Adapting DAW-Driven Musical Language to Live Coding: A Case Study in EarSketch

Takahiko Tsuchiya
anna.xambo@gatech.edu
jason.freeman@gatech.edu

Georgia Institute of Technology
Center for Music Technology
840 McMillan St., Atlanta, GA 30318

ABSTRACT
A number of musical live-coding performances attempt to communicate with the audience by means of using visualizations to help them understand the ongoing creative process. EarSketch, a web-based platform for computer science and music education that supports live-coding capabilities, provides a visualization inspired by a digital audio workstation (DAW) interface to aid the student or audience to understand the code. This paper discusses the benefits and challenges of combining DAW-inspired linear composition techniques with nonlinear live coding practices, providing a case study of a live coding session in EarSketch.

1. INTRODUCTION
Text-based live coding languages can often be impenetrable to understand for lay audiences. They may appreciate the virtuosity involved in creating music on the fly through code but fail to understand the code that produces the music they are hearing. Live coders, therefore, often use animations and visualizations to try to make the connections between code and music clearer (Wang and Cook 2004), or even create live coding languages deeply integrated into the logic of visual systems such as Fluxus (Griffiths 2007). This aligns with the expectations of a live coding session in which the live coder’s workspace should be made visible to the audience.1

Digital Audio Workstations (DAWs) are pervasive music production tools that employ a standardized strategy for visually rendering multi-track musical timelines of audio, MIDI and effects that is readily comprehensible by lay users and audiences. Programmatic techniques have also effectively been integrated into DAW interfaces (e.g., Max for Live2, ReaScript3). In this paper, we consider how to combine live coding with a DAW framework using the EarSketch environment (Freeman and Magerko 2016). The approach is to explore the DAW-inspired visualization and the DAW-driven language API of EarSketch to make the musical results of live coding execution and the relationship between music and code more comprehensible to a general audience. In particular, this paper discusses the challenges and potential benefits of shifting from a time-lined musical composition’s paradigm to a live-coding paradigm. Following this, we present a case from an exemplar live-coding session, which illustrates a process of adaptation to a loop-based nonlinear music performance.

---

2. EARSKEtCH

EarSketch\(^4\) is a web-based educational platform for introductory computer science (CS) and music education, primarily at the high-school level. The platform features a Python / JavaScript API, a text-based and blocks-based code editor, an extensive audio-loop library in popular musical genres and a multi-track timeline visualization similar to a DAW, among many other features (see Figure 1). Students develop structured knowledge of basic computational and musical designs through the curriculum materials, classroom lectures and with the interactive programming environment.

Recently, EarSketch introduced a live-coding mode with loop points and non-stop crossfading audio rendering, allowing more real-time feedback to students about changes in code as well as opportunities for live coding EarSketch performances with DAW-driven visualizations (Freeman and Magerko 2016). This addition, however, creates some conflicts between loop-based and time-lined sequencing, and between continuous modification and batch processing of the entire song, which is discussed in this paper.

3. BENEFITS OF LIVE CODING IN EARSKEtCH

As an authentic learning environment for CS, the EarSketch’s curriculum conforms to the Advanced Placement Computer Science Principles (Astrachan and Briggs 2012). The EarSketch environment encourages the student to develop skills for properly structuring their music program ahead of execution time, and to test their conceptual design against the resulting sound, visualization (i.e., the DAW view), and text output in the debugging console. Introducing live coding in this formal structure may benefit the learning experience in various dimensions. Freeman and Magerko (2016), in a previous discussion of the pedagogical opportunities afforded by live coding in EarSketch, focused on the rapid iterative creation cycles common to both programming and composition and how modern DAWs and experimental software development environments (IDEs) already support simultaneous modification and execution to speed this iterative cycle. Live Coding, therefore, is a natural extension of these existing activities in both domains. Xambó et al. (2016) explore the potential of live coding environments in the classroom to facilitate collaboration, moving beyond group projects and pair programming paradigms to support simultaneous authoring and execution of the same project by multiple students. Guzdial (2014), in contrast, considers the educational role of live coding not in the classroom but in live performance settings, where audience members witness a virtuosic live-coding performance, are exposed to the computational logic and development process which underlies the music, and may be engaged to pursue further learning opportunities after the event.

\(^4\) http://earsketch.gatech.edu (accessed Aug 29, 2016)
4. ADAPTATION TO LIVE CODING

The workflow of a traditional EarSketch programming session operates in a batch mode: the student authors code in the code editor, presses a run button to execute that code to generate a DAW timeline of the resulting music, and then presses a play button to hear the music. To modify the music, the student modifies the code, then presses run and play again. This behavior aligns to the educational standpoint of teaching traditional programming languages, where the student has to conceptually plan the working program ahead of time before the execution. Live coding, however, may demand a significantly different style in programming from the regular EarSketch workflow. Brown and Sorensen (2009), for example, argue the general requirements in algorithmic live coding, such as succinctness, parametrization, and modifiability of code. Collins et al. (2003) discuss the characteristics of the computer languages used in live coding performance including flexibility, rich grammars and arbitrary changes of structure in real time.

In the live-coding adaptation of EarSketch, code editing, execution, and audio playback can all happen simultaneously. The DAW timeline (or a time slice of it) is looped continuously. Each time the student (or live coder) re-executes the code, the timeline and audio are updated non-disruptively without stopping playback or resetting the play head.

The dominant style in live-coding compositions uses short, repetitive but modifiable patterns that provide a musical continuity (e.g., Magnusson 2011; Roberts and Kuchera-Morin 2012). In EarSketch, the extensive audio-loop library and the clip-sequencing functions make it trivial for a student with no prior knowledge in music to generate repeating patterns. However, continuously modifying such patterns programmatically or in real time requires non-trivial effort and considerations. Particularly, the coder's focus may be shifted to different hierarchical levels than the whole-song arrangement, by either restricting or temporarily hiding the song structure in EarSketch. For instance, the global time line becomes less important but the intra- or inter-clip time may become more important.

5. CASE STUDY OF A LIVE CODING SESSION

As a demonstration of the potential of DAW-inspired live coding and a case study in the design challenges and opportunities afforded by such environments, we now analyze a recording of a live-coding session with EarSketch\(^5\). This example illustrates a process of crafting the musical "hocket" effect algorithmically, a commonly used musical technique in sampling-based DJ performances\(^6\). The algorithmic hocket technique extends the concept of note-level sequencing in EarSketch, which uses the `makeBeat` function\(^7\), but also incorporates additional musical properties, such as varying note length and audio-clip phase.

In the video, the live-coding performance begins with a simple 4-bar loop (0:48). Then, instead of extending the clip arrangement in the timeline, as students do in a regular EarSketch practice, the live coder attempts to modify the pattern within the 4-bar loop by first alternating the audio clips manually (0:58) and then programmatically (1:13). As the original `fitMedia` function\(^8\) is not supporting the hocket technique's goal, the coder decides to use an alternative function, `insertMediaSection`, to achieve the rhythmic effect and the inter-clip phase continuity (2:55). As the musical expressiveness grows, it also poses unique programming challenges such as aligning rhythms to musical beats using conditional statements (4:23, 5:30). In the end, when the generative hocket algorithm is mostly established, the coder can start to adjust the parameters, such as the note-length or the audio-clip array, to quickly and iteratively make variations. In each musical or programming challenge, the coder here is urged to apply the available knowledge of programming (Python or JavaScript), and even reach out to the information beyond the curriculum materials (e.g., type casting, variable scopes). The video also shows a few moments of how a live coder handles with

\(^7\) A function to operate on rhythms with string-based sequencing.
\(^8\) Cf., https://earsketch.gatech.edu/earsketch2/?curriculum=2-3-0 (accessed Aug 29, 2016)
programming errors (0:40, 1:20, 6:10), in which real-time debugging is required without disrupting the musical loop.

It is important to note that the coder’s problem-solving and debugging may be largely informed by the real-time DAW visualization, sometimes even before an unintentional sound is heard (4:20). Even in a short-cycle live-coding loop, the visual outlining of the rhythmic arrangement can guide the live coder’s compositional plans and future actions in a swift manner, while at the same time aiding the audience to understand the code using a well-known graphical user interface metaphor. The combination of a code editor with a familiar DAW interface allows the audience to understand, and potentially learn about CS and music concepts, from observing a live coder’s performance in the wild.

6. CONCLUSIONS

We discussed the benefits, challenges, and potential impacts on both programming and musical styles of live coding in EarSketch. In addition to the situated musical and programming challenges, live coding practices complement the formal learning experience of EarSketch users with the informal and impromptu learning. The live coder may also benefit from the DAW visualization, which guides the composition, problem solving, and communication of the ongoing creative process to the audience. As future work, we hope to extend this study to other musical environments that feature DAW-like framework, connecting further linear composition and nonlinear live coding practices.

Acknowledgments

EarSketch receives funding from the National Science Foundation (CNS #1138469, DRL #1417835, and DUE #1504293), the Scott Hudgens Family Foundation, the Arthur M. Blank Family Foundation, and the Google Inc. Fund of Tides Foundation.

REFERENCES


