

## expansion within Restriction: conviction and poesis in computer poems

### Abstract

Coding processes used by poets in the 1960s will be introduced to show types of structures in which digital poetry has thrived and managed to perpetuate itself. I'll present an analog template that can be used to recreate one of the earliest computer poems designed by an American (Emmett Williams "IBM" 1966) as a hand's on example of how coding has been employed to create a particular but constantly varying outcome. Other descriptions of early formulas and processes will be complimented by illustrations and brief demonstrations of programmed and, time permitting, software-built works. A range of restrictions are invented by the authors, with varying degrees of purpose and effectiveness. I know not the endless possibilities for code in literature, which seems well situated within the limits confronted by any text. Code is a mysterious presence, a labor that forces derivation and limitation on artistry devised and prepared in the contemporary landscape. The reason this is important? Beyond the possible importance of the history itself, by now perfectly legitimate (if not beautiful) pieces of digital poetry and literature are made without the author needing to type any sort of code (which remains even further behind the scenes than usual). Yet, tellingly, most of the sophisticated artists who have succeeded in the field do work with code, or at least combine automated scripting with hands-on programming. Does manipulating the restrictions through the intricacies of coding lead to more intricate, staying creations?

Chris Funkhouser

New Jersey Institute of Technology

April 4, 2008

The earliest works of digital poetry strictly involved coding as there were no other possibilities. Poets and/or programmers initially used computer programs to combine a database and a series of instructions that established a work's content and shape—the language generated by programmed poems, or computer poems, were assembled to the specification of the programmer. Formal, precise commands were coded to perform particular tasks, which at this juncture included, as Ted Nelson describes in *Computer Lib*, “loops, tests and branches, and communication with external devices” (15). These tools essentially permit elaborate permutation, or the transformation or re-ordering one set of base texts or language (i.e., word lists, syllables, or pre-existing texts) into another form—code included instructions for the production of the poem, which would include any formation of nuance.<sup>1</sup> As in any type of poetry, the computer poem relies on the author/programmer's senses, thoughts, inspirations, and structural inclination to form words, phrases, and poems. As always, the poet enacts language amidst a range of possible treatments. Using coded elements in language was familiar grounds for poets. Use of defined structures in classical poetry, across a wide range of styles—from the type of works Florian Cramer celebrates in *Permutations*, to sonnets and other metrical forms—is easy to find. Concurrent to dawning of the computer era, “code” poems were theorized and created by Concrete poets in Brazil.<sup>2</sup> [show “Birth of God”]

Almost immediately after the ball of digital poetry began rolling, three major things happened. First, the frames were used to frame wildly unconventional poetry. Second, the aesthetics of computer poems quickly evolved to include significant visual components. Thirdly, poets who had little or nothing to do with computers began to use certain types of commands and code-like motifs in non-computerized poetry.

Computers and programs used in the first generation of coded works are now obsolete. Zuse Z 22s or IBM 7070s, mainframes, punchcards, and so on: relics of a long gone era. Now artists

have more access to machines, and (one could say), freedom, as there are numerous approaches to digital expression available. Nonetheless, in certain respects (as I argue in my book), the outcomes in digital poems have not strayed so far from the intricate contrivances created in the 1960s. What has advanced is the technology that enables and delivers various new forms, such as videopoetry and different types of visual and hypertext works, to emerge. Interestingly, now there are many instances of digital artistry in which the author(s) can produce accomplished, sophisticated works without having ever having to engage with a single bit of computer code. For instance, Lionel Kearns's visual poem "Birth of God/uniVerse" (1965), uses the most basic elements of binary systems (rather than elements of coded language itself or implementing code on the computer) to fashion a simple yet striking image that suggests the close relationship that exists between text, image, and code in the new forms of contemporary expression.

[show picture] The diagram that accompanies the first citing of computer poetry in the mainstream media in the United States, which appeared in *Time* magazine in 1962, is accompanied by a caption that reads, "'A.B. Computer? Unfree verse" (A.B. stands for "Auto Beatnik," the program reviewed in *Time*) (99). Since I have been writing about restriction in computer poems, I was particularly startled by encountering the phrase "unfree verse," although in context the article is commenting at the high price of computing in the early 60s.<sup>3</sup> The tone of the article is curious, and concludes by surmising, "there is a chance of a whole new school of poetry growing up," yet the unnamed author essentially seems to make fun of the process, treating it with only a *faux* seriousness, suggesting, "the machine needs help" (a point I happen to agree with). Where the contents of this article directly intersect what I am thinking about is where it says, "by drastically cutting down its choice of words—so that the incidence of a subject word reappearing is greatly increased—engineers can make the machine seem to keep to one topic". So, in arguably the very first piece of criticism on the subject, the creation of the computer poem was seen as something that

involves limitation in order to succeed (or, it could perhaps be said, even to exist). This idea, of course flies in the face of the viewpoint that celebrates the endless possibilities of electronic text sometimes heard in discussions on the topic.

With that in mind, let me present—as an example of programmatic expansion and limitation, and also as a text that embodies *all* of the major developments in the early era of the genre mentioned above—one of the very first computer poems created by an American, Emmett Williams’s “IBM”.<sup>4</sup> The process here involves randomly choosing twenty-six words (which he does by chance, although he admits he may have cheated) and then associating each of them with a letter of the alphabet to create “an alphabet of words” (Valentine 3). A three-letter title is chosen, and the first line of the poem is determined by substituting words for letters in the title. Letters of words in one line are then used to make subsequent lines.

To demonstrate and practice the mechanics involved with this poem is the best way to become familiar with the types of conventions imposed on machinated works. To that end, I have recently prepared a few “IBM Poems” (one of which I will be performing tomorrow night) and have created an analog template for “IBM” that you can take home and practice with—and I should mention it would be a marvelous idea, since the original program has not yet surfaced, for someone to remediate this piece on the Web.

You can see Williams’s initial vocabulary and results on the back of the handout. The three words and ten letters generated from the title lead to three lines containing ten words; those ten words (containing forty-six letters) lead to the next ten lines, which contain forty-six words; the volume of the poem expands exponentially.<sup>5</sup> A visual aspect was added when Williams used a Datype machine to increase the size of the word each time it is repeated. He writes of the computer as “the muse’s assistant,” and that the poem as such amounts to an “eternal project,” but considered it “no great accomplishment” (4).<sup>6</sup> [show picture of another iteration]

What happens is that the repetition of specific words leads to repetition of lines, which would not be as interesting, were it not for the visual component. Williams's combinatoric texts feature self-contained generative dimensions at the level of the letter. Words are presented not to convey syntactically correct meaning, but are the result of the combination of letters found in a preceding iteration of the contrived structure, and are reused as many times as the letter initially appears. As words are repeated and increase in size, verbal and visual amplification is presented so that they appear to contain more weight and become, perhaps, thematic (as do "fear" and "idiots" in fig. 1). Using specific, pre-meditated equations, as did writers involved with Oulipo, Williams add qualities to the work that would not be present otherwise.<sup>7</sup>

If you experiment with the template you may discover immediately the benefits of using words with fewer letters. If you want a bigger poem, you will learn to use the longest words you can find (and quickly discover that the size of the lines, if you want them to be legible, will go beyond the boundaries of a typical printed page). [show example of printout] You might also end up—depending on what kind of poet you are—wanting to strategically manipulate the placement of words in the dictionary, and might also consciously select words which minimally repeat letters. If you want to use all of the words in the vocabulary, you will have to incorporate every letter in the alphabet in the words you choose. While I don't have much trouble with frequent repetition—which is unquestionably a characteristic of computer poetry and which can certainly serve a purpose—smaller words that each contain unique letters perhaps enables a more manageable poem for those who are looking for more verbal diversification. As soon as someone who is not committed to total randomness (or pointlessness) becomes engaged with such a process, s/he begins to consider ways to maximize the impact of the choices at hand. This is the craft of digital poetry in a nutshell.

There is an imposed "world" in computer poetry, in which the authors impose a variety of filters and gates that gives the work its substance and direction. Confines of the encodings are

revealed by content and output that is viewed. The evolution of software engineering has probably influenced a sense of restriction in digital writing practices, but even some of the best works coded for the Web, such as Jim Carpenter's *Erika* or Charles O. Hartman's *Pyprose* are unable to break through the tendency towards inevitable redundancy and repetition. In the analog world, on the other hand, it is clear that certain *code* artists, such as Mary Ann Breeze (*mez*) have been able to conceive of an expansive, flexible sense of language without writing in computer languages or using sophisticated software programs. Code-like effects are imparted in print, as they have been since "The Birth of God," Edwin Morgan's "Simulated computer poems," and Archie Donald's "Timesharing" in the 60s and 70s. [show pictures]

Working with programs that automate encoding has led authors to create simple and complicated original approaches, many of which are not meant to be expansive but rather made for contemplative or educational purposes, or to make social commentary through multimedia, or perhaps just for experiment or folly. I am now, as I did at the end of my book, however, considering the issues of textual confinement alongside aesthetic expansion and formal transformation in poetry. Can we, by somehow coding a vast text, possibly go beyond confronting a certain, pre-set set of circumstances in the digital poem? Or, is the recognition, understanding, and identification of the confinement itself what establishes the form, allows genre typologies to be built, and formal advancements to be made? I continue to search for digital poetry, or *system* of digital poetry, that not only contains self-generated aspects—something that contains a mechanism that allows for the expansion of expressive qualities in any text, or perhaps even from "scratch."<sup>8</sup>

In addition to positioning computer poems as expansive but restricted, the issue I wish to introduce involves the relative value of the use of code in literary and artistic works. Is the work of automatically adjusting parameters, through manipulating settings with a graphical interface, hold as much promise as approaching and handling it through its most inner workings? Casually, I'd say that

it doesn't matter whether the actual coding is done by the human or machine, but I'm not so sure that this is the truth. Code applies on so many levels: the linguistic, the algorithmic, in performance, as translation, through tagging, our use of search strings, etc.—but for the artist and participant(s) in the text interacting through software and the surface of the text is pervasive. Does software equal surface, and does this matter? How much coding actually goes (and will go) into digital literature, and by who? Should something that is not coded belong in a genre of its own (it sometimes seems the hardcore programmers practically scoff at something that isn't coded)? We know all artists are not technically based technicians, thus will there be, for those who are essentially unengaged with code, a tension between the means and the end result? This is such a non-issue for non-discerning users who are normally not required to think about code, but it is a type of demon, or at least a seriously unresolved issue, for me.

---

<sup>1</sup> In *Computers and Creativity* (1973), Carole McCauley effectively summarizes the three essential stages in the process of creating such works: “determining a frame (single words, lines, or stanzas with particular grammatical features); creating a dictionary of words for use in the frame; and finally, adding any extra qualities or instructions (the machine should choose only certain rhymes or words beginning with a particular letter or should print the results in certain patterns on the page)” (113).

<sup>2</sup> In the concretist works I am referring to, a lexical key in the form of graphical symbols is provided by author or authors, and, in one of the best examples of this, Luiz Angelo Pinto's 1964 “code” poem, is an arrangement of diagrams and is read accordingly, despite the absence of language in the work proper. The objective of artists like Pinto and Décio Pignatari, as stated in their semiotic poetry manifesto of 1965 (and referenced in *An Anthology of Concrete Poetry*), was a view that conceived of a language in which visual signs “might be designed so as to determine the syntax of a work” rather than verbal signs (n. pag.).

<sup>3</sup> The article claims the auto-beat computer costs \$100,000, thus “the output will certainly not be free verse” (99).

<sup>4</sup> “IBM” is documented and versioned in Williams's book *A Valentine for Noël* (Something Else Press, 1973)

<sup>5</sup> Williams, however, was not particularly interested in this possibility beyond processing the letters of each of the words in second set of lines one more time—probably because configuration of the lines becomes quickly redundant as a result of the small vocabulary.

<sup>6</sup> In the rendition of “IBM” shown in *A Valentine for Noël*, Williams enhances the process by imparting a “cylindrical” effect by shifting the vocabulary attached to the letter after the third process of substitution is made (i.e., at that juncture the initial “a” word becomes the “b” word, “b” becomes “c,” and so on) to create new poems. In the version of “IBM” that appears in the book, each letter is matched with each word in the vocabulary one time, thus twenty-six distinct pairs of poems appear, each stemming from “IBM” as input.

<sup>7</sup> The Oulipo group, founded in France in 1960, advanced various forms of non-computerized procedural poetry and writing that employed arithmetic and other programmatic constraints.

<sup>8</sup> If what Brian Stefans suggests in *Fashionable Noise* is true, then authors of computer poetry will “aspire” to make texts that feature convention because of the works' active and “parasitic” relationship to its primary hosts, publishing and academic institutions. Setting aside some obvious projects that prove there are numerous artists who are their own hosts (e.g., *VISPO*, Young-Hae Chang Heavy Industries), and are making digital poems for their own creative purposes and reasons, I think there's even more to be added to Stefans's solid observation. One could

---

reasonably argue that many digital and computer poems are self-parasitic. I have written elsewhere about digital poetry as a textually cannibalistic form, and that is true—but sometimes what we see, through various forms of permutation, is the text feeding back into itself, consuming itself to progress. And when it isn't busy eating and regurgitating itself, it is a text that is so much about itself, containing technologically driven internal commentaries (direct or indirect) through the text or textual objects. It is clear that digital poetry (and all instances of computer poetry) tend to feed off of something, whether it is historical poetry or other inscribed texts, or itself. I would not qualify this view as necessarily conservative, or negative, but as a certain perspective on the subject. The self-reflexive nature of works created may be prohibitive—keeping it too self-contained—but there is certainly artistry within it. Authors have proven that the implementation and combination of good input texts can lead to the creation of inventive electronic poetry. Thoughtful consideration, and embracing limitation, gives rise to form, to the identification of structure—indeed the plausibility of a genre of writing that includes multimedia and interactivity.