Code.surface || Code.depth

by Rita Raley

This essay begins by identifying a central idea in the critical discourse on code art and code poetry: code is a deep structure that instantiates a surface. The AP Project’s Jonathan Kemp and Martin Howse, for example, explain that their work makes “manifest underlying systematics,” that can make the digital “physical, audible and visible through geological computing.” In what sense, if at all, can we trace a computing operation down to a foundation, bottom, or core? Why do we maintain this cultural imaginary of code and how has it come into being? Moreover, how have the metaphors of software engineering – particularly the notion of structured layers and multitier architectures – been put to artistic use? The thematizing of layers, surfaces, and spatial metaphors has become quite intricate in new media writing practices, as I will demonstrate in a reading of “Lascaux.Symbol.ic,” one of Ted Warnell’s Poems by Nari, and recent projects by John Cayley, including Overboard and Translation. These readings, among others, will point to a logical tension between, on the one hand, the discourse of the foundational architecture of code, a “geological computing” that mines the depths to produce a geology (or a mythology) of surface and, on the other, the discourse of computational code in terms of inaccessible, inscrutable processes.

1. Code.surface || Code.depth
2. Lascaux.Symbol.ic
3. Reading code
4. Overboard
5. Translation
6. Writing for complex surfaces
7. Black boxing code

1. Code.surface || Code.depth

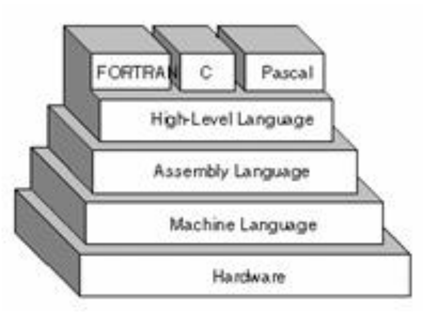
Jean-Luc Godard's *Le Mépris*/*Contempt* (1963) opens with a long shot of Giorgia Moll on a studio lot. She reads from a book (perhaps a script) as she walks along dolly tracks toward the front of the screen, tracked by Raoul Coutard's camera and an attending crew. The voice-over recounts the primary credits – actors, cinematographer, writer, editor, director – noting as well that the film was shot in Cinemascope and naming the lab where it was processed. Once the actress and crew reach the front of the screen, we hear a quote attributed to André Bazin on cinema 'substituting for our gaze' and the camera turns to face the audience, moving in for a close shot and incorporating us as spectators and filmic subjects. Such does *Contempt* appear to lay bare the material conditions of filmic production, a gesture of revealing echoed by the immediate jump to the 'real' beginning of the film, which features Brigitte Bardot in bed and spectacularly nude. The next scene will show us Moll's character walking in the very same studio lot, this time in a transparent rather than constructed frame. If the first calls our attention to mediation and ostensibly exposes the mechanism of filmic production, this scene takes on the point of the view of the camera within the frame and produces a sense of immediacy. However, we have not at the outset been privy to an exposure of the actual conditions of production. Rather, the opening of the film lays bare the symbolic conditions of production. What has been brought to the fore, or in the rhetoric of new media, the surface, is not the mechanism itself but a representation of the mechanism.

This filmic scene perfectly captures the impetus of contemporary code art, which is, put simply, to reveal codes, to make the mechanism of production visible to the viewer. As it would have been for Godard, such a move endeavors to puncture the insularity of the representational frame. But such an impulse is also partly corrective: even now, code artists might say, there is a tendency to regard the work of art as separate from the work of software engineering and situating code on the interface complicates the notion that a program is merely a tool with which to produce the 'real' art. The Whitney Artport CODEDoc show was in this respect paradigmatic in that it prompted viewers to consider algorithm and output in generative relation: since each "enter project" button was located at the end of the different programs, viewers had at a bare minimum to pass their eyes over the artists' code as they scrolled to the bottom to reach the link to the visual display.[i] Viewers were thus prompted to consider where they located cultural, artistic, and institutional value: with the code (instruction sets for translating a message from one symbolic form to another), execution (machinic process), or output (object). So, too, TOPLAP's "live coding" concerts combine coding and performance and write both under the sign of artistic improvisation; as their manifesto professes, "Live coding is not about tools. Algorithms are thoughts."[ii] But the investment in making code visible is far more pervasive and powerful than the question about the division of labor and disciplinary distinctions might suggest. It reaches beyond the problem of the work of art to the insight that "code is law" and, as such, an architecture for control by government and technocratic experts.[iii] Bringing code to the interface, moving it from background to foreground, in this respect bears a strong relation to free software and open source movements. (It might go without saying that codework's contemplation of the conditions of production does not usually extend to the machine itself, to the socio-economic problems of hardware sweatshops, nor to the ecological problems of e-waste.[iv]) As even the most preliminary web search would make clearly evident, code in the broad sense of programming languages has become an object and medium not only of artistic and literary production, but also of critical inquiry and political engagement.

Such a general statement, however, must surely seem commonplace after more than a decade of exposure to Jodi's work and the extensive critical discourse these artists have inspired. We can extract from an early analysis of their projects by Hans Dieter Huber the now-axiomatic notion that code is a deep structure that instantiates a surface: "In principle, all of the Internet-based works are based on the difference between code and surface. The source code represents a kind of notation or musical score that is interpreted by the computer when a page is called up by a specific browser such as Netscape, Internet Explorer or Opera. Like a virtual conductor or a symphony orchestra, the browser performs the score and displays it on the surface of the monitor. What we see is only the surface of a specific interpretation."[v] Huber is here discussing the relation between HTML source and interface but the tension between visible and hidden structures, between the surface and the depths, is paradigmatic. Witness Talan Memmott in a discussion of his notable *Lexia to Perplexia*: "The encoding is multi-layered. There is the code-base of the application, which certainly participates in the narrative construction of the work through interactive functionality. The code-base also bubbles through to the surface, to the superficial narrative – the readable text – by what you have called 'overprocessing.'"[vi]

In works such as Memmott's, code is brought, or "bubbles," to the surface as a static linguistic and aesthetic artifact rather than as a functional program. Such a move situates natural and programming languages within the same semiotic frame and presents the interface as a kind of "intersection of words and symbols," as the epigraph to Giselle Beiguelman's meta-critical work, *The Book after the Book / O Livro depois do Livro* indicates.[vii] The visual chaos of her piece suggests that the intersection or encounter is not without interference; as she explains, the layers, the linguistic and digital "substrata," leak into each other.[viii] The resulting text presents itself as something to be read while still alluding to the programming languages that generate the text on the interface, for example by including lines of binary code or symbolic elements of high-level programming languages such as forward slashes. For example, incorporating the double pipe, the logical 'or' condition (||), within a document written in English invites the reader to think in terms of an either/or structure, not an aporia exactly since the logic is substitutive (if the conditions of 'a' are not met, run 'b'), but at the very least pointing to a tension that may or may not be resolvable.[ix] Another rationale for the foregrounding of codes is provided by Jessica Loseby's *Code Scares Me*, in which she incorporates elements of the site's HTML so as to confront the strangeness of "what lies buried within the under texts," within the "depths."[x] Substrata and depth may be suggested by the display of codes and coding elements but, paradoxically, this type of code writing practice isolates the screen as surface. [xi] Beiguelman's *Book* points to this construction of pure surface: "Any page on the web seems to be only surface. The very metaphor of the screen with the page reinforces the assurance. Nevertheless, it is just an optical illusion. What is shown is not there. It is hidden. It is the source."[xii]

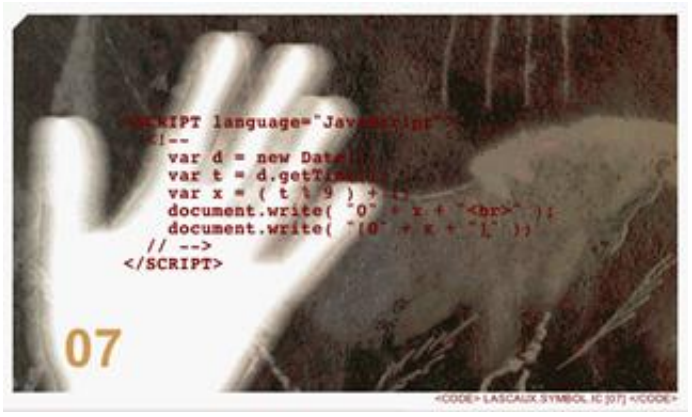
Exposing the mechanism of production, then, instantiates a surface or, as the I/O/D project's Matthew Fuller names it, a "visceral façade."[xiii] What the façade of the code surface masks is the deep structure of code, the tower of programming languages that descend from software to hardware. What is the deep structure of computing and are we able to see or otherwise access it? Are



there coding practices that can, as the AP Project's Jonathan Kemp and Martin Howse profess, "manifest underlying systematics," that can make the digital "physical, audible and visible through geological computing"?[xiv] In what sense, if at all, can we trace a computing operation down to a foundation, bottom, or core? To pick up on Kemp and Howse's metaphor, is such an excavation possible? Can we think in terms of a deep structure of code through which we can trace an archaeology of surface? As this essay will indicate, there is a logical tension between those who claim a foundational architecture for code and those who point to code as an inaccessible black box. For example, Fuller speaks to the "subscopic," "invisible," and "inscrutable" aspect of software implementation, to our inability to achieve anything like a comprehensive view of its operations: "it is worth noting that simply because they occur at the level of electrons the axes of software are impossible to find for the average user. Just as when watching a film we miss out the black lines in between the frames flashing past at 24 per second, the invisible walls of software are designed to remain inscrutable….these subscopic transformation of data inside the computer are simultaneously real and symbolic." If we wanted to construct a more apposite filmic analogy for this issue of exposing the mechanism of production and mapping its "geological" structure, then, we would perhaps have to subject the celluloid used for the opening of *Contempt* to microscopic examination.

To what extent is there a correlation between spatial metaphors of surface and depth and machinic architecture? Why do we maintain this cultural imaginary of code and how has it come into being? General perusal of the ACM proceedings of the 1950s and early 1960s indicates that programming had not yet evolved into software layers. Modular programs and subroutines, in other words, did not necessarily lead people to think in terms of building layers of abstraction. At what point, then, do notions of tiers or layers come into play? At what point do we begin to see people thinking in terms of building layers of abstraction? Martin Campbell-Kelly pointed me to the 1968 Garmisch software engineering conference for clues to the research and industrial environment of the period.[xv] Conference proceedings suggest that the notion of multiple layers of software emerges with structured programming and theories of abstract data types. In a working paper, "Complexity Controlled by Hierarchical Ordering of Function and Variability," Edsger W. Dijkstra emphasizes that "the software of our multiprogramming system can be regarded as structured in layers" (182).[xvi] He goes on to explain: "The subsequent ordering in layers has been guided by convenience and was therefore, as said, more hardware bound. It was recognized that the provision of virtual processors for each user program could conveniently be used to provide also one virtual processor for each of the sequential processes to be performed in relatively close synchronism with each of the (mutually asynchronous) pieces of I/0 equipment. The software describing these processes was thereby placed in layers above the one in which the abstraction from our single processor had to be implemented" (183). With the notion of multiple layers of software comes the notion that layers leak into each other: "what is put in layer 0 penetrates the whole of the design on top of it and the decision what to put there has far reaching consequences" (184).

2. Lascaux.Symbol.ic



```
<SCRIPT language="JavaScript">
  <!--
    var d = new Date();
    var t = d.getTime();
    var x = ( t % 9 ) + 1;
    document.write( "0" + x + "<br>" );
    document.write( "[0" + x + "]" ));
  // -->
</SCRIPT>
```

This brings me then to my central questions: How have the metaphors of software engineering – particularly the notion of structured layers and multitier architectures – been put to artistic use? What compels the discursive and physical instantiation of a surface? What is at stake in such an artistic project? The thematizing of layers and spatial metaphors has become quite intricate in new media writing practices, as we shall first see in a reading of "Lascaux.Symbol.ic," one of Ted Warnell's *Poems by Nari* ("by Nari" = binary). Initial decoding of the text requires the recognition of the connection between the displayed numbers on the one hand and the variables and arithmetic operators on the other. At first look, it appears that the script that executes the numeric text is displayed on the interface surface, as if the code that regenerates the numbers in the bottom left-hand corner and below the visual frame is not only visual but operational. "Lascaux" purports, then, to make all of the code processes of the visual poem manifest to the reader, to open its source and show us how it works. In fact, however, the code is placed within HTML comment tags that would generally be used to hide the script from older browsers.[xviii] It should be noted that my reading here requires the rather safe assumption that Warnell made a decision to comment out rather than turn the script itself into an image file. Here is an explanation of the comment tags on Lascaux's surface:

```
<SCRIPT language="JavaScript">
<!-- Hide the following script from old browsers:
var d = new Date();
var t = d.getTime();
var x = ( t % 9 ) + 1;
document.write( "0" + x + "<br>" );
document.write( "[0" + x + "]" );
// End the hiding here. -->
</SCRIPT>
```

The two forward slashes (//) indicate a comment and stop the JavaScript interpreter from compiling the line, which in an older browser would have resulted in the display of the script as page content. Comment tags would have been good etiquette at a time when programmers could assume that a fair number of users were using browsers not yet able to support JavaScript, which was first possible with a December 1995 release of Navigator 2.0 (JScript was developed for Internet Explorer version 3.0, released in August 1996).[xix] In that commenting out prevents compilation, here the Script tag situates the code lines as visual rather than functional. As such, it is content rather than etiquette. Like Godard's *Contempt*, then, "Lascaux" suggests a symbolic rather than an actual laying bare of the conditions of production. What is apparently brought to the surface and displayed for the reader is in fact commented out, hidden. And embedded within this comment tag is the intricate history of browser technologies.

Demarcating a break between the eras of “old” and “new” browsers deepens the historical narrative of a piece that situates prehistoric art and programming languages within the same artistic frame. The dominant visual backdrop of the piece is a black-and-white image of a horse from the cave wall at Lascaux, c. 15,000 bce.[xx] Layering then produces the effect of writing on the wall, as if the JavaScript were inscribed on its surface or even as if coding were a kind of tagging. There are limits perhaps to the comparison of cave paintings and code – certainly one would not want to suggest a transcendental signifying system that would render each legible – but their layering in this context reminds us of the extent to which each requires specialized literacy practices. Both are understood to be at once communicative and expressive, functional semiotic systems but also “art.” As the intermediary hand would imply, both are in a sense forms of writing requiring craft and technique. Its transparency reminiscent of the Xeroxed image, the representation of the hand also invites us to think in terms of tactility, touching the interface, leaving behind the ghostly traces of one’s presence.[xxi] The hand is thus cave writing in another sense, invoking the Australian aboriginal practice of leaving an imprint of the hand by spraying pigment over its surface.[xxii] Code-as-inscription necessarily emphasizes code’s materiality, an idea that would be familiar to critics such as Matthew Kirschenbaum, Katherine Hayles, and John Cayley.

Next we can turn our attention to the missing upper-left corner, which suggests that the visual frame is mounted. It is as if the cave wall is not in fact the background but itself mounted on another background, the default white screen of the browser window. Such is the complexity of the layering in this piece, then, that the very concepts of “foreground” and “background” are rendered fundamentally unstable. The missing corner in “Lascaux” also calls up another historical era: the near-antiquity of punch-card computing. Virtual archaeological relics in our current IT imaginary, punch cards are here analogized to cave walls, each a “new” medium for the recording of data.

```
//SYSUT2 DD DCB=C066.DIS.MACLIB(CONFFM),DISP=SHR
```

3. Reading code

In his programmable media and his critical essays on codework, John Cayley has endeavored to call our attention to the material difference between code addressed to a reader (pretend) and code addressed to the machine (genuine). The opening lines of a HyperTalk experiment illustrates that difference through the form of a poem-program that is operational within the realms of both natural and programming languages.

```
on write
  repeat twice
    do “global “ & characteristics
  end repeat
  repeat with programmers = one to always
    if touching then
```

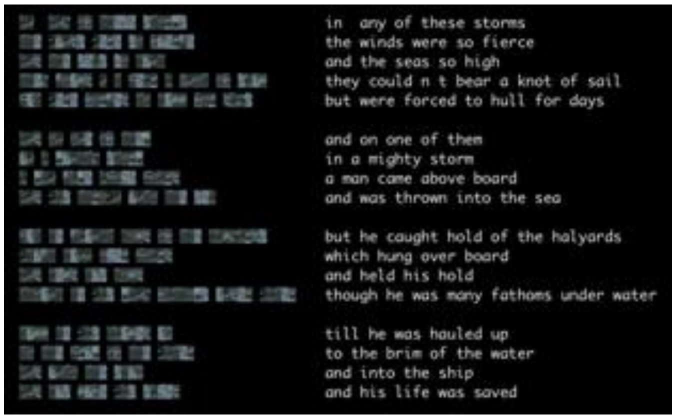
```
      put essential into invariance
    else
       put the round of simplicity * engineering /
synchronicity + one into invariance
    end if
     if invariance > the random of engineering and
not categorical then
      put ideals + one into media
      if subversive then
        put false into subversive
      end if
      if media > instantiation then
        put one into media
      end if
    else
      put the inscription of conjunctions + one into
media
    end if
```

In its complete form, this poem-program signifies and functions within both symbolic systems in that it is capable of altering either one; it is interpretable, working code that is also human-readable. Its address, then, is ambiguous. The code could alter the behavior of the system if it were included as one routine in a text generator, with the acknowledgment that its execution might be inconsequential.[xxiii] And the code could also be interpreted by a human reader, with the acknowledgment that the lines are less meaningful than they are allusive. Designed both for reading and for compilation, then, this text collapses and yet also situates the difference between program and poem. It prompts two different readings, two different interpretations.

Is code then semiotic? Is it, contra Ellen Ullman, a “text for an academic to read”?[xxiv] Do the linguistic sensibilities that have informed the development of programming languages mean that we can regard code as we would any natural language and analyze its grammar, logic, and rhetoric?[xxv] Or would thinking in these terms reiterate the fallacy of *The Matrix* and imply that One with special insight may render code absolutely legible, even figurative? I am thinking here of the moment that the Neo character realizes his gift: instead of seeing three Agents at the end of the hallway, he sees human figures composed of vertical strings of code. The failure of the film on this score is its inability to keep code within its proper signifying system: it must, rather, acquire dimensionality and be made to take human shape. Jutta Steidl writes in an essay for the ‘I Love You’ virus exhibition that “the hermeneutics of source code do exist”: if we were to read code properly, which is to say with the proper literary sensibility, then we would be able to recognize “the true beauty of human language.”[xxvi] Far from alone in expressing the general sentiment that program code is itself artistic material rather than the functional process by which the ‘real’ work of art is produced, Steidl would have rather less support for the notion that one could read code precisely as language, much less for the notion that an unmediated encounter with code as “the original, primal text” would be possible at all. While it is certainly the case that high-level programming languages such as HyperTalk are readable in a general sense and that linguistic knowledge is continually translated into putatively neutral programming structures, however, critics such as Adrian Mackenzie and Katherine Hayles have pointed to the limits of signification as it is conventionally understood with regard to code.[xxvii] So, for example, we might ask if the linear nature of signification still pertains in the context of a program that does not ‘read’ from left to right, or that is linear not to the reader but to the program as it executes. And therein we can locate the *difference* of the sign system of code: its executability, its operative transformation of a message from one symbolic form to another. Or to return to Cayley's experiment, and to stay within the perimeters of his work: code and language alike may amuse, astonish, inform, and delight; both may be written and read; both are performative and may initiate changes in the world; but one can be executed by the computer and one cannot.

Mackenzie notes that the "readability of code relates to execution, to how it circulates, how quickly it can be read and understood by other programmers, and how it affords revisions, modulations and modifications" (16). But at this stage we might ask: what of codes that are not executed or even code whose purpose is not to function but to crash?[xxviii] Is there not a crucial difference between code and computation? In the computing context, we can understand code in general terms as a sequence of commands that tell a computer what to do. But, as Michael Mateas and Nick Montfort's analysis of obfuscated programming reminds us, code may exist for years without ever being implemented on a computer.[xxix] (Alexander Galloway's entry of a fork bomb into the Whitney Artport CODEDoc show makes the same point about the difference between code-as-text and code-as-operation.[xxx]) Code then cannot ultimately be reduced to mere execution, not only in such cases as its function is precisely not to function, but also in such cases where it lies dormant. In this sense, Mackenzie's account of code in terms of the linguistic performative is inspired, indicating as it does a gap between expression and execution. (Thinking 'you're fired' is one thing; actually saying it in a boardroom is quite another.)[xxxi]

4. Overboard


in any of these storms
the winds were so fierce
and the seas so high
they could n t bear a knot of sail
but were forced to hull for days

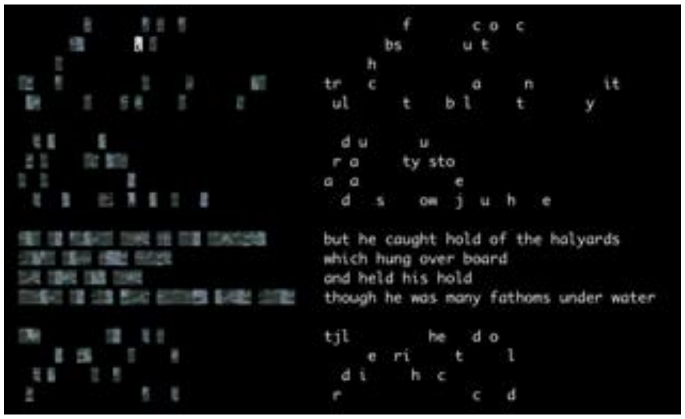
and on one of them
in a mighty storm
a man came above board
and was thrown into the sea

but he caught hold of the halyards
which hung over board
and held his hold
though he was many fathoms under water

till he was hauled up
to the brim of the water
and into the ship
and his life was saved

We can turn to Cayley's recent work in order to consider further the relations between code and text, surface and depth, as they are articulated in the context of new media writing. On the surface, *Overboard* performs continually evolving mutations of a verse passage adapted from William Bradford's *Of Plymouth Plantation*. In its interior, *Overboard* is an algorithmically generated text, a "kinetic language painting" characterized by its "operative performance."[xxxii] It is also "literal art": the name Cayley gives to an art practice that explores morphological and symbolic connections among words and letters.[xxxiii] This practice has evolved and become more complex from his earlier *riverIsland* to *Overboard* and the more recent *Translation*. In all, the screen space may be organized by stanza and poetic line, but the letter is the primary unit of analysis. *Overboard* often pairs letters that are proximate and/or bear a morphological relation to each other, e.g. "i" and "j" or "b" and "p," and these letters – the grid cells – operate in a sequenced transposition. Using relatively simple algorithms to produce a complex surface makes this project retroactively a perfect example for Hayles' analysis of the relations between analog and digital textuality. In "Simulating Narratives: What Virtual Creatures Can Teach Us," she explains that analogical relations are structured on a depth model; that is, the analogical requires links between the surface and depth units (13). For the analogical, complex codes produce a simple surface, and here we might think of the mythology of the Author that holds that a kind of complex interiority lends the text its depth. For the digital, on the other hand, a complex surface is produced by underlying simple models.

Though *Overboard* implies a teleological structure in its “ambient states” of “floating,” “sinking,” and “surfacing,” it does not definitively proceed from opacity to clarity, or from clarity to opacity. Indeed, a ‘complete’ realization of the text is markedly temporary: the Bradford text comes to the surface as an integral unit only briefly before the ‘proper’ letters begin to permutate and slip away. The effect of this is circularity, as if it were programmed with continual replay loops, but the algorithms in fact initiate an interplay between the random and the nonrandom. Although their primary unit of textual generation was the word rather than the letter, Jackson Mac Low and John Cage can be read as precursors to this kind of quasi-randomization of text.[xxxiv] In all we can see an interest in chance operations and an investment in the mechanisms by which pattern, structure, and order emerges spontaneously. In all we can see a visual presentation of rule-based behavior. However, the programmable aspect of Cayley’s text-generation procedures renders them practically and theoretically different from print-based experiments.[xxxv] The use of generative algorithms in his work results in texts that in a significant sense program and emerge from themselves. As he notes, chance operations and the accrual of data input mean that “the procedure ‘learns’ new collocations and alters itself” (“Beyond Codexspace” 180).

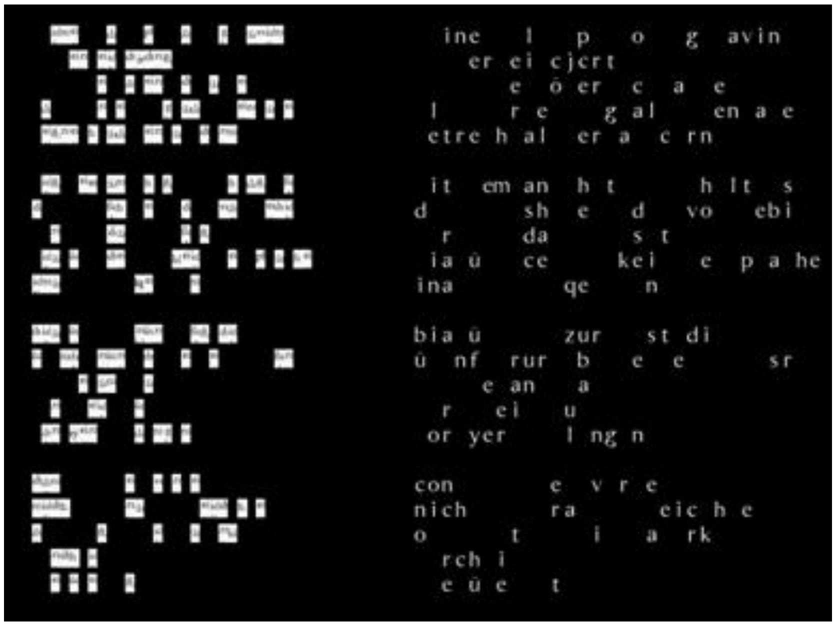


but he caught hold of the halyards
which hung over board
and held his hold
though he was many fathoms under water

While the morphological and even phonic relations among the letters serve to stabilize signification to a certain extent – insofar as the reader can intermittently pick up some of the patterns inherent in the transformation of the text – legibility is only partially insured on the side of production. That is, the time-based text sequences in *Overboard*, the “ambient states” of the piece, are generated by algorithms that are simple but that nevertheless produce complex semiotic effects.[xxxvi] To reiterate: rather than proceeding from legibility to illegibility, or from completeness to fragmentation, and thereby stabilizing its own instability, *Overboard* is inherently unstable. Cayley explains: “There is a stable text underlying its continuously changing display and this text may occasionally rise to the surface of normal legibility in its entirety. However, *Overboard* is installed as a dynamic linguistic ‘wall-hanging,’ an ever-moving ‘language painting.’ As time passes, the text drifts continually in and out of familiar legibility - sinking, rising, and sometimes in part, ‘going under’ or drowning, then rising to the surface once again.”[xxxvii] Rather than generating language per se, Cayley’s transliteral text generation procedures enact “liminal, hybrid linguistic phenomena.”[xxxviii] These linguistic phenomena are legible as language only to the extent that the reader/user understands them to be approximations of such. We might then reach even further and consider Cayley’s texts as autopoietic systems, self-generated and stable despite the continuous flow of matter and energy.

Overboard is not only a fluid textual sculpture, but also a fragmented visual image and sound composition. The thematizing of surface is not limited to the text sequences, however, but also performed by the visual image of an ocean surface rendered as a cut-up. The verso-recto split screen between oceanic and textual surfaces suggests that one medium resonates off of another, but there is not a strict 1:1 correspondence between the two. The page layout simulates a kind of translation and equivalence between the two sides, as if they formally recognized and responded to each other. Recognition or equivalence, however, occurs at the coding level, with bell sounds and the cursor linked conditionally to the appearance of certain instantiations of language.

5. Translation



I turn now to Cayley's *Translation*, another literal art project that in this instance "investigates iterative, procedural 'movement' from one language to another."[xxxix] Running the same algorithmic processes as *Overboard*, *Translation* cycles texts through the three states of floating, sinking, or surfacing. On the surface, *Translation* performs continually evolving translations – or symbolic translations – among English, French, and German versions of excerpts taken from both Proust and Walter Benjamin's essay "On Language as Such and the Language of Man."[xl] The verso features a scanned image of a page from a German-language version of Proust and the recto the Benjamin and Proust excerpts, suggesting perhaps that translations are being performed on the fly. Although it is possible to summon a complete English, French, or German text to the surface by keystroke, any act of translation would necessarily be both incomplete and symbolic. (The motifs of incompleteness, transformation and mutation carry through to the 'versioning' of the project, variations of which include Ukrainian translations and fairly monochromatic graphics in the place of the cut-up German codex.) Informed by the contemporary critical discourse on cross-cultural translation, and Cayley's practice as a translator, these translations enact the remainder, the incompleteness of linguistic exchange, and the spatial movements of language. The translations of Proust, for example, notably do not pass through English en route from French to German, as would be the case for any standard machine translation, informed as it would be by technologies that developed on the basis of Warren Weaver's understanding of translation as a problem of decryption. (In Cayley's descriptions of the coding of this work, source and target implicitly allude to, but do not directly reference, Weaver.) Rather, the visual representation of pages from the German translation implies a proximity to the original French that would not necessarily be implied by a transcribed version of the text.

The 'source text' for *Translation*, insofar as one can fix a source for a piece that calls into question the very status of a primary or originary text, is Benjamin's essay on language, the English-language version of which can be pulled to the surface with the shift-e command:

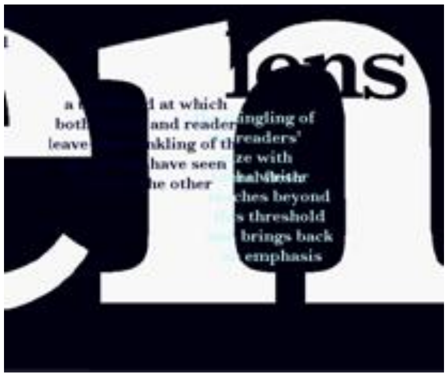
> Translation attains its full meaning in the realization that every evolved language can be considered a translation of all the others. By the relation of languages as between media of varying densities the translatability of languages is established. Translation is removal from one language into another through a continuum of transformations. Translation passes through continua of transformation, not abstract areas of identity and similarity.[xli]

In its basic operation, I will suggest, *Translation* is an enacting of Benjamin's conception of translation as the generation of language. The project thematizes and performs translation as transformation: one translation, one transformation, produces another.[xlii] Moreover, the translation from one sign into another is also the translation from one medium into another.

In that the source text is itself a kind of translation of Benjamin – Cayley has done some lineation – its status as a 'version' also calls into question the notion of an original. The adaptation is not quite as dramatic as the versification of Bradford in *Overboard* but here we must focus on what I take to be the most important of the changes: the omission of "(with the exception of the word of God)" from Cayley's transcription of Benjamin. The 'word of God' is for Benjamin the originary language in a mystical sense, the pure language in which word and referent coincide. By omitting the reference to the word of God, Cayley eliminates, or rather simply appears to eliminate, the transcendental. Excluding the word of God, in other words, leaves no non-translated language. Rather, all languages are translations of other languages. But the omission of the Benjaminian transcendental also leaves open the possibility that there might be a substitute operating, as it were, beneath the surface. In other words, for a critic deeply invested in the distinction between pretend and genuine code, between symbolic and operational code, code itself must certainly be understood as a generative sign system, as a language that generates all others and forms rather than "passes through continua of transformation." That the text describing the transformation is the very text being transformed, however, should alone indicate that Cayley would refuse the claim for code as a transcendental sign system. In other words, eliminating the transcendental, the metacode, suggests its impossibility.

6. Writing for complex surfaces

Two of Cayley's recent works in progress, *Lens* and *Torus*, will introduce another aspect of the code surface: inscription on complex surfaces and its corollary, the letter becoming its own complex surface. *Lens* is an interactive QuickTime piece in which the word "Lens" is literally that – a movable, scalable, seemingly translucent lens through and on which one can read the four poetic, epigrammatic stanzas of the work. As Cayley describes it: "by making a letter large enough within the programmatic structures of lens, the region of colour defining the letter-shape becomes an entirely different type of surface – it becomes a surface of inscription for other texts that had been perceived 'underlying' it. In doing so, literal surfaces subvert our experience of space and relative distance. Surfaces that were 'in front' now form surfaces for



other texts."[xliii] One of the stanzaic texts in this piece – "the letter is a threshold" – provides further guidelines for reading. The letter crosses the threshold from the two-dimensional space of printed text to the three-dimensional space of the virtual word, its volumetric projection in this work achieved by the shifting of scale. The material signifier, the form and shape of the letter, is itself a threshold. The lens is also a threshold in the sense of a portal; that is, the dynamic word-object "lens" functions as a portal into the text *Lens*. It is the functioning within *Lens* that gives the word "lens" meaning. This, then, is a poem with a portal into itself: quintessentially technotextual in its self-reflexive engagement with its own inscription technologies.[xliv] A threshold is also situated between writer and reader, where both are said to "leave some inkling of the glory they have seen within the other." The threshold then bears material traces of reader and writer. The writer reaches back beyond the letter as a single unit to the other, and the meaning of 'other' fluctuates: the writer and reader are other but so, too, is the letter.

Cayley's critical and aesthetic exploration of the literalizing of the letter continues in his Cave project *Torus* (2005-). Like his previous work *Translation*, *Torus* incorporates fragments from English and French editions of Proust's *The Way by Swann's* and performs continually evolving translations – or symbolic translations – among English, French, and German versions of excerpts of the text. There are five textual fragments at any one time on the vanes of the torus; proximity establishes some relation among them and they are also dynamically altered with the transliteral letter substitutions that are Cayley's artistic signature. As with his previous work, there are moments in the torus when the text is in its "natural language state," or, to use the rhetoric of *Overboard*, when the text is "on the surface." At these moments, too, the recitation of the text is quite clear; in other moments the sounds of the recitations are densely layered and the individual voices difficult to distinguish. The reader can penetrate to the inside of this three-dimensional text; she can "seem to be inside an inside," at which point the text is silent.[xlv]

Cayley has a long-term critical interest in what he calls "writing for complex surfaces"; and the torus, the donut-shaped, closed surface that is the product of two circles would be precisely that. Invoking non-Euclidean geometry in a virtual environment situates the letter in and on a three-dimensional space without edges or vertices. Cayley points out during a narrated video of the project that "letters in the *Torus* have no thickness"; that is, the reader "cannot see their rear surfaces nor even view them obliquely." Here the letter is flat but it is inscribed on a surface of complex folds. We can also contrast the flatness of the letters in *Torus* with the volume that words appear to attain in *Lens*. If words inscribed on the complex surface of *Torus* are without depth, words in *Lens*, by contrast, become themselves complex surfaces.

7. Black boxing code

I mentioned at the outset that there was a logical tension between, on the one hand, the discourse of the foundational architecture of code, a "geological computing" that mines the depths to produce a geology (or a mythology) of surface and, on the other, the discourse of computational code in terms of inaccessible, inscrutable processes. I turn now to the question of code as a black box whose inside cannot be penetrated and begin with the pursuit of universal translatability, which, we might agree, is one of the logics of new media. This is transcodification as media theorist Vilém Flusser articulates it in his *Writings* [*Die Schrift*], the translation from one message into another, the translation from a denoting code, which has a singular meaning, to a connotating code, which is open and ambiguous.[xlvi] Without the possibility of transcodification, we would be left with the notion that either a singular, closed message or a multivalent, open message in some way reflects the natural order of the world. This is not to say, however, that transcodification is without limits, that it allows for total transmissibility or that there is a 1:1 correlation between messages. Rather, as Flusser explains, "one universe may have two or more codes to communicate messages about it, and that in some codes there may be an overlap of universes. They are partly translatable into certain codes, and partly not. And the limitation of translations show that no code refers to all the universes, and no universe is referred to by all codes" (14). He engages the idea of a metacode in his *Philosophy of Photography*, in which he unambiguously proclaims such a universal, transcendent code impossible: there can be "no 'final' program of a 'final' apparatus since every program requires a metaprogram by which it is programmed. The hierarchy of programs is open at the top."[xlvii] Here we might see a clear echo of Cayley's treatment of Benjamin and understand again why he has omitted the transcendental word of God from his transcription.

There can be no metacode or totalizing apparatus situated in an outside, but this would not preclude our recognition that the apparatus remains a black box and in this sense ultimately unknowable.[xlviii] Functionaries control and are controlled by it, but its interior remains impenetrable. A comprehensive overview from a position of topsight is impossible and the smallest particles are also inaccessible: "apparatuses that, on the one hand, assume gigantic size, threatening to disappear from our field of vision (like the apparatus of management), and, on the other, shrivel up, becoming microscopic in size so as to totally escape our grasp (like the chips in electronic apparatuses)" (*PP* 21). What Flusser makes apparent is that the interior, the black box, is not only impenetrable but also not subject to modification.[xlix] There is thus something indeterminable about the photographic apparatus in Flusser's analysis, something within the black box that cannot be manipulated. His analysis of the unknowable, indeterminate apparatus strongly correlates with Friedrich Kittler's critique of the GUI as a "system of secrecy" that hides "a whole machine from its users" by concealing its depths or bottom layers.[l] During "the descent from software to hardware," then, we move "from higher to lower levels of observation [that] could be continued over more and more orders of magnitude" (150).

In the introduction of assembly and machine codes, we have lost a great deal: our capacity to see and alter the functioning of the mechanism and thus in a certain sense our capacity to grasp the entirety of our writing practices, the sum total of actions initiated and completed by a single keystroke.[li] Recognizing that the bottom layers are not simply concealed behind a curtain that has only to be thrust aside in order for us to see the wizardry underneath, Kittler nonetheless points to the conscious construction of such a barrier: "these layers, which like modern media technologies in general, have been explicitly contrived in order to evade all perception. We simply do not know what our writing does" (148). Again, what we cannot reconstruct, what we "simply do not know," is the precise sequence of human and machinic processes that result in our seeing our own words appear on a screen before us. In his reading of Kittler, Mackenzie has recourse to the rhetoric of "backgrounding" with respect to code's invisibility, both in a technical sense and with respect to its embedding as ideology within the social fabric. It is invisible because it functions within "inaccessible interior spaces" over which no "panoptic view" can be made available (25, 28). This is a problem not only of space but also of time. Analogizing the hierarchy of programming languages to one-way functions in mathematical cryptography, Kittler explains that "such functions, when used in their straightforward form, can be computed in reasonable time, for instance, in a time growing only in polynomial expressions with the function's complexity. The time needed for its inverse form, however (that is, for reconstructing from the function's output its presupposed input), would grow at exponential and therefore unviable rates. One-way functions, in other words, hide an algorithm from its result" (151). Whether conceived as "secret," "inaccessible," or an imperceptible background element, the 'deep' layers of software, the bottom floors of the tower of the programming languages, elude our cognitive reach. Matthew Fuller is on the same page as Kittler when he notes that "the axiomatics that channel and produce the behaviour necessary for use of computers happen at both human and subscopic scale," which is to say that we can produce a coherent and empirically grounded media archaeology of new media writing with respect to the GUI but such an archaeology would come up against a certain limit at the "lower levels of observation."[lii] The representation of codes, whether binary, assembly, or high-level, can therefore only ever be that: a representation of what is happening at the machine level.

The opacity of code holds true even at the pragmatic level of programming. This is partly the result of the massive proliferation of coding languages – "code babble" as Mackenzie names it (25). But it also results from issues of programming style that mean, as Alan Sondheim notes, that "you'd have to be the author to follow it."[liii] Katherine Hayles has a statement that thoroughly documents the extent to which code is opaque even (or especially) to those who write it:

> people who have spent serious time programming will testify that nothing is more difficult than to decipher code that someone else has written and insufficiently documented; for that matter, code that one writes oneself can also become mysterious when enough time has passed. Since large programs – say, Microsoft Word – are written by many programmers and portions of the code are recycled from one version to another, no living person understands the programs in their totality….In the case of evolutionary algorithms where the code is not directly written by a human but evolved through variation and selection procedures carried out by the machine, the difficulty of understanding the code is so notorious as to be legendary.[liv]

Code may be mysterious, cryptic, and in a sense unknowable, but it is, as Warnell's "Lascaux Symbol.ic" reminds us, made. Analogizing the cave painting to code, "Lascaux" reminds us that the hand – craft, skill, technical expertise – comes in between code and surfaces of inscription, here the wall of the cave. Code may in a general sense be opaque and legible only to specialists, much like a cave painting's sign system, but it has been inscribed, programmed, written. It is conditioned and concretely historical. Whether or not non-human agents have had a 'hand' in its formulation, code remains not only a constructing force but also that which is constructed. Mackenzie has a variation on this insight: "Code can be read as permeated by all the forms of contestation, feeling, identification, intensity, contextualizations and decontextualizations, signification, power relations, imaginings and embodiments that comprise any cultural object" (*CC* 5).

In Flusser's typology, there are three types of codes: visual (alphabetic), auditory (spoken language, music), and mixed audiovisual (theater, televisual). These codes then have three structures: diachronical, the ordering in linear sequence, such as is the case with languages and alphabets; synchronical, the ordering on surfaces, as is the case with ideogrammatic writing; and the three-dimensional synchronical, which is ordered in space, as is the case with architecture (*W* 15). The late twentieth century has brought about a crisis, Flusser suggests, in that writing in the diachronical sense of "lining-up of letters and other writing signs" faces a planned obsolescence. Less convenient for storage, less adept at transmitting information than the new codes, "the codes of writing, like the Egyptian hieroglyphs, or the Indian knots" are likely to be "put aside," to give way to the codes that improve the production, circulation, and reception of knowledge. Kittler puts this even more starkly: "we do not write anymore," he announces, writing's "last historical act" the design of the first microprocessor on blueprint paper (147). "At its alphabetic beginning, a camel and its Hebraic letter gamel were just two and a half orders of decimal magnitude apart," but now, "our writing scene may well be defined by a self-similarity of letters over some six orders of decimal magnitude" (147). On or around 1968, the year of ruptures, writing in the sense of manipulating alphabetic letters on a page comes not only to be hidden but to disappear. While writing seems to have no future, it is nonetheless associated with a historical consciousness, in fact brings that consciousness into being. Since writing is linked to historical consciousness, code is by implication linked to post-historical consciousness. The possibility of transcodification, of converting 'older' media forms into codes, presents writing not only with the taint of its own obsolescence but also with two routes away from itself – back to the image or forward to the code, "back to the imagination or forward into calculation." What Flusser's *Writings* suggest, however, is that "these two directions can merge surprisingly into one another: figures can be computed to images. From textual writing/thinking we can try to escape into imagined calculations."[lv] It is toward this possible future for writing that Ted Warnell's visual poetry and John Cayley's transliteral projects gesture.

---

[i] CODeDOC show curated by Christiane Paul (September 2002), http://artport.whitney.org/commissions/codedoc/index.shtml. A related text is Robert Nideffer's, *The Fine Art of Appropriation* (UCSB, 1997), which met university requirements for a printed and bound MFA thesis by submitting the HTML code used to produce his visual artwork and therein posing compelling questions about code as both mechanism and object of knowledge.

[ii] Not only a nice reversal of 'laptop,' TOPLAP is an acronym for (Temporary|Transnational|Terrestrial|Transdimensional) Organisation for the (Promotion|Proliferation|Permanence|Purity) of Live (Algorithm|Audio|Art|Artistic) Programming). See http://toplap.org/.

[iii] Lawrence Lessig, *Code and Other Laws of Cyberspace* (New York: Basic Books, 1999).

[iv] As just one example of the critical investigation of e-waste, see Lisa Parks, "Falling Apart: Electronics Salvaging and the Global Media Economy," *Residual Media*, ed. Charles Acland (Minneapolis: University of Minnesota Press, 2006).

[v] "Only!4!!!!!!!!!!!!!!!!!!!!!!4-for YOUR Private Eyes. A structural analysis of http://www.jodi.org," http://www.hgb-leipzig.de/ARTNINE/huber/writings/jodie/indexe.html

[vi] Mark Amerika, "Active/onBlur: An Interview with Talan Memmott," http://trace.ntu.ac.uk/newmedia/interview.cfm

[vii] http://www.desvirtual.com/thebook/english/epigrafe.htm

[viii] http://www.desvirtual.com/giselle/english/prologue.htm

[ix] The incorporation of elements of coding languages is present in print as well. Two recent examples: Salvador Plascencia's *People of Paper* presents binary code as readable language and Mark Z. Danielewski's new novel, *Only Revolutions*, brings the double pipe into play as a cryptic shorthand for his title (OR).

[x] http://www.kanonmedia.com/news/nml/code.htm

[xi] Also see Katherine Hayles's analysis of print and digital textuality in terms of metaphors of surface and depth in "Print is Flat, Code is Deep: The Importance of Media-Specific Analysis," *Poetics Today* (Fall 2001). For exploration of the architectural idea of "deep surface" in the context of new media, see Stuart Moulthrop, *Deep Surface* and Lev Manovich, *The Language of New Media*, 31-34.

[xii] See http://www.desvirtual.com/thebook/english/text.htm

[xiii] Fuller, "Visceral Facades: Taking Matta-Clark's Crowbar to Software" (1997), http://bak.spc.org/iod/Visceral.html. All subsequent excerpts from Fuller are taken from this essay (no pagination). On I/O/D's *Web Stalker*, see Josephine Berry, "Bare Code: Net Art and the Free Software Movement," http://netartcommons.walkerart.org/article.pl?sid=02/05/08/0615215&mode=thread

[xiv] http://nodel.org/orgs.php?ID=99

[xv] Personal email, February 21, 2006.

[xvi] Peter Naur and Brian Randell, eds., *Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968* (Brussels: NATO, 1969). Also see the d'Agapeyeff's inverted pyramid (23).

[xvii] Peter Naur and Brian Randell, eds., *Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968* (Brussels: NATO, 1969).

[xviii] *JavaScript Guide* for JavaScript 1.1 http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/getstart.htm#1006443.

[xix] http://en.wikipedia.org/wiki/JavaScript

[xx] The image appears to have been taken from the ceiling of the Axial Gallery. See *Virtual Lascaux*, http://www.culture.gouv.fr/culture/arcnat/lascaux/img/da-plafond.jpg. For a fuller description of the cave paintings, see Georges Bataille, *Lascaux, or the Birth of Art*, trans. Austryn Wainhouse (Lausanne: Skira, 1955). The text-only version of Warnell's poem was part of an online exhibition, *Our Digital Lascaux*, curated by Jennifer Ley (July 2000). Available from http://www.heelstone.com/lascaux.

[xxi] Ley similarly notes that "our digital handprints cover the walls of this, our common Lascaux." http://www.heelstone.com/lascaux/about.html

[xxii] Warnell explains that he had in mind this particular cultural context. Personal email, April 14, 2006.

[xxiii] This idea might be profitably compared to Geoff Cox, Alex McLean, and Adrian Ward, "The Aesthetics of Generative Code," which comments on the inconsequential, prosaic execution of many code poems, such as those written in Perl. See http://www.generative.net/papers/aesthetics.

[xxiv] Ellen Ullman, "Elegance and Entropy: Ellen Ullman Talks with Scott Rosenberg About What Makes Programmers Tick," *Salon* (October 9, 1997),

http://www.salon1999.com/21st/feature/1997/10/09interview.html.

[xxv] See Friedrich Kittler, "Protected Mode," *Literature, Media, Information Systems: Essays by Friedrich A. Kittler*, ed. John Johnston (Amsterdam: G&B Arts International, 1997); Raley, "Machine Translation and Global English," *The Yale Journal of Criticism* 16:2 (Fall 2003).

[xxvi] Steidl, "If () Then ()," digitalcraft exhibition, "I love you - computer_viruses_hacker_culture" (2003), http://www.digitalcraft.org/?artikel_id=293.

[xxvii] Katherine Hayles, *My Mother Was a Computer: Digital Subjects and Literary Texts* (Chicago: University of Chicago Press, 2005); Adrian Mackenzie, *Cutting Code: Software and Sociality* (New York, Peter Lang, 2006).

[xxviii] Alan Liu writes extensively of virus art in these terms in *The Laws of Cool: Knowledge Work and the Culture of Information* (Chicago: University of Chicago Press, 2004).

[xxix] Mateas and Montfort, "A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics," DAC paper (2005). In contrast, Florian Cramer's *Word Made Flesh: Code, Culture, Imagination* (Rotterdam: Piet Zwart Institute 2005) asks whether execution is a process particular to the computer. Available from http://pzwart.wdka.hro.nl/mdr/research/fcramer/wordsmadeflesh/.

[xxx] http://artport.whitney.org/commissions/codedoc/galloway.shtml

[xxxi] See "The Performativity of Code: Software and Cultures of Circulation," *Theory, Culture & Society* 22:1 (2005), 71-92.

[xxxii] Cayley, "*Overboard*: An Example of Ambient Time-Based Poetics in Digital Art," *dictung-digital* 2 (2004), http://www.dichtung-digital.com/2004/2-Cayley.htm.

[xxxiii] See Cayley, "Literal Art," *First Person: New Media as Story, Performance, Game*, eds. Noah Wardrip-Fruin and Pat Harrigan, republished in *electronic book review* (2004), http://www.electronicbookreview.com/thread/firstperson/programmatology. Also see Cayley, "Inner Workings: Code and Representations of Interiority in New Media Poetics," *dictung-digital* 3 (2003), http://www.dichtung-digital.com/2003/issue/3/Cayley.htm.

[xxxiv] Cayley persuasively argues that the "digital instantiation" of his work makes for substantive, "non-trivial differences" between his text-generation procedures and those of Emmett Williams, Mac Low and Cage, which achieve a relative fixity through print: "any aleatory of 'chance operation' aspect of such work is only *fully* realized in a publication medium which actually displays immediate results of the aleatory procedure(s). Such works should, theoretically, never be the same from one reading to the next (except by extraordinary chance)" ("Beyond Codexspace" 173).

[xxxv] For a contemporary instance of procedural text generation that works with software, see Jim Carpenter, "Electronic Text Composition Project," Slought Foundation, http://slought.org. The ETC project "uses a probability-based approach to constructing syntactic constituents," which, "absent authorial intent and divorced from any underlying message, assume their status as poems only as they are read." As with Cayley, legibility in a general sense is insured on the side of production.

[xxxvi] This is to emphasize a stark contrast (which, though obvious, is still useful) between the dynamic movement of the letters in Cayley's piece and the kinetic visual poetry of writers such as Brian Kim Stephans and Ana Maria Uribe who work in authoring environments such as Flash.

[xxxvii] Cayley, "*Overboard*: An Example of Ambient Time-Based Poetics in Digital

Art."

[xxxviii] Personal email, February 14, 2006.

[xxxix] All published versions of the project and description available from http://www.shadoof.net/in/translation.html.

[xl] "On Language as Such and the Language of Man," *Walter Benjamin: Selected Writings, Volume 1, 1913-1926* (Harvard UP, 1996), excerpted text pp. 69-70.

[xli] The following line – "The translation of the language of things into that of man is not only a translation of the mute into the sonic" – bears an interesting relation to the generative sounds of *Translation*.

[xlii] Jakobson's thinking on the "Linguistic Aspects of Translation" – that "the meaning of any linguistic sign is its translation into some further, alternative sign" – would also be a precursor to this operative theory of translation (56).

[xliii] Project description of *Lens* available from http://homepage.mac.com/shadoof/lens/lens.html.

[xliv] See Katherine Hayles on technotext in *Writing Machines*. Janet Zweig's flip-book *Sheherezade* (1988) uses magnification to produce a similar portal-effect. Finally, a typology of words and letters as spaces would include illuminated manuscripts, pop-up books, and even the Narnia chronicles.

[xlv] See the QT video with voice-over narration, available from shadoof.net. This is soon to be replaced by a video online at TIRW (August 2006).

[xlvi] *Writings*, trans. Erik Eisel (Minneapolis: University of Minnesota Press, 2002).

[xlvii] Flusser, *Towards a Philosophy of Photography*, trans. Anthony Mathews (London: Reaktion, 2000), 29.

[xlviii] "Apparatuses are black boxes that simulate thinking in the sense of a combinatory game using number-like symbols; at the same time, they mechanize this thinking in such a way that, in future, human beings will become less and less competent to deal with it and have to rely more and more on apparatuses. Apparatuses are scientific black boxes that carry out this type of thinking better than human beings because they are better at playing (more quickly and with fewer errors) with number-like symbols" (*PP* 32).

[xlix] Also see Michael Hardt and Antonio Negri on FOSS: "Since proprietary software owned by corporations does not expose its source code, the proponents of open source maintain that not only can users not see how the software works but they also cannot identify its problems or modify it to work better" *Multitude (New York: Penguin, 2004),* 302.

[l] "There Is No Software," *Literature, Media, Information Systems: Essays by Friedrich A. Kittler*, ed. John Johnston (Amsterdam: G&B Arts International, 1997), 151. All subsequent references are to this essay.

[li] On the other side of this question, Gene Kan writes of the advantages of Gnutella as operating on the surface, requiring little specialist knowledge for basic use.

[lii] Ibid.

[liii] Email communication, "essay on codework" (February 2, 2004).

[liv] "Traumas of Code," *Critical Inquiry* (Fall 2006), xx.

[lv] Afterword.
