Live Patch / Live Code

Charles Celeste HUTCHINS
University of Kent
celesteh@gmail.com

ABSTRACT

Modular synthesiser live-patching has gradually been accepted into the big-tent of live coding practice, due to a number of natural similarities to computer-based live coding. These similarities especially include flexibility, complexity and sudden stops. In my own performance of live-patching, I have sought out other areas of digital live coding practice to apply to the modular synthesiser. These include starting from a blank slate, showing my cables (sometimes with projection), graph changes and the use of conditional triggers to create audio events.

1. Introduction

The definition of live code has been gradually growing over the years of contemporary practice. In 2008, McLean struggled to name any non-digital live coding practice. (McLean 2008) Then, in February 2011, there was a very brief discussion on the TopLap email list as to whether live patching modular synthesisers was an analogue form of live coding (Rohrhuber 2001). By June of that year, Nick Collins was treating the question as settled in *Leonardo*, listing live electronics band Loud Objects as an example of live coding. (Collins 2011) He notes, when talking about live algorithms, 'Computers remain a primary touching point, but let us also keep in sight examples that center on human rule-making independent of digital computability.' (Collins 2011) Given this very broad potential pool of music-making activity, live coding has become extremely inclusive. Thor Magnusson referred to 'herding cats' in the title of one his papers, as a reference to this wide diversity. (Magnusson 2014) Indeed, I heard a joke recently that it would be faster to make a list of all the things that *aren't* live coding, rather than the things that are.

2. Synthesiser as a Live Code Platform

Given how open live coding has become, it may become useful to discuss points of commonality between divergent live coding practices. In many ways, live synthesiser patching seems different from live coding with a computer, given the vastly different interface. There is a greater amount of tactility, which, to me at least, feels closer to playing a traditional musical instrument than creating an algorithm. It is also worth noting that the synthesiser is specifically designed for music making, unlike a computer, and thus has a limit on the kinds of operations and calculations it can do. In this way, a synthesiser is both a chainsaw and an idea. (TopLap 2010)

Thus chainsaws vs ideas become less of a binary opposition and more a continuum. Tools, such as chainsaws, are perceived as extensions of the body of the user. (Cardinali et al. 2009) My experience of playing the tuba very much matches this. By contrast, my experience of text-based computer live coding feels abstract and unembodied. Live-patching the synthesiser lies in between. The afore-mentioned tactility does give it some feelings of embodiment. But plugging in cables and deciding how to route signals primarily involves abstract decision-making. Playing a patch, therefore, is chainsaw-like. Making the patch is idea-like.

2.1. Commonality

A modular synthesiser, therefore has many points of similarity with digital live coding practice. The specific design of a synthesiser for only music making may not parallel computer hardware, but the specificity certainly parallels the way some live coding languages have developed. While you can't use your synthesiser to check your email, this is also impossible in several live-coding languages and with early analogue or other non-digital computers. And, indeed, the modular synthesiser does have a general purpose element within the musical sphere. Common practice, when I was an

undergraduate, was to spend time in the studio, carefully making the perfect sound. This is also how I was taught to write code.

Historically, modular synthesisers do not seem to have been seen as live electronics systems to the same extent as were some particular pieces, such as *Runthrough* by David Behrman (1971), which is realised live with DIY electronics. The programme notes for that piece state,

> No special skills or training are helpful in turning knobs or shining flashlights, so whatever music can emerge from the equipment is as available to non-musicians as to musicians . . .. Things are going well when all the players have the sensation they are riding a sound they like in harmony together, and when each is appreciative of what the others are doing (Behrman 1971).

The piece is based entirely on discovery and has no set plan or written score (ibid). The piece-ness relies on the equipment. Scot Gresham-Lancaster describes pieces like this one as "direct use of electronics as a 'score' " (Gresham-Lancaster 1998).

Historical live electronic pieces like *Runthrough* do differ from live-patching in that a modular synthesiser is designed to be a more general purpose tool and its use does not imply a particular piece. Interestingly, however, some live coding platforms, like ixi lang, have been compared to graphic scores due to their use of space (Magnusson 2011) and thus do have a certain piece-ness about them. However, unlike ixi lang and *Runthrough*, understanding the interaction between synthesiser modules is not designed for ease of use for non-musicians. It is a specialist skill which does imply that expertise is possible. However, the idea of finding a sound and following it is similar to live electronics and all of these practices make use of improvisation. These are all points of commonality between live patching and digital live coding.

Additionally, the interface differences between modular synthesis and computer languages do not create as large a gap as it might at first seem. While synthesiser modules are not like text-based code, neither are all languages. Scheme Bricks, a live coding language by Dave Griffiths (2012) is an example of a coding language that relies very heavily on graphic elements rather than dense lines of text. A step closer to the synthesiser patching interface (and not uncoincidentally so) are the patching languages MAX/MSP and Pd. A version of one of these languages which had only a set and limited number of unit generators and control generators would not be logically different than a modular synthesiser. Even the model of making connections is the same.

2.2. Complexity

Both digital systems and analogue modular systems allow a user to create a graph of unit generators to generate sounds electronically. In both systems, the graphs and their control mechanisms can quickly grow in complexity, past the point where the player can easily understand what is happening. The late Click Nilson hints at this in his 2007 paper. In a footnote, he quotes a personal communication from Julian Rohrhuber, 'The main difficulty of live coding I find is to keep aware of the relation between text and sound - this is a balancing act between the ability to change and the ability to understand a certain situation' (Nilson 2007) While certainly live patching lacks text, there is some critical number of patch cables beyond which it becomes increasingly difficult to keep all the connections in mind. Patch cables turn into a kind of electronic spaghetti, where even the colour codes of my cables are not helpful in comprehending what is connected to what. I attempt to compensate for this by physically tracing my hands along the wires, but even that has its limits. Given the extreme flexibility for feedback and chaos with analogue signals, the routing can become intensely complex. Pd programmers are familiar with a similar effect of a visual spider web forming on screen.

This complexity also leads to a situation analogous to 'crashing', wherein the synthesiser suddenly stops making sounds in performance, in a way that can be difficult to predict. Experimenting with an oscillator's 'hard sync' switch during a set may lead to a sudden and interesting timbre change or a sudden silence. Similarly, long running LFOs can drop below a threshold where they unexpectedly cause things to fall silent. It is often not at all obvious what has caused the sound to stop. In situations where there are a large number of wires and multiple modes of interconnected sound-generation, it can be more beneficial during performance to treat a sudden, unexpected silence as a stopping point and start anew with new sounds after a brief pause to remove cables.

Although it would be impossible to exactly replicate a previous live patch in performance, I do sometimes try to repeat steps that have lead to a 'crash' in past situations, as they make a sudden and final-sounding way to end. I've seen many digital code-based live coders end sets through crashes and at least some of them do court disaster on purpose. Click Nilson has written of the desirability of a '"code out" ending of deliberate coded crash'. (Nilson 2007) And some live coding languages have embraced crashes as language functionality. ixi lang, for example, has a built-in 'suicide' command, which creates a percentage chance of crashing on each bar. (Magnusson 2012)

3. TopLap Influences

Many of the points of similarity above are inherent in live patching, due to a synthesiser's evolution from the analogue computer. However, the courting of crashes is one way in which my practice has been specifically influenced by the live code movement and the TopLap Manifesto in particular. (TopLap 2010)

3.1. Show us your screens

Like blank slate digital live coders who start positioned behind a laptop with a blank screen, I start with all of my patch cables hung around my neck. Thus we both experience a short pause from starting the set to starting making sound. I also emulate the 'show us your screens' directive, so I turn my synthesiser at an angle to the audience, so the majority of friendly experiencers can see my actions to at least some extent. This orientation is a compromise position such that my instrument is largely visible, but I am not turning my back on the audience. For many lines of sight, it is probably more visible than simply standing in front of the synthesiser. I have experimented also with using a web cam to project a video image of the synthesiser as I patch.

In order to make things more transparent, at the Live Code Festival in Karlsruhe in 2013, I paired this with a manually triggered set of slides of super-imposed text. These started with a description of what colour cable I intended to use for what kind of signal. Copying the Royal Conservatory of the Hague's coding system, I projected that I would be using blue for Control Voltages, black for Audio Voltages and red for Triggers. In addition, I projected the name and a photograph of the module that I was using when plugging into it for the first several connections I made.

While audience feedback was generally positive about these good intentions, it was difficult to find a web cam angle that was not blocked by my body as I used the instrument. Even more disappointing, the combined low quality of my web cam and of the projector meant that the different cable colours were not visible to people watching the video display. In low light situations, such as one often has on stage, black cables often become completely invisible. Although a better camera may solve colour issues, I have instead reconsidered what information I should be giving to audiences. Although McLean notes that programmers can take steps such as 'using a language with a simplified syntax, choosing highly descriptive variable names and so on', (McLean 2008) I have never seen a show-us-your-screens live coder project any explanations as to their syntax colouration or what various commands mean.

3.2. Graph Changes

When discussing approaches to digital live coding, Magnusson writes,

> The argument . . . is that a true 'liveness' requires that the performer is not simply manipulating pre-written code in real time, but is actually writing and modifying algorithms during the execution of the program. This is arguably a prerequisite of live coding, as simply running code and changing parameters in prewritten code is more comparable to operating buttons, sliders, and knobs on a screen interface or a MIDI controller. (Magnusson 2014)

This demand for algorithm modification also appears in Collins's live coding analogues, wherein he describes several examples of changing the algorithm mid-play. In one such example, specifically, 'a live-coding twist would be to perturb the algorithm halfway through'. (Collins 2011) Perhaps, however, it would be useful to take a step back and examine what is meant by 'algorithm' in an analogue context.

Plugging in cables creates a unit generator graph. The output of one circuit is connected to the input of another, eventually creating a complicated sound generation and control process. This is the creation of an algorithm, just as combining unit generators in a patching language creates an algorithm. However, playing this patch, by twiddling knobs on a synthesiser is exactly like operating knobs on a MIDI interface. Indeed, the physical interface is 'pre-written'. This is not to say that a physical interface is necessarily alien to live coding. Marije Baalman's live-coded piece, *Wezen-Gewording*, uses sensors attached to her hands, to capture the physical gesture of her specifically performative gestures and of her typing code. (Baalman 2014) However, setting something up and relying solely on parameter changes, while musically valid, is dubious under the heading of 'live coding.' Therefore, Baalman changes her code over the course of the piece. And in my practice, I do not rely on the algorithmic rules established at the start of the set. Instead, I 'perturb the algorithm' by setting up and switching to parallel signal paths.

Figure 1: *An image I projected in Karlsruhe.*

3.3. Decisions

Many computer programs involve decision making. This is not strictly necessary for audio processing, which only needs follow a signal graph. However, I seek to include decision making in my patches, in order to create stronger similarities to coding practice. My synthesiser includes a MOTM 700 voltage controlled router – a module that specifically does binary signal switching based on a comparison of two signals. However, although this module is an obvious choice for seeking to automate some decisions, this kind of behaviour is also found via triggers and envelope generators. When one signal changes, this changes another signal. This seems roughly analogous to an if statement. Because I often use both random generators and FM chaos, I cannot always predict when a trigger will occur. Thus the synthesiser picks up a degree of autonomy and behaves in an unpredictable way, increasing a distance between it and a chainsaw. The instrument may be a tool, but a patch is more like an expression of an idea. (TopLap 2010)

3.4. Scheduling

Many live coding languages offer the possibility of scheduling future events and some have argued that this is therefore a necessary component of live coding or live coding-related practice. In essence, this is a situation where the performer decides to invoke or stop a particular algorithm at a future time. A synthesiser user who wishes to mimic this behaviour has several options, including step sequencers or using LFOs to trigger future events or graph changes. As with deccisions, one can make the timing less predictable by using random generators or chaos. This is not something I tend to use either in my live coding or my live patching practice, but is certainly possible to implement.

4. Conclusion

While discussing live coding metaphorically, McLean wrote, 'We can think of live coding in the sense of working on electricity live, re-routing the flows of control . . ..' (McLean 2008) In modular synthesiser patching, this practice is not a metaphor. Flows of control are literally plugged and unplugged. Complex systems develop and are replaced by new complex systems. It's no longer necessary to retreat into our imaginations to experience live coding without a computer.

References

Baalman, Marije. 2014. "Wezen – Gewording." Performance; STEIM. https://www.marijebaalman.eu/?p=404.

Behrman, David. 1971. "Runthrough." In *Sonic Arts Union*. LP; Mainstream Records. http://www.ubu.com/sound/sau.html.

Cardinali, Lucilla, Francesca Frassinetti, Claudio Brozzoli, Christian Urquizar, Alice C Roy, and Alessandro Farnè. 2009. "Tool-Use Induces Morphological Updating of the Body Schema." *Current Biology* 19 (12): R478–R479.

Collins, Nick. 2011. "Live Coding of Consequence." *Leonardo* 44 (3): 207–211.

Gresham-Lancaster, Scot. 1998. "The Aesthetics and History of the Hub: the Effects of Changing Technology on Network Computer Music." *Leonardo Music Journal*: 39–44.

Griffiths, Dave. 2012. "Scheme Bricks." http://github.com/nebogeo/scheme-bricks.

Magnusson, Thor. 2011. "Algorithms as Scores: Coding Live Music." *Leonardo Music Journal* 21: 19–23.

———. 2014. "Herding Cats: Observing Live Coding in the Wild." *Computer Music Journal* 38 (1): 8–16.

———. 2012. "Ixi Lang." Accessed March 3. http://www.ixi-audio.net/ixilang/.

McLean, Alex. 2008. "Live Coding for Free." *Floss+Art. London: OpenMute*: 224–231.

Nilson, Click. 2007. "Live Coding Practice." In *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, 112–117. ACM.

Rohrhuber, Julian. 2001. "[Livecode] Analogue Live Coding?" http://lists.lurk.org/mailman/private/livecode/2011-February/001176.html.

TopLap. 2010. "ManifestoDraft." http://toplap.org/index.php/ManifestoDraft.