There is no hardware.

A thesis paper submitted in partial satisfaction of the requirements for the degree of Masters of Fine Arts in Digital Arts and New Media

Nik Hanselmann 2010
abstract

bodyfuck is a computer programming environment that translates bodily movement into computer programs. Each physical motion committed by the programmer/performer becomes an operation in the esoteric programming language brainfuck. Theoretically, because bodyfuck is Turing-complete, all computable operations are possible. Despite this, physical difficulty in implementing more complicated programs may prove to concede the theoretical limitations of the potentially computed output.

demonstration and exhibition history

Documentation of fifteen bodyfuck programs were exhibited at the Things That are Possible MFA show from April 30th through May 9, 2010. Each program was presented as a video capture of bodyfuck software interpreting my physical gestures. The fifteen programs presented were: save, heart, going nowhere, smile, walking to cell 68, help, copy, path, hello world, R^PcX (failed crawl), 2+2, unmoving, endless, undo, and 2000.

introduction

“Software Art” has endured through being contextualized as the spiritual successor of a profusion of 20th century projects. Of these, Formalism and Conceptualism seem to be predominant, as the environment in which software is conceived to operate is within the realm of a continuation of the linguistic and conceptual dematerialisation of the art object. The cultural context of the computer as a symbol of its history as technological statis object also influences the positioning of Software Art in these contexts, where the reappropriation of interrogative material practice is often perceived to be employed in order to subvert and realign cultural notions of what it means to live with technology. These contexts, while useful to admit software into the dialogue of Art, are not sufficient for explaining the contemporary “materiality” in which software operates.

This paper will attempt to reconcile 20th century historical precedents of material inquiry in both the visual arts as well as literary theory to reposition software praxis outside of the recapitulation of mise en abyme in favor of a conception of software performativity. This conception of software as performance will elucidate the relationship between inscription and execution as well as the ‘site’ in which software performs. This model of analysis will serve to contextualize my MFA work, bodyfuck, and provide a framework of mutative software praxis.

This paper recounts historical models of “sites of execution” in visual, literary, and performance based work to reconsider what it means to encode and perform language. This analysis is employed in the examination of the context in which “code” becomes performative and provide a model that positions the body as the site of software execution. This framework is used to evaluate my work, bodyfuck, to explain relationships between encoding and performance in context to what it means to execute/perform software. The discussion of bodyfuck makes connections to performance art as a means to explain the subjectivities of performance and what that means in context to computation. Finally, my paper concludes with reflections on my MFA work as well as the future of critical software.

It is my hope that this inquiry into materiality and software proposed by my MFA work is not read as the recapitulation of conceptual devices of the past avant-garde, but a reconsideration of cultural engagement with technology of the present moment.
**bodyfuck description**

*bodyfuck* is computer software. As such, it takes form as a binary executable that is written to run on Apple’s OS X. As a binary executable it performs the operations of a basic programming language interpreter. However, instead of using plaintext programming source files as input, *bodyfuck* accepts input through a web camera that is then analyzed to produce programming code. By itself, like any programming language interpreter, *bodyfuck* is inert. *bodyfuck* requires some sort of interpreted code to act on, which provided by means of physical movement.

At its core, *bodyfuck* is a reimplementation of the esoteric programming language *brainfuck*. *brainfuck* was initially written by Urban Müller in 1993 in an effort to write the smallest possible programming compiler for the Amiga OS. In the original *brainfuck* implementation, the programming language was originally afforded eight commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>increase register at pointer location</td>
</tr>
<tr>
<td>-</td>
<td>decrease register at pointer location</td>
</tr>
<tr>
<td>&lt;</td>
<td>move pointer backward</td>
</tr>
<tr>
<td>&gt;</td>
<td>move pointer forward</td>
</tr>
<tr>
<td>[</td>
<td>if previous register is zero, jump to</td>
</tr>
<tr>
<td>]</td>
<td>if previous register is not zero, jump to</td>
</tr>
<tr>
<td>.</td>
<td>output register at current pointer location</td>
</tr>
<tr>
<td>,</td>
<td>input into register at current pointer location</td>
</tr>
</tbody>
</table>

These minimal eight commands have since been proven to afford Turing-completeness. Theoretically, this places *brainfuck* in a class of computation where it is capable of computing all computable functions. Despite this potential for functionality, *brainfuck* has been mostly relegated to a hobbyist language, where programmers compete in the of coding *brainfuck* to write programs that are self-modifying, programs that output their own source, and other implementations of Turing Machines.

*bodyfuck* forces programmers to use their body to perform simple operations with the *brainfuck* programming language. Jumping increases the memory register, ducking decreases the memory register, moving left or right either swaps the memory register or sets a point in the ‘magnetic strip’ by which the head will return upon the zeroing of the preceding register (a loop). Since *bodyfuck* is based on *brainfuck*, all Turing-computable functions can be computed. 28 bodily movements can produce the output HI while 3,847 movements could potentially factor a large integer, provided of course, the programmer is very careful to choreograph their process.

Because *bodyfuck* is inert in itself, *bodyfuck* can only be exhibited by means of interesting programming code. This amounts to physical movements that correspond to *brainfuck*-computable programs. For the Things That Are Possible MFA exhibition, I chose to implement fifteen separate programs: *save, heart, going nowhere, smile, walking to cell 68, help, copy, path, hello world, R^PcX (failed crawl), 2+2, unmoving, endless, undo, and 2000*. These programs were exhibited as recorded video documentation of fifteen performances. These performances occurred over the course of three weeks in the solitude of the Light Lab at the Digital Arts Research Center.

*bodyfuck* as well as the performances that it engendered places conventional computability in tension with movements of the body. While *bodyfuck* maintains its ability as to automatically compute, the process of transcribing the code into the computer becomes an arduous task. The embedding of physical difficulty into the creation of a computer program reifies the gaps between machine performance and physical performance. The computer didn’t get sore. I did.
historical contexts of encoding and execution

We are undertaking to publish a journal and beginning to discuss how and what. In a view of the fact that in it we intend to reduce everything to zero, we have decided to call it Zero. We ourselves will then go beyond zero.4

Kazimir Malevich
In a letter to Mikhail Matyushin, 1915

For Kazimir Malevich, “Zero” simultaneously marked a disconnect with previous Russian avant-garde movements as well as the beginning of a linguistic revolution. No longer was language to be representative; language itself was to be performative act. It was the Russian Futurist opera Victory Over the Sun that provided a foundational moment for Suprematism and it was there that Malevich’s Black Square was first exhibited.

When put into context with Italian and Russian Futurism, Kazimir Malevich’s Black Square is seemingly static. Rejecting conventions of spatially and temporally compressed kinesthetic movements, Black Square does not call upon a polemic between two-dimensional representation and four-dimensional moments of dynamism. Malevich was not concerned with painting dreams of mechanized progress, war, or frustrations of Modernity. Malevich’s Black Square was just that: a black equal-sided quadrilateral. Within Suprematist works the experiential component was “pure” engagement with form. To say that Suprematist forms are “abstract” would be a misstep; Suprematist work was to be an agent of Modernity by means of “direct experience.”5

Guiding Suprematist gestures towards direct experience through concept and action was the notion of intuitive irreducibility: “the square is not a subconscious form. It is the creation of intuitive reason.”6 This utopian ideal permeated later Russian Futurist and Suprematist work, culminating in Malevich’s visual projects as well as Aleksei Kruchenkykh and Velimir Khlebnikov’s zaum language. These experiments, looking to the past as well as the future, had the intention of activating linguistic performativity in regards to a suspected universal language.7

In many ways, Malevich’s Black Square as well as Kruchenkykh and Khlebnikov’s zaum experiments are generative systems. Founding the formal rules in utopian unattainables such as intuition and universal truth, the visual and linguistic lexicon of Suprematism was meant to function as progenitors for a range of “pure” communication. Black Square was an artifact of this gesture, demarcating a square and calling for an infinitude of geometric permutation as a gesture towards universalized visual language. The zaum language functioned similarly, as the components from which it was constituted functioned as a construct designed for intuitive permutability. This “intuitive permutability,” while debatably universal, was meant to perform the potential of a formal system that could be “executed” by the viewer.

In “Concepts. Notations. Software. Art.” Florian Cramer positions code as simply a set of “executable instructions [and] logical scores” that “by no means [are] limited to formal instructions of computers.”8 Cramer contextualizes Tristan Tzara’s Dada method for poem construction as an algorithm and poses that the translation of such a set of rules into any language, programming or natural, maintains the status of the method as software. Taking this consideration into account in with the Suprematist and zaum language experiments, the mechanisms that which allow “execution” are brought into the foray; even though Suprematism and zaum relied on a mystic universal as the means for “execution” of language, what can be reconciled out of this construction of language and this particular
conception of language as performance is the means by which software performs, that is the “hardware” of the site in which the software is encoded and produced. This “hardware,” be it assumed sublime truths, the mind, a greater social body, digital computation, et al. is the site in which software is rendered material through action.

In “Code as Performatve Speech Act” Inke Arns cites Codework by artists such as Jodi or Mez that aesthetically identify as code while not being technically executable by a computer. She contextualizes this work as performative, even though the programming code itself that is inherently mechanically performative has been subverted. Similarly, Florian Cramer cites the appearance of code in programming reference manuals as being “executable”. Both these readings of code position the performative engagement with code as a matter of locating “execution” of a formal code within the conceptual devices of the audience. Whereas the location of execution for the works of Jodi or Mez happen in opposition to a programming-literate subjectivity, the mechanically functional code of a programming manual “execute” with knowledge of the conceptual machinery of the programming language itself. In the context of the Malevich, Kruchenykh, and Khlebnikov, the conceptual machinery at play is the sublime beyond Zero or the “intuitive reason” that in which was meant to serve as a means for executing the visual and literary “code” of the Suprematist avant-garde.

computation as context

If software is subject to perform by the means in which it is executed, then software that is computer-executable is therefore in dialogue with what it means to “compute”. The contemporary convention for digital computation relies on the model put forth by Alan Turing in his seminal paper: “On Computable Numbers, with an Application to the Entscheidungsproblem.” Turing’s paper mathematically proposes a theoretical model in which numbers may be machine-computable. In the context of software, Turing’s model functions as conceptual machinery for “mechanically” expressing mathematical procedure. In this case, a Turing-machine could be considered as a particular mode of “reading” or “execution”: just as “intuitive reason” was the mechanism by which Malevich assumed his visual code would be encountered by the audience, Turing’s model for mechanical computation functions as a particular application of “reading” and “executing” in which formal codes could be activated for means of mechanical procedure.

It is important to emphasize the word “particular” in the circumstance of Turing’s model for computation as the mathematical proof used to describe his model does not occlude other means in which mathematics could be mechanically expressed. Turing’s model only prescribes the mechanics of a Turing Machine to a theoretical degree in which a Turing Machine could be capable of automating procedural instruction. The various particularities of contemporary computing, such as the bit, serves as one set of an infinitude of symbols capable of being computed by a Turing Machine. In addition, conventional computational conceptions of mathematics such as addition and subtraction by no means reflect any essential behavior of Turing-Machine, as the means in which these actions are carried out are only prescribed by a particular code that is Turing-Machine interpretable.

In this instance, it could be posited that any critique of Turing-computation is a critique of a particular lineage of epistemological methodology stemming from the history of mathematics. However, the essentialization of Turing-computation as logically reductive negates the capacity for which other methodologies can be inscribed into, act with, or act contra Turing’s theoretical mode of reading, executing, and performing software. In context to the aforementioned site in which software performs, software that functions in dialogue with Turing Machines could be contextualized in a greater schema that considers the context in which Turing Machines operate as merely one mode of engaging
“code” in general. It should also be considered that the site of software performance does not occur in the digital vacuum of computer hardware, but through multiple layers of conceptual devices that mediate the encoding and execution of the mechanically embedded process: the ascription of affect to input or output ultimately is produced by the conceptual devices at play within user-subjectivity. This relationship is summarized by Geoff Cox, Alex McLean, and Adrian Ward: “Coding requires human intervention and full access to the means of production. In this formulation, the human subject gains agency (the power to act), as one who assembles the apparatus as much as is assembled by it.”

The contextualization of Turing-computability as merely a genre of reading and execution as well as software as non-machine-specific processes further emphasizes the series of contextual relations in contemporary computing. Florian Cramer argues these relations inherent in computer software reflect the postmodern condition, that is, software appropriates, defamiliarizes, and activates other software as the material in which it operates. In context to conventional computer software, these efforts are made apparent by software that functions as a product of multiple levels of encoding, reading, and execution. Operating systems, machine code, higher level programming languages, computer hardware interface, and libraries mediate the means in which computer processes are in dialogue with one another. These processes, despite the appearance of being a product/producer of an insular digital environment, function as a product of a series of assumptions that are situated by a particular lineage of encoding, reading and execution. These assumptions operate in tandem with other modes of encoding, reading, and executing in a greater social schema that can serve to mutually appropriate, defamiliarize, and activate software processes across multiple genres of executable language; the means in which one language relies on its ability to become encoded/executed can be used to identify gaps and rupture the assumptions of the means in which another linguistic model encodes, reads, and executes.

An example of this interrogative relationship is visible in the history of Modernist and Postmodernist Art, where the unstability of the art object engendered a reconsidering of material and practice in regards to the paradigms in which art was encoded, read, and executed. While Modernist Art focused on the material and linguistic limits as a reification of the autonomous art object, Conceptual Art reconsidered the status of the autonomous art object where the reflexive nature of testing the linguistic limits of the art “text” became unsavory in light of “context.” This repositioning culminated in a critical reevaluation of the 20th century institution of art by posing other models for art encoding, reading, and execution external to the institution as a means of critique.

**Performance and the Edges Between**

For a Conceptualist critique of context, I will discuss Bruce Nauman, whose work was concerned with both the reconsidering of art/artist/institution relations as well as subjectivities of bodied performance/execution. In an 1980 interview with Michele De Angelus, Nauman remarked:

I spent a lot of time at the studio kind of re-assessing, or assessing, why, why are you an artist and what do you do, and finally that’s what the work came out of -- that question, why is anyone an artist and what do artists do. And so some of that early work after I got out of school had to do with how I spent my time. I paced around a lot, so I tried to figure out a way of making that function[ed] as the work. I drank a lot of coffee, so those photographs of coffee thrown away ...of hot coffee spilled ....

Nauman’s work redefines “what the artist did” by structuring the labor performed in the studio as the artwork itself by means of posing the canonical framework for reading, encoding, and executing art in
tension with the artist’s questions about what it means to be an artist. Nauman’s *Coffee Thrown Away Because It Was Too Cold* or *Stamping In The Studio* emphasized lapses between these contextual frameworks through the conflation of artwork and art work. For Nauman, these lapses, or ‘edges between’ are considered to be the moments in which the interrogation of a discipline can become interesting.15

Nauman’s examination of the ‘edge between’ contextual frameworks extends into his performance work, where the interrogated context shifts towards the subjectivities of the body. This investigation is witnessed in Nauman’s piece, *Body Pressure*, where he invited gallery visitors to follow a series of directions:

Press as much of the front surface of your body (palms in or out, left or right cheek) against the wall as possible.
Press very hard and concentrate.
Form an image of yourself (suppose you had just stepped forward) on the opposite side of the wall pressing back against the wall very hard.
Press very hard and concentrate on the image pressing very hard.
(the image of pressing very hard)
press your front surface and back surface toward each other and begin to ignore or block the thickness of the wall. (remove the wall)
Think how various parts of your body press against the wall; which parts touch and which do not.
Consider the parts of your back which press against the wall; press hard and feel how the front and back of your body press together.
Concentrate on the tension in the muscles, pain where bones meet, fleshy deformations that occur under pressure; consider body hair, perspiration, odors (smells).
This may become a very erotic exercise.

*Body Pressure.* 1974.16

As installed in the Konrad Fischer Galerie, *Body Pressure* was exhibited as a wall and series of instructions printed in German and English. *Body Pressure* invites the execution and appreciation of gaps between individual experience; each individual experience becomes its own experiential occasion that is linked to the bodied subjectivity of that gallery visitor. The directions to contemplate and appreciate one’s own interaction as a sensuous event further emphasizes the gaps between each visitors’ experience as the enjoyment of the piece is situated in what makes each visitor’s experience unique: their bodied execution of Nauman’s directions.

In context to the terms employed earlier in the discussion, *Body Pressure* emphasizes the means of execution for each performance rather than the mapping of the “code” to an expected output. The “code” that Nauman uses welcomes individual subjectivities to perform. In regards to computer software and Turing-computability as a merely a genre of encoding, reading and executing, it could be surmised that programming a computer can welcome an affect similar to Nauman’s *Body Pressure*. Despite the precedence of the machinic underpinnings of Turing’s theoretical framework, these mathematic assumptions can function to engender a profusion of user-performances, in accordance with, opposition to, internal, and/or external to machinic modes of execution.
The importance of a contextually-based user-subjectivity appears in the lineage of *Body Pressure* in its enactment by Marina Abramović as a part of her *Seven Easy Pieces* (2005). Recounting five performances from her contemporaries (Bruce Nauman, Vito Acconci, VALIE EXPORT, Gina Pane, Joseph Beuys) as well as two of her own pieces, Abramović presented seven days of seven different performances for seven hours. Abramović's performance of each respective work suggests a re-execution of a work based on her own subjectivity; Nauman's *Body Pressure* and Acconci's *Seedbed* are encountered and reinterpreted by Abramović to interrogate what it means for performance art to be dislocated from its position in historical ephemera to be re-documented through a different body: her own.¹⁷ The contextualization of the performances in *Seven Easy Pieces* as “documentation” of historical performance problematizes the relationship between body and subjectivity as the medium in which performance is executed. Again, the means in which “code” (directions) comes to perform is complicated the frameworks of execution.

**software and performance**

*Microcodes* is a series of code-based artworks created by Pall Thayer that bring means the of execution into the foray. Each artwork is short, discreet, and written in Perl. Thayer’s website presents each piece with a title and date.

```perl
#!/usr/bin/perl
use IO::Socket::INET;
$pid = fork();
if($pid==0){
  $myself = IO::Socket::INET->new(Proto=>'tcp', LocalPort=>4321, Reuse=>1, Listen=>5);
  while($conn = $myself->accept()){}
}else{
  sleep(5);
  $specter = IO::Socket::INET->new(Proto=>'tcp', PeerAddr=>'localhost', PeerPort=>4321);
  waitpid($pid,0);
}

*Introspection*. 2009.¹⁸
```

Typically, code that is interpreted or compiled by a computer is called "source code"; however, in *Microcodes*, this term becomes hard-pressed. Due to the textual exhibition in which *Microcodes* is presented on Thayer’s website, the mode in how a viewer is intended to experience each respective artwork is entirely dependent on their chosen level of engagement. Those unfamiliar with programming may still glean understanding of the relations between computational and expressive process through contextualization and the poetics embedded in the code itself. For example, what is encoded in Thayer's *Seedbed* is situated in both the history of art as well as computer culture:

```perl
#!/usr/bin/perl
use Cwd qw(realpath);
$myself = realpath($0);
while(1){
  `touch $myself`;
  print `ls -l $myself`;
}

```
As a Perl homage to Vito Acconci’s famous performance, what *Seedbed* does in regards to computation is quite simple: *Seedbed* repeatedly issues the "touch" command to the file containing the Perl script from which *Seedbed* is run from. The particular file is "touched" until the user quits the program.¹⁹

Thayer’s *Seedbed* is concerned with representation on multiple fronts through its reliance on encoded information that exceeds the computer itself (an homage to masturbation), while at the same time being subject to being computable. The performance of the code as an executed program is reified by making explicit its inner-workings; the inner-workings are multi-coded for consumption other than that of the computational. The inner-workings, or "source code," is therefore hardly "inner" or "source," as it is the code itself (as well as the contextualization as code) that can execute for the viewer, both with and without computational aid.

It is within this inversion of executed program/source code where Thayer shifts the level of representation. Generally, an executed program is considered to be the mode of interaction for most software users. Only a small percentage of computer users compile their programs from source code and even fewer take the time to glance at the source code before transmuting it into its functional compiled counterpart. With *Microcodes*, one of the most "functional" stages at which they operate is at the stage of disassembly before the textuality of the code is munged by the machine.

In context to the subjectivities in which code is performed, Thayer’s *Seedbed* lampoons the bodied execution of the original performance. Through purely textual/machinic execution, the qualities Acconci’s performance engaged with are notably absent, specifically: proximity, uncomfortableness, endurance, and eroticism. In this sense, Thayer’s reworking of *Seedbed* neuters Acconci’s work, as the means in which Thayer *Seedbed* operate are notably reduced to be computationally digested.

Artist collective Social Fiction confronts the means of execution in a different manner: by placing *Walkware* applets in the context of a psychogeographical computer. With *Walkware* humans act as agents of algorithm upon the “hardware” of a cityscape. Each human applet runs a .walk script.

```
// Classic.walk
Repeat
{
  1st street left
  2nd street right
  2nd street left
}
```

*Classic.walk* instructs the human applet to walk to the first street, turn left, walk two streets and turn right, walk another two streets and turn left, and repeat. These .walk codes engage city space in the spirit of dérivé as well as John Cage: directions without a map.²¹ As a human applet transverses the streets of a psychogeographical computer, the algorithm is injected with the structural incongruities of urban space. Each operation given by a .walk code becomes embedded with the surrounding environment of the human applet, mutating and multi-coding the *Walkware* code through physical action on the apart of the human algorithm. The performance of .walk code is thusly dependent on the architecture of the psychogeographical computer and the respective experiences of each applet. Rarely do *Walkware* applets result in their intended outcome; there are theoretically “perfect” grids on which *Walkware* applets can operate, but to operate on these grids would be to negate the injection of the experience of urban exploration into the code.²²

Thayer’s *Microcodes* and Social Fiction *Walk* approaches to coding supersede their respective output. The “computed” outcome functions to be an artifact of the coding process, the code and coding
process function to be more worthwhile than the “computed” outcome. This is not to say that computability is trivialized by these projects; on the contrary, computability reifies the space afforded through the intersection of code, encoding, and execution.

**bodyfuck performances**

*bodyfuck* conflates two contexts: body and computation. Unlike *Microcodes* and *Walk*, *bodyfuck* requires a level of discipline to reify moments of incongruities between the two contexts. These incongruities manifest as severe physical difficulty in performing the task of computational inscription as well as difficulty in the creation of a compilable program. This tension is manifest in the fifteen programs that I created with *bodyfuck* where both programmer’s body as well as the *bodyfuck* program are engaged in multiple levels of feedback. This feedback functions to reify moments in the edges between bodily and computational moments of performance.

For some of the *bodyfuck* performances, I created a set of programs that typically refer to contextual verbs used in everyday computing: *save, help, undo*, and *copy*. These contextual verbs are generally available in most contemporary computer programs as a means for the user to manipulate their computing context. The verbs ‘help,’ ‘undo,’ ‘save,’ and ‘copy’ function the same every time: ‘saving’ generally inscribes bytes in memory to the hard drive, ‘undo’ removes the last operation in a linear fashion, ‘help’ returns a manual, and ‘copy’ takes an exact amount of information and duplicates it into a new computational context. These specific computational metaphors were selected for their connotation of action, that is, they are meant to indicate to the user their ability to perform. The *bodyfuck* implementation of these verbs stresses the gaps between the contexts in which these verbs operate. The *help* performance took fifty-two gestures to inscribe and execute the simple program which only functions to output the text of the program’s title. The machinic execution of my inscribed operations was simultaneously reduced and produced by present bodily predicament of being exceptionally sore from previously performed programming exercises. The presence of extreme pain in the reductive algorithmic setting of a compiled program reifies a gap in the means in which the code was executed through my own bodied subjectivity as well as the compiled program. The verb ‘help’ in the circumstance of the *help* performance thusly contrasts a bodily dire situation with computational operation.

In contrast to help, *going nowhere* works in a different manner by correlating computational operation to the gestures required to produce them. The operations in *going nowhere* produce no output; the *brainfuck* program only commits a series of computational actions internal to the machine itself. By walking in circles, my body inscribed a set of operations that only amounted to moving the pointer back and forth. What this resulted in was a repetition within two contexts: the circularity of my physical movements as well as the circularity of movement within the compiled *bodyfuck* program. The contexts in which both bodied gesture and computational gesture occurred in parallel.

There were a series of performances that resulted in extreme difficulty such as *crawl* and *push*. *crawl* was exhibited as $R^\text{PCX}$ (failed crawl) due to the difficulty of inscription. These performances required the most physical capacity to perform. *crawl* was meant to be inscribed by means of crawling which meant that no register could be increased, only decreased. Because the memory registers in *bodyfuck* wrap around from -127 to 127, this meant that there needed to be an excessive amount of inscription and computation. As my body had to spend a lot of time on the floor, the compiled *bodyfuck* program had to do a great deal of subtraction to get the memory registers to their appropriate values. The physical requirements for these performances exceeded my ability and through multiple attempts left me writing programs such as *help*. It is within these performances that the tension created between
computational and physical contexts are pushed into the foray; the soreness amassed by these attempts at inscription impacted both my physical ability to perform as well as execution of the *bodyfuck* program. The output of *crawl*, ‘R^PcX’, is thusly an artifact of the troubles of physical translation proposed by performing with *bodyfuck* software.

The most computationally complicated performance, 2+2, illustrates the physical gestures required to represent the addition of two numbers. The performance of adding two and two begins with setting the value of the first two registers to two. The memory cells are then non-destructively added to each other into the third cell, where they are then appended through a loop to the ASCII value of zero. This results in forty-eight plus four, which results in the output of the program to be visually represented as four. Coincidentally, the number of physical gestures required to inscribe a program capable of adding two plus two in a representable fashion is the same number of gestures that would be required to simply output the number four without any actual computation of addition. The various steps incurred in the physical inscription of 2+2 were exacerbated by making the computed result visually expressible; what the inscription of 2+2 amounted to was a physical working-through of multiple representational levels in order to reify the ephemera of computational process. The physical inscription necessitated the most disciplined inscription of all *bodyfuck* performances, requiring my physical gestures to be exceedingly precise in order to produce exact calculations.

The *bodyfuck* performances ultimately present various stages of negotiation between bodily gesture and computation. Programs such as *save, help, undo,* and *copy* question verbs of conventional computer interface. *Going nowhere* maps a direct correlation between physical gesture and computation. My attempts at *crawl* reifies the context of physical constraint of bodied performance. And 2+2 emphasizes computational complexity and discipline for means of representing computational process. Each of these discussed programs shift the means of execution, whether more closely tied to my physical ability or towards the context of computation, to contrast different moments of performance/execution.

**subjectivities of inscription**

It is important to note that the bodily gestures performed in the *bodyfuck* performances are merely means of inscription of code and not computation. By posing bodily gestures as inscription, *bodyfuck* conflates the multiple levels of mediation incurred in the execution of software. In a sense, *bodyfuck* extends N. Katherine Hayle’s notion of the ‘flickering signifier’ external to the context of computation: the act of inscribing is no longer a single act that engenders a chain of mutable signs in the computational context. With *bodyfuck*, this chain of mutable signs extends into the physical performance where mutability of signifiers is engendered by the physical performance of the body. The conflation between physically performed signifiers with computational signifiers further emphasizes a tension of physical/computational context. Whereas the ‘flickering signifier’ is capable of being easily alterable by means of computation, the signifiers produced by the physical gesture are constrained by the particular subjectivities of the programmer’s body. With *bodyfuck*, this incongruity is produced as physical affect.

The attribution of physical affect into the inscription of computable code imbues a particular moment of subjectivity into each inscribed operation. In context Abramović and Nauman’s work, the particular subjectivity imbued with *bodyfuck* was the conditions of my body in computational context. This imbuing of subjectivity functions contrarily to projects such as *Microcodes* and *Walk* in that *bodyfuck* challenges Turing-computability as a reading with bodied subjectivities. As with Abramović’s *Seven Easy Pieces*, the particularities of performance in *bodyfuck* (in and outside of the computer) question where software is truly performed.
conclusions

I have traced a lineage of encoding, execution and performance from the early Russian avant-garde to works of Bruce Nauman, Marina Abramović, as well as contemporary Software Art. In this discussion, I have emphasized the subjective means in which software operates in terms of both execution and inscription. *bodyfuck* operates within this framework, illustrating different moments in performativity in regards to the workings of software.

*bodyfuck* as well as this paper are a reaction to the contextualization of formal Software Art that Florian Cramer, Inke Arns, and as well as many others employ to understand so-called generative works that pride their formal constructions of algorithm. As with the generative visual and literary language of the early Russian avant-garde, it is important to understand that language necessitates both a means to be inscribed as well as enacted. In this capacity, criticism of computation and software is not relegated to formal logics of computer hardware as claimed by Friedrich Kittler in "There is No Software," but the construction of “information” technology in its entirety.

Considering the construction of information technology as an amalgam of subjectivities engenders a conception of computation beyond being rote, mechanical, and reductive operation, to a mere component in a greater culturally embedded apparatus. *bodyfuck* engages with Turing-computability as well as cultural conventions of computation (contextual verbs, mathematics) as quotidian and fundamental operations that are often overlooked as sites of critique. By shifting how we perceive these operations into a physically subjective context, *bodyfuck* engages with a broader field by interrogating assumptions that have been embedded into contemporary computational lexicon.

Unfortunately, the means by which *bodyfuck* interrogates convention is absurdity and uselessness. Performing with *bodyfuck* merely adds another layer of complexity in the interface to an already unusually complicated programming language. Despite this, the framework employed in *bodyfuck* and explained in this paper can be used to engage with software that focus on the relationships it produces in a greater context. By considering the site of software performance outside the computational execution, software can be written that assumes its execution and performance in the greater cultural context in which it truly functions.

There is no hardware.
1 As proposed by Troells Degn Johansson as the “dead end” that Cramer invites in his 2002 Transmediale paper. Essentially, spinning wheels within the quagmire of "language at its limits". See: Troells Degn Johansson, "Mise en Abyme in Software Art: A Comment to Florian Cramer", http://art.runme.org/1107862109-3200-0/johansson.pdf

2 http://esoteric.voxelperfect.net/wiki/Brainfuck


13 “Conceptual Art was an experimental and critical praxis fundamentally connected to the communication processes, information economies, representational systems, ideological power structures, and codes of the surrounding social and cultural reality, including the art institution” -- Lillemoose's reading of Brian O'Doherty's “Inside the White Cube. The Ideology of the Gallery Space” (1981)


15 Nauman continues: “I think that with art or philosophy or any kind of -- at the edges of any discipline, if you think of art as discipline, the people that are interesting are the people that are exploring the structure of the discipline. In that sense they're breaking the discipline down, too, as they're expanding it. They tend to break down what's there. Certainly there are artists who function entirely within the discipline. I would find those people uninteresting. Not that they're not talented or skilled or all those things, but it’s not of interest to me. In that sense there's a great deal of confusion, because it doesn't require being able to draw or being able to paint well or know colors, it doesn't require any of those specific things that are in the discipline, to be interesting. On the other hand, if you don't have any skill at all, then you can't communicate, either, so it's an interesting edge between -that edge is interesting for those reasons.”

16 http://www.e-flux.com/projects/do_it/manuals/artists/n/N001/N001A_text.html

17 Jessica Santone, Marina Abramović’s Seven Easy Pieces: Critical Documentation Strategies for Preserving Art’s History (Leonardo, Volume 41, Number 2, April 2008) 147-152.

18 Pall Thayer, Microcodes, http://pallit.lhi.is/microcodes/

19 touch is a simple tool that sets the "last modified time" of a file to the present time on the computer.
Walkware rarely executes as planned. During the 2003 Image Festival in Toronto, 25 small teams tried to make it to Trinity-Bellwoods Park using a map created from a previous .walk applet. Due to the incongruities of algorithm and space, this .walk was particularly afflicted by confusion and resulted in several glitches. “Some groups turned out to be shitty map readers because we never saw them again, somebody told us that at not far from were we were several groups did actually meet. But that was certainly not at the right place; that was where we’re standing.” See: Social Fiction, “Report Toronto Walks”, http://www.socialfiction.org/psychogeography/dotwalktoronto.html

push was not exhibited at all.

N. Katherine Hayles’ “flickering signifiers” “[exist] as a chain of markers bound together by the arbitrary relations specified by the relevant codes.” Here, ‘arbitrary’ in this circumstance implies the programmer’s agency within the application and ‘relevant codes’ implies the cohesive connection to the logical faculties of the digital computation device. There is no one-to-one correlation as found on a typewriter; instead each unit of input allows for transformational shifts to occur anywhere along the path of mediation. A single command can be given to transform the string of signifiers into a file, copied into an email, edited, or reconfigured towards “unexpected metamorphoses”. See: N. Katherine Hayles, How We Became Posthuman: Virtual Bodies in Cybernetics, Literature, and Informatics, (U. of Chicago Press) 30.