2015

Glîtchéd in †ranslation [Glitched in Translation]

Matt Applegate Ph.D.
Molloy College, mapplegate@molloy.edu

Glîtchéd in †ranslation

Matt Applegate, Ph.d.
Assistant Professor of English & Communications
Molloy College

In this paper, I think precarity in digital communication on two overlapping registers. The first is perhaps best described as an aesthetic intervention at the level of critical code studies, or, as Mark C. Marino describes it, "an approach that applies critical hermeneutics to the interpretation of computer code, program architecture, and documentation within a socio-historical context."[i] What I am interested in examining here is the representation of natural languages by computer languages (specifically Unicode), but also natural languages' ambiguation by computer languages in the production of aesthetic objects. The second intervention follows from the first. There is an architectural play at work in text-based glitch art, specifically where natural languages and the code that manifests them alter the shape and function of their interface. I am interested in thinking the glitch aesthetic in particular as one that not only identifies the limitations of our most common interfaces, but one that speaks an erratic and unstable visual language of design. In this text/code/text translation, the language of the glitch aesthetic manifests as computer code transforms letters, numbers, diacritical marks, and more, into the building blocks of aesthetic objects. This process speaks back to the form and function of language itself.

Critical Code Studies (CCS)

In 2006, Marino theorized CCS as a sub-discipline of (DH, software studies, media archaeology, etc.) with reference, and in some instances by contrast, to an aesthetics of code made manifest by figures like John Cayley and Mez Breeze. In his article, simply titled "Critical Code Studies," Marino is concerned with the functionality and efficiency of code, an aesthetic problem on his view, but he is also concerned with how code manifests as an aesthetic object. Marino cites Rita Raley on this point, arguing that code's aestheticization need not result in an executable process, especially as it brings what's hiding in the background of our favorite websites, apps, digital lit, and so on, to the foreground.[1] The question of what computer code is when it doesn't produce an executable command, is therefore a linguistic problem as much as an aesthetic one. If code is not necessarily an operable set of symbols and characters, what is its status *as a language*? Marino draws a helpful distinction between what he calls codework and CCS on this point, stating: "I would like to propose that we no longer speak of the code as a

[1] In her article "Code.surface || Code.depth," Raley argues that 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." This claims allows for some variability in understanding code *as a language*, but it also allows us to think the legibility and morphology of coding languages in a more nuanced way. To the point of this paper, the argument concerning execution transforms into an argument about a coding language's legibility from the backend to the front.

text in metaphorical terms, but that we begin to analyze and explicate code as a text, as a sign system with its own rhetoric, as verbal communication that possesses significance in excess of its functional utility."[ii] Codework, then, is situated as a metaphorical engagement with coding languages in their function *as a language*. Speaking diagrammatically, codework is often characterized as the surface level representation of code in Marino's argument, which is also often characterized as a literary exercise. CCS, on the other hand, situates code within a varied landscape of semiotic systems, showcasing its own syntactical rules and grammar. Again, CCS focuses on and refers to *the morphology of code itself*.

The problem that this paper identifies, and one that I work to articulate, is already manifest. If code need not result in an executable command in order to be considered an extension of *a coding language*, then the question that follows is clear: how can we think code's representational value "on the surface" as an extension of a particular coding language's morphology? On the one hand, the answer seems to be a simple one. The grammar and syntax of a particular coding language does not manifest on the interface it creates, therefore the two are divided. On the other hand, there are moments in which the representational value of a particular language's grammar and syntax upsets the interface that manifests it, demanding that we clarify what we mean when we talk about code *as a language* and what its linguistic properties are.

For the work I am interested in examining, Marino's focus on what Nick Montfort calls "obfuscated code" and what Michael Mateas calls "weird languages" is perhaps the most pertinent. For Monfort, obfuscated code is often dense and indecipherable. It, in Mateas' words, "exploits the syntactic and semantic play of a language to create code that, often humorously, comments on the constructs provided by a specific language."[iii] Weird languages are those in which "programmers explore and exploit the play that is possible in programming language design."[iv] They, too, often comment on the features of existing coding languages. The allure of these practices for both theorists is primarily characterized by their ability to "double code," or their ability to execute valid programs in two or more coding languages. Obfuscated code and weird languages blend coding languages a means of making syntactic and semantic commentary about code *in code itself*. They also demonstrate the variability of code, its inherent multi-linguality, and in some cases, weird languages and obfuscated code further programming language design.

I don't want to get caught up in the examples that Montfort and Mateas provide to articulate these concepts. They are difficult and specialized. Rather, I'd like to turn to a linguistic problem that is situated somewhere between thinking code's representational value on the surface, on the one hand, and theorizing the function of double coding, on the other--making the discourse of CCS somewhat more complex, but also more precarious. Take Glitchr's 'tweet art' as an example (https://twitter.com/glitchr_). Since 2011, Glitchr has exploited errors in varying social media's Unicode renderings, a digital language database comprised of character encodings that modify depending on their visualization (by way of a brief definition: Unicode encodes all of the characters that comprise a symbol, rather than a symbol itself, and leaves a given character's visual rendering up to a given platform and its programming languages that manifest it, i.e. Microsoft Word, Twitter, Facebook, etc.). At its base, Unicode operates as a set of encodings that rely on other languages to manifest on screen. The Unicode consortium's motto, so to speak, is "Unicode provides a unique number for every character, no matter

what the platform, no matter what the language."[v] In this way, Unicode is a global standard for encoding and representing text in almost every interface we use on a computer, but it also offers an alternative to the focus on "double coding" in CCS. At the backend, Unicode does not comment on the grammar or syntax of coding languages themselves, it is the numeric value that stands between coding languages and natural languages. It is the operator and proof that digital languages are multi-lingual. On the frontend, Unicode is used primarily for text processing, but it can also be used to inform a given website's features and layout. What Unicode executes via various programming languages is therefore a variation on "code switching," literally "code switching," with and beyond what one would see in the interplay of natural languages. Utilizing Unicode to render text is the inherent process through which programming languages combine to manifest text across various platforms and languages.

Glitchr's tweet art is a stark visual expression of this process: it mobilizes Unicode encodings to alter the text space of his Twitter account by creating text-based shapes and architectural doodles. Consequently, Glitchr's objective and principal aesthetic intervention is to use social media's coding languages to alter its architecture via the digital rendering of text, even if only briefly. The take away here is that, if we are to think glitch art as an intervention in CCS, then text-based glitch art calls into question what we mean when we talk about language in and of a digital milieu. This is precisely what Rosa Menkman proclaims in her *Glitch Studies Manifesto*: "Once the glitch is understood as an alternative way of representation or a new language, its tipping point has passed and the essence of its glitch-being is vanished. The glitch is no longer an art of rejection, but a shape or appearance that is reorganized as a novel form (of art)."[vi] The language of the glitch aesthetic to which Menkman refers is difficult to pin down. Certainly, it carries with it a technical vocabulary that is associated with various coding languages, hardware, and software. Moreover, the art form's relation to code and the technologies that manifest it complicate any order, command, or representative set of characters that would form a universal 'glitch grammar.' In the instances of glitch art's visual representation, code is the art form's double, so to speak; code is what manifests glitch art in digital landscapes as glitch art often exposes the presence of 'code switching' that brings it into being. However, Menkman's claim to understanding the art form *as a language itself* is far more tenuous than these techno-linguistic aspects of the medium. Certainly, the viewer cannot simply understand glitch art as one would a natural language with an alphabet and established syntax. Nor is it a coding language with similar characteristics. The visual representation of glitch art often intervenes at multiple levels of articulation: in the architecture of a website, in the compositional elements of an image, and in the organization and design of text by the coding languages that manifest them. Where the linguistic and spatial elements of glitch art are broadly articulated, then, the medium must be understood on at least two overlapping registers.

On the one hand, the visual representation of text-based glitch art is the simultaneous ambiguation and disambiguation of code. Not a process of 'double coding' but of code switching, glitch art mobilizes various Unicode sets to showcase a given interfaces inoperability. Therefore, code switching is both an inherent feature of how programming languages represent text, but one that can be manipulated when a particular Unicode encoding is inserted into a context in which it doesn't belong, leading us closer to what Montfort calls obfuscated code. This is to say that code manipulation is

simultaneously obfuscated and made manifest where the glitch is made visible, subjecting both the visualization and obfuscation of code to interpretation as both an aesthetic process and linguistic curiosity. On the other hand, the visual representation of glitch art often manifests as an architectural play of spaces, orchestrated by the rearrangement of text. The linguistic elements of the medium are therefore figurative and spatially oriented, frequently divorced from both the code and natural languages they are commonly associated with at the point of their representation.

The process through which Glitchr produces his architectural doodles is multi-layered. It requires one to initialize the Unicode Hex Input keyboard on her computer; reference a unicode-table that features both the numerical Unicode encoding and its symbol; manually arrange a set of symbols and characters as one desires in Microsoft Word; and finally, it requires that one transfer the doodle from Word to its desired location. Here, the author types the Unicode encoding for the desired symbol, allowing each symbol to stack onto the next, and ultimately rendering it as a single character. The proposition for CCS here is twofold. First, when we explicate code as text, what manifests on the surface is an extension of a particular coding language's morphology. Glitch art is the artifact and proof that Unicode remains functional within the coding languages that manifest it albeit dysfunctional at the level of natural languages. It is the artifact and proof that a conversion has taken place between the characters to which Unicode refers and the visual language that one creates by manipulating Unicode encodings at the level of the interface. This set of concerns underscores a kind of linguistic precarity that still renders Unicode and other coding languages functional, but points to a kind of representational inoperability. Second, glitch art requires that we reimagine what what we mean when we say that we want to explicate code *as a language*, especially where code, as Marino claims, is a mode of "verbal communication that possesses significance in excess of its functional utility."[vii]

Speaking a Language of Digital Design

If glitch art is a language, it is a language that is similar to the discussion of code's inoperability that opens this paper. I will state the question that follows from this problem again as a reminder of what's at stake. If code need not result in an executable command in order to exist *as a coding language*, how can we think its representational value "on the surface" as an extension of a particular coding language's morphology?

Deleuze and Guattari's discussion of the order-word is helpful place to begin. On Deleuze and Guattari's account, all of the intonations, sounds, markers, and symbols attributed to a given language become codified by the production of the order-word. On its first definition, the order-word is the "semiotic coordinates possessing all of the dual foundations of grammar (masculine-feminine, singular-plural, noun-verb, subject of the statement-subject of enunciation, etc.)," and, "the elementary unit of language—the statement."[viii] Stated differently, the intonations, sounds, markers, and symbols attributed to a given natural language are retroactively translated into a dominant signifying regime. It is the formation of a 'language' that imposes a system of representation onto its eventual semiotic components.

In her 2008 "On 'Sourcery', or Code as Fetish," Wendy Hui Kyong Chun affirms this comparison by interrogating the use and function of source code (i.e., instructions for computers written in a computer language that humans can also read, typically as text) as

an executable command for a computer. Here, she writes that the translation from source code to a command that functions in a computer process is a retroactive procedure: source code "only becomes source after the fact," precisely because it has to be conscripted into the command structure of a unified coding language, piece of hardware, or software, to make it operable.[ix] On Chun's view, code is therefore a "resource" rather than a 'source'—a set of symbols that only produce something tangible when they are properly aligned with a command structure to translate them.

When we disconnect the eventual semiotic components of a language from the command structure giving it linguistic value, however, we, in Deleuze and Guattari's words, 'deterrtorialize' the signifiers of one signifying regime and 'reterritorialize' those signifiers into another. With reference to glitch art in particular, what happens on the surface or at the level of the interface is precisely a linguistic process of deterritorialization and reterritorialization. Characters, symbols, etc., are rended from their function in and of a natural language and repositioned within a much more malleable signifying regime. Glitch art is then perhaps closer to what Deleuze and Guattari's call a collective assemblage of enunciation, working at the inverse of the order-word; glitch art is the co-manifestation of "all the voices present within a single voice," ultimately undermining the perceived unity of a signifying regime.[x] In this sense, glitch art is a schizophrenic exercise, but one that targets the design of digital interfaces and their technologies of production. Glitch art aestheticizes the coding languages that prefigure the manifestation of digital objects and space, manifesting them by refusing their full translation. In the case of Glitchr's tweet art, code, text, and grammatical marks are granted a collective enunciative property within the architecture of a social media platform. Each new piece of tweet art proclaims that users can communicate in a language of digital design, and further, that digital design need not conform to a 'one size fits all model.'

To bring this back to the question of aesthetics and space, however, and to bring us to a conclusion, we must return to the question of what languages are for. Glitchr is by no means the most famous glitch artist, nor is this kind of architectural play with text a unique computer interface problem--text-based glitch art builds on a much broader set of aesthetic practices that rend characters from their natural languages in order to mobilize them as building blocks for the production of visual art. In fact, how we think the linguistic value of a coding language or set of coding languages "on the surface" and as an extension of a particular coding language's morphology seems to require a media archaeology of various writing interfaces. This is akin the kind work Lorin Emerson embarks on in her book, *Reading Writing Interfaces*. It is perhaps a variation on what Friedrich Kittler means when he writes of typewriters as storage devices: "Typewriters do not store individuals; their letters do not communicate a beyond that perfectly alphabetized readers can subsequently hallucinate as meaning."[xi]

To move to this paper to its conclusion, I want to show a couple of text-based art objects that prefigure glitch art and underscore an architectural play at work when text is taken from its function in and of a natural language to its function as an aesthetic building block.

Example 1: H.N. Werkman, Bauhaus Student, Construction Exercise

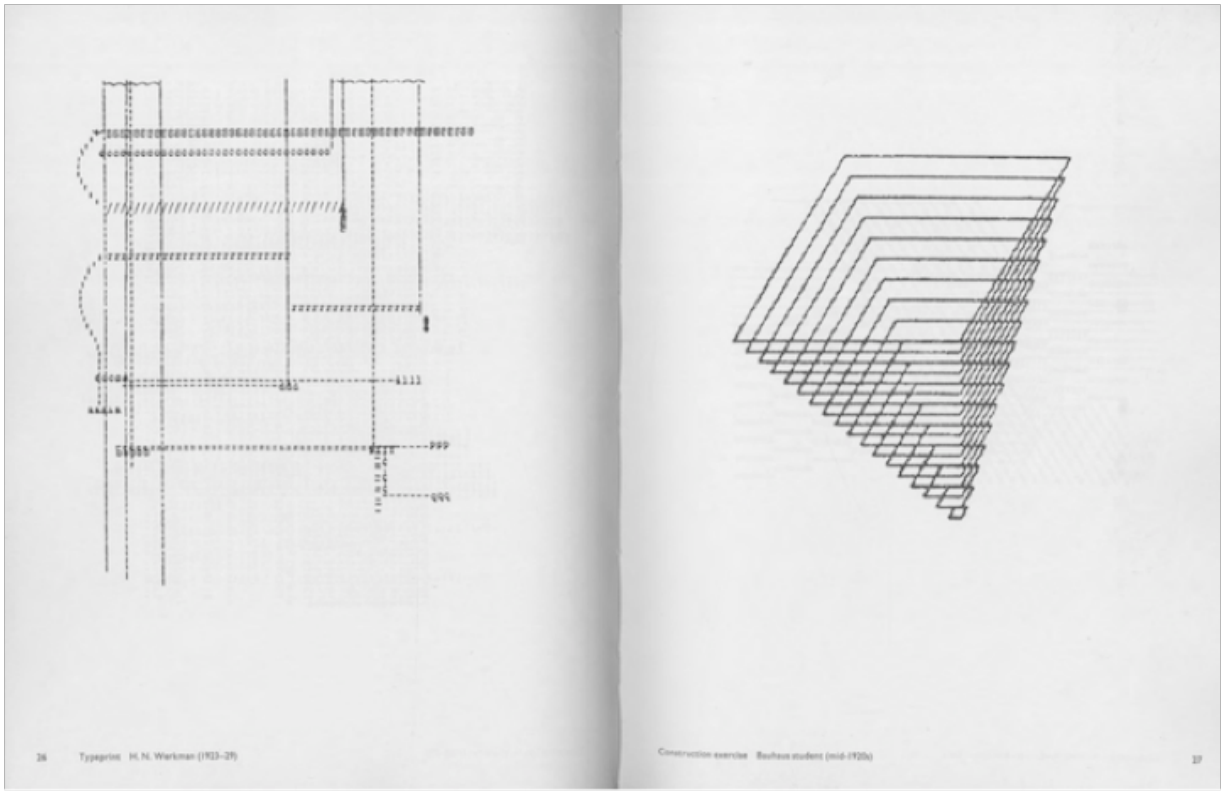

Figure 1 Left to right: H. N. Werkman, "Typeprint" (1923–29) and Bauhaus student, "Construction exercise" (mid-1920s).

Source: *Typewriter Art*, Edited by Alan Riddell, p. 26-27. (screenshot via monoskop.org)

Example 2: Kenneth Knowlton and Leon D. Harmon, the forefathers of ASCII Art (American Standard Code for Information Exchange) Art

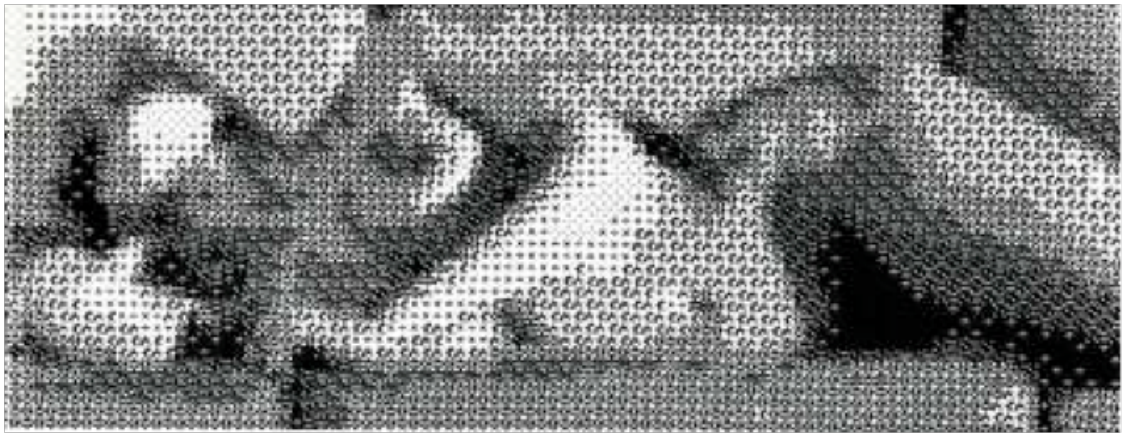

Figure 2 "Nude" by Kenneth Knowlton and Leon D. Harmon, 1966

Source: Compart: Center of Excellence Digital Art

Example 3: Screen shot of ASCII Sector

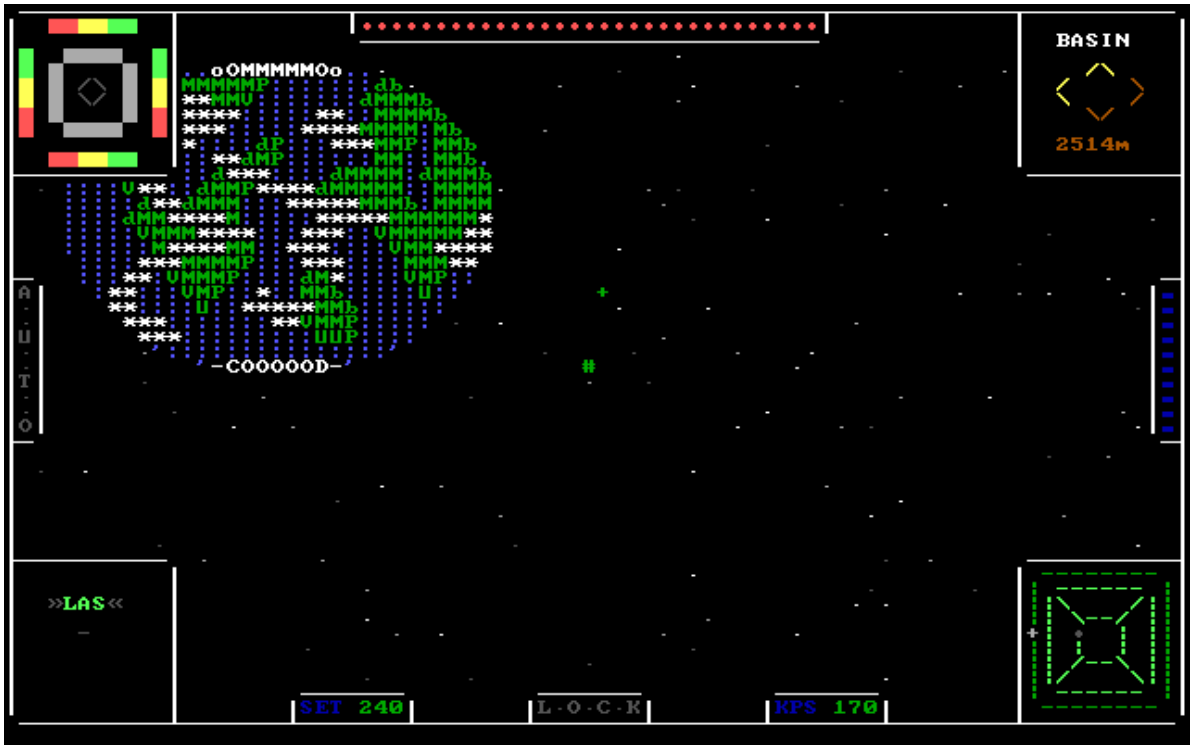

Figure 3

Soruce: http://www.mobygames.com/images/shots/l/560976-ascii-sector-dos-screenshot-flying-the-ship-in-space-note.png

Thinking the surface level representation of what code manifests as part of that language's morphology starts to make more sense when we look at examples like this. It underscores the precarity of language itself, the precarity of language in and of a digital milieu, but also reveals that our contemporary computer interfaces are not as seamless as they appear.

[i] Mark C. Marino. "Critical Code Studies." *Electronic Book Review*, December 4th, 2006. Accessed October 1 2015. http://electronicbookreview.com/thread/electropoetics/codology

[ii] Mark C. Marino. "Critical Code Studies." *Electronic Book Review*, December 4th, 2006. Accessed October 1 2015.

[iii] Michael Mateas. "Weird Languages." Accessed October 1 2015. https://users.soe.ucsc.edu/~michaelm/publications/mateas-software-studies-2008.pdf

[iv] Ibid.

[v] The Unicode Consortium. "What is Unicode?" Accessed October 1 2015. http://unicode.org/standard/WhatIsUnicode.html

[vi] Rosa Menkman. "Glitch Studies Manifesto," in *Video Vortex Reader II*, ed. Geert Lovnik & Rachel Somers Miles (Amsterdam: Institute of Network Cultures, 2011) 336-347.

[vii] Mark C. Marino. "Critical Code Studies." *Electronic Book Review*, December 4th, 2006. Accessed October 1 2015.

[viii] Gilles Deleuze & Felix Guattari. *A Thousand Plateaus: Capitalism and Schizophrenia*, trans. Brian Massumi (Minneapolis: University of Minnesota Press, 1987) 84.

[ix] Wendy Hui Kyong Chun. "On 'Sourcery,' or Code as Fetish," *Configurations* 16.3 (Fall 2008): 307.

[x] Gilles Deleuze & Felix Guattari. *A Thousand Plateaus: Capitalism and Schizophrenia*, trans. Brian Massumi (Minneapolis: University of Minnesota Press, 1987) 80.

[xi] Friedrich Kittler. *Gramophone, Film, Typewriter*, trans. Geoffrey Winthrop-Young and Michael Wutz (Stanford: Stanford University Press, 1999) 14.