https://doi.org/10.5281/zenodo.7843896

Proceedings of the 7th International Conference on Live Coding (ICLC 2023)
April 19th - 23rd – Utrecht, Netherlands

Asymmetric Performance in Virtual Reality and Code

Leonard Geier
Hasso Plattner Institute, University of Potsdam
leonard.geier@student.hpi.uni-potsdam.de

Paul Methfessel
Hasso Plattner Institute, University of Potsdam
paul.methfessel@student.hpi.uni-potsdam.de

Tom Beckmann
Hasso Plattner Institute, University of Potsdam
tom.beckmann@hpi.uni-potsdam.de

Robert Hirschfeld
Hasso Plattner Institute, University of Potsdam
robert.hirschfeld@hpi.uni-potsdam.de

ABSTRACT

Virtual reality enables a rich 3D user experience where immediate feedback can yield lively interactions. For live coding, however, the rigid, text-based nature of source code is still a serious impediment to achieving such experience, as text input in virtual reality is significantly slower than on a physical keyboard.
We present an asymmetric live-coding environment, in which a performance can benefit from both the fluid and flexible direct manipulation capabilities of virtual reality and the expressive power of text-based code. Here, one performer interacts with parameters and code blocks of the system using their hands in virtual reality, but is ultimately constrained by the code defined by another performer immersed in a dedicated programming and runtime environment with full access to the source of the system. We present a proof-of-concept implementation of such a system and describe future directions for its development.

1 Introduction

A unique property of live coding is its deep and unlimited access to the computing resources for audio or visual production (TOPLAP 2005). However, as a result, the live coding performer is in charge of the entire spectrum of parameters that are involved in the performance, including scheduling or timing of notes.

In a proof-of-concept, we implemented a live coding environment where control over some parameters is given to a second performer that is in virtual reality. The other performer remains in a traditional ("flat") environment with access to a computer monitor and a mouse and keyboard. There, they define and edit code blocks that can emit MIDI events. The performer in virtual reality is in charge of activating or deactivating the code blocks by placing them in designated areas in the virtual space. Each area has different, pre-determined properties that the performer can experiment with. In addition, the distance of the code block from the ground controls its amplitude, thus allowing the performer to dynamically fade in sequences. This asymmetric performance, with one performer in a flat development environment and another in virtual reality, enables the performance to benefit from both the depth of expression enabled by live coding music and visuals with established tools, but also the rich and direct means of interaction and feedback that virtual reality affords.

As we want code that is currently playing to remain editable, our live coding environment reconciles changes from code to the living object system that is emitting MIDI notes through a virtual DOM-like approach as popularized by ReactJS.

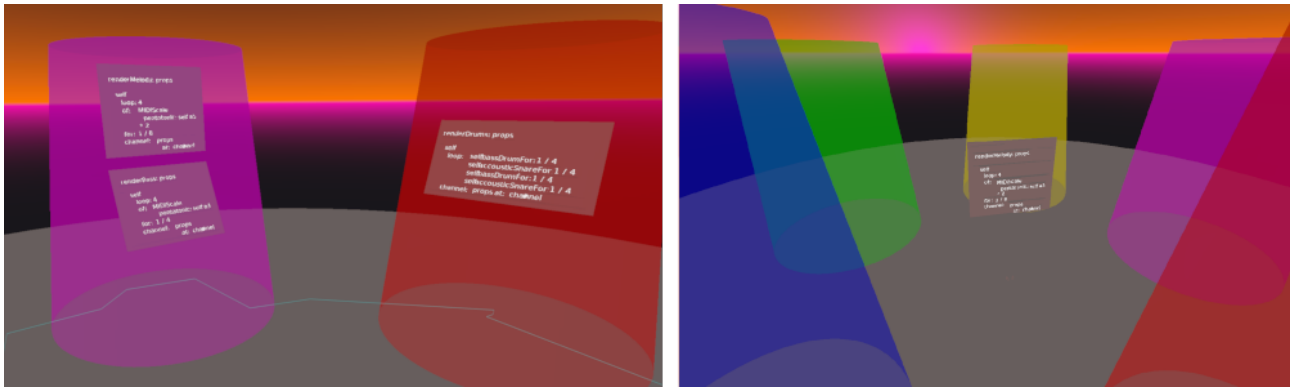

Figure 1: *Left: three active methods, assigned to two different instruments. Right: all five instruments, arranged in a circle. In the center a code block that is not yet assigned.*

This approach allows us to identify the fine-grained, minimal changes to the static source and apply them to the living object system without need to restart a sequence.

In this paper, we will first discuss related work, then describe a walkthrough of a performance in our prototypical live coding environment, as well as its properties and implementation, and finally outline future directions we think are interesting for asymmetric performance between virtual reality and traditional environments. Source code and a demo of the system are available on Github[1] under an open-source licence.

2 Related Work

The possibility of integrating virtual reality in the creation of programs (Elliott, Peiris, and Parnin 2015; Castillo et al. 2021; Geier et al. 2022) or for supporting art through code (Fischer 2016; Iannini 2016; Twomey et al. 2022) has been considered before. A majority of the work, however, still relies on textual editing in virtual reality, which is typically significantly slower without access to a mouse and keyboard. Through our approach, we aim to combine the speed at which performers can express new ideas in code but also the benefits of continuous input in virtual reality.

Similarly, the use of one's body to enhance a performance has been considered (Olaya-Figueroa, Zapata Cortés, and Nemocón 2019). In this work, audience members were able to influence the performance by moving on a physical mat and observing the result on projected screens or via the audio. Virtual reality has the potential to make the relation between one's actions and the outcome even clearer, as a full digital space provides continuous and potentially immediate feedback.

Systems based on or inspired by Croquet, a virtual, collaborative space (Smith et al. 2003), have been used in a variety of forms in the context of live coding. Most similar to our work, it was used to allow a performer to click on a set of objects in virtual reality, which in turn possessed behavior to trigger sounds (Suslov 2019). A performer outside of virtual reality was able to modify and extend the behavior that was triggered. The system, in principle, would thus be able to realize a similar setup as to the concrete setup we are presenting, where virtual code blocks act as a pluggable behavior when attached to other objects.

Our MIDI React system is one means to ensure a performance can continue uninterrupted even though the code defining the performance is being changed while it is running. It is thus similar to Sonic Pi's `live_loop`, or TidalCycles' patterns (Mclean 2014), and other approaches that enable live reloading. Functional reactive programming, to which ReactJS is related, has also been applied to live coding (Murphy 2016). Our approach aims to appear familiar to developers who know ReactJS and automates as much of the syncing issues as possible, presenting a less pure approach compared to functional reactive programming-based systems.

3 Walkthrough

In this section, we outline a walkthrough of a performance using our prototypical asymmetric live coding system.

To prepare, the performers launch a MIDI synthesizer on the host system. We used QSynth in our experiments. Then, we start our live coding environment. It consists of two parts: one, the Squeak/Smalltalk (Ingalls et al. 1997) live

[1] https://github.com/hpi-swa-lab/react-midi

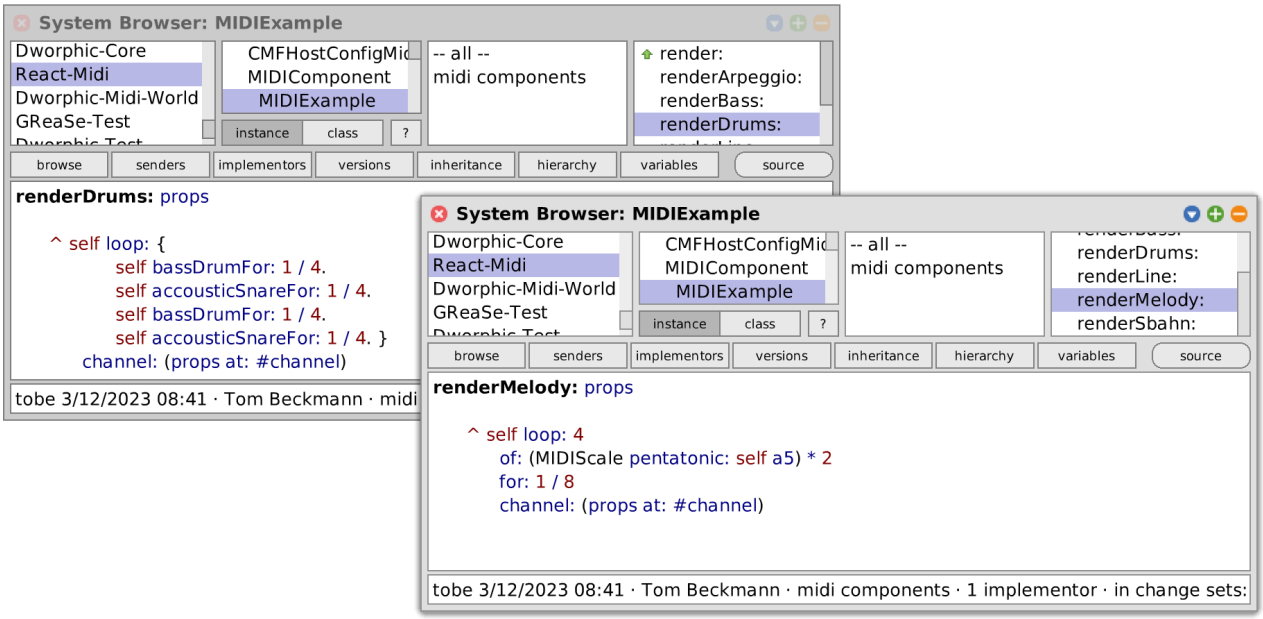

Figure 2: *The performer in the traditional, "flat" development environment has full access to pre-existing development tools. Here, some MIDI components are open for editing within the Squeak/Smalltalk interface.*

programming system (Rein et al. 2019), where one performer has access to the full source code driving the system. And second, a small kernel of code written for the Godot game engine (Linietsky, Manzur, and contributors 2014), which renders the visualizations defined on the host system. The Godot part is launched separately and either runs on the host system as well, to integrate with for example SteamVR, or on a standalone headset that is connected to the host system via USB cable.

The code driving the MIDI output is structured in separate methods that resemble ReactJS functional components, which we call MIDI components, for example:

```
renderOctave: props
    ^ self loop: {
            self note: self c4 for: 1 / 4.
            self note: self c5 for: 1 / 4.
            self note: self c4 for: 1 / 4.
            self restFor: 1 / 4. }
        channel: (props at: #channel)
```

Here, the performer has defined a loop of three-quarter notes and a rest. Note that MIDI channel, and thus the instrument, to which the loop is being sent is parametrized via the dictionary of properties this MIDI component receives.

The performer in virtual reality is presented with five cylinders representing different instruments, arranged in a circle around them. All MIDI components that the performer in the flat environment has defined appear clustered in the center of the circle as live-updating 3D code blocks. The performer in virtual reality can now pick up any code block using the virtual reality controllers and move them. As soon as a code block intersects one of the five cylinders, the MIDI component starts playing on the corresponding instrument.

As the performer in the flat environment introduces new MIDI components, they too spawn at the circle's center, such that the performer in virtual reality can pick them up and assign them to an instrument. By quickly moving a held code block in and out of a cylinder, the performer can let the code block activate for only one or multiple beats during the live performance.

To introduce a new sequence gradually, the performer in virtual reality can begin intersecting the cylinder closer to the floor. The lower quarter of each cylinder represents a 0% amplitude, the top quarter a 100% amplitude, and the center 50% allow a gradual increase or decrease of amplitude.

When the performer in the flat environment changes aspects of the sequences and saves the method, such as changing the pitch of a note, the change becomes instantly live in the system, so the next time the beat of that note is hit, it will play the adapted pitch. All loops are quantized against a global beats-per-minute value set by the performer in the flat

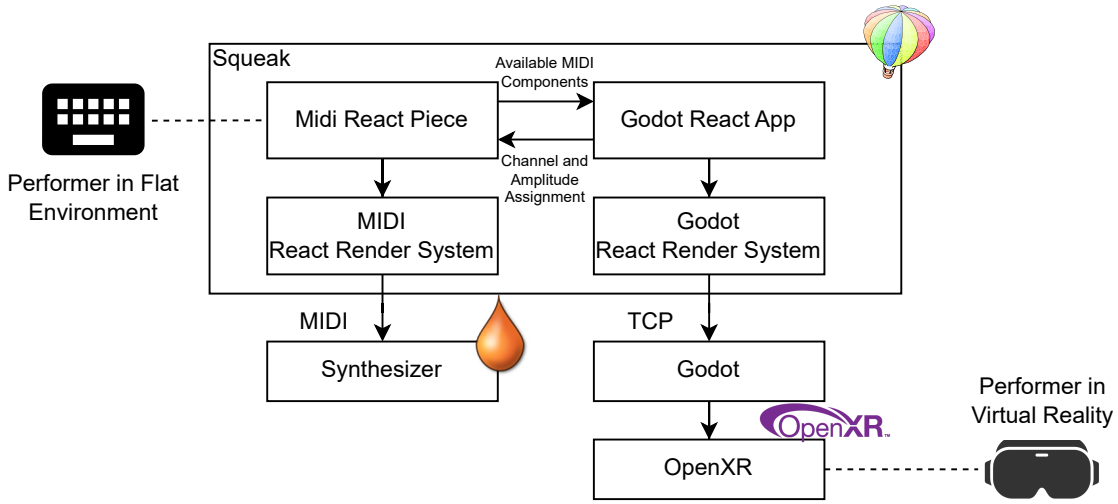

Figure 3: *Overview of our architecture. The piece being live performed is rendered via our MIDI React system and subsequently output to a synthesizer. The application rendering the virtual reality interface is defined in a Godot React app in Squeak, rendered via Godot, and output to OpenXR. The MIDI piece provides the VR application with the available components, while the VR application communicates the assignment of channels and amplitudes back to the piece, which is determined by the location of the components in virtual reality. One performer is interacting with the system via the VR interface, the other via a "flat" or desktop environment.*

environment, so changes do not lead to potentially annoying-to-fix timing issues, which may otherwise occupy most of the time of the performer in virtual reality who is in charge of timing and scheduling.

Throughout the performance, the performer outside of virtual reality essentially presents offers to the performer within virtual reality through the form of new loops but can also react to their decisions by modifying already running loops. The performer in virtual reality is constrained by the offers given but can decide how to weave them into the musical performance, perhaps previewing a MIDI component at a low amplitude or reading its code before deciding on when and on which instrument to play it.

4 Setup and Implementation

Three systems are collaborating to create performances in our prototypical live coding environment: Godot for visual output, a synthesizer such as QSynth for audio output, and Squeak/Smalltalk for controlling both. Both visual and audio output is controlled through a system inspired by ReactJS: Godot-React outputs Godot scene trees and MIDI-React outputs an object tree describing the currently living MIDI playback nodes.

4.1 MIDI React

As performers change the piece's code, we want all active aspects of the system to keep playing, if at all possible. To enable this, one approach is to execute code fragments throughout the performance and save the system state in objects. In this example from SuperCollider, a synth definition exposes properties that can be modified at a later point to influence the synth's behavior while it is running:

```
(
SynthDef("control", { arg freq1 = 440, freq2 = 443, mul = 0.12;
    Out.ar(0, SinOsc.ar([freq1, freq2], 0, mul))
}, [4, 5]).add;
)
~aSynth = Synth("control", [\freq1, 550, \freq2, 344, \mul, 0.1]);
~aSynth.set(\freq1, 600, \freq2, 701, \mul, 0.05);
```

As another alternative, seen for example in Sonic Pi, performers define a `live_loop` that every time it reaches the end of its defined sequence may be updated with a changed definition.

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8
  sleep 8
end
```

In contrast, in our React MIDI, all objects are created declaratively: all objects defined in code are always instantiated. Objects that are no longer declared are automatically removed. The visible code as such always represents the state of the system, unlike the above two examples where statements are executed, apply their effect, and can in principle now be forgotten.

Performers can influence the system state in React MIDI by changing the code and saving, at which point the operations to make their change, and only their exact change, live in the system. For example, changing the pitch in a loop will not restart the loop, it will only modify the pitch of the note. Via data bindings, performers can have the system state evolve without modifying constants. In the below example, an arpeggio is played starting from MIDI note c4 and progressing chromatically to a higher pitch each measure.

```
renderArpeggio: props

    | base |
    base := self useState: self c4.
    ^ (self loop: {
            self note: base get + 0  for: 1 / 8.
            self note: base get + 4  for: 1 / 8.
            self note: base get + 7  for: 1 / 8.
            self note: base get + 12 for: 1 / 8.
            self note: base get + 7  for: 1 / 8.
            self note: base get + 4  for: 1 / 8. }
        channel: (props at: #channel)) onMeasureStart: [base set: [:previous | previous + 1]]
```

In the example, just before the start of each new measure, the notes in the running loop will all be assigned a new pitch; everything else remains unchanged.

MIDI React creates and maintains an object tree that encapsulates the scheduling of MIDI note and control outputs. Our prototype provides users with four primitive classes that they can combine and program against:

1. A `Player` object forms the root of the object tree. It controls the global beats-per-minute value and sends beat and measure information to all its child nodes.

2. A `Note` object contains a duration, velocity, and pitch. It maps to corresponding MIDI noteOn/noteOff events. Durations are given as fractions of a measure. A note with pitch `0` acts as a rest.

3. A `Loop` object takes sequences of note objects as its children and plays them in sequence based on their duration. In addition, it is assigned a channel to which the notes should be sent.

4. A `Control` object sends a MIDI control event as part of a loop. Beyond that, it acts as a note with a zero duration.

As in ReactJS, after every state or code change, a description of the concrete structure of this object tree as described by the code is produced. This description is then compared to the previous description to determine the minimal changes that need to be applied to the actual living object tree. As a result, most programming errors will leave the living object tree untouched and allow the system to keep playing using the last valid description. A proper restart of the system, which is near instantaneous but will reset the clock, is only required if a programming error corrupted the system's state: the error did not manifest when building the object tree description but invalid data was inserted in some property of the description, resulting in a corrupted live system.

4.2 Virtual Reality Controls

The controls for the performer in virtual reality are realized through standard facilities of the Godot game engine, with little extra code for arranging objects, adding interactivity, and displaying code blocks. Code blocks display the up-to-date source code as also seen in the Squeak programming system. We make them interactive by allowing the performer to grab them by pressing the VR controller's grab button.

The instruments are rendered through translucent cylinders, which inform our system of intersections with the code blocks. When an intersection is detected, the MIDI component corresponding to the intersecting code block is assigned the channel of the instrument in the MIDI React system. In addition, to provide feedback to the performer, we let the block pulse in sync with the beats-per-minute to show that the code block is now active.

5 Conclusion and Future Work

In our tests, the prototype showed promise: the asymmetry of the setup allowed one person to concentrate on creating low-level details of a composition in the form of the MIDI sequences, whereas the other was in control of the larger picture, by scheduling provided sequences at different times. There are multiple aspects we believe are interesting for future investigation.

At the moment, the instruments the circles are playing are static. Instead, they could either be defined in code or evolve automatically throughout a performance. For example, the system could randomize the parameters of the instruments at a specific interval, prompting the performers to adapt their performance. This may in particular require the performer in virtual reality to adapt quickly to maintain a coherent soundscape.

In the prototype as described, the performer in virtual reality benefits primarily from a fast gesture for toggling and assigning instruments to musical fragments. As position data in virtual reality provides continuous, analog values, it would be interesting to not only map these to amplitude as in our prototype but for example, allow the performer in virtual reality to edit constants in the code by selecting them and moving the controller. Thanks to our MIDI React system, the changes would instantly become live, so it may for example allow the performer to slide around the cutoff frequency of a low pass filter or any other values that are hardcoded.

Finally, the MIDI React system in its current state is limited by the few primitives we have defined for the prototype. By taking inspiration from more mature systems for sequencing notes such as sclang, overtone, or pure-data we think that it may be possible to identify a more flexible and powerful set of primitives.

References

Castillo, Víctor Stefano Segura, Leonel Merino, Geoffrey Hecht, and Alexandre Bergel. 2021. "VR-Based User Interactions to Exploit Infinite Space in Programming Activities." *2021 40th International Conference of the Chilean Computer Science Society (SCCC)*, 1–5.

Elliott, Anthony, Brian Peiris, and Chris Parnin. 2015. "Virtual Reality in Software Engineering: Affordances, Applications, and Challenges." In *Proceedings of the Conference on Software Engineering (ICSE) 2015*, 2:547–50. ICSE '15. Piscataway, NJ, USA: IEEE Press. https://doi.org/10.1109/ICSE.2015.191.

Fischer, Michael. 2016. "Inception: A Creative Coding Environment for Virtual Reality, in Virtual Reality." In *Proceedings of the ACM Conference on Virtual Reality Software and Technology (VRST) 2016*, 339–40. VRST '16. New York, NY, USA: ACM. https://doi.org/10.1145/2993369.2996354.

Geier, Leonard, Clemens Tiedt, Tom Beckmann, Marcel Taeumel, and Robert Hirschfeld. 2022. "Toward a VR-Native Live Programming Environment." In *Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments*, 26–34. PAINT 2022. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3563836.3568725.

Iannini, Luke. 2016. "Rumpus." https://store.steampowered.com/app/458200/Rumpus/. https://store.steampowered.com/app/458200/Rumpus/.

Ingalls, Dan, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. 1997. "Back to the Future: The Story of Squeak, a Practical Smalltalk Written in Itself." In *Proceedings of the 12th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 318–26. OOPSLA '97. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/263698.263754.

Linietsky, Juan, Ariel Manzur, and Godot Engine contributors. 2014. "Godot." https://godotengine.org/. https://godotengine.org/.

Mclean, Alex. 2014. "Making Programming Languages to Dance to: Live Coding with Tidal." In *FARM 2014 - Proceedings of the 2014 ACM SIGPLAN International Workshop on Functional Art, Music, Modelling and Design*. https://doi.org/10.1145/2633638.2633647.

Murphy, Tom E. 2016. "A Livecoding Semantics for Functional Reactive Programming." In *Proceedings of the 4th International Workshop on Functional Art, Music, Modelling, and Design*, 48–53. FARM 2016. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2975980.2975986.

Olaya-Figueroa, Juan, Laura Zapata Cortés, and Camilo Nemocón. 2019. "Full-Body Interaction for Live Coding." In. https://doi.org/10.5281/zenodo.3946303.

Rein, Patrick, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2019. "Exploratory and Live, Programming and Coding - A Literature Study Comparing Perspectives on Liveness." *The Art, Science, and Engineering of Programming* 3 (1): 1. https://doi.org/10.22152/programming-journal.org/2019/3/1.

Smith, D. A., A. Kay, A. Raab, and D. P. Reed. 2003. "Croquet - a Collaboration System Architecture." In *First Conference on Creating, Connecting and Collaborating Through Computing, 2003. C5 2003. Proceedings.*, 2–9. https://doi.org/10.1109/C5.2003.1222325.

Suslov, Nikolai. 2019. "LiveCoding.space: Towards P2P Collaborative Live Programming Environment for WebXR." In. https://doi.org/10.5281/zenodo.3946356.

TOPLAP. 2005. "ManifestoDraft." http://toplap.org/wiki/ManifestoDraft. http://toplap.org/wiki/ManifestoDraft.

Twomey, Robert, Tommy Sharkey, Timothy Wood, Amy Eguchi, Monica Sweet, and Ying Choon Wu. 2022. "An Immersive Environment for Embodied Code." In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA '22. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3491101.3519896.