



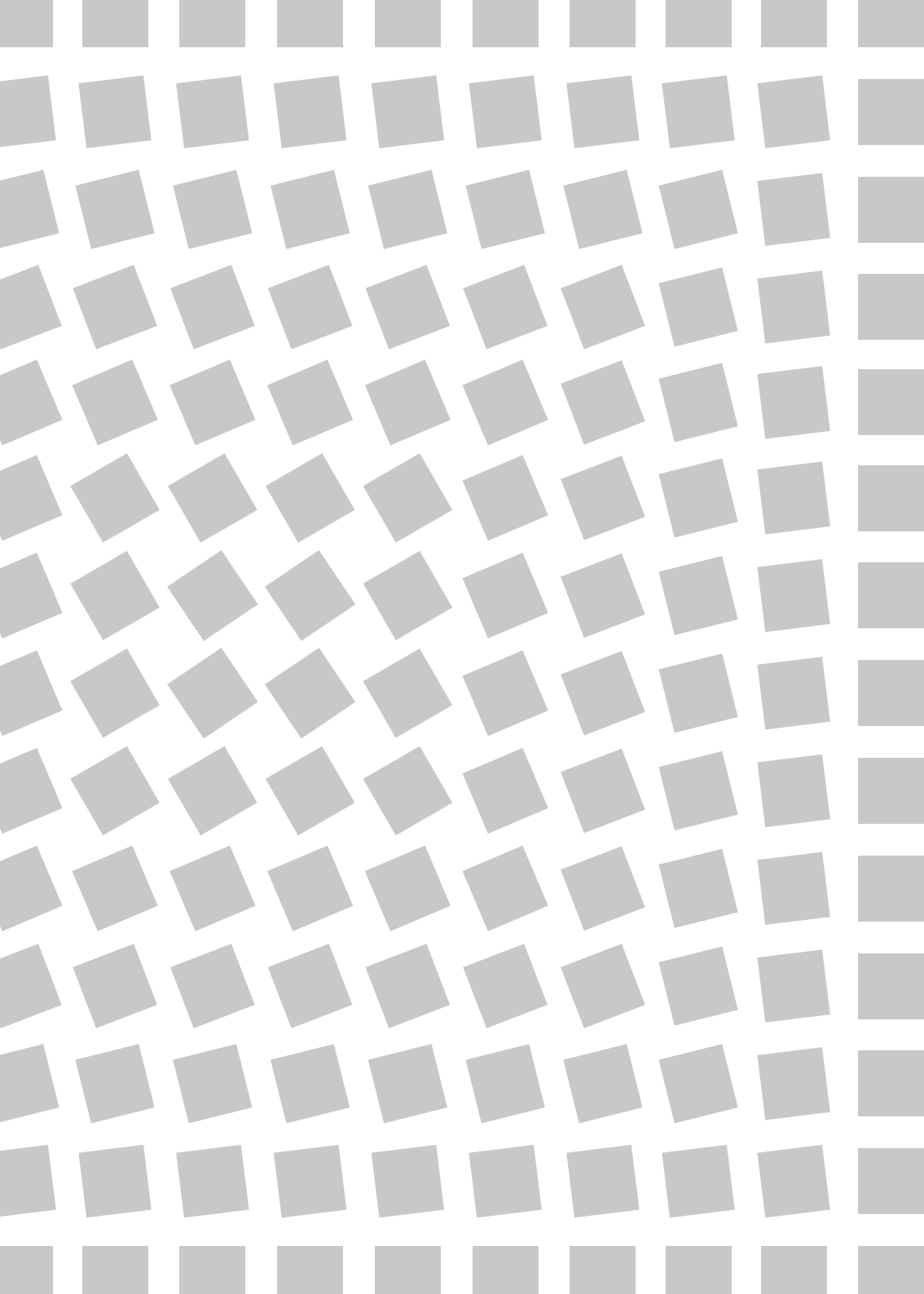
**Luiz Velho
Vitor Rolla**

Proceedings
of the

live =>
**coding
music;**

seminar

**Rio de Janeiro
IMPA
2019**



Proceedings
of the

live =>
coding
music;

seminar

IMPA
Instituto de Matemática Pura e Aplicada

Director: Marcelo Viana

Vice Director: Claudio Landim

VISGRAF
Vision and Computer Graphics Laboratory

Leading Scientist: Luiz Velho

Department of Scientific Events

Coordinator: Suely Torres de Melo dos Santos Lima

Live Coding Music Seminar

Organizers: Vitor Rolla <vitorgr@impa.br>

Luiz Velho <lvelho@impa.br>

Luiz Henrique de Figueiredo <lhf@impa.br>

Marcelo Cicconet <marcelo_cicconet@hms.harvard.edu>

Proceeding editors: Luiz Velho <lvelho@impa.br>

Vitor Rolla <vitorgr@impa.br>

Sponsors



**Luiz Velho
Vitor Rolla**

Proceedings
of the

**live =>
coding
music;**

seminar

**Rio de Janeiro
IMPA
2019**

Copyright © 2019 by IMPA

Ficha catalográfica

V436l Velho, Luiz
Proceedings of the Live Coding Music Seminar / Luiz Velho;
Vitor Rolla, editores. - 1. ed. -- Rio de Janeiro: IMPA, 2019.
38 p.
ISBN 978-85-244-0466-5
1. Computer Music 2. Algorithmic Composition 3. Live Coding
CDD: 519 CDU: 519.6

Second Edition
Printed in Brazil

Cover and graphic design: Júlia Rabetti Giannella

Distribution

Instituto Nacional de Matemática Pura e Aplicada (IMPA)
Estrada Dona Castorina, 110
22460-320 Rio de Janeiro, RJ Brasil
<http://www.impa.br>

Foreword

This booklet contains the proceedings of the **1st Live Coding Music seminar**, which was organized and sponsored by Instituto Nacional de Matemática Pura e Aplicada (IMPA), in cooperation with Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

We selected six papers for plenary presentation and live coding performance. We have also chosen three short-papers for oral presentation and live coding performance.

The manuscripts present in this booklet are a starting point to comprehend the programming language, the main functions involved in the composition, the pre-loaded code of the live coding performance, and the musical techniques used to compose the musical pieces presented in our seminar. Essentially, each manuscript describes a piece of music composed algorithmically.

Vitor Rolla

Table of content

SECTION I | PLENARY PRESENTATIONS

- page 12 **Cross-categorized-seeds**
Iván Paz
- page 16 **Ode's ode**
Pablo Riera
- page 20 **Hyperphase**
Avneesh Sarwate
- page 22 **0 to 1 : An Infinite Space?**
Shawn Lawson; Ryan Ross Smith
- page 25 **yi': 'callejón de luz' (alley of light)**
Rodrigo Velasco
- page 29 **CYOF**
Shelly Knotts

SECTION II | SHORT-PAPER PRESENTATIONS

- page 33 **Ghost**
Santiago Bosch
- page 35 **“On/Off”**
Martim Galvão
- page 37 **Crisis: Electroacoustic Improvisation**
Molly Jones; Derek Worthington

A decorative pattern of gray squares arranged in a grid, slightly offset from each other, covering the top portion of the page.

SECTION I

PLENARY PRESENTATIONS

A decorative pattern of gray squares arranged in a grid, slightly offset from each other, covering the bottom portion of the page.

cross-categorized-seeds

Iván Paz¹

¹Soft Computing Research Group – BarcelonaTech
Computer Science Department
C/ Jordi Girona 1-3. Edifici Omega
Barcelona, 08034

ivanpaz@cs.upc.edu

***Abstract.** Live coding builds and conducts the sound by real-time intervention of parametric devices (such as synthesizers). Coding a piece on-the-fly requires to bridge the cognitive gap associated with devices' huge parameter spaces and the possible nonlinear sound variation built-in within them. A possible approach is to have some pre-selected parameter combinations, of which the aural result is known, as a starting point for the performance. However, collecting/memorizing many combinations is time consuming and using only a few can be perceived monotonous. cross-categorized-seeds uses an inductive rule learning algorithm to extend a set of categorized combinations (or seeds) offering new material perceived consistent with the categories used. In this way, the selected seeds draw the contour of the piece and the extracted rules fill the different parts. The tensions and relaxations that arise from the recombinations of the material shape the inner dynamics of the sections.*

1. Background

Live coding uses parametric devices to create music and sound [Brown and Sorensen 2009, Rohrhuber and De Campo 2009]. The parameters are the channels for real-time interaction [Rohrhuber and De Campo 2009]. By tweaking them, the coder explores, categorizes, and selects the appropriate combinations of parameter values for different musical contexts. Thus, a piece can be seen as the succession of combinations that creates its spatial and temporal structures.

Coding a piece *on-the-fly* requires guiding the sound by changing the parameter settings. This imposes cognitive challenges due to the huge size of devices' parameter spaces, and to the possibility of non-linear sound variation built-in within them [Dahlstedt 2009]. Therefore, it is normal to have some pre-selected parameter combinations, of which the aural result is known, as a starting point from which to explore during the performance. However, when only a few combinations are used, the performance can quickly become repetitive and boring for the listener, and again, remembering many combinations imposes practical challenges. Generative algorithms [McLean and Dean 2018] have been used to address this problem by automating the production of low-level material, while the coder works on guiding the high-level evolution of the piece.

cross-categorized-seeds is a musical piece that uses “RuLer” [Paz et al. 2019], a rule learning algorithm to inductively generalize small seeds of pre-selected combinations. To create the seeds, a linguistic perceptual label that describes the characteristics of the sound (e.g. calm, harsh) or the musical context (e.g. intro, break) is assigned to each

combination. The algorithm produces a set of new unheard combinations intended to create variation, but that still remain consistent with their respective labels. The algorithm searches for patterns in the given seeds based on a dissimilarity function, and constructs new rules that generalize the input data. The production of rules crossovers the original material in a similar way as genetic algorithms¹. The user controls the generalization level of the rules through the dissimilarity function, and by listening and evaluating the material *on-the-fly*. The form of the piece is created by calling the combinations in the corresponding moments using the perceptual labels. The tensions and relaxations created by the recombination levels of the original material can be used to shape the inner dynamic of the sections. Hence, cross-categorized-seeds is also a directed search, in a vast space of possibilities, used as a compositional tool.

2. Learning rules

2.1. Rule representation

The algorithm represents each categorized parameter combination as an array of size N . The first $N - 1$ entries contain the set of possible values of the corresponding parameter, and the last entry contains the category (or class) assigned to the combination. Each datum is also considered a single rule. For example, a rule $r = [\{3\}, \{5\}, \text{intro}]$ is a single rule. A rule $r = [\{1,2,3\}, \{7\}, \dots, \{3\}, \text{intro}]$ could be interpreted as $r[1] = 1 \text{ OR } 2 \text{ OR } 3$, $r[2] = 7, \dots$, $\text{label} = \text{intro}$.

2.2. Rule learning algorithm “RuLer”

The input of the algorithm is a set of labeled combinations. As mentioned, the algorithm iteratively searches for patterns among the rules. It iterates until no new rules are created. This is done in the following way:

1. Take the first rule of the rule set.
2. Compare the selected rule with the other rules using the dissimilarity function. If a pattern is found, i.e the dissimilarity between the two rules is less or equal that a threshold established by the coder, create a new rule using the *create_rule* function (See Sections 2.3 and 2.4).
3. Eliminate the redundant rules from the current set. A rule r_1 is redundant with a rule r_2 (of the same class) if $\forall i \in \{0, \dots, N-1\}$, $r_1[i]$ is a subset of $r_2[i]$. Note that this eliminates those rules from which the new rules were created.
4. Add the created rules at the end of the rule set.

2.3. dissimilarity function

The *dissimilarity* function receives two rules (r_1, r_2) together with a threshold $d \in \mathbb{N}$ and returns True in case the rules have the same category and $\text{dissimilarity}(r_1, r_2) \leq d$. It returns False otherwise.

The *dissimilarity* function used in cross-categorized-seeds counts the number of empty intersections between the corresponding entries in the rules. For example, if $r_1 = [\{1\}, \{3,5\}, \text{intro}]$ and $r_2 = [\{1,3\}, \{7,11\}, \text{intro}]$, $\text{dissimilarity}(r_1, r_2) = 1$.

¹Other ways of creating material are also possible, see sections 2.4 and 2.5.

2.4. *create_rule* function

This function verifies that no contradictions (i.e. rules with the same parameter values but different label) are created during the generalization process. It also allows the user to control the generalization level by defining a percentage of the cases contained in any created rule that should be present in the original data. The function receives two rules and creates a new rule if both conditions are met. In the case of cross-categorized-seeds, if r_1 and r_2 are the input rules $r_{new}[i] = r_1[i] \cup r_2[i]$ if $i < N$ and $r_{new}[N] = \text{class of } r_1$.

2.5. Domain specific functions

Note that the *dissimilarity* and *create_rule* functions can be changed according to the needs of the objects being compared and the desired generalization procedure. For example, in [Paz et al. 2018] a previous version of the algorithm used the Hamming distance. Selecting the functions is useful since, according to the objects being compared, different dissimilarities (or similarities) can be used (e.g. for melodic analyze the spectral content, and for rhythmic the temporal dimension). See for example [Toussaint 2004] for a comparison of rhythmic similarity measures.

3. on-the-fly

To create the piece the contour is drawn by selecting and labeling some parameter combinations for its sections. These may have subsections for which the same procedure is applied. For example, 3 + 4 + 4 would be the structure of a piece with three sections composed respectively by 3, 4 and 4 subsections.

The piece is played by calling the rules that result from applying the algorithm to the selected material. If a section is composed by subsections intro, main, break, and end, the rules for these labels are used in turn. To develop each subsection the recombination levels (controlled by the threshold d) of the original material are used. They also provide the repetition/novelty balance. The automation of the low-level settings allows the coder to focus on the high-level development of the piece. Thus, cross-categorized-seeds explores the possibilities of playing using perceptual categories, creating variations that seem to drive us out of the path but that always allow us to return (if we want).

References

- [Brown and Sorensen 2009] Brown, A. R. and Sorensen, A. (2009). Interacting with generative music through live coding. *Contemporary Music Review*, 28(1):17–29.
- [Dahlstedt 2009] Dahlstedt, P. (2009). Thoughts on creative evolution: A meta-generative approach to composition. *Contemporary Music Review*, 28(1):43–55.
- [McLean and Dean 2018] McLean, A. and Dean, R. T. (2018). *The Oxford Handbook of Algorithmic Music*. Oxford University Press.
- [Paz et al. 2018] Paz, I., Nebot, À., Mugica, F., and Romero, E. (2018). Modeling perceptual categories of parametric musical systems. *Pattern Recognition Letters*, 105:217–225.
- [Paz et al. 2019] Paz, I., Nebot, À., Mugica, F., and Romero, E. (2019). Generalizing successful parameter combinations: directed sound design for algorithmic music. *Submitted for revision*.

- [Rohrhuber and De Campo 2009] Rohrhuber, J. and De Campo, A. (2009). Improvising formalisation: Conversational programming and live coding. *New Computational Paradigms for Computer Music*. Delatour France/Ircam-Centre Pompidou.
- [Toussaint 2004] Toussaint, G. T. (2004). A comparison of rhythmic similarity measures. In *ISMIR*.

Ode's Ode

Pablo Riera¹

¹Laboratorio de Inteligencia Artificial Aplicada,
Instituto de Ciencias de la Computación,
Universidad de Buenos Aires, CONICET,
Argentina

pablo.riera@gmail.com

Abstract. *Dynamical systems, either continuous or discrete, are known to exhibit a plethora of behaviors such as periodicity, recurrent structures, bifurcations, chaos, stochasticity, stability, coexistence phenomena and many more. The music piece here presented is an open piece aimed to intensify those behaviors by generating and controlling sound with the flows of ordinary differential equations (ODEs) and maps (discrete-time ordinary difference equations). This piece is executed in a live coding fashion by writing the vector fields and their parameters in a data serialization language and connecting the inputs and outputs of different dynamical systems. Thus, allowing the generation of smooth transitions or abrupt changes (through bifurcations) to build evolving musical textures or complex gestures.*

1. Introduction

This paper describes the use of ordinary differential equations and maps for real-time sound synthesis and control in a live coding musical performance (*Live ODEing*). It presents *LODE*, a system specifically designed to allow users to write, compile on the fly, listen and visualize the numerical solutions of ordinary differential equations and maps while changing their parameters and definitions in real time. It also permits connecting different equations by controlling the parameters from one ODE through the dynamical variables of another, simplifying a process otherwise hard to implement by modern sound synthesis software such as *Supercollider* or *Pure Data*¹ which require extension plugins or external objects to modify or add new equations.

1.1. Related works

The use of ODEs for music and sound synthesis began around 1990. Xavier Rodet and colleagues made use of the dynamical systems formalism for real instrument modeling and musical purposes [Rodet 1993][Rodet and Vergez 1999]. Most recently, in 2010, new approaches appeared. Stefanski describes a full system for generating sound with ODEs similar to the one proposed here [Stefanakis et al. 2015] and Medine lists some of the benefits of generating sound by solving ODEs compared to other methods that do not rely on this formalism [Medine 2016]. ODEs also appear in works that explore complex systems sonification, for example, using spiking neural networks for control [Kerlleñevich et al. 2011] and synthesis [Miranda and Matthias 2009]. The present effort tries to further explore the use of dynamical systems for musical purposes in the context of live coding practice. Section 2 describes the musical aspects and section 3 the details of the system. The current alpha version is available at <https://github.com/pabloriera/lode>.

¹and also hardware modular synthesizers [Slater 1998]

2. Live ODEing music

A core aspect of the live coding practice is the continuous two-way communication between the coder and the code as a suitable scenario for improvisation, experimentation and trial and error procedures to occur [Collins et al. 2003]. In that sense, including mathematical methods with possible unknown behaviors like nonlinear ODEs seems natural.

However, using dynamical systems for sound and control has some inherent difficulties, for example, system sensibility to parameters due to intricate configurational spaces and instability. An approach for dealing with these problems consists in choosing an appropriate scale when changing parameters and departing from a known stable equation by carefully adding and turning new terms. Controlling the equation and the parameters in this manner may end in continuous changes in sound but also in abrupt transitions through bifurcations. Considering that, the performance tries to explore the behaviors and sororities of different types of ODEs in a more or less incremental manner.

2.1. Playing with ODEs

This section describes common aspects of dynamical systems that may serve for musical purposes. The dynamic variables of the ODEs can be used to generate sound or to control the parameters of other ODEs. For example, the possibility to modify the time scale allows to decide, among other things, if the oscillations will be audible or used for control. One of the most salient phenomena of dynamical systems is chaos. It is common to use systems that exhibit chaotic trajectories for both synthesis and control. In the case of synthesis, the resulting sounds usually combine a tonal and a stochastic characteristic. In the case of control, the chaotic trajectories can serve to control the parameters of other ODEs and to generate variations with some degree of unpredictability. Adding noise terms (white noise or filtered noise) in the ODEs can have different types of behaviors. In some cases, it can end in a noisy output. However, in others, the noise can be used to disturb a parameter and trigger bifurcations. Some DSs, particularly excitable ones, can generate discrete events (such as neural action potentials) that can be used to generate rhythmic sequences. A compelling case is bursting, where complex patterns of events may occur. It is also possible to generate complex patterns with small networks of neurons [Kerlleñevich et al. 2011]. Adding external forces is a classical way to perturb a dynamical system to generate new behaviors such as resonance, synchronization, chaos and other complex solutions. Musically speaking, this is very useful because it allows adding a modulation with a controlled frequency. Systems with delay are also a great source of complex behaviors. Many acoustic systems use delays in their equations relating it to the periodicity of the sound.

3. LODE system description

Defining sound generators and making connections between them is a widespread practice in modern sound synthesis frameworks like Supercollider. The LODE system was designed following the same ideas but keeping simplicity in mind.

The following example describes a system of an ODE and a map. The ODE is based on the normal form of the supercritical Hopf bifurcation (eq. 1) which depending on parameters has a stable limit orbit with sinusoidal trajectories and the map is the logistic equation (eq. 2) which has periodic or chaotic solutions depending on the parameter r value.

$$\begin{aligned}\dot{x} &= 2\pi\omega y + \epsilon x - x(x^2 + y^2) + i \\ \dot{y} &= -2\pi\omega x + \epsilon y - y(x^2 + y^2) + i\end{aligned}\tag{1}$$

$$z_{n+1} = rz_n(1 - z_n)\tag{2}$$

Where ω is the frequency, ϵ is related to the amplitude of the limit cycle and i is an external input. In the current version the code interface uses *YAML* syntax which for this system looks like:

```
hopf:
  equation:
    x: 2*pi*y*w + e*x-x*(x*x+y*y)+i
    y: -2*pi*x*w + e*y-y*(x*x+y*y)+i
  parameters:
    w: 300 + logistic.x*100
    e: 10.0
    i: sinosc(2.0)*100 + noise*2 - hopf.x[-0.005]*4.0
  output:
    x: {gain: 0.5, pan: -1}
    y: {gain: 0.5, pan: 1}

logistic:
  discrete: true
  equation:
    x: r*x*(1-x)
  init:
    x: 0.3
  parameters:
    r: 3.9
    hz: 4.0
```

The main key of the definition is the name of the unit, *hopf* and *logistic* in this case. The next keys describe the system equation, initial conditions (*init*), and parameters. In the case of a map, the discrete flag must set to *true* and the *hz* parameter is used to set the update frequency. The *output* key allows setting which dynamic variables will sound.

As an example, the Hopf ODE has an input parameter i that shows how to add an external periodic forcing *sinosc*, which accepts a frequency argument, noise which is white noise, and delayed connection, in this case with itself (*hopf.x[-0.005]*), where *hopf.x* indicates the dynamical variable x of the Hopf ODE and the negative time in braces indicates the delay in seconds. The w parameter has also an external influence, in this case from the x variable of the logistic map.

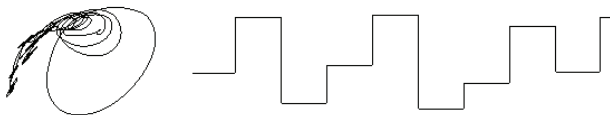


Fig. 1. Visualization of the system . Left phase plane of the Hopf oscillator, right the logistic series time series

3.1. ODE sound synthesis

The system uses *Supercollider* as the sound synthesis engine back end with the addition of a new unit generator extension called *Oderk4*, specially developed to load ODE equations and integrate them with the Runge-Kuta 4 method. The programming language used for the parsing and synths definition management is *Python*. After writing an ODE definition, the program translates the equation definition to C++ and compiles an external library. Next, an *Oderk4* unit generator builds up a node in the *Supercollider* server to receive the name of the external library and load it.

References

- [Collins et al. 2003] Collins, N., McLean, A., Rohrhuber, J., and Ward, A. (2003). Live coding in laptop performance. *Organised sound*, 8(3):321–330.
- [Kerlleñevich et al. 2011] Kerlleñevich, H., Riera, P. E., and Eguia, M. C. (2011). Santiago—a real-time biological neural network environment for generative music creation. In *European Conference on the Applications of Evolutionary Computation*, pages 344–353. Springer.
- [Medine 2016] Medine, D. (2016). Dynamical systems for audio synthesis: Embracing nonlinearities and delay-free loops. *Applied Sciences*, 6(5):134.
- [Miranda and Matthias 2009] Miranda, E. R. and Matthias, J. (2009). Music neu rotechnology for sound synthesis: Sound synthesis with spiking neuronal networks. *Leonardo*, 42(5):439–442.
- [Rodet 1993] Rodet, X. (1993). Sound and music from chua’s circuit. In *Chua’s Circuit: A Paradigm For Chaos*, pages 434–446. World Scientific.
- [Rodet and Vergez 1999] Rodet, X. and Vergez, C. (1999). Nonlinear dynamics in physical models: Simple feedback-loop systems and properties. *Computer Music Journal*, 23(3):18–34.
- [Slater 1998] Slater, D. (1998). Chaotic sound synthesis. *Computer Music Journal*, 22(2):12–19.
- [Stefanakis et al. 2015] Stefanakis, N., Abel, M., and Bergner, A. (2015). Sound synthesis based on ordinary differential equations. *Computer Music Journal*, 39(3):46–58.

Hyperphase

Avneesh Sarwate¹

¹Georgia Tech Center for Music Technology
840 McMillan Street NW
Atlanta, GA 30318

avneeshsarwate@gmail.com

***Abstract.** This paper describes an audiovisual live coding piece based on the idea of "continuous time patterns." It is a musical improvisation where the graphics are fully derived from the musical information.*

1. Musical Motivation and Overview

The core of this piece is based on using frequency modulated low frequency oscillators (FMLFOs) to control the tempo of looping rhythms, causing various coordinated speed modulation and rhythmic phasing effects. The sequencer can support tempos of over 1000 bpm, effectively causing wave-table synthesis, at which point the FMLFOs act as pitch modulators. The piece also involves using MIDI sliders to manually control audio effects on each rhythm. This live-played parameter manipulation can be recorded, saved, looped, and transformed via Python. All live coding is done in a Jupyter notebook [Kluyver et al. 2016].

2. Rhythmic Sequencing

The definition of the rhythms used in the piece is done via a partial port of TidalCycles [McLean and Wiggins 2010] to Python and SuperCollider [McCartney 2002] (referred to from here on as Pydal). Figure 1 shows the code to play a Pydal pattern. Python is used to parse the TidalCycles style string into a "Pydal pattern," which is a stream of time stamped events that are scheduled using SuperCollider. A Pydal pattern is played on a particular "Pydal channel", which is instantiated with a particular key and a unique clock in SuperCollider (the channel clock is initially synced to a master clock). In this piece, the individual "events" are MIDI note messages used to trigger samples in Ableton Live. The speeds of individual channels/patterns can be changed via OSC messages [Wright et al. 1997]. The phase and/or tempo of the individual channels can be synchronized to each other or to the master clock, as shown in Figure 1. This master clock is shared between Pydal, the FMLFO and live looping components, described next.

```
ch1.play(read("bd [bd bd]"))  
sendMsgSC("/syncClocks", "both", 'master', 1, 2, 3)
```

Fig. 1. A Pydal pattern playing on channel 1, and a message to sync both the phase and tempo of channels 1, 2, and 3 to the master clock

3. Frequency Modulated LFOs

The FMLFOs are also implemented using Python and SuperCollider. Figure 2, cell A shows the syntax for defining a composition of sinusoids. When the resultant wave object

is played, the Python object generates the code for the equivalent SuperCollider function, and the wave is played in SuperCollider. Each FMLFO can be associated to the key of a Pydal channel and be used to modulate its tempo (Fig 2, Cell A).

4. MIDI CC Live Looping

SuperCollider is also used to implement the MIDI CC live looper used in this piece. A Novation Launchpad is used as an interface for recording and replaying MIDI CC curves played on the MIDI slider interface. The Python environment has access to these loops and can be used to arbitrarily transform them (Fig 2, Cell B). When loops are played, the MIDI CC data is forwarded to Ableton Live to control audio effects.

<pre>w1 = Tri(freq=0.4, phase=Sin(freq=0.2))*20 + 120 startWave(w1, 1) stopWave(1)</pre>	A	<pre>loop1 = getLoop(3) loop1a = reverseLoop(loop1) setLoop(loop1a, 4)</pre>	B
--	----------	--	----------

Fig. 2. Cell A: Python operator overloading allows for the expression of these waves in a syntax similar to mathematical notation. This wave is used to modulate channel 1. Cell B: An example of how to access, transform, and save a new loop.

5. Visual Mapping

The visual element of this piece reflects the real time state of the rhythms being played and the audio effects. The greyscale rows show the rhythms being played on each channel, and the rotation of the rows reflects their phase offsets against the master clock. Each audio effect also corresponds to a particular visual transformation, and the level of the transformation is directly mapped to the amount of the audio effect applied. The visuals are all implemented using Max-Jitter, and respond to real time MIDI and OSC messages.

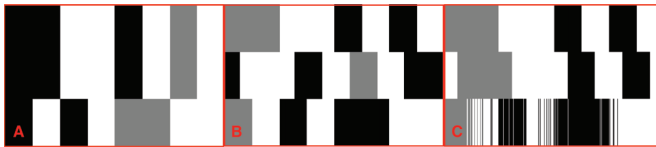


Fig. 3. A visualization of 3 patterns that are- A: all in phase, B: pattern 2 out of phase, C: pattern 2 out of phase and pattern 3 played with granulation

References

[Kluyver et al. 2016] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., et al. (2016). Jupyter notebooks—a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90.

[McCartney 2002] McCartney, J. (2002). Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68.

[McLean and Wiggins 2010] McLean, A. and Wiggins, G. (2010). Tidal—pattern language for the live coding of music. In *Proceedings of the 7th sound and music computing conference*.

[Wright et al. 1997] Wright, M., Freed, A., et al. (1997). Open soundcontrol: A new protocol for communicating with sound synthesizers. In *ICMC*.

0 to 1 : An Infinite Space?

Shawn Lawson¹, Ryan Ross Smith²

¹Rensselaer Polytechnic Institute
Department of Arts
110 8th Street
Troy, NY, USA, 12180

lawsos2@rpi.edu

²Monash University
Wellington Road
Clayton VIC 3800
Melbourne, Australia

ryanrosssmith@gmail.com

Abstract. *The authors discuss high-level technical and conceptual aspects of the audiovisual live-code performance, 0 to 1 (2019). Details include descriptions of our custom programming environment, the programming languages we use, and the audio & visual techniques we employ in our work.*

1. Introduction

This paper briefly describes the collaborative audiovisual work *0 to 1* (2019). We will highlight the conceptual ideas of the performance, the live-coding environment, including details regarding audio regarding visuals.

Conceptually we see *0 to 1* as a metaphor for an infinite set of numbers/possibilities, while also constrained by an inability to reach either extreme. In other words, what may appear as a binary distinction can be refined into a spectrum of increasingly smaller divisions, and we are attempting to discover the immense within the simplistic. How do we move through an infinite space, of sorts, or compute one of any possibility? These are questions that are directing us through this performance.

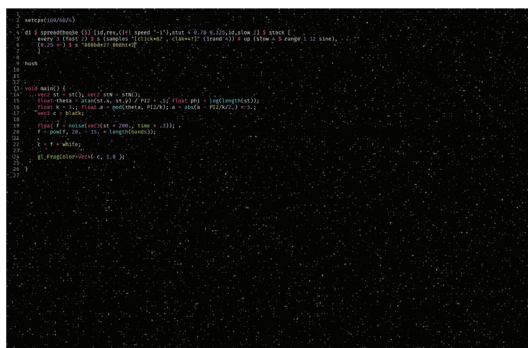


Fig 1: Near the start of the piece the imagery is dark as the performers build up the piece from near-scratch.

2. Programming Environment

The integrated development environment (IDE) for performing this work is *The Dark Side* [Lawson and Smith 2017]. This IDE affords us the ability to collaboratively live-code in multiple programming languages simultaneously in a single text buffer. Moreover, it permits each performer to have access to edit each other's code, providing a full range of congenial to confrontational behaviors.

3. Audio

The live-coded audio is programmed using Alex McLean's TidalCycles [Alex McLean], a pattern-based, functional language built on Haskell. Patterns are created through the sequential triggering of discrete audio samples, which can then be transformed into different patterns through a chain of function modifiers. Each pattern block is sent to (executed by) the Haskell interpreter, and the result is performed by Supercollider.

TidalCycles provides a comprehensive function library for the transformation and manipulation of samples and patterns. The language is extendable with custom functions if so desired, although nothing additional was added for *0 to 1*.

The sample library for this piece includes a collection of custom samples designed for this work, as well as samples pulled from earlier works by the performers. Samples were created using a variety of commercial software, hardware and modular synthesizers.

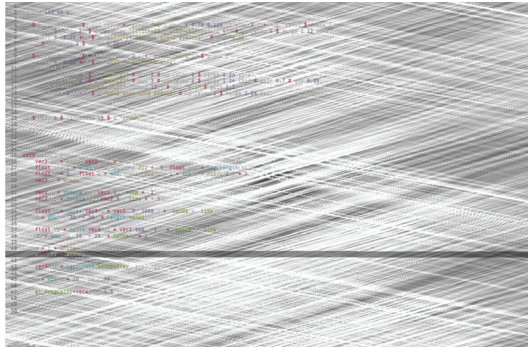


Fig. 2: Partway through the piece there are multiple layers of image interaction.

4. Visual

The live-coded visuals are programmed in the OpenGL Fragment Shader language. Within this scope, each pixel is processed in parallel, meaning that the entire image output can be programmed as a whole instead of each pixel iteratively. The OpenGL shader code is auto-compiled at regular intervals, such that updates are continually being displayed while typing.

Primitive shapes, noise generators, colors, and so on are not included in the OpenGL Shader language, but these functions and values are accessible through a custom, predefined header file attached to the shader just prior to compilation. In addition, audio data, system time, or any continuously-changing values are passed to the shader through OpenGL's uniform handle.

From a visual standpoint, *0 to 1* begins in black and slowly builds, see figure 1. The piece moves through the range of values between zero and one, similar to telling a story or narrative with high and low points, see figure 2. Finally, it concludes with white as it approaches the end, see figure 3.



Fig. 3: Nearing the end of the piece, color is introduced and frequently whiting out the imagery.

References

[Alex McLean] Alex McLean, e. a. Tidal.

[Lawson and Smith 2017] Lawson, S. and Smith, R. R. (2017). The dark side. In *Proceedings of the Third International Conference on Live Coding*.

yi' 'callejón de luz' (alley of light)

Rodrigo Velasco¹

¹Department of Design and Computation Arts
Concordia University
Montréal, Quebec, Canada

yectocsalev@gmail.com

***Abstract.** yi' is a collaborative process where difference finds a place. This paper shares conceptual and philosophical concerns, among which the influence of the 'minor gesture' stands out, resisting formation long enough to allow us to see the potential of worlds in the making. Besides, it delves into the description of algorithmic musical ideas and patterns live coded through TidalCycles. yi' moves these ideas from the analog to the algorithmic realm with an interest in not to recreate, but to find new openings, giving shape to unexpected encounters through rhythms, sonorities and resonances in movement.*

1. Introduction

yi' is a making-thinking process that involves a constant evolving audiovisual exploration through live coding and collaborative practices. A conversational approach allows new modes of encounter through emergent and rhythmic shades of experience. The project began in 2017 with a particular focus on exploring chord progressions influenced by musical ideas inspired by a wide range of genres (e.g. bolero, bossa nova, jazz and salsa).

Sound and visuals are fields of relation where textures and rhythms allow the existence of navigable structures, emergent alleys where to improvise, opening to flux pre-existing algorithmic ideas between polyrhythms, chord progressions and audio-reactive 3D shapes that have an organic resemblance and inspiration. yi' has been written mainly in Haskell through TidalCycles, however as part of a conversational and spontaneous process between different ideas and software (e.g. SuperCollider, Processing, Reaper, VCV Rack and Fluxus). In May 2018, I had the opportunity to present an early exploration performing for 'Noche en Blanco : Latinx Remix' organized by Never Apart and Eastern Bloc as part of 'Nuit Blanche' in Montréal.

However, yi' has always been linked to an interest in exploring this processes through live coding performances and printed matters, including the next presentation as part of 'Live Code Experience' at the National Institute for Pure and Applied Mathematics (IMPA) in Rio de Janeiro, which is an incredible opportunity to share this experience. yi' also involves the publication in physical formats through digital prints and an audio cassette that has been being recorded.

2. Table and Equation example

Regarding algorithmic music for yi', it is important to mention the influence that non-linear processes and difference as conceptual and creative potentials have brought to

find new ways of making-thinking towards interstitial, collaborative and relational experiences. Since 2014 I started using TidalCycles, an open source software created by Alex McLean to describe sound patterns through functional programming; thanks to the knowledge that Alex McLean and Ernesto Romero shared with me at the beginning of my incursions in live coding, I began to rewrite ideas that were born in my guitar through SuperCollider and TidalCycles. The approach that yi' shares with Tidal lies in the capacity of vary complex processes through simple algorithmic transformations, therefore and due to my interest in traditional percussion, especially in Afro-Latin rhythms I have found in TidalCycles and in its potential to implement Euclidean algorithms to the organization of sound in time a really great opportunity.

Delving into Euclidean rhythms, namely computing the greatest common divisor of two given integers [Toussaint 2005].

This pattern is playing two different patterns together through a 'stack' function, the first one is 3,8 = '[x . . x . . x .]', where 'x' represents a sound and '.' a silence and it generates a Cuban tresillo, the second one is 9,16 = '[x . x x x . x . x x x . x .]' a cow-bell pattern in the Brazilian samba. In this sound exploration djembe samples are distributed through the function '# n (run 17)' that chooses one different sample for each sound of the pattern. Besides, diverse functions are spontaneously or at specific moments iterating, reversing or adding silences to the process, for example, 'within (0, 0.5) (density 2)' increases the density adding the feel of a drumroll in the first half of the cycle. The functions are fundamental to open the pattern to unexpected experiences, in this pattern the functions are also used within the effects to randomly control the gain, time and feedback of a delay, in addition to panning the sounds through a sine wave that slowly moves in a stereo sound system from one speaker to the other.

As for the chord progressions, yi' has a strong concern to establish a fruitful dialogue between analog and algorithmic processes, reinterpreting and writing musical ideas that emerged through an acoustic guitar into algorithmic music, however when I started to write chords through algorithms, TidalCycles did not have a function for this specificity, therefore, I began to translate the chords through a 'stack' function, including each note of the chord and sending the pattern to a synthesizer (SynthDef) in SuperCollider, later, thanks to new implementations of the TidalCycles user community, we are able to use different strategies to live code chord progressions.

In the previous pattern we can appreciate the use of the slow function that allows adapting the main tempo of Tidal, reducing its speed, giving space to a subtle progression of chords that evolve through a Euclidean rhythm, however the structure '< 5 9 >, 16', results in the combination of the rhythm 5,16 = [x . . x . . x . . x . . .] the Bossa-No-va necklace rhythm that is played during the first half and the cycle and the necklace samba rhythm 9,16 = ' [x . x x x . x . x x x . x .]' played during the second half, this TidalCycles function creates a new pattern: [x . . x . . x . . x x . x . x .].

When moving an idea from the analog to the algorithmic realm, my interest aims not to recreate, but to find new ways in which the idea unfolds through the process giving shape to unexpected encounters with rhythms, sonorities and resonances. However, playing the guitar is a process that involves the body and what is transmitted is a sensitive experience. Algorithmic music is written on a keyboard, its relationship with the body is different, but it is further linked to thought. Fostering this dialogue comes from the need

to find in this balance, a space in between, an alley of light.

Following this exploration, and very inspired by Latin music, I wrote an algorithm reinterpreting the 'tumbao' rhythm through TidalCycles, however, my knowledge in music and mathematics is mostly sensible and empirical, so the process included the interpretation of some musical scores but giving greater importance to careful listening and feeling, after a long journey I arrived at the following pattern.

3. alley of light

yi' involves different approaches that share an interest in exploring computational three-dimensionality through analog, digital and algorithmic processes. In the visual part, yi' has involved the exploration of two different approaches, both of which have been written through 's2hs2', a collaborative project that I had the pleasure of creating with Alex McLean, an OSC interface between TidalCycles and Processing, which is focused on visual poetry and electronic literature through live coding, however, this exploration focus in the potential of 's2hs2' for 3D live coded graphics that allows the emergence of vibrant and not predefined shapes unfolding through time and sound. yi' has a great conceptual influence by the 'minor gesture', which operates in different layers, in a dialogue between making and thinking, where the force of the form remains emergent. [Manning 2016].

The conceptual origin of yi', goes back to the idea of the 'callejón de luz' (alley of light), and consists of contemplating the alley as an in-between space, a place that allows the discovery of the otherness of our experience, bringing with it the possibility of encountering other forms and routes, approaching a raw vision where what is not accepted on the surface, such as graffiti (tag and bombing) or living in the streets, exists; we also find places stopped in time. Sometimes at night, among the sounds of birds the city is transformed, the calm is breathed and there is light at the end of the path. The alleys are places where difference finds a place, and that's where yi' appears, a word that means light in Zapotec and that is also a way of honoring my ancestors, especially my grandfather 'Ignacio Ramírez', a guitarist who introduced me to arts by sharing me music knowledge in my young age and my great-grandmother 'Consuelo Paz'. Both were born in 'Yahuiche' located in 'Ixtlán de Juárez', in the 'Sierra Norte', at 60 kilometers from the city of 'Oaxaca'. yi' is a way of thanking and celebrating an important part of my roots, a light that I share with my ancestors and an opportunity to find and explore different paths through it.

Later on, as the processes of yi' were unfolded through diverse reflections. I found in the work of Patrick Gauvin a good friend and admired artist, a possibility to explore all that inspiration that comes from the alley of light, specifically through his work with 3D photogrammetry. Before we had done a couple of explorations to establish dialogues between live coding and photogrammetry, so after talking about yi', we both thought it was an incredible opportunity to go to the street and take photogrammetries that could later be part of yi'. This process was extremely important since it opened the project to transversality and collaboration, allowing the crossing with other ideas and processes. In addition, interestingly, we both agree on the relevance that psychogeography and situationalism arouse in our work and decided to take a drift through alley-ways in Montréal without any specific direction. Patrick took some photogrammetries of specific fragments of walls or traces of snow on the ground, most of the shots occurred in an alley

between Clark St. and St.Laurent, at Av. des Pins, It is an interesting site since the alley is completely full of graffiti and the atmosphere is the reflection of that otherness. The shot of the photogrammetry was difficult, the wind did not help us, but at the same time it generated quite a lot of glitches in the objects, which from our perspective is something fortunate, since it reveals that movement, the deformity and complexity of the situation.

Curiously and unexpectedly the project had to stay in a state of pause for about 6 months since we both had to do other projects, however, during this time I focused on the sound part of yi' doing a couple of live coding performances. After a while Patrick sent me the files and this brought great surprises. It is important to mention that the only way I have to visualize 3D files is through Processing ('s2hs2'), this makes the process long to arrive at the discovery of the photogrammetries, writing code for about an hour and it is up to the moment in which I evaluate the last piece of code that I could expose myself to that discovery of a fragment of reality captured half a year ago, and definitely finding memories and spaces that have opened new possibilities for yi'.

References

- [Manning 2016] Manning, E. (2016). *The Minor Gesture*. Thought in the Act. Duke University Press.
- [Toussaint 2005] Toussaint, G. (2005). The euclidean algorithm generates traditional musical rhythms. In *In Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 47–56.

CYOF

Shelly Knotts¹

¹Music Department
Durham University
Durham, UK

knotts.shelly@gmail.com

***Abstract.** CYOF is an algorithmic performance system that uses analysis of the performer's previous and current live coding performances as a basis to infer improvisational novelty, and predict potential futures of the live performance in realtime. Musical Information Retrieval (MIR) and text analysis techniques are used to analyse an archive of code and audio files from the performer's previous performances. The piece was developed as exploration of originality in live coded improvisation, and aims to give performer and audience visual feedback on the performer's innovation (in comparison to previous performances).*

1. Introduction

In CYOF, a live coding performance (using SuperCollider's JITLib [Wilson et al. 2011] to live programme sound synthesis) is augmented by a visualisation which shows a representation of the past, present and potential future of the current performance. The system aims to gently encourage the performer into more innovative improvisation by using archive material to determine the originality of the current performance in relation to the performer's own history. Realtime audio feature data (e.g. chromagram, loudness, MFCCs) relating to the past, present and potential future is mapped to greyscale blocks, with features in the y axis and time in the x axis. The performer's current code is projected as it is being typed alongside the most likely (in orange) and least likely (in blue) possible future code.

The visualisation aims to inform the performer of the similarity of their current performance to past performances and was developed as part of a research project exploring computational methods to aid improvisers in advancing their skill and flexibility [Knotts 2015]. The author's past and ongoing work includes algorithmic performance systems, and visualisations which show augmented information relating to live coding performances.

1.1. Visualising Live Coding

CYOF is influenced by several live coding systems which explore visual representation. Scheme Bricks [McLean and Wiggins 2010] and Gibber [Roberts et al. 2014], use visual cues to augment the live coding interfaces, relaying information related to live coding activity to the audience.

1.2. Prior Work

The piece builds on research undertaken by the author in algorithmic and live coded performance between 2012 and the present. The author's previous work embraces experimental approaches to live coded sound synthesis in SuperCollider, considering aspects of

human interaction with algorithmic systems including aspects of error, failure and temporality.

Previous performance systems used: visualisations to display augmented information relating to live coding performances; data, including live MIR, to inform and influence performances; and computational methods to aid performers in long term development of improvisational skill and flexibility.

2. Description

CYOF uses a large archive of the performer's past performances to build a data set of likely code combinations, audio feature combinations and trajectories in the performer's live coding sets. The live performance is analysed in realtime in 10 second windows using SuperCollider's SCMIRLive Library [Wilson et al. 2011]. The live data is compared to past performances, and the visualization shows data relating to the most closely related past performance alongside the data relating to the current performance. The live performance data is colour coded on a scale of blue (most original) to orange (least original).

In the visualization audio feature data is mapped to greyscale blocks with features shown in the y axis and time in the x axis. Time is visualised in 10 second blocks. At the

beginning of a performance a random past performance is chosen as the prediction data to be visualised. At each 10 second interval the prediction data for the remainder of the performance is replaced by data relating to the performance in the archive with MIR data most closely related to the current performance, showing the most likely future data for the current performance given the performer's history. This 'prediction' data is replaced with data from the current performance as the performance progresses.

Each block of the data relating to the current performance is colour coded on a scale of blue (most original) to orange (least original), showing the performer how close to data from past performances the previous segments were.

Another aspect of the visualisation shows the performer's current code as it is being typed alongside the most likely (in orange) and least likely (in blue) possible future code according to the database of past coding performances.

CYOF aims to gently encourage the performer into more innovative improvisation by using archive material to determine the novelty of the current performance in relation to the performer's own archive. In a performance of CYOF the performer live codes audio synthesis using SuperCollider using the visualisation as a stimulus to experiment with new ideas should it suggest too high a degree of self-similarity. Though the visualisation itself does not depend on a particular audio programming language for performance it has not been developed sufficiently to enable easy intergration with languages other than SuperCollider.

Video documentation of the first performance of CYOF at the International Conference on Live Coding at CMMAS, Morelia, MX can be found at: <https://vimeo.com/264561088>.

References

- [Knotts 2015] Knotts, S. (2015). Changing music's constitution: Network music and radical democratization. *Leonardo Music Journal*, 25:47–52.
- [McLean and Wiggins 2010] McLean, A. and Wiggins, G. (2010). Bricolage Programming in the Creative Arts. In *22nd Psychology of Programming Interest Group 2010*.
- [Perkins et al. 2014] Perkins, T. J., Foxall, E., Glass, L., and Edwards, R. (2014). A scaling law for random walks on networks. *Nat Commun*, 5.
- [Roberts et al. 2014] Roberts, C., Wright, M., Kuchera-Morin, J., and Höllerer, T. (2014). Gibber: Abstractions for creative multimedia programming. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 67–76, New York, NY, USA. ACM.
- [Wilson et al. 2011] Wilson, S., Cottle, D., and Collins, N. (2011). *The SuperCollider Book*. The MIT Press.



SECTION II

SHORT-PAPER PRESENTATIONS

Ghost

Santiago Bosch¹

¹Sección de Biofísica
Facultad de Ciencias, Universidad de la República
Montevideo, Uruguay

stgo.bosch@gmail.com

***Abstract.** A musical piece composed in the Chuck programming language, inspired by the work of musician and illustrator Louie Zong, is described in musical and programming terms.*

1. Introduction

I was asked to compose this piece for IMPA's "live => coding music;" conference taking place during the first week of February 2019, because of my participation in the live coding workshop at the 2018 Imaginary Conference held in Montevideo, Uruguay. This composition grew out of a melodic line which I came up with during that workshop, along with some backing tracks, some of which were composed shortly after that.

The main inspiration for the melody, and after that the whole piece, was the work of musician and illustrator Louie Zong, who produces some beautiful music videos in his youtube channel: <https://www.youtube.com/watch?v=kXF3VYYa5TI>. His tracks often have sweet melodies with soothing backup tracks and a smooth feel, accompanied by his animations, which possess a signature style. He has created a couple of pieces featuring ghosts, in particular, one titled "ghost choir" [Louie Zong, 2018] which uses sine oscillators to simulate voices singing in harmony, and which was the main inspiration for this song, from which it takes its name.

2. Musical description

The song starts simple and grows more complicated with the addition of further shreds which remain synced and in harmony with each other. The backing harmony and the melody prevail present mostly during the whole piece. Throughout the song, the main melody is played through the right audio channel, with the rest of the harmony being played on the left channel.

The piece starts with a 16 bar melody which switches between B minor and F# minor, which will be present during most of the song. Afterward, some light percussion starts, soon followed by a backing harmony which repeats every 8 bars. After this, a steady bassline and a kickdrum beat are introduced, as the light percussion from the beginning is removed.

As the song grows in volume, more percussion is added, and the melody is replaced by a second 16-bar melody which switches between D major and F# minor. Eventually, the original melody is brought back, and a melody similar to the second melody is left as part of the harmony backing tracks. Finally, the shreds are removed one by one, leaving only the backing harmony to finish the song.

3. Program description

The composition consists of several scripts written in the Chuck [Wang 2008] programming language [Ge Wang, 2008] which are first coded live, sequentially executed in parallel and updated during the performance to add complexity.

To keep the shreds synced, a script created by a member of the Laboratório VIS-GRAF is used, which creates a class that automatically syncs when time starts to pass in different shreds, as well as define general notes duration to control the speed of play of different shreds.

One objective for this piece was to try and keep it within the predetermined Chuck samples to keep the sound simple. Because of this, almost all the sounds used in this composition come pre-installed with the language. Because of the inspiration, I took, mostly sine oscillators are used, with the exception of the percussion scripts, one of which uses samples and the other one which uses synthesized instruments via the unit generator - Shakers. This piece is available at: <http://w3.impa.br/~vitorgr/livecode>

References

- [Wang 2008] Wang, G. (2008). *The Chuck Audio Programming Language. "a Strongly-timed and On-the-fly Environ/Mentality"*. PhD thesis, Princeton, NJ, USA. AAI3323202.

“On/Off”

Martim Galvão¹

¹Department of Music
Brown University
Providence, Rhode Island, United States

`martim_galvao@impa.br`

Abstract. *The computer cursor is reimaged as a site for musical performance through a distributed mapping of performer inputs to cursor actions.*

1. Introduction

This paper describes the techniques utilized to transform musical gestures into user inputs in “On/Off”. The software created for the piece uses pitch information from four performers to control movement of the computer’s cursor, allowing them to use their playing to navigate the cursor in relation to an on-screen target. The piece explores distributed control and goal-oriented behavior as a means for generating musical output.

2. “On/Off”

In this piece, the group is split up into two teams, each consisting of one pianist and one percussionist. One team is given the goal of moving the computer’s cursor so that it “flips” a light switch on-screen, and the other team is given the goal of preventing the first team from accomplishing their goal.

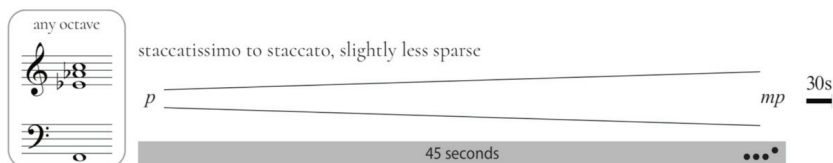


Fig. 1. Excerpt from “On/Off”.

One player on each team is given control of the up/down axis of motion, and the other player is given control of the left/right axis. By working together, the players are able to guide the cursor towards or away from the light switch on screen. In order to move the cursor up or right, a player must play two consecutive ascending pitches (*p*) (ex. C4 -> F4). The software will assign an upward or rightward direction to the input according to the input channel. Descending intervals move the cursor leftward or downward, respective to input channel. The distance the cursor moves is also affected by the interval of the pitches played, with greater intervals causing the cursor to move a greater number of pixels in the designated direction.

The game is structured as a series of rounds, in which one team can move the cursor while the other team must wait their turn. The rounds become progressively shorter as the game goes on. The final round is a “free-for-all” where players make one last

attempt to accomplish their goal before the game ends. If a team manages to flip the light switch before the game is over, the other team’s goal becomes to flip it back.

During each round, players are given a specific pitch set to use. Only notes from that pitch set will be recognized by the computer as valid inputs, allowing them to move the cursor. The pitch sets are notated on a score and change for each round, with the exception of the “free-for-all” round, where all pitches are accepted.

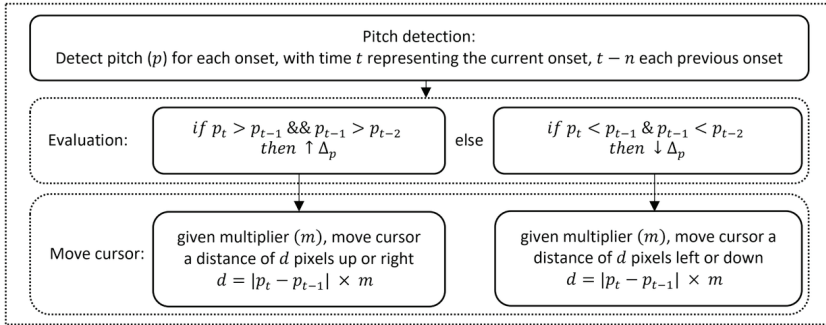


Fig. 2: Flow chart of pitch detection to cursor movement in “On/Off”.

Along with a dynamically shifting set of valid inputs, the system adjusts a multiplier for calculating the distance the cursor will move each round. In the initial rounds, the multiplier serves to create a strong bias in favor of moving the cursor toward the switch. For example, ascending (upward and leftward) movements will have a larger multiplier than descending (rightward and downward) movements for the team attempting to flip the switch, since they would move the cursor further towards their objective. As the game progresses the bias begins to shift, eventually favoring movement away from each team’s objective. This dynamic evaluation encourages the teams to make the most of their early rounds, since it becomes more difficult to accomplish their goals later in the game.

3. Conclusion

“On/Off” serves as an opportunity to explore a rules-based environment in which musical gestures inform the behavior of an on-screen user interface. While many of the techniques developed for the piece will make their way into future works, there will also be changes to rules that overly constrained performers. This includes relaxing strict adherence to pitch sets, reconsidering the turn-based system employed in the piece, developing more expressive input to cursor mappings utilizing amplitude tracking, and further exploring collaborative actions.

Crisis

Electroacoustic Improvisation

Molly Jones, Derek Worthington

¹Detroit, Michigan, USA

mojones.e@gmail.com

***Abstract.** Derek Worthington and Molly Jones are improvisers, composers, and electroacoustic musicians based in Detroit. Using SuperCollider, ChucK, trumpet, and saxophone, Crisis explores intersections of live processing and digital synthesis. The non-pulse-based music draws from aesthetics of improvised music, new music, and experimental jazz. Live-coded processing of live-generated acoustic sound embraces the spontaneity and immediacy of both acoustic and electronic music creation.*

1. Introduction

Improvisers Molly Jones (saxophones) and Derek Worthington (trumpet) have collaborated acoustically and electroacoustically since 2011, participating together in numerous ensembles with Detroit area musicians including experimental jazz, new music, and free improvisation projects. Instrumental performers by background, their live-coding duo has grown out of Worthington's extensive work with SuperCollider [Wilson et al. 2011] and Jones' exploration of ChucK [Wang 2008] and Max/MSP in solo performance. In *Crisis*, the performers balance the aural and aesthetic presence of their acoustic instruments with that of digitally manipulated and synthesized sounds.

2. Table and Equation example

Worthington uses SuperCollider, an open-source platform for real-time audio synthesis and manipulation, to perform live processing of acoustically generated sound. He has built a collection of patches that focus on immediate control as opposed to predetermined reactions or generative algorithms, allowing the computer to be used as an improvising instrument, responsive to the demands of group improvisation. By recording live sound into buffers, performing simple processing, and re-capturing the results, Worthington creates iterations of transformation, sending frequency, timbre, and temporal relationships progressively further from their source. Manipulating parameters on custom-built loops, delays, granulations, and fast Fourier transforms creates nested alternations of stasis and discontinuity.

Jones employs ChucK, a strongly-timed language developed at Princeton and Stanford by Ge Wang and team, to create layers of digital sound that blend with Worthington's processed live audio. By granulating simple oscillators and altering the timing of filters, pitch selection functions, and other effects pre-loaded in an arsenal of individual patches, Jones adds high- and low-frequency material and noise not easily accessible on acoustic instruments. ChucK's MiniAudicle format provides simple control of layering and looping shreds, individual iterations of patches, allowing just a few patches to create a wide variety of sounds and densities via a kind of additive synthesis.

3. Figure and URL example

In addition to the electronic traditions of glitch, drone, and musique concrete, the aesthetic of Crasis is influenced by Worthington and Jones' work with experimental jazz and new concert music. Both compose for a variety of improvising and chamber ensembles, acoustic and electroacoustic, and play in Balkan bands. All these improvisational traditions value spontaneity and preparedness; like technologists, improvisers must accept that their ideas may not work as envisioned and respond gracefully. Discussion of the lack of immediacy in electronic music performance is somewhat answered by live-coding, and the presence of acoustic instruments in this piece further anchors it in the realm of music performance. Crasis therefore embraces the immediacy of multiple processes at multiple levels: sound generation, computer interaction, and decision making.

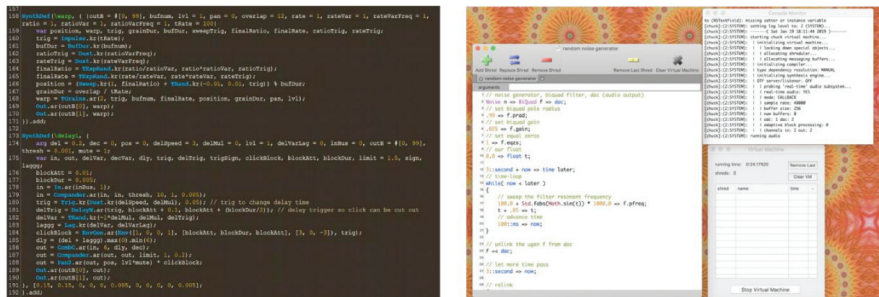


Fig. 1: (a) Two of Worthington's pre-loaded SuperCollider patches. (b) Chuck window with noise generator patch.

References

- [Wang 2008] Wang, G. (2008). *The Chuck Audio Programming Language. "a Strongly-timed and On-the-fly Environ/Mentality"*. PhD thesis, Princeton, NJ, USA. AAI3323202.
- [Wilson et al. 2011] Wilson, S., Cottle, D., and Collins, N. (2011). *The SuperCollider Book*. The MIT Press.

