Tangibility and Low-Level Live Coding

Georgios Diapoulis
No affiliation
ydiapoulis@gmail.com

Ioannis Zannos
Ionian University
Department of Audiovisual Arts
zannos@gmail.com

ABSTRACT

Current advances in digital fabrication are accompanied by efforts to bring about increased facility in the fabrication of digital circuits. In this context, tangibility can apply to the intimate contact with objects as programmable entities forming part of the human-material loop in the sense of physical computing. In this paper, we explore the possibilities of making music with very simple circuits, using an equally minimal interface for live interaction with the hardware.

Our aim is to find new ways for experiencing the behavior of circuits and for navigating inside the data space of generative algorithms with musical devices by using minimal interfaces, while involving both human and machine in the perception of the musical output. In our experiment we have focused on the lowest level of the machine language [1], that of manipulating bits in real-time. Furthermore, we attempt to tighten the loop between human and machine by introducing a machine listening component which processes the output of the human-machine interaction. This splits the perceptual feedback loop into a human and a machine part, and makes the final output a joint outcome of both.

1. INTRODUCTION

1.1 LHC and Tangibility in live music-making

Live coding as a practice blurs the limits between composition and performance [2]. It is also considered as a new notation form [3] or an approach for developing notations, albeit in its infancy. Live coding from scratch usually follows a bottom-up approach, building musical elements incrementally from simple to more complex ones. Yet from the point of view of the computer, the musician uses high-level abstractions for coding (for example: SinOsc, Synth, Pattern, Routine, Scale). In this paper we present an approach that attempts to start from low-level computational elements, namely bits and their manipulation through simple automata that make up the building blocks of digital machines. Writing code at the lowest level of machine language seems hardly a practical method for constructing music, and no more a usual method for constructing programs. Our experiment is motivated by the question, whether it is possible to create an intimate link between musician/coder and machine which, while still based on digital programming principles, addresses both the human and the machine in an integrative manner. By that we mean integrating different functions or faculties such as sensation (input), perception (human/machine listening), planning (computation), action and comprehension as closely connected as possible, in order to form a closely knit whole. This is a prerequisite for forming the closest possible interactive link between human and machine, one that creates a "tangible" contact between the two.

In this experiment we are not applying any design metaphors from computer music, but look for grammatical interfaces that open the way to hacking, as suggested by Stowell and McLean (2013) [4]. Our starting point is not music, but computational processes. Our question is how the performer can use computational processes as raw material to form into something approaching a musical performance. Therefore, we developed devices and interfaces that serve to communicate to computer musicians some fundamental notions and processes of computing machinery, such as bits, symbols, binary representation of numbers, serial transimission, decoding and encoding process, and lexical analysis. We do not ask of the performer any previous experience in programming. Such an apparatus might have educational uses [5].

In our design, we sought to obey the general rule of presenting the user with 7 ± 2 elements at any moment (usually refered as Millers law). We use three input switches, one reset button, a potensiometer and two LED displays which reflect the current state of the machine. Equipped with this interface, we sought to develop a bottom-up minimal programming language and environment for live music-making.

We presented first results of this low-level approach to live coding in Diapoulis and Zannos (2012). The language we used likely is being generated by type-0 grammars [6] in the Chomsky hierarchy. Whether or not this language can be followed by humans remains an open question; we would like to thank Nick Collins who set this question during live.code.fest in Karlsruhe, 2012. The basic design that serves as starting point is a 3-bit minimal interface that drives a counter coupled to a decoder as generator of musical structures. As a next step, our aim is to develop interfaces that enable us to explore and experience the behavior of these processes as musical processes at the building block level [7], that is, as musical phrases or sections com-

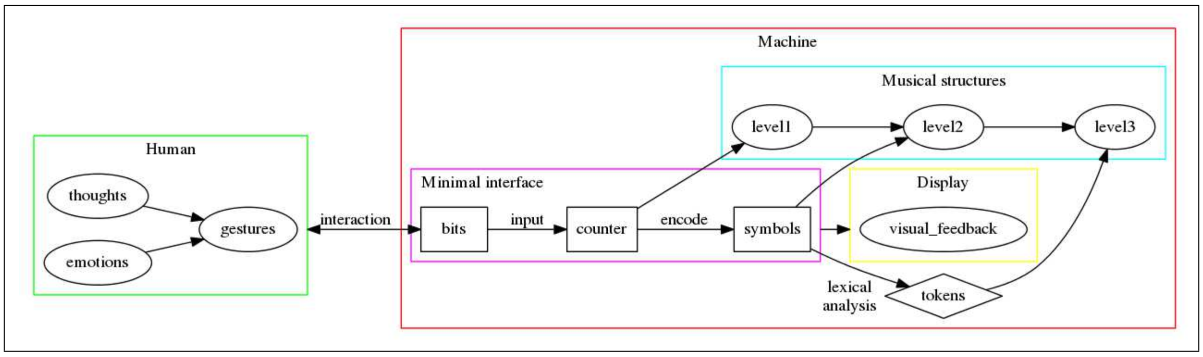

Figure 1. Schematic setup of our expreriment and its main components.

prised of groups of single note events. Here we present first results of this approach, which add higher-level processing of the initial output by the machine. This can in turn be used as musical material, thereby enriching the final outcome. Figure 1 presents schematically the overall setup of our experiment and its main components.

Futhermore, we have developed a new hardware interface which allows more intimate and direct tactile interaction with the digital circuit through flexible buttons that require very small and light movement, and can be operated while remaining almost entirely in contact with the device.

2. EXPERIMENTAL SETUP

Our experiment is based on the combination two elementary blocks of digital design: A counter and a decoder. Both are sequential circuits which can be represented by a finite state machine [8]. The counter is a 3-bit counter machine which operates as the modulo 8 function using 2's complement. The decoding machine is a Huffman decoder which operates with variable length code and uses a combinational encoding process to procudes symbol sequences from an alphabet of four symbols with specific weights. The human agent provides a 3-bit parallel input to the counter by means of three switches and a potentiometer. We have developed two different machines, one with a fixed clock and one with a variable clock rate. The potentiometer controls the counter's positive edge clock. It it is an external module which applies only to the machine with the variable clock rate. The output from the counter machine is read in serial order by the decoding machine. The decoder has a single bit input, and an output alphabet of four symbols (A, B, C, D).

In the original experiment, both machines were developed using prototype circuit boards and TTL technology. The output from the counter and the decoder are sent to an arduino board, which is connected to SuperCollider via USB cable. We use SimpleMessageSystem arduino's library which is controlled from ArduinoSMS class in SuperCollider. SuperCollider is responsible for real-time sound synthesis. We have mapped counter's output, numbers 1 to 7 to the seven diatonic degrees and 0 (zero) to silence (pause). The four symbols produced from the decoder provides us the opportunity for senondary mapping. The software used for this experiment was packaged as a Quark for SuperCollider [9], named LHC [1] [2], see Figure 2.

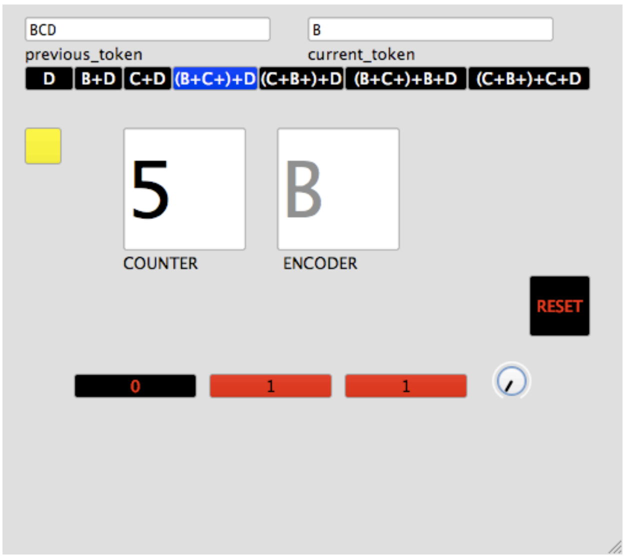

Figure 2. GUI interface implemented as a Quark for SuperCollider

Following diagram gives an overview of the experiment (Figure 3).

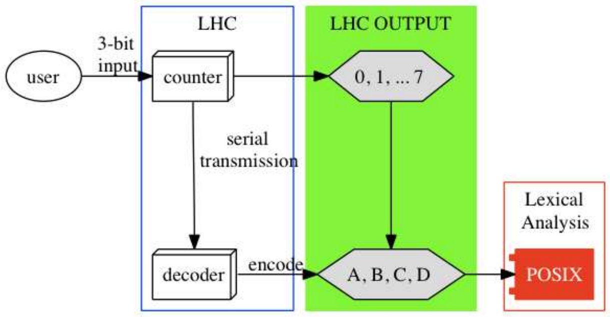

Figure 3. A high-level diagram of the system

[1] https://github.com/iani/SC/tree/master/Quarks/iz.projects/LHC
[2] https://gist.github.com/yorgosdiapoulis/11365609

3. PROCESSING AND OUTPUT

The input is provided by the human agent in terms of a 3-bit parallel input. This drives a counter machine which implement the modulo 8 addition function in 2's complement. The counter transmits in serial order the 3-bit output to Huffman decoder machine. The final output is a stream of symbols whose alphabet consists of the four symbols (A, B, C, D). Overall, the output produced by our initial device (2012) had three levels:

1. The output of a variable-rate counter programmed by the 3-bit switch interface and the potensiometer.

2. The output of sampling the states of the counter at a steady-rate, programmed only by the 3-bit switch interface.

3. The decoding of the sampled states by a Huffmann decoder into a stream whose alphabet consisted of the 4 symbols A, B, C and D.

4. OBSERVATION OF OUTPUT: EXPLORING PALINDROMES

From the three levels of output described above, output level 1 presents some interesting characteristics. The variable-rate clock of the counter is adjusted by a knob, while the rate at which the state of the (variable-rate) clock is sampled by the software system that receives its output is steady. This creates a downsampling-artefact which results in quasi-palindromic structures shown in Figure 4. We created a class LHCV [3] in order to simulate this process and explore it in greater detail.

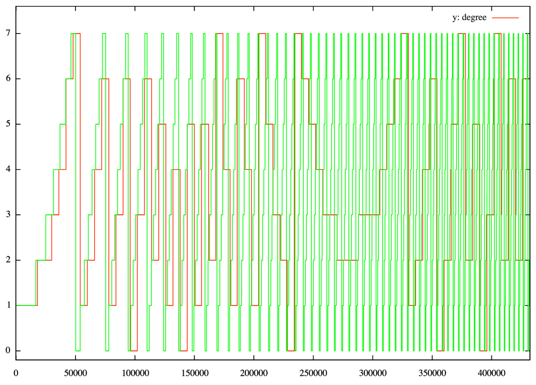

Figure 4. Original values of variable-rate clock in green, latched values by steady-rate clock in red

Through this simulation, we can confirm in software the emergence of quasi-palindromic structures which was observed in hardware. In this paper we give the formal framework for describing these phenomena.

Such an approach has applications in education but also in design at all levels. It also opens new ways to approach live coding [10]. The value of low-level approach has already been noted [11]. Here we try to take this approach to the limit.

At the core of the LHCV-sampling simulator is the following algorithm - as coded in SuperCollider:

```
{ Latch.ar( Stepper.ar( Impulse.ar( Line.kr
    (1,99,9) ) ), Impulse.ar(8) ) }.plot(9)
```

The above code produces quasi-palindromic structures as demonstrated in the following plot (Figure 5). The X-axis represents the number of samples and Y-axis represents the diatonic degrees from 1 to 7, and 0 (zero) is for pause.

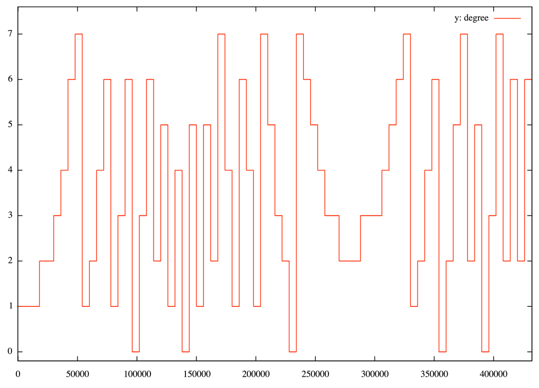

Figure 5. Quasi-palindromic structures. Y-axis: diatonic degrees. X-axis: number of samples

The palindromes were a natural first outcome of the mechanism, and illustrated a way in which such an elementary process can be induced to produce structures that are recogniseable at a higher level - a kind of "emergence". The next question in this respect is to determine the ratios of counter rate and sampling rate at which such palindromes occur. The first argument of the Latch UGen is the input, while the second is the trigger for latching the value. The Stepper operates as the modulo 8 function and its first argument is the trigger. This observation demonstrates that by applying a linear function into the frequency argument of the trigger (Stepper) is an approach for generate quasi-palindromic structures. A characteristic audible example is the following code excerpt in the form of a SuperCollider "tweet" (see `https://ccrma.stanford.edu/wiki/SuperCollider_Tweets`):

```
play{p=Impulse;SendTrig.ar(Changed.ar(a=
    Latch.ar(Stepper.ar(p.ar(Line.kr
    (99,1,40,1,0,2))),p.ar(8))),0,a)};
    OSCFunc({|m|(degree:m[3]).play},'/tr')
```

Code excerpt below builds a GUI for trying out various parameter configurations of the counter-sampling algorithm interactively [4].

```
SynthDef(\lhcv, {|clk=1 xclk=1.1 input=1|
    var p = LFPulse;
    var signal = Latch.ar(Stepper.ar(p.ar(
        xclk), step: input), p.ar(clk));
        Out.ar(0, SinOsc.ar(100*signal))
}).synthGui(
```

[3] `https://github.com/aucotsi/sc3/blob/master/LHC/LHCV.sc`

[4] The code makes use of the Lilt2 Library by Iannis Zannos, `https://github.com/iani/Lilt2`

```
    specs: [
        clk: [0.1, 2.0],
        xclk: [1.0, 20.0],
        input: ControlSpec(0, 7, \lin, 1)
]);
```

5. IN SEARCH FOR FURTHER PATTERNS: A MINI-LANGUAGE FOR LHC (MLHC)

The observations about the emergence of patterns made at the first stage of the experiment above led to the question whether the machine could also detect patterns in the signal, using algorithmic ways of processing the output. Since the patterns of output level 1 were recognizeable by humans our "bet" was what kind of patterns the machine could recognize from the symbol stream that is the output of level 3. To analyse the string of symbols we employed the techniques of regular expressions, which are one of the first tools of choice for such tasks. These expressions define regular languages, that is formal languages that are equivalent to non-deterministic finite automata (NFA) [12]. We thus defined a mini-regular-language for musical live coding.

"mLHC" is a regular language in Chomsky hierarchy. The alphabet of that language consists of the output symbols from the decoder/encoder. Each word is being recognised at run-time through lexical analysis with POSIX expressions.

5.1 Alphabet

The alphabet consists of three letters (symbols). Symbol A is mapped to the empty string ε (A $\rightarrow \varepsilon$). In such a way we can reduce the complexity of the tokens. So the alphabet is $\Sigma = \{B, C, D\}$.

5.2 Language

We define the language L as a set which contains every product of the alphabet Σ^* and ends with the letter D, as follows:
L = $\{w \epsilon \Sigma^* : w\ every\ word\ that\ ends\ with\ a\ D\}$

5.3 Regular expressions

Using the following POSIX expression we can recognize every token that ends with a "D", which is used as an end-marker. The set of the accepted words have an infinite cardinality, though they can be expressed by a finite state machine.

```
// POSIX expression
D|B+D|C+D|(B+C+)+D|(C+B+)+D|(B+C+)+B+D|(C+B
    +)+C+D
```

Where plus (+) symbol, stands for "at least one".

5.4 Graph for lexical analysis

Figure 6 shows the non-deterministic automaton (NFA) which describes visually the recognition process on the ongoing output string from the encoder.

The start state is q0 and the final state is q3.

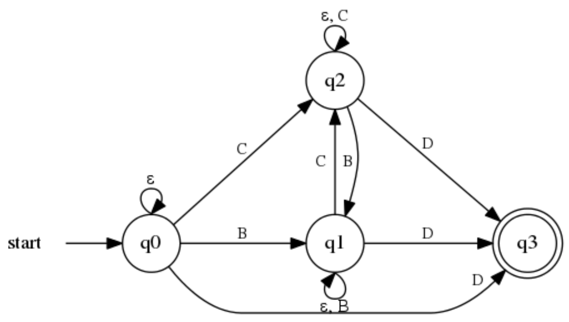

Figure 6. NFA for lexical analysis

6. DISCUSSION: PHYSIOLOGY, PERCEPTION, INTERACTION

The crucial question underlying these experiments concerns the relationship of unconscious and consious processes in musical experience. Is it possible to conduct music making through programming in a similar way as traditional live music making activities, that is, to involve the intuitive (unconscious) and physical levels of the creative process together with the highly analytical processes of programming? Already our interface has been pushing in this direction, since it is possible to run the clock rate at the limits of the perception of individual notes.

6.1 Fast vs. Slow

It is inevitable that next generations will be faster in their interaction with machines. We could imagine future systems of HCI that will improve our capabilities into this [5]. Speed matters in evolution but this is not the case in art practices. Slow code [6] represent a completely different perspective into this. But we are making music. Music is a complex phenomenon and a really demanding task. "Should music interaction be easy?" [13].

7. CONCLUSION

In this paper we presented experiments in combining software and hardware coding, aided by visual representations of the behavior of the coupled hardware-software processes. The observations made through the present experiments relate to several questions regarding the character of live coding as an individual experience and as a cultural activity. For example, it has been asked whether live coding is just a state of mind [14] or a self-referential cultural activity [15]. Live coders possibly open a new approach to use of technology in art through the policy of "show us your screens", that is, through the public display of programming activity during performance. This approach does away with any buffers or security cushions that protect from users' "mistakes". In live coding, concerns on safety are raised at a different level than in other environments and creative settings [16]. Perhaps in this sense live coding redefines the

[5] Video by Click Nilson https://www.youtube.com/watch?v=gi3jMQs0Gfs
[6] http://www.ludions.com/slowcode/

original meaning of the word "program" as a public practice which is being announced simultaneously to the writing act itself. Magnusson (2014) has pointed out the relationship of this practice to the etymology of "program": "as the Greek root, *prográphein*, signifies the activity of public writing" . One may add that this sense of the "programming" activity is still found in modern Greek in the phrase "programmatic statements", which means "policy statements", made publicly in political campaigns.

A further question concerns the role perceptual and cognitive processing limits in live coding and human-machine interaction. We used as a typical time frame 0.5 seconds (tempo = 120bpm), and by accellerating beyond that, limits of music perception [17] could be felt. Experimenting using such an apparatus for live music-making relies on subconscious processes. Whether or not this can be used as an expressive way to live coding is a question still open to further research. But we believe that by "designing constrains" [18] using grammatical interfaces for musical expression is a promising field for experimentations as it is a new area of musical practices based on computation. In our setup we could also observe the computational algorithms juxtaposed to the simulation of signal processes that belong to the domain of Calculus.

Into this scope "constraints are seen as compositional rules" [18]. Whether or not this is for real-time or non real-time usage is a matter of the composer/performer. Interesting applications could be developed also on the microscopic level. This may lead to states of training where programming will become as effortless as swimming in a "pool of code" . Perhaps this is at least one part of the essence of live coding and interactive programming. The deduction of the cognitive effort [19] plus a journey in minimal expressions. A typical duration for a live coding performace is ten to twenty minutes. Code expressions must be elegant and short, in order to be coherent and easy to debug. Whether or not it is feasible to write programs unconsciously is a topic for research relevant to human-computer interaction design, but also addresses broader philosophical concerns regarding embodiment.

Acknowledgments

We would like to thank family and friends.

8. REFERENCES

[1] G. Diapoulis and I. Zannos, "A minimal interface for live hardware coding," in *Live Interfaces*, ICSRiM, U. of Leeds, 2012.

[2] A. Blackwell and N. Collins, "The programming language as a musical instrument," in *Proceedings of PPIG05 (Psychology of Programming Interest Group)*, 2005.

[3] T. Magnusson, "Algorithms as scores: Coding live music," *Leonardo Music Journal*, vol. 21, pp. 19–23, 2011.

[4] D. Stowell and A. McLean, *Live music-making: a rich open task requires a rich open interface*. Springer, 2013.

[5] N. Collins, "Trading faures: Virtual musicians and machine ethics," *Leonardo Music Journal*, vol. 21, no. 3, pp. 35–39, 2011.

[6] D. Grune and C. Jacobs, *Parsing techniques: A practical guide*. Springer, 2007.

[7] E. Miranda, *Composing music with computers*. Focal Press, 2001.

[8] U. of Crete. (2011, September) Cs-120: Digital design. [Online]. Available: http://www.csd.uoc.gr/~hy120/11f/engl_cont.html

[9] D. Cottle, S. Wilson, and N. Collins, *The SuperCollider Book*. MIT Press, 2012.

[10] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, vol. 8, no. 3, pp. 321–330, 2003.

[11] T. Bovermann and D. Griffiths, "Computation as material in live coding," *Computer music journal*, vol. 38, no. 1, pp. 40–53, 2014.

[12] M. Sipser, *Introduction to the Theory of Computation*. Course Technology, 1996.

[13] J. McDermott, T. Gifford, A. Bouwer, and M. Wagy, *Should Music Interaction Be Easy?* Springer, 2013.

[14] T. Magnusson, "Herding cats: Observing live coding in the wild," *Computer Music Journal*, vol. 38, no. 1, pp. 8–16, 2014.

[15] N. Collins, "Live coding of consequence," *Leonardo*, vol. 44, no. 3, pp. 207–211, 2011.

[16] R. Wieser and J. Rohrhuber, "Personal accounts on the history of live coding," in *Collaboration and learning through live coding*, Dagstuhl Seminar 13382, 2014.

[17] S. Koelsch and W. A. Siebel, "Towards a neural basis of music perception," *Trends in Cognitive Sciences*, vol. 9, no. 12, pp. 578–584, 2005.

[18] T. Magnusson, "Designing constraints: Composing and performing with digital musical systems," *Computer Music Journal*, vol. 34, no. 4, pp. 62–73, 2010.

[19] A. McLean, "Stress and cognitive load," in *Collaboration and learning through live coding*, Dagstuhl Seminar 13382, 2014.