Coding Livecoding

Ben Swift
R.S. Computer Science, CECS
Australian National University
ben.swift@anu.edu.au

Andrew Sorensen
Inst. Future Environments
Queensland University of Technology
a.sorensen@qut.edu.au

Michael Martin
R.S. Finance, Actuarial Studies and Applied Statistics
Australian National University
michael.martin@anu.edu.au

Henry Gardner
R.S. Computer Science, CECS
Australian National University
henry.gardner@anu.edu.au

ABSTRACT

Livecoding is an artistic programming practice in which an artist's low-level interaction can be observed with sufficiently high fidelity to allow for transcription and analysis. This paper presents the first reported "coding" of livecoding videos. From an identified corpus of videos available on the web, we coded performances of two different livecoding artists, recording both the (textual) programming edit events and the musical effect of these edits. Our analysis includes a novel, transition-matrix visualisation of the textual and musical dimensions of this data to create a "performer fingerprint". We show how detailed transcriptions of livecoding videos can be made which, we hope, will provide a foundation for further research into describing and understanding livecoding.

Author Keywords

Creativity Support Tools; End-user Programming

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces – Evaluation/methodology

INTRODUCTION

Livecoding is a "new direction in electronic music and video" [1] in which the performer(s) write computer code in front of an audience to generate music and visuals in real time. The act of programming is part of the performance, and performers are exhorted to "show us your screens. . . code should be seen as well as heard" [1]. Academic treatments of livecoding can be found in [3, 2, 8] and elsewhere.

In spite of its revolutionary manifesto, livecoding is now well over a decade old and has become part of the establishment of computer music. Allied forms of musical performance are electronic music, minimalist music, and jazz. Modern, software-enabled DJ/VJ performance is a more mainstream relative of livecoding, and there has been recent interest in the "top-down" HCI design of software interfaces for DJ/VJ and other similar artistic practices [4, 5]. In contrast to these interfaces, the genesis of livecoding was as a reaction against the strictures of "designed" interfaces. In the words of Collins et. al: "Not content to submit meekly to the rigid interfaces of performance software. . . (livecoding artists) work with programming languages, building their own custom software, tweaking or writing the programs themselves as they perform." [3]

The documentation and sharing of livecoding performances takes the form of performance videos, together with (often indistinct) images of projected computer screens. A few artists have taken to producing direct-feed screencasts of livecoding performances, whose quality is sufficiently high to permit manual 'coding' of these performances after the fact. The existence of these higher-quality videos motivates the following questions:

1. Can a "bottom-up", edit-level analysis of livecoders' activity reveal insights into the into the way that livecoding performances evolve in real time as the artists write computer code to create and manipulate music and video?
2. Are there ways in which differences in "musical style" between different artists can be represented from this interaction data?

PROCEDURE

In this note we describe a systematic study of a collection of livecoding videos, each of which was manually annotated to produce a time-stamped transcript. Each video in our corpus satisfied the following requirements:

- We restricted ourselves to complete, musical livecoding pieces rather than shorter 'tech-demo' style videos. This also ruled out livecoding videos where the artistic outcome was visual, rather than auditory, a choice which was made not as a value judgement but rather to simplify the comparisons between pieces in our analysis.
- One norm in livecoding is to start performances from a blank editor screen [7]. We restricted our focus to livecoding videos that started in this way for the sake of bringing out common aspects of these performances. This restriction meant that the start-up phases of the videos contained

CHI 2014, April 26–May 1, 2014, Toronto, Ontario, Canada.

http://dx.doi.org/10.1145/2556288.2557049

a large number of insertion events.
- To look for elements of style across multiple works by the same artist, we required that the artist have at least four eligible videos to be included in the corpus.

Identifying a comprehensive corpus of livecoding videos is a challenge in its own right. While livecoding has a semi-official home at `toplap.org`, there is no canonical list of videos there. There are other websites[1] which attempt to maintain curated lists of livecoding videos, but none of these lists are comprehensive. In the end, our list of potential videos was assembled from livecoding artists' individual websites, Google, YouTube, and from our knowledge of the TOPLAP community. Our restrictions meant that our videos were direct screen recordings.

We selected the videos of two artists: Andrew Sorensen[2] (AS) and Ben Swift[3] (BS), who between them made up approximately three quarters of all the videos which met the criteria described above. In addition, both of these artists used the Scheme-based livecoding software environment Impromptu [8]. This allowed for a simplified coding schema and facilitated an easy comparison between the two artists. In the end our dataset comprised 13 livecoding videos, with a total running time of approximately three hours. A full list (including URLs) of the videos analysed to produce the final dataset is given in Table 1.

Coding

Each of the selected videos was analysed frame by frame to produce a series of time-stamped event codes[4] describing the activity of the livecoder. We chose an event-level granularity for these codes, since the framerate of the videos did not always allow us to delineate individual keystrokes. Event boundaries were determined by the times at which the source code was syntactically correct. For example, if the livecoder started to write a new function, that event was considered to last until the function was complete (i.e. callable). When editing existing code, smaller events were possible, such as the changing of a literal value or the addition of a new clause to a `case` statement.

Our coding schema had the following fields:

- **timestamp**: the *start* time of an event (in seconds), and (optionally) the *end* time of that edit, for edit events longer than one second.
- **textual meaning**: was an event an *insertion*, *deletion* or *evaluation*, or was it a *quick edit*, (such as changing an integer literal `60` to `70`)?
- **musical meaning**: did an edit affect the *pitch*, *rhythm*, *dynamics* or *timbre* of the music, or *all* of the above (as in the introduction of a new instrument)?
- **instrument**: in most cases it was possible to delineate different musical instruments/voices in the piece (e.g. *bass*, *drums*, *piano melody*, *piano accompaniment*).
- **comment**: a free-form text field for any salient features of the edit which were not captured in the other fields.

The rationale for this coding schema was to keep the cardinality of each coding "dimension" as small as possible, and to allow the interactions of these dimensions (4 textual codes $\times$ 6 musical codes) to provide richness and complexity in the data.

Event coding was completed by a professional composer, with tertiary degrees in both music composition and computer science. Although this person had some livecoding experience, they were not the creator of any of the pieces studied. While determining the textual meaning of each edit was fairly straightforward, the musical and instrument fields required subjective judgements, and we also provided a free-form "comment" field to enable notes to be recorded about possible ambiguities and other issues. It was possible to provide *multiple* musical or instrumental codes for a given textual edit, e.g. when the modification of a global parameter affected both the pitch and dynamics of the piece.

RESULTS

In total, 2577 edit events were recorded in the dataset, with an average of 15 edits/min. Figure 1 shows the way that the distribution of these events changed over the course of each piece by breaking each piece into three equal segments: the beginning (first third), middle (middle third) and end (final third) of each piece.

In the textual dimension data exhibits a shift from having a large number of insertion events (writing new code) at the beginning of each piece towards primarily evaluation and quick edit events at the end of the piece. Both artists exhibited this trend. Musically, however, there are different edit proportions between the artists: AS favoured dynamic (loudness) edits, while BS preferred rhythmic ones and edits which affect whole instruments (the "all" musical category).

Figure 2 shows the frequency of some of the different edit types between pieces, separated into the beginning-middle-end time chunks. This figure shows the same trends as Figure 1, with code insertion events early in the piece and evaluations and quick edits in the latter stages. AS makes noticeably more dynamic (musical) edits in the latter stages of his pieces—a trend not evident for BS.

Transition matrices

Transition matrices (Figure 3) can help us look for "signatures" of artists in terms of their edit *transitions* (the sequence of edit events) in both the textual and musical dimensions. This is a new contribution of this work which expands on the use of transition matrices for chord sequences by Nichols et. al. [6]. A key aspect of our approach is the creation of a temporal sequence (beginning-middle-end) of transition matrices to show the evolution of an artist's activity within their pieces.

[1] such as `http://livecoding.co.uk/doku.php/videos`

[2] `http://vimeo.com/andrewsorensen`

[3] `http://vimeo.com/benswift`

[4] In this paper we use the term "code" to describe both the textual source code of a program and the (event code) label given to a particular event in the "coded" transcript.

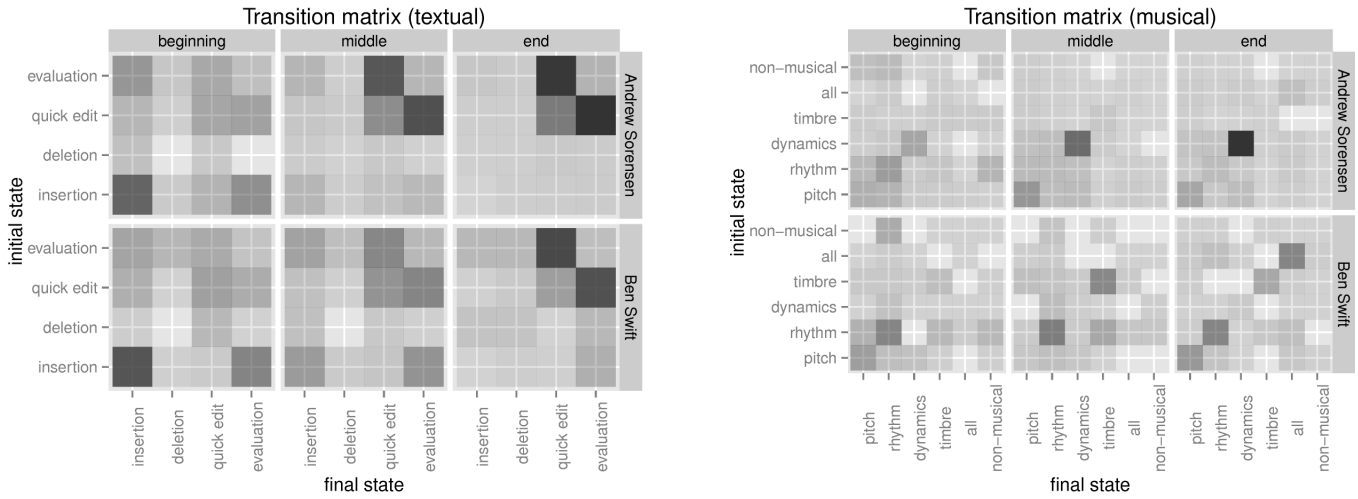

Figure 3. Transition matrices for both the *textual* and *musical* dimension of each edit event. A darker cell in the matrix indicates a more common transition (initial state → final state) in the dataset.

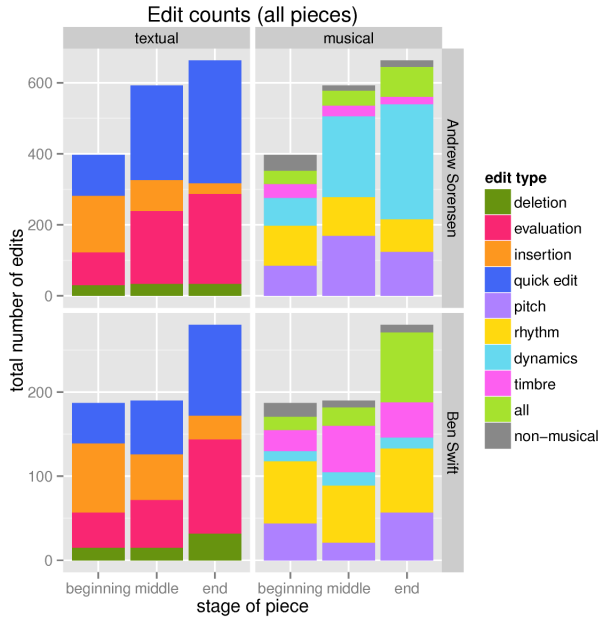

Figure 1. Total number of edits across the dataset. The y-axis scale is different between artists due to the fact that each artist had a different number of pieces in the dataset. There is a trend towards a greater number of edits toward the end of a piece, as well as a general shift in proportions of the edit types (both textually and musically) across the different stages of each piece.

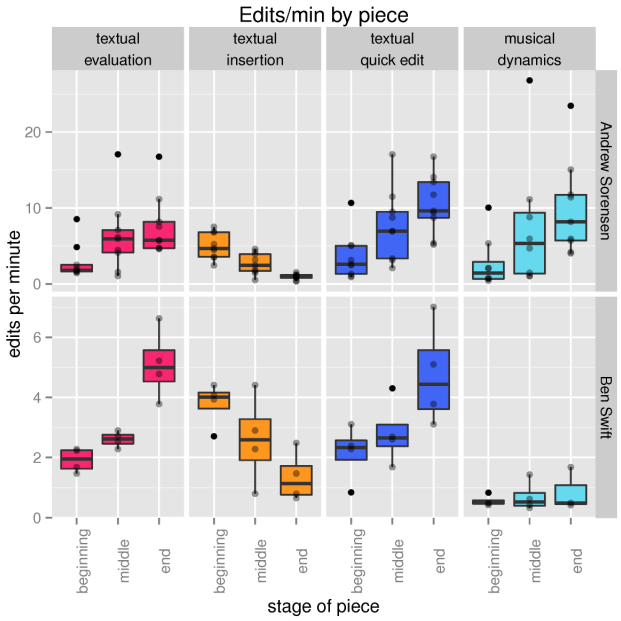

Figure 2. The edits/minute activity across pieces, broken down by artist and both the textual and musical meaning of their edits. Since the pieces are different lengths, we have used an edits/min variable instead of a total edit count to facilitate comparisons between pieces (and piece segments) of differing lengths. The raw values for each piece are shown with black dots, overplotted with box plots to show the distribution. These boxplots may be misleading in the bottom row (artist BS) due to the small number of pieces ($n = 4$ for BS, while $n = 9$ for AS). Note also that the y-axis scale is different for each row—in general AS (top row) makes more edits per minute than BS.

Figure 3 (left-hand side) shows the textual transition matrices for each artist. Both artists exhibit a progression from insertion events (bottom left-hand corner) to quick edits and evaluations (top right-hand corner) as each piece develops. Also visible in the transition matrix is the 'quick edit → evaluation' (and back again) loop in state space. This suggests that the quick edit and evaluation events in the latter stages of a piece tend to alternate, rather than being contiguous in time. This observation makes sense given the nature of the Impromptu software environment, in which changes to the source code must be evaluated before they take effect.

Figure 3 (right-hand side) shows the same transition matrices, but this time from a musical perspective. Interestingly, there is much less similarity between the artists. Over time, AS's transition matrix becomes increasingly dominated by 'dynamic edit → dynamic edit' self-transitions, while BS's matrix remains diffuse, even in the later stages of his pieces. This suggests that, from a musical perspective, AS starts out making code edits which affect many different aspects of the music, but in the later stages of his pieces he 'zooms in' on dynamic edits, adjusting the already-running musical voices up and down in dynamic range as dictated by his aesthetic goals. In contrast, BS continues to jump around more in the musical nature of his edits, although there seems to be a slight trend towards the on-axis (i.e. self-transitions) values in the transition matrix in the final third of his pieces.

DISCUSSION

Figures 1, 2 and 3 all show visible differences in edit activity over the course of the pieces. The most interesting aspect of the analysis is that while there are recognisable patterns in both the textual and musical dimensions of the edits, only in the musical dimension do there seem to be recognisable differences *between* the artists (see Figure 3). While it might seem, on the surface, that a textual programming language may provide too broad an interface to the musical power of the computer, it is interesting that the differences between the artists are not so visible based on their activity on a purely textual level, but come out clearly when considering the musical meaning of their edits.

A potential future use for the sequence of local transition matrices is the detection of natural change points in live-coded pieces, through both the textual and the musical activity of the livecoder.

CONCLUSION

This paper has presented the first reported coding and event-level analysis of a decade-old artistic endeavour which is centred around programming at a computer interface. Although our data set is small, being limited by the available web presence of livecoding, we have examined the works of two different livecoding artists with a one second temporal resolution along both the programming (text editing) and musical dimensions. We have also demonstrated the utility of transition matrices for visualising stylistic differences between artists.

The possibilities of this data collection approach are exciting, and we acknowledge that this note presents only a preliminary exploration of these possibilities. Future work could extend our study through the instrumentation of the programming interface to enable automatic data collection, refining the coding scheme (e.g. breaking "quick edit" into subtypes), exploring the relationship between the textual and musical dimensions of the scheme, and also to examine and contrast livecoding with other source code editing activities. As more data is collected, we hope to be able to answer with more confidence questions about the nature of style and the specific patterns of interaction which characterise livecoding.

ACKNOWLEDGEMENTS

The authors would like to thank Matt Rankin for his contribution to this work.

Artist	Title	Vimeo ID
AS	Strange Places	2503257
AS	Dancing Phalanges	8732631
AS	A Study in Keith	2433947
AS	Jazz Ensemble Study	15679078
AS	Variations on a Christmas Theme	18008372
AS	A Christmas Carol	8364077
AS	Day of the Triffords	2735394
AS	Orchestral	2579694
AS	Antiphony	2503188
AS	A Study in Part	2434054
BS	I am an Elder Brother	7525775
BS	Archtop	6940393
BS	A Few Days Late	7196155
BS	Fear of Sucking	7039223

Table 1. The livecoding videos which were coded for analysis. Each video can be seen online at `https://vimeo.com/<VimeoID>`

REFERENCES

1. *TOPLAP Manifesto*. `http://toplap.org/wiki/ManifestoDraft`.
2. Blackwell, A., and Collins, N. The programming language as a musical instrument. *Proceedings of PPIG05 (Psychology of Programming Interest Group)* (2005).
3. Collins, N., McLean, A., Rohrhuber, J., and Ward, A. Live coding in laptop performance. *Organised Sound 8*, 03 (2003), 321–330.
4. Hook, J., Green, D., McCarthy, J., Taylor, S., Wright, P., and Olivier, P. A vj centered exploration of expressive interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 1265–1274.
5. Hook, J., McCarthy, J., Wright, P., and Olivier, P. Waves: exploring idiographic design for live performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 2969–2978.
6. Nichols, E., Morris, D., and Basu, S. Data-driven exploration of musical chord sequences. In *IUI '09: Proceedings of the 14th international conference on Intelligent user interfaces*, ACM Request Permissions (Feb. 2009).
7. Nilson, C. Live coding practice. *Proceedings of the 7th international conference on New interfaces for musical expression* (2007), 112–117.
8. Sorensen, A., and Gardner, H. Programming with time: cyber-physical programming with impromptu. In *OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications* (2010), 822–834.