On Codework: A Phenomenology of an Anti-Genre

Talan Memmott

1. Codework as Phenomenology

In his essay "Change the Object Itself" (1971), Roland Barthes states that "myth consists in overturning culture into nature or, at least, the social, the cultural, the ideological, the historical into the 'natural' […]" (p. 165). We could read this quote as a recognition that culture itself is a set of abstract codes and functions that position man outside of nature and that the purpose of mythology is to make these codes understood in relationship to nature. In addition, we could consider culture and its byproducts should as an originating technology of sorts that separates man from nature; it is through this separation, and awareness of it, that man becomes Man. To return to nature, then, with a nod to Rousseau, is to loosen the constraints of culture until one becomes unaware of the strata of codification and intermeshing cogs of protocol between the social, cultural, and ideological. Though Barthes is most likely directing his comments to something entirely different – indeed, the entire essay is a response to his earlier work on mythologies – the quote could have potential implications for the electronic literary practice of what is called codework and its relationship to computing culture and digital culture in general.

In an issue of the *American Book Review* (June 22, 2001) that focused on codework as a practice, Alan Sondheim, the originator of the term, claims that "Code refers to a translation from natural language to an artificial, strictly defined one" (1). To a certain extent this is a reversal of Barthes's claim for mythology – which, according to him, transforms culture (and its

codes) into nature, or the natural – but it does present an opening in which mythology in the Barthesian sense can be contrasted with Sondheim's notion of the practice(s) of codework, which is perhaps more a phenomenology of computer-based inscription than a genre of electronic literature or a specific writing practice.

Sondheim recognizes the definition he has provided as necessarily narrow and expands further on the idea of code and codework, stating:

> "Code" can refer to just about anything that combines tokens and syntax to represent a domain. In a sense, natural language encodes the "real," gives us the ability to move in environments constantly undergoing transformation. (1)

He also reminds us that:

> the syntax of Morse code […] has no room for anomalies or fuzziness. Computer programming generally requires strictly defined codes that stand in for operations that occur "deeper" in the machine. (1)

What is perhaps most interesting in these two quotations is that they interplay open and closed notions of code and codework – one semiologically expansive, the other technologically defined. In either case we are still talking about code as a token or stand in for deeper operational correspondences – be they cultural or computational. Through this, Sondheim points to the artificiality of language itself by stating that natural language encodes the "real." That the real is presented in quotations is telling from a phenomenological perspective since it puts into question any originating position outside of language, or – in Sondheim's broad view – outside of code. Origin, then, is replaced, as Barthes already demonstrates in his 1957 work *Mythologies* and again in his 1971 essay, with *doxa* – a whole set of doctrines that are negotiated through culture and language. As such, any discussion of natural language and the natural itself is already made

unnatural, codified by the series of protocols through which the notion is processed and filtered.

We could now perhaps rewrite the quote from Barthes to read something like this: myth consists of the recoding of culturally encoded ideas of nature through a decoding and re-encoding of cultural codes. I understand that this heavily parsed phrase may seem to lead us nowhere, or lead us into recursive loops of consideration for encoding and decoding. That being said, it does indicate a fundamental difficulty in defining codework as simply a genre of electronic literature. As stated above, I believe that codework should not be considered a genre, but an evidentiary phenomenology of computer-based inscription. Almost any work of electronic literature may fall under the general rubric of codework based purely on the material location of the authorial/applied/readerly transaction. Indeed, works that have been considered codework by a number of scholars vary widely in their intentional, aesthetic, and procedural constructions.

Later in his introduction to codework, Sondheim proposes a three-pronged ontology for the form that includes the following criteria: a) works using the syntactical interplay of surface language, with reference to computer language and engagement; b) works in which the submerged code has modified the surface language – with the possible representation of the code as well; and, c) works in which the submerged code is emergent content (p. 4). As you will note, collectively these criteria have built in paradigmatic redundancies. We see a differentiation between natural or surface language and computer languages or code, with the latter imagined as submerged – instrumental in the realization of the work but intended to remain unseen. What is provocative in Sondheim's vision of codework is that the code does emerge, it is made visible, and commingles with natural language. Sondheim indicates various formulas for this commingling of surface and submerged, natural and coding languages – through syntactic

interplay, surface modification, and code as content – citing a number of practitioners whose work he sees operating under each of the conditions he has outlined.

Among the practitioners listed by Sondheim is the Australia-based internet artist Mez (Mary-Anne Breeze). Mez is best known for a writing style that is heavily interrupted by square brackets, full stops, IRC (Internet Relay Chat) abbreviations, and the formatting of various programming languages – something she calls mezangelle. Sondheim places Mez's work under the first prong of his ontology, believing that mezangelle operates primarily though a syntactic interplay between surface and computer language. What is perhaps more significant in Sondheim's inclusion of Mez's work under this particular prong of his ontology is that the conditional aspects in her work are not only about an interplay between surface and submerged texts but also about engagement. Or, that this interplay is the result of engagement with the Internet apparatus, based in a willing and sustained participation.

It is through understanding that the syntactic strata of natural and coding language correspond, cooperate, and collide in codework that the discussion can shift from considering its originality in the literary sense to its residual or forensic value in relationship to the network. And, through this shift, we can start to understand codework not as a genre, but as an evidentiary phenomenology in which the *doxa* of the apparatus is foregrounded and naturalized.

2. Inscription as Encryption

Much of Mez's work, as well as much of the codework produced from 1996-2003, made use not only of fragments and syntactic elements of code but also of appropriated email and email list correspondence. As I argued in my own essay in the codework issue of the *American Book Review* (June 22, 2001), this use of email lists for the source and distribution of content

places the author at the fulcrum between correspondences – as a sort of human shuttle, processor, or mediator between dispersals across the apparatus, through the network (see Memmott). This form of conductivity, rather than any intentional drive toward literary or experimental text production, locates the text outside of poetry. Certainly, codework can be critically endowed with a poetics, but as Sandy Baldwin claims, "the qualities typically emphasized by critics in (these) works are not qualities of digital poetry, but are instead part of the phantasmatic role poetics plays for criticism" ("Against Digital Poetics" n.p.). Thus, the critical engagement with codework as literary text has, for the most part, been focused on the production of mythologies residing outside of the work itself.

Other than Sondheim, two of the most prominent scholars who have addressed codework are Rita Raley and John Cayley. Though both keep their definitions of codework general and provisional, prefacing their descriptions with "broadly" (Raley, "Code.surface || Code.depth" n. p.) and "potentially" (Cayley, "The Code is Not the Text (Unless It Is the Text)" n. p.), once we begin to dig into their theories we discover some rather significant differences in terms of ontological privilege.

To a certain extent Raley is interested in surface effects and the disruptions that the intervention of code causes for literary text, while Cayley is more interested in the maintenance of the executable aspects of code and its consideration in relationship to literary production. It should be added here that John Cayley is not only a scholar but a practitioner of electronic literature and this may have some effect on how he positions the procedural and computational. As Cayley and Raley would likely agree, neither interest nor condition are mutually exclusive: there are codework pieces that operate on both levels and the ontological differences are primarily directed toward issues of address. Is the code addressed to the human reader or to the computer?

These kinds of arguments of address are important, but maintain the division between surface and submerged language forms and presuppose a distinction between the reader of the text and the operator of the machine. At the computer terminal the reader is also the operator. The code does not execute unless the machine is turned on and there is a human interacting with the device. In Mez's case, however, as Raley indicates, "If 'net.wurked' life requires a cognitive adaptation and naturalization to the machine, her 'net.wurk' aims to disrupt its disciplinary and regulatory 'sensory reverberations' and offer instead an 'infoalert': informatic reverberations that shock and thus gesture toward new, and potentially liberatory, modes of cognition" (Raley, n.p.).

Though the emphasis here is on how Mez's work and codework in general push the reader/operator to begin to understand their phenomenological position at the computer terminal, the text is made somewhat conservative through the use of words like "disrupt" and "shock." On the other hand, Raley does recognize "potentially liberatory, modes of cognition" inherent in codework, concluding her section on Mez by stating, "Part of the mezangelle codework project is to awaken us to – also to comment upon and recompile – the varied and various data streams that we engage, filter, and disregard while multi-tasking"; and, "Within its specific online environment, then, digital media experimental writing, and specifically Mez's codework, offers us a glimpse of a mode of reading, cognition, consciousness, and even pedagogical praxis that is not yet fully available to us" (Raley, n.p.).

This attitude towards the work, as well as this understanding of the work as a primarily phenomenological rather than literary project, is reinforced by Mez herself. In the introduction to her work *_][ad][Dressed in a Skin C.ode_* (2002), Mez indicates that "the texts presented [in the work] act as residual traces from net.wurk practices that thrive, react N shift according 2 fluctuations in the online environment in which they][initially][gestated" (Mez, n. p.). That the

project moves beyond a literary endeavor is even further emphasized in various interviews. When asked by Josephine Bosma about a preoccupation with language and poetry Mez almost recoils, answering, "Regarding poetry, it's a label I'm uncomfortable with" (Mez, Interview n.p.).

It is interesting, then, that codework, and Mez's work in particular, is historicized within a literary context. At some level I do understand why this occurs – for institutional purposes it is important that electronic writing practices be considered and analyzed within some discipline, and drawing connection between print poetry and electronic texts can aid in the development of affective context. But, at the same time, grouping codework with other forms of literarture runs the risk of simplifying more significant cultural changes in regard to writing practice, technology, and available and emergent systems of inscription at iconic, indexical, and symbolic levels (to borrow Charles Sanders Peirce's terms).

There are many examples of this sort of historicizing, or as I have referred to it above, mythologizing of electronic textuality. Proposed antecedents for electronic literature include practitioners ranging from Stéphane Mallarmé to John Cage, James Joyce to Ezra Pound, Tristan Tzara to Steve McCaffery, etc. In her essay on codework, Raley uses e.e. cumming's *r-p-o-p-h-e-s-s-a-g-r* as a text to demonstrate formatting and punctuation used in an ideogrammatic or at least iconic manner similar to Mez's works; from a purely visual point of view, we can find similarities between texts by Mez and the cummings poem. Raley claims through this comparison that the formulaic difference between these works is "between the typewriter and the computer, the difference of what the medium allows" (n.p.). Though I do not disagree with this assessment, I do see differences between the works in terms of their syntactic and ideogrammatic interplay.

The cummings poem, though disruptive in terms of formatting, spelling, and the placement of punctuation, grants the marks themselves traditional, recognized values. That is, the punctuation in the cummings example comes from the same syntactic structure, the same language form as the overt though cryptic, letter-based text. In the case of Mez's texts, and much of codework for that matter, the various punctuation marks come from a different syntactic structure (e.g,. programming languages and other systems-based protocols) and are invested with different meanings.

A simple example of this can be found in Mez's use of [ ] (square brackets), along with numerous other codework practitioners (including myself). In writing, the [ ] may be used as in [sic] to indicate an error in a cited original, […] to indicate the exclusion of a portion of the original text, or to indicate modifications to a cited text. By contrast, in many programming languages brackets are utilized to denote a character class or elements of a variable array. It is in this second computational regard that we find square brackets most used in Mez's work. They serve as indicative marks related to the introduction of variability in the text, and potential emergent polysemic results. With the exception of the second to last line in the cummings poem (which reads, "rea(be)rran(com)gi(e)ngly"), we do not see the sort of polysemy so evident in Mez's work. And even here there are differences in how Mez and cummings embed encrypted value.

We could argue that there is an overlap between the traditional and computational uses of punctuation, particularly at the level of modification, but the origin and effect of the marks in codework is fairly explicit. As Mez clearly states, her texts are "residual traces from net.wurk practices." There is no mention of how her language is related to modernist poetry, or how her use of punctuation is related to prior conventions. Rather, there seems to be an internal logic at

work here, one that is subjective and based in a dedicated observation of what I would call the cyborganic relationship between human and networked computer.

What we see in a Mez work is a sort of subjective parsing that is not so much about surface text (natural) and submerged text (code) as it is about a collision of syntactic structures that are both given equal value, a lateral rather than hierarchal move. The problems in reading these works are primarily based in the degree to which they are iconic. This is a significant, and signifying aspect of codework, as Sondheim recognizes: "[T]he interstitial / liminal between the meaning-sememe and the ikonic provides the content of the work; in fact, the meaning-sememe and ikonic-sememe are interwoven, inseparable, and contributory […]" (Sondheim, "Further Notes on Codework" n.p.).

In Sondheim's view, it would seem that codework introduces a degree of encryption into inscription and through this procedural aspects – for both writer and reader – are emphasized. For the writer, the document is provided as documentation of a performative writing practice (encoding); for the reader the document introduces an interpretive process (decoding) through its iconicity and encryption. In this regard I would argue against a superficial reading of codework that positions the work outside of the procedural and regards it as incapable of producing a meaningful reading experience. When Raley states that codework practioners are "less concerned with offering a reading experience than they are with working with the language of code to offer comments on form and the materiality of language" (Raley, "Code.surface || Code.depth" n.p.), she is right to claim that the materiality of language is in question, but she overlooks the residual effect of interaction with the Internet apparatus (i.e., the embedded network phenomenology) and the intended process of reading as decryption. It should be noted that higher level languages are ultimately readable by humans. And, if we make claims about the openness of the web, it is

important to understand that, despite its disruption of current natural language structures, code has become, or is becoming something other than a specialized language. Or, as Michael Mateas and Nick Montfort put it in their *A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics*:

> [M]odern computer programs are written in a form, usually textual, that is also meant to be manipulable and understandable by human beings. For a programmer to understand what she herself is writing, and to incorporate code that others have written, and to simply learn how to program with greater facility and on a larger, more complex scale, code has been made legible to people. While a computer system may compile or interpret code, it is important to the nature of code that it is interpreted by people as well." (online text, n.p.)

3. Obfuscation as Clarification

As stated above, much of the debate around codework is centered on issues of address, or as Cayley puts it, "[the] pretended ambiguity of address" (Cayley, n.p.) between code-as-text, and code-as-operation. This ambiguity seems in actuality to be a default condition of code. For Raley, and perhaps because of an ontological privileging based on discipline, code need not always be executable. As she states, "Code […] 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" (Raley, "Code.surface || Code.depth" n.p.). In this statement, which is in fact a reiteration of comments made by Mateas and Montfort in the article cited above, Raley is pointing to a number of code related phenomena that play into how certain electronic literary works are considered codework. In this quote we can read "dormant" in two different ways:

first, as code that has not yet been implemented into an application and as such has not performed its intended function and second, as code that has been implemented as surface, interface, or readable text and is not intended to be made executable in the computational sense.

In this second sense what is produced is a sort of obfuscation through the syntactic collision of two language systems – code and natural language – which results in increased iconicity, an esoteric appearance, and perhaps, ironically, the exposure of writerly intent. This iconic obfuscation is at times mirrored in programming communities, though in these communities the code is still generally operational and the obfuscation is more about elaboration. Mateas and Montfort make reference to the International Obfuscated C Code Contest (IOCCC), which ran, on and off, between 1984 and 2006. The example they provide is based on the simple "Hello, world!" program that is generally used as an introductory exercise in programming courses. The IOCCC example is obfuscated through the use of what could be called an intentionally anti-grammarian increase in complexity that includes obscure methods for expressing zero, redundant conditions, and meaningless or unnecessary math. Nonetheless, the output of the program remains "Hello, world!" This example has some parallels to Mez's work in terms of the obfuscation of inscription but also, perhaps, through the more representative examples of IOCCC entries which suggest the wider practices of what we are calling codework.

In 1998 the IOCCC "Best of Show" winner was a flight simulator programmed by Carl Banks. There are a number of unique qualities to this flight simulator, mostly technical – the program is less than 2 kilobytes, the scenery files contain fewer than 1000 lines of input, etc. – but what is perhaps most interesting about the program is that the source code is formatted to display an ASCII art representation of an airplane.

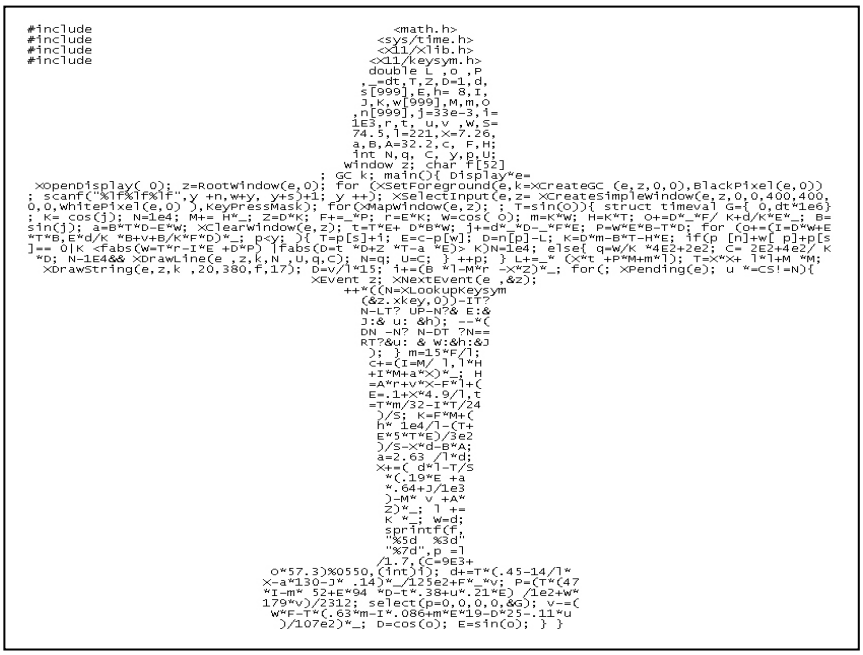

Fig. 1. Source Code from the Flight Simulator Program by Carl Banks

Image in the Public Domain

Though this formatting does not necessarily make it an easy read for a programmer, and has no bearing on how the program runs, it does in fact indicate, iconically, what the program is. Of course, if a user of the program did not view the uncompiled source, he or she would have no knowledge of this obfuscation as iconic clarification. There are three different levels at which this program can be read. We can read the image rendered through the formatting, we can read the code as code, or we can play the flight simulator. In each case the address is rather explicit and only one of these actually requires that the code be addressed to the machine. Only in the case of playing the flight simulator, which is, indeed, the purpose of the program though not the intent of the code's formatting, is the code hidden and actualized as process.

What is significant here in relationship to codework is the contextual association between the formatting of the code and the operations of the program. When we view the source and find

the airplane icon displayed we are indeed confronted with an ambiguous address. At the iconic level "[t]he code has ceased to function as code" (Cayley, "The Code is Not the Text (Unless It Is the Text)" n.p.), while at the literal level it remains code. Still, the address is to the human reader in this regard – we are looking at and reading the source code as text (or image) – and the formatting gives this code-as-text (or image) the added value of being a calligram. None of this rich content has anything to do with machinic address – the machine could not care less that the code displays an airplane. So, one must ask, at what point is the code made procedural? And, does the procedure require that the code be addressed to the machine?

We could consider the formatting of the source code as a sort of suspended calligram, or a calligrammatic riddle. Without knowledge of C code (even with knowledge of C for that matter), we privilege the iconic value of the airplane formatting. It is only through compiling the code and running the program, playing the flight simulator, that the intent of the formatting and the purpose of the program are brought into alignment. In this regard, a combination of human and machinic address is required for the overall piece to be realized. This complex process cannot be reduced to questions of surface and submerged, natural and encoded languages, or inscription versus encryption. Rather, the overall effect of the project is emphatically procedural and performative regardless of questions of address.

4. Aesthetics vs. Poetics

Bank's flight simulator is not generally considered codework, but it does possess many of the qualities associated with codework, and to a certain extent software art. That being said, as Mateas and Montfort indicate, the term "software art" is rarely used in programming communities and "it seems unfair to apply the term 'art,' with all of its connotations, to their

work" (http://users.soe.ucsc.edu/~michaelm/publications/mateas2-dac2005.pdf n.p.). I imagine the same could be said about applying the terms "literature" or "poetry." Yet, codework as a practice has been embraced by the electronic literature community at the level of genre. This is primarily based on the specifics of the origin of the term, the various codework practitioners' willingness to have their work considered within the field of electronic literature, and the willingness of electronic literature scholars to consider codework, and code for that matter, within the domain of literary studies.

There is a degree of ambivalence, however, among codework practitioners when it comes to how the attributes of the work are addressed within literary studies. Mez makes it quite clear that she does not feel comfortable with her work being labeled poetry, yet she is willing to accept that her work may be considered literary at the level of inscription. There are other practitioners whose work has been considered codework who are not willing to accept the literary label at all, preferring instead to define the work as "net.art"; there are also codework practitioners who fully embrace poetry as the proper rubric for their work. Such is the case with Ted Warnell, who calls his work "code poetry" rather than codework despite aesthetic and programmatic similarities.

As stated, codework is not really a specific genre of electronic literature, and the representative works that fall under this umbrella are diverse. While much of Mez's work is presented as flat web or text documents, and this tradition continues with codework practitioners such as Bjørn Magnhildøen and Alan Sondheim (though flat web or text documents are clearly not the only type of output produced by Sondheim), there are many examples that take full advantage of various digital media and the procedural aspects of code. Somewhere in the middle, between aesthetically rendered static document and mediated, procedural application, comes Warnell's work.

Looking at three of Warnell's projects produced between 2001 and 2011 (*Viru2*, *The Eden Database*, and *db.11x8.5*), we find a long term commitment to a particular aesthetic and poetic/aesthetic agenda. In general, all of these works have a formal, minimalist appearance – they are not cluttered with bells and whistles, over-mediated so to speak – but are rich in conceptual framework. Each of these works asks questions about the propriety of the interface, the position of text and code, and the conflict between aesthetic and poetic values in regard to interface legibility and readability.

The earliest piece, *Viru2* (2001), is probably the work that can most easily be categorized as codework, especially considering the time of its production. As I argued in my essay for the *American Book Review*, *Viru2* uses text not as readable units but as color mass. "Areas of red, black and blue text mark a stark white screen, drip, and flood the screen in strokes reminiscent of paintings by Clifford Still" (6). This is text to be looked at, not read. As such, the interface text is used primarily for aesthetic rather than poetic effect. The only readable interface

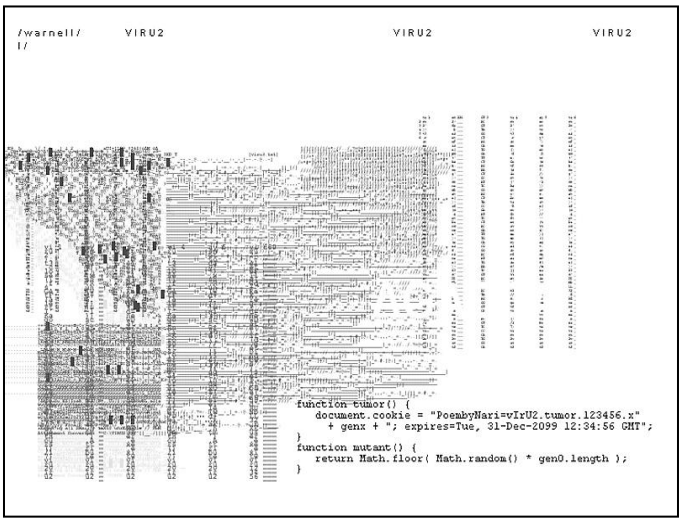

Fig. 2. Screenshot from Warnell's *Viru2* (2001) Permission granted by the artist.

text is in fact code, and even here the text is only partially readable and comes from different sources. In the lower right of the screen we find a barely visible genetic sequence, and just beneath it a fully visible JavaScript function. The text as color mass, the DNA codons, and the JavaScript are connected conceptually to the title (Viru2 = virus) and context of the work. What we are offered is not so much poetry, as a holistic environmental grammatology on a theme.

The JavaScript presented through the interface is in fact a surface reiteration of the functioning JavaScript of the application. While the functioning code actually deposits a cookie into your browser cache – with an expiration date of December 31, 2099 – at the surface, the code operates as text, letting the user know what the functioning code is doing:

```
function tumor() {
document.cookie = "PoembyNari=vIrU2.tumor.123456.x"
+ genx + "; expires=Tue, 31-Dec-2099 12:34:56 GMT";
}
function mutant() {
return Math.floor( Math.random() * gen0.length );
}
```

With function names like “tumor” and “mutant,” the surface code-as-text provides readable cues connecting it to the overall concept of the work. The “tumor” function points to the depositing of the cookie into the system, while the “mutant” function refers to the single interactive feature of the work. The DNA codons displayed at the surface are actually dynamically generated when the page is loaded, and the displayed codons provide a link that reloads the page, regenerating (mutating) the DNA sequence. The effect is subtle, considering that the displayed DNA code-as-text is barely visible, but does demonstrate correspondence

between surface code-as-text and the functioning code.

In a 2002 interview for the trAce Online Writing Community, Warnell argued that "Code is visual, to be looked at" (Interview by Randy Adams and Rita Raley n. p.). And in *Viru2*, we see a variety of code being displayed – DNA sequences, JavaScript, etc. In fact, if we view the source code of *Viru2* we find that the areas of color mass are formed by way of parsed binary code from digital images. This strategy of employing the codebase of digital images rather than the images themselves is further utilized in Warnell's work *The Eden Database* (2002). This piece uses as its premise the idea that, "[i]f the codes that make up digital images are unique like the people in the images are unique, then we might imaginatively think of these codes as a form of digital DNA" (from the preface, n.p.). To a certain degree this is an extension of the ideas Warnell investigates in *Viru2,* but here he seems less interested in the procedural mutation of code than in the representational substitution of images with their codes, as qualitative tokens or stand-ins for the actual image and what it represents. One could ask, what is an image when left unrendered, or rendered as code to be read as text? Does it still have potency as image?

Fig. 3. Screenshot from Warnell's *The Eden Database* (2002). Permission granted by the artist.

Without the ability to read the code of a JPG image as output through a TXT file we cannot be assured of the authenticity of the images used in the piece, their origins, or the process through which they have been transformed. Not only is the potency of image represented as code put into question, but the process itself seems directed toward an excessive level of abstraction. In fact, when we view the source code from various pages of *The Eden Database* we do not find the binary image code left intact; rather, we find that it has been heavily parsed by Warnell's hand. Characters have been removed, replaced by underscores, spaces or other characters, and the image code has been implemented into variable arrays. This introduction of a subjective authorial or editorial method into the code makes it impossible for the images to be recompiled – even through a secondary parsing of the code. We cannot be sure of Warnell's parsing routine here, so we cannot tell what has been removed and what characters have been replaced by others.

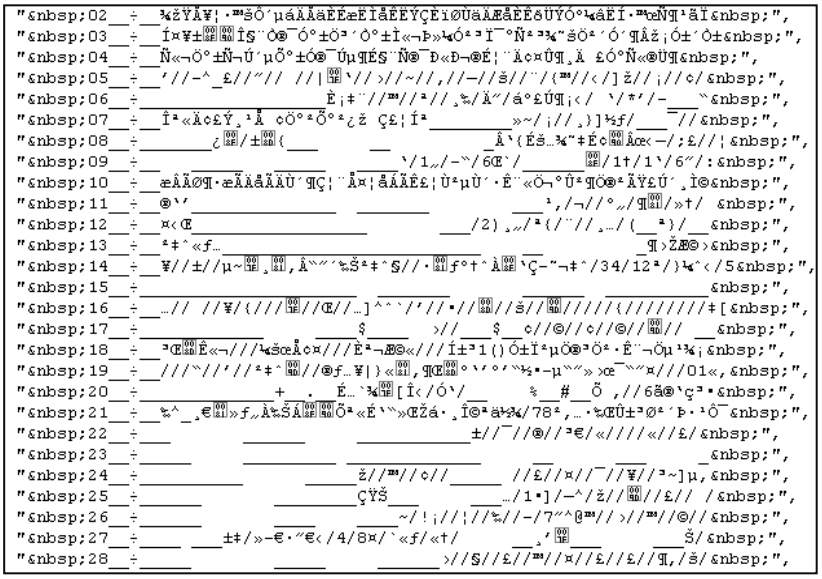

Fig. 4. Source Code from Warnell's *The Eden Database* (2002). Permission granted by the artist.

Both of these early Warnell works, *Viru2* and *The Eden Database,* use displayed code for aesthetic ends. And, to a certain degree, they are similar to each other in both concept and outward appearance. Regarding the subjective parsing in these works, we can draw parallels to Mez's texts. Inscription for both Mez and Warnell is directed toward a sort of encryption that allows computer language to enter into natural language and, in Warnell's case, to operate visually at an elemental level. Unlike Mez, Warnell's style is painterly and consistently directed toward the design and configuration of the interface. Code is used almost exclusively for its material value, its density and mass when displayed at the level of the interface.

More current work by Warnell continues with this painterly approach, taking it to more sophisticated aesthetic levels. Since 2005, Warnell has been developing a series (nearly 500 to date) of single page works under the title *db.11x8.5*. Though the series foregrounds the aesthetic concerns initially established in *Viru2* and *The Eden Database*, they are quite different in terms of how code is utilized and exposed. Where the earlier work makes code an overt formal gesture by exposing it at the level of the interface as visible text, the works in *db.11x8.5* are more subtle in their method and perhaps more in line with aesthetic computing and visualization than with what has been classified as codework (in the literary sense).

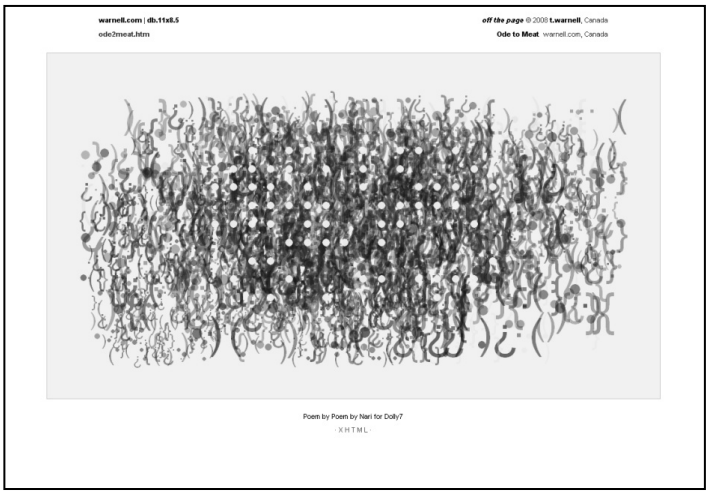

Fig. 5. Screenshot from *ode to meat* (2008), one of nearly 500 single page works in Warnell's db. *11x8.5 series*. Permission granted by the artist.

Still, this newer work does fit under the second prong of Sondheim's codework ontology. Through its use of CSS, XHTML, JavaScript and HTML5, the "submerged code has modified the surface language" and the source material for Warnell's visualizations is largely textual. Among his "CONTRIBUTING & COLLABORATING ARTISTS & OTHERS," Warnell lists Dante Alighieri, Mary Shelley, Allen Ginsberg, and Edgar Allan Poe – not to mention Johnny Cash, Kurt Cobain, Marcel Duchamp, Kurt Schwitters, and Mez herself. Of course, what Warnell means by "contributing" and "collaborating" is based on appropriation of text as pure data and how the work is aesthetically and procedurally informed.

In these works we still have inscription as encryption, but here the encryption is procedural and the result is visual rather than literary.

5. Process and Procedure

To conclude this essay, I would like to discuss some differences between process and procedure in relationship to codework. As I have stated towards the beginning of my essay, I see codework as a sort of evidentiary phenomenology of engagement with the network. The fact that the code (sometimes) appears at the surface does not necessarily suggest an operational transfer of functioning code from machine to humans, so much as a reflection on the syntactic duality of network-based subjectivity.

The works of Mez and other codework practitioners are not programs, they are projects. Mez's subjective process is significant specifically because it is subjective and addressed to the human reader. The process is strongly writerly, however, which leads to the impression that the

work is somehow positioned between machine and human – that the encryption of the text is intended to baffle rather than address certain critical issues of network engagement. Still, as John Cayley states, "[The reader] can appreciate, through more-or-less traditional hermeneutic procedures, the references and allusions to technology, technoscience and the issues with which they confront us" (Cayley, n.p.). As such, the issues of inscription in codework practice are more than mere rhetorical devices. They are strategic moves that not only gesture toward issues of network subjectivity, but signal their complexity.

It would be quite reductive to consider codework as a purely computational form or to try to simulate aspects of codework through a purely procedural application. Yet, this has been the initiative of Edde Addad, one of a number of programmers associated with Gnoetry Daily – a blog dedicated to poetry generator programs. Addad is a self-proclaimed fan of Mez and has attempted to reproduce her heavily bracketed style procedurally, doing so with some enthusiasm and a bit of bias towards the potential for computation in order to match the Mez's. As he says, "If I was a poet from Australia and didn't know how to code I'd probably want to be just like her. But since I can't be her maybe I can write a program to be her instead" (eddeaddad, n.p.).

Addad's program, *codework parenthetical insertions* (see under eddeaddad) runs in the JanusNode text generator. It is fairly efficient in its functionality and at a superficial level does seem to modify texts to have them look like texts by Mez.

Taking the quote from Barthes at the beginning of this essay:

> "myth consists in overturning culture into nature or, at least, the social, the cultural, the ideological, the historical into the 'natural'…"

running it through the program produces the following results:

myth consists in :o.ver[whelmed]turning culture into nature or, at :l.ea[der]st, the social, the cultural, the ideologi:c.a[r]l, the his:t.or[nado]i:c.a[r]l into the natural

While working on this essay, I asked Mez if she would be willing to turn the quote into M[ez]ang.elle. Within hours Mez replied with the following text:

myth[os drenched.in.the.liMin(im)All] con[+re:]sists.

over[::passes::]turn[::ed+urved_curled::]ing.

c[o]u:l:ture _vs_ nat[L]ure:

>>the soc[D]ial[led.up, 1nce],

>>the cult.[du.chump]ur[in]al,

>>the ideolog[rammat]ical,

>>the historica[u]l.

By turning Mez's subjective parsing method into a scripted procedural function what is lost is Mez herself. The complexity of the process and the fickleness of the polysemy are limited by the program, and Mez's phenomenology is replaced, written through the procedural poetics of a secondary reader. Some of the problems are based on the limitations of the parsing routines of JanusNode, but Addad's text mapping does not produce the same sort of embedded critical, philosophical, variable and subjective text that Mez is best known for. What are reproduced are the most banal formal aspects of Mez's technique – the bracketed insertions – and though the mapping is interpretive, the output lacks the mastery and prerogative of a text by Mez herself. The simulation is, therefore, more readerly than it is writerly.

RE:mark

Whether or not a work is directed toward the literary, there seems to be a certain cultural aspect to the production, use, and sometimes abuse of code. We can think of this as critical or creative; we may address these issues from the point of view of literature and inscription or from that of programmers. What is essential to understand here is that code-based practices, be they defined specifically as codework or not, have entered into the general cultural economy of expression. Codework, which can only be defined as a certain taxonomic convenience, cannot be limited to the literary genre or to the domain of literary practice. Sondheim's statement that in codework "the interstitial / liminal between the meaning-sememe and the ikonic" are interwoven and contributory ("Further Notes on Codework" n.p.) makes it clear that codework is a form that commingles semiological, aesthetic, poetic, and procedural methods that emphasize the location of practice – at the computer terminal. As such, the literary historicization of the methods at hand can only work to mythologize and serve to diminish the investigatory, expressive, and evidentiary phenomenology embodied in the work.

Works Cited

Baldwin, Sandy. "Against Digital Poetics." *Electronic Book Review* (2009): Web. Accessed 03 Dec 2010. http://www.electronicbookreview.com/thread/electropoetics/absorbant.

Barthes, Roland. "Change the Object Itself." *Image, Music, Text*. Ed. Stephen Heath. New York: Hill and Wang, 1978. 165-69.

Barthes, Roland. *Mythologies*. Paris: Editions de Seuil, 1957. Trans. Annette Lavers as *Mythologies.* London, Paladin, 1972.

Cayley, John. "The Code is Not the Text (Unless It Is the Text)." *Electronic Book Review*

(2002): Web. Accessed 11 Nov 2010. http://www.electronicbookreview.com/thread/electropoetics/literal..

cummings, e.e. *r-p-o-p-h-e-s-s-a-g-r* . http://www.poets.org/viewmedia.php/prmMID/15402.

eddeaddad (Edde Addad). "codework parenthetical insertions." *Gnoetry Daily*. 11 Jan 2011. Web. Accessed 12 Jan 2011. http://gnoetrydaily.wordpress.com/2011/01/11/codework-parenthetical-insertions/.

Mateas, Michael, and Nick Montfort. "A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics." In *Proceedings of the 6th Digital Arts and Culture Conference*. Ed. Alexandersen and Diddle. Copenhagen, Denmark: IT University of Copenhagen. 2005. 144-53. Available at: http://users.soe.ucsc.edu/~michaelm/publications/mateas2-dac2005.pdf

Memmott, Talan. "E_RUPTURE://Codework"."Serration in Electronic Literature." *American Book Review* 22.6 (2001): 1, 6.

Mez (Mary-Anne Breeze). *_][ad][Dressed in a Skin C.ode_* . Center for Digital Discourse and Culture. 2002. http://www.cddc.vt.edu/host/netwurker/

Mez. Interview with Josephine Bosma. *Readme* 3 (Summer 2000). http://home.jps.net/~nada/mez.htm

Raley, Rita. "Code.surface || Code.depth." *dichtung-digital - journal für digitale ästhetik* (2006): Web. 27 Dec 2010. http://dichtungdigital.mewi.unibas.ch/2006/01/Raley/index.htm..

Sondheim, Alan. "Codework." *American Book Review*. 22.6 (2001): 1, 4.

Sondheim, Alan. "Further Notes on Codework." 10 Feb 2004 . Online Posting to *nettime* Web. Accessed 4 Nov 2010. http://www.mail-archive.com/nettime-l@bbs.thing.net/msg01653.html.

Warnell, Ted. *db.11x8.5*. Web. 2005-. Accessed 12 Nov 2010. http://warnell.com/db11x85/index.htm.

Warnell, Ted. *The Eden Database.* 2002. Web. Accessed 12 Nov 2010. http://warnell.com/eden/eden.htm.

Warnell, Ted. Interview by Randy Adams and Rita Raley. *trace Online Writing Centre*. 2002. Web. Accessed 04 Jan 2011. http://tracearchive.ntu.ac.uk/showcase/index.cfm?article=24.

Warnell, Ted. *VIRU2*. 2001. Web. Accessed 10 Jan 2011. http://warnell.com/syntac/viru2.htm.