRAPH PAPER ON ACID

At the other extreme from the lowly screensaver and keyboard are sophisticated animation suites like Macromedia Director, which offer a readymade, programmable toolkit with plenty of built-in features. Most interactive designers, however, restrict their use of this powerful tool to creating pulsing logos or morphing navigation bars for e-commerce sites. Seen in this context, Turux is a misuse of Macromedia's software only in so far as it demonstrates the raw visual horsepower of these tools when they're not yoked to some mundane purpose. Like Levin's AVES, each of Turux's scores of interfaces allows users to click and drag their way to a dynamic abstract image; unlike AVES, Turux's excessive animation is staccato and rectilinear, governed less by organic growth than by statistical irregularity. Clicking through a Turux work is like trying to plot points on a graph while dropping acid.

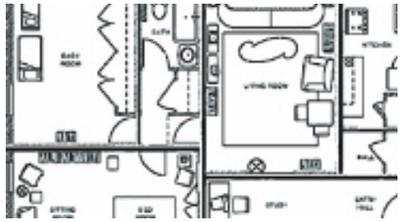
[[divider]] **Code as Experience: Web**

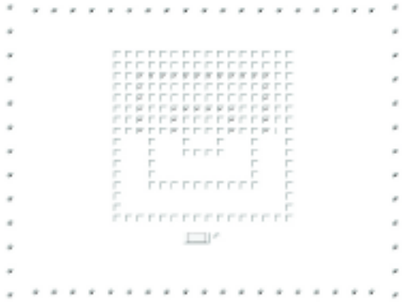
HyperText Markup Language is the lingua franca of all Web pages, a code so simple that most techies don't consider it programming at all. Yet the misuse of HTML tags can allow insights into online politics, community, and vision. In Chapter 04, we'll see how Internet artist Heath Bunting misused the anchor tag <a href> to comment on the politics of domain names. In the meantime, however, here is a brief primer of everything a Web-design class won't teach you about HTML.

[[head2]]
HTML
[[subA2]]
<FRAMESET>

A fixture of first-generation Web pages, <framesets> were supposed to allow one part of the page (such as a navigation bar at the left) to remain the same while another part (such as the content displayed at the right) changed based on the user's clicks. Olia Lialina's hypertext story 'My Boyfriend Came Back

from the War' (http://www.teleportacia.org/ war/ war.html) undermines the intended purpose of frames — to produce a sense of stability as the user navigates through a Web site — by multiplying them indefinitely, nesting new ones within previous ones and subverting the navigation-to-content hierarchy, with the result that users aren't sure where to click next. Darcy Steinke's online story 'Blindspot' (http://adaweb.walkerart.org/project/blindspot) explores a similar misuse of frames, but here their proliferation is more deliberate and systematic. As the story unfolds from one pane to another, the irregular rectangles making up the <frameset> are gradually replaced by a blueprint of the apartment that serves as the setting and emotional anchor for





the story.

[[subA2]] **<button>**
Another holdover from the Web's Jurassic period is the <button> tag, a stodgy grey rectangle used in countless online forms along with radio buttons, checkboxes, and input fields. Long reviled by Web designers for their clunky and inconsistent appearance across different browsers, these crude building blocks of HTML forms are elevated to the status of vocabulary for abstract art in Alexei Shulgin's *Form Art Exhibition* (http://www.c3.hu/hyper3/form). The results of Shulgin's invitation to Internet artists are an uncurated set of Web pages built entirely with form elements: pages tiled over with submit buttons or pimpled with radio buttons, where every click leads to a new 'forest of signs'.

[[subA2]] ** **
To view the Internet through one of Mark Napier's interfaces is to glimpse a landscape of unlimited visual possibilities. Napier's work is richer and more complex than the familiar print-inspired pages offered by corporate browsers like Netscape and Explorer. For example, his *Shredder* chops up any Web page into slivers of text and image, de-emphasizing the public face of a site while foregrounding such fine print as button icons and JavaScript code that make the site function (www.potatoland.org/shredder).

When combined with **cascading style sheet**s, the tag that *Shredder* exploits was originally intended to pin down the elements of a Web page so

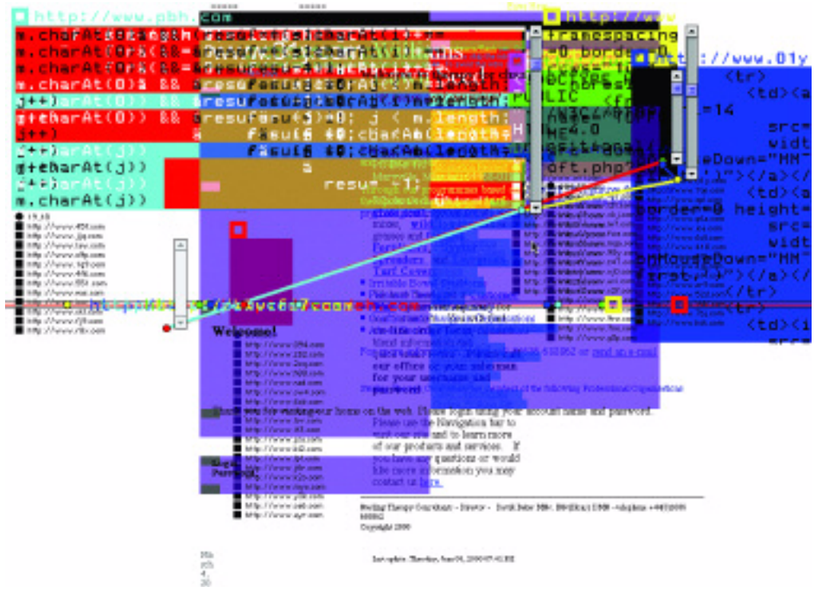
that designers could specify the same fixed page layout on different computer screens. *Shredder*, however, turns this page metaphor inside out by switching the placement of scripts and images to s of its own making, thus revealing what Webmasters and designers have deliberately concealed.

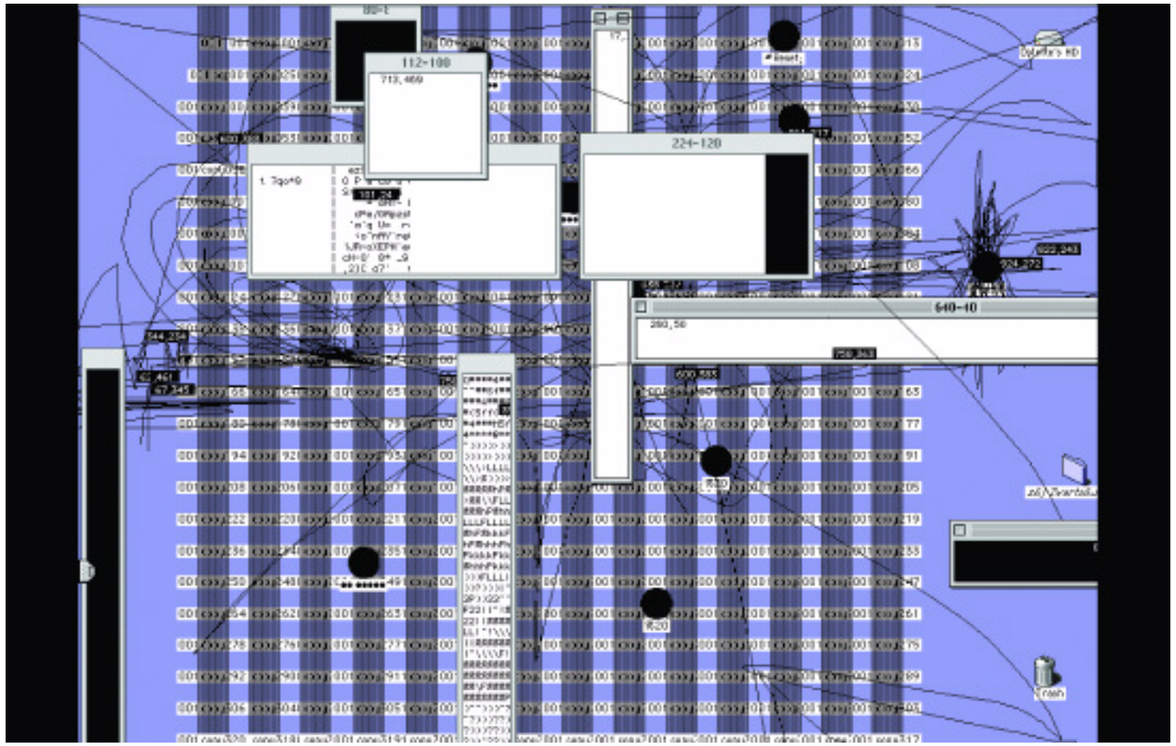
[[subA2]] **JavaScript**

If HTML is the skeleton of a Web page, Java is the muscle. Unlike tags like and <button>, which determine the location and style of static items on your screen, JavaScript is a fully functional programming language. Most Web pages employ it for superficial tasks like changing icons on mouse rollover, repositioning text or images in response to user clicks, and popping up new pages or error messages. JavaScripts can also handle sophisticated calculations or data processing, but some of the most successful misuses of the language focus on a single 'method' such as Math.random() or window.open().

One of the most notorious JavaScript methods is the ability to pop open a new browser window — a technique so overused by advertising companies and porn purveyors that around 2002 browsers began to offer the option of disabling this feature. While most popup and popunder windows are merely annoying, in the hands of Joan Heemskerk and Dirk Paesmans, the infamous glitch artists of jodi.org, they create absolute havoc. A visit to oss.jodi.org triggers a window.open() command that pops up a blank blue square. But the square is only visible for a fraction of a second, after which jodi's mischievous JavaScript executes a window.close() and reopens an identical blue square window a few pixels over. By looping this process in fast motion, jodi create the illusion of a square zigzagging across the screen, thus kludging an animation technique out of a handful of commands.

If the result were just a syncopated low-tech animation, oss.jodi.org would simply be an update on Jeffrey Kurland's *The Dotted Line*, a crude but compelling animation from the early 1990s in which mousing down on the browser's scroll bar animated





a series of dots in the manner of Thomas Edison's hand-cranked Kinetoscope. But jodi's runaway animation has no hand-crank to control it; the excessive popups stutter across the screen too quickly for most mortals to catch and close them manually. The only resort is to force-quit the browser or shut down the computer — surely one of the most perverse examples of narrative closure in the history of art.

oss is only one example of jodi's cornucopic misuse of code. Cascading sheets of green ASCII text over a black screen, flashing 404 error messages, proliferating form buttons and text boxes — this is what awaits the unsuspecting user who downloads one of their hacked games, inserts one of their CD-ROMs, or visits their Web site. Although much of jodi's work thrives on obfuscation and illegibility,[47] oss is one of their most legible works to date, for it misuses a single atom of code — the window.open() method — and for this reason it is also one of their projects most likely to agitate viewers into reassessing their relationships to their 'obedient machines'.