Collaborative Live-Coding Virtual Worlds with an Immersive Instrument

Graham Wakefield
KAIST
Daejeon, Republic of Korea
grrrwaaa@gmail.com

Charlie Roberts
University of California
Santa Barbara, USA
charlie@charlie-roberts.com

Matthew Wright
University of California
Santa Barbara, USA
matt@create.ucsb.edu

Timothy Wood
University of California
Santa Barbara, USA
fishuyo@gmail.com

Karl Yerkes
University of California
Santa Barbara, USA
yerkes@mat.ucsb.edu

ABSTRACT

We discuss live coding audio-visual worlds for large-scale virtual reality environments. We describe *Alive*, an instrument allowing multiple users to develop sonic and visual behaviors of agents in a virtual world, through a browser-based collaborative code interface, accessible while being immersed through spatialized audio and stereoscopic display. The interface adds terse syntax for query-based precise or stochastic selections and declarative agent manipulations, lazily-evaluated expressions for synthesis and behavior, event handling, and flexible scheduling.

Keywords

Live coding, collaborative performance, immersive instrument, multi-agent system, worldmaking

1. INTRODUCTION

Over the last decade live musical practice has demonstrated the value of run-time malleable code for improvisation, bringing concepts formerly associated with studio composition and software development into the realm of live performance [2]. We are inspired to extend this practice to collaboratively authoring virtual worlds on-the-fly within immersive environments.

How can we code worlds while we are immersed within them? What combinations of constraints and affordances will best serve performers with mere minutes to bring forth engaging worlds? To address these challenges we propose and evaluate *Alive*, a multi-user browser-based editing interface presenting a multi-paradigm programming model. We emphasize flexibility and terseness while manipulating an agent-based simulation model with distributed spatial audio and stereoscopic rendering.

The improvisatory nature of live coding calls for a dynamic approach to specification and modification of the world. We use data-oriented concepts of entities (*agents*), associations (*tags*), selections (*queries*) and behaviors (*expressions*) in code fragments that may be manually triggered or scheduled for future execution using coroutines.

The project is informed by our work as researchers in the AlloSphere, a three-story, immersive, spherical virtual reality environment at the University of California, Santa Barbara [12]. Performers stand alongside the audience on a bridge suspended through the center of the sphere that can comfortably accommodate thirty people, surrounded by over fifty loudspeakers and twenty-six stereoscopic projectors, to create a powerful immersive experience (Figure 1).

Although one specific venue inspired *Alive*'s development, it is open-source[1] and may be easily applied to other facilities.

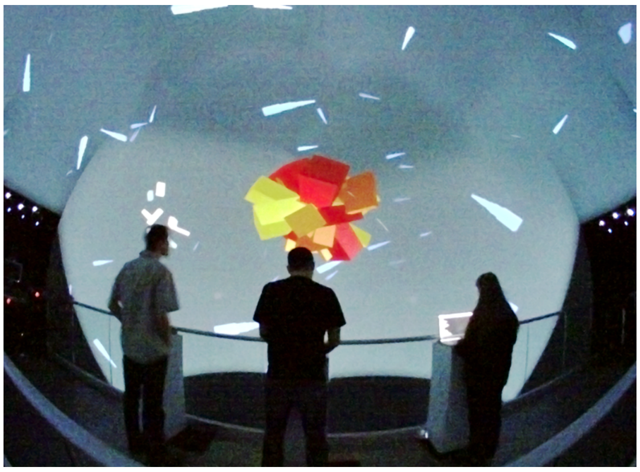

Figure 1: Fish-eye photograph of three performers standing on the bridge of the AlloSphere and live coding the world that surrounds them.

2. RELATED WORK

Computer music software has been variously extended [19] or coupled with game engines [5] for immersive compositions and installations, but we have found little evidence of live-coding immersive performance. The SmallTalk-derived SuperCollider audio programming language is extensible via SCGraph[2], which extends the unit generator concept to modulate 3D OpenGL primitives. This could be used to live code immersive systems.

Extempore[3] and LuaAV [20] are general systems for authoring efficient audio-visual applications, and also support live coding. Both have been used for networked interac-

[1]`github.com/AlloSphere-Research-Group/alive`
[2]`github.com/scgraph/SCGraph`
[3]`extempore.moso.com.au`

tive multi-display installations, such as Extempore in "The Physics Playroom" [16], and LuaAV in "Artificial Nature" [11]. In both cases live coding was used during design and development, but not for improvised performance.

Fluxus is a graphics-oriented live coding environment and "a 3D game engine for livecoding worlds into existence" [10]. Fluxus is extensible via the Fluxa[4] module for audio synthesis and sequencing. In 2009, performance groups Slub and Wrongheaded used Fluxus and other audio software to perform inside the Immersion Vision Theater, a former planetarium. Dave Griffiths mapped projections via a fisheye lens to cover the planetarium's screen[5], in what may have been the world's first immersive live coding performance.

Gibber [17] and Lich.js[6] are browser-based platforms for multi-user live coding that support both audio and graphics. LOLC [7] enables performers to share code snippets to iterate ideas within an ensemble. Republic [4] players write SuperCollider scripts to generate sounds distributed across all machines, using a chat system for social intervention.

3. DESIGN AND IMPLEMENTATION

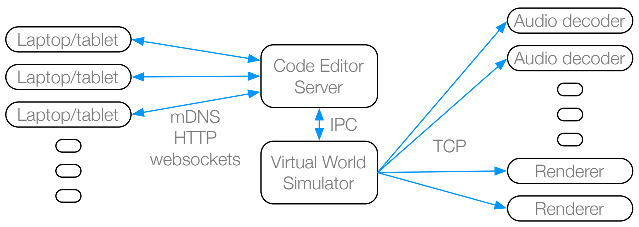

Figure 2: Architectural overview. Client browsers on performers' laptop or tablet devices (left) display code editors requested from a central server (center). Edits to the code are sent to the server and shared back to all clients. The server hosts a virtual world simulator as a sub-process, to which code edits are forwarded for execution. The simulator distributes virtual world updates to all audio and visual rendering machines (right).

Performers use web browser clients to retrieve the live-coding interface from a server application on the local network. The server forwards all code fragments received from clients to a simulation engine by interprocess communication. The simulation engine continuously updates a virtual world of mobile, audio-visual agents via subsystems of movement, synthesis, collision, and event propagation, then distributes the resulting world state to all audio and visual rendering machines installed in the venue (Figure 2). (The interface, server and simulator can also be run on a single computer for solo practice.)

3.1 The Live-Coding Interface

The *Alive* code interface runs in any modern web browser, communicating with the server application by websockets. Performers can add code fragments to the editor pane, and send the currently-selected code for execution by pressing the `Command+Enter` keys or double-clicking. If no text is selected, the whole paragraph containing the cursor is sent. This makes it very easy to rapidly trigger prepared code fragments. A typical performance is a mixture of authoring, invoking, modifying, and copying fragments.

To encourage collaborative interplay, each performer shares the same global namespace and also sees and edits the same live "document" of code. Edits sent to the server are merged and updates sent back to clients. The shared document thus collects all code produced by the group performance, including potentially re-usable code fragments.

The interface also includes a console pane reporting commands executed (in white) and error messages (in red) from the server. Another pane presents reference documentation along with short copyable code fragments to assist rapid experimentation.

3.2 The Virtual World

The principal elements or entities of the world are *agents*. This term is variously defined in scientific simulation, computer science, robotics, artificial life and game design [23, 6], but general to most multi-agent systems is the concept of mobile autonomous processes operating in parallel.

Our agents are transient, spatially-situated identities to which users associate properties and behaviors. Easily constructed and destroyed, their properties can be manipulated directly by assignment or indirectly through the use of "tags" and "queries" (described below). Properties of a particular agent might include sonic and visual appearance as well as behaviors of movement, morphology, and reactivity. Using an associative scheme side-steps constrictive issues of categorization in favor of dynamic action of wide applicability [21].

The world has a 3D Euclidean geometry, but is finite and toroidal. Autonomous agents can easily navigate away off into the distance, but a toroidal space ensures that movement is never limited nor activity too far away. It is implemented such that no edges to the world are ever perceived, despite free navigation.

3.3 Audiovisual Rendering

The simulation engine implements a pragmatic approach to spatial audio not dissimilar to [14] and [19]. The sounds of each agent are processed to apply distance cues, and their directional information is encoded using higher-order ambisonics. Ambisonic encoding/decoding supports scalability in distributed rendering: unlimited numbers of agent sounds are encoded into just a handful of domain signals, which can be more easily distributed to multiple decoders with scalability up to hundreds of loudspeakers. A per-agent delay, indexed proportionally to distance, simulates Doppler shift. Sounds are also attenuated and filtered according to distance-dependent curves.

Distributed visual rendering (required to drive large numbers of projectors) depends on updating all visual rendering machines with the minimal state to render the scene. Each machine renders a stereoscopic view of the world properly rotated and pre-distorted for its attached projectors, resulting in a coherent immersive world seamlessly spanning the venue [12].

4. LANGUAGE INTERFACE DESIGN

Best practices of language and interface design for live coding differ significantly from general software application development [1]. Typical application code is verbose and extensively commented since it is more often read than written [9], but in a live scenario every keystroke counts. For creating whole worlds in real-time, low granularity of control could easily be overwhelming.

Additionally, where application development uses constraints such as strong typing to ensure correctness, in live

[4]en.flossmanuals.net/fluxus/ch027_fluxa/
[5]www.listarc.bham.ac.uk/lists/sc-users-2009/msg57333.html
[6]github.com/ChadMcKinney/Lich.js

performance it is better to interpret permissively than throw errors; accidents can have creative value! Ephemeral live coding values the multi-paradigm, ad-hoc, and in-place "hacks" that application developers try to avoid [15, 3].

In line with these insights, our interface extends the Lua programming language (featuring a highly flexible data structure, dynamic typing, re-entrancy, first-class functions, coroutines, and automatic memory management) with terse yet flexible abstractions for live-coding agent-based worlds.

4.1 Properties and tags

Agents have various properties, represented using Lua's associative dictionary tables. Some property names have specific built-in semantics for the simulation engine, including amplitude ("amp"), forward and angular velocity ("move" and "turn"), instantaneous forces ("push" and "twist"), color and scale, visibility and presence ("visible" and "enable"), as well as some read-only properties such as unit vectors of the coordinate frame ("ux", "uy", "uz") and nearest neighbor ("nearest"). Users can add other arbitrary property names and values as desired.

For managing multiple agents we drew inspiration from HTML/CSS web technologies, which afford a sophisticated system for declarative configuration of groups and individual document elements.[7] Each agent can also be associated with one or more "tags", akin to the classes of CSS. Tags are dynamically modifiable associative tables of properties. Tags serve as templates for new agents: once a tag has properties defined, any new agents created with that tag will be initialized with these properties. Tags can be added to and removed from agents dynamically:[8]

```
-- create an agent associated with two tags:
a = Agent("foo", "bar")
-- modify the foo tag (and thus all "foo" agents):
Tag.foo.amp = 0.5
-- modify the tags the agent associates with:
a:untag("bar")
a:tag("baz")
```

4.2 Queries

In addition to operating on individual agents and the various tags in play, performers can operate over arbitrary selections of active agents. Queries serve a role similar to relational database queries or lenses and patterns in functional programming, and are partly inspired by the expressive jQuery[9] web technology.

The `Q()` function constructs a query, with optional arguments indicating a set of tags to start filtering; if no tags are given, all agents are included. Chained methods refine the filter predicate, such as `first()` and `last()` to select the earliest and most recent members of the collection, `has()` to select members with specific property names or values, and `pick()` to select random sub-collections. The result of a query can be cached and re-used. Query results are used to call methods or set properties on all members:

```
-- set "amp" of the most recent foo-tagged agent:
Q("foo"):last().amp = 0.3
-- terminate all agents with a "chorus" property:
Q():has("chorus"):die()
-- set the frequency of about 50% of all agents:
Q():pick(0.5).freq = 200
-- set "freq" of four random "spin" agents:
Q("spin"):pick(4).freq = 500
-- stop one randomly chosen agent from moving:
Q():pick():halt()
```

4.3 Expression Objects

Our instrument supports a declarative form of function definition through objects similar to unit generators[10] and LISP M-expressions.

A declared expression object represents a computation to be performed. This expression object can be assigned to properties of agents, tags, and queries, as well as performing as a unit generator graph for sound synthesis (building upon prior work in [20]). Our expression constructors can accept numbers or expressions as arguments. They also accept strings, which are used to index properties of agents, and utilize operator overloading to support a more readable syntax:

```
-- construct an expression:
e = (Max(Random(10), Random(10)) + 2) * 100
-- assign to "mod" for all agents tagged "foo"
-- (each receives distinct evaluations of Random)
Q("foo").mod = e
```

Many common math operations and signal processing behaviors are provided as standard. Some expression constructors, such as oscillators and filters, evaluate to stateful behaviors (functions of time) rather than atomic values. A distinct instance of the behavior is created for each agent property assignment, and invoked continuously until replaced or destroyed. Behaviors thus bridge unit generator and agent concepts with an easily composable model for live programming that is terser than function definition.

4.4 Procedural Time and Events

In addition to immediate invocations, our instrument provides a means to schedule the execution of procedural code via *coroutines*, largely borrowed from [20]. Coroutines are akin to functions that may be paused in the middle and later resumed at that point, allowing other coroutines to run in between. We extend the yield/resume semantics with sample-accurate scheduling and event handling. With `wait()` a coroutine can yield for a specified time in seconds, or until a specified event occurs, identifed by a string and triggered from anywhere via `event()`. (Agents can also use the `on()` method to define callback handlers for asynchronous named events.) Coroutines can embed loops and conditions to structure time and behavior, and return by calling themselves or other functions to implement temporal recursion.

5. PERFORMER EVALUATION

Several performers, all with prior experience in computer music and immersive systems, reported on their experiences with the instrument via questionnaire.

Performers described the system as playful or fun, and commented positively on immediate "liveness", one describing it as "the first time that I've felt that the coding wasn't getting too much in the way of the experience." Several players also described it as "stressful" or "frustrating," commenting on the difficulties of realizing an imagined complex behavior. All performers saw good potential and expected that further practice would be very rewarding. Desirable extensions mentioned included higher-level behaviors such as physics simulation, and richer agent geometries, to combat the limited time available in a performance. Performers

[7]See www.w3.org/TR/html5 and www.w3.org/TR/CSS.

[8]If an agent has multiple tags with values for a property, the last (most recent) tag applied determines the value.

[9]jquery.com

[10]Unit generators [13] essentially offer a practical, declarative form of single-valued function (or closure) definition.

also suggested adding capabilities to the editor such as autocompletion and visual representations of active agent/tag associations. However the most frequent request was the provision for virtual interfaces and physical sensors for continuous control.

In contrast to live-coding systems such as Fluxus that superpose the code visually over the animated graphical content, the distributed nature of our system effectively divides attention between the editor window and the surrounding immersive environment. Performers reported dividing attention fairly evenly between modes of observing the world, authoring code, and observing/debugging code; however maintaining focus in the transition between world and editor was difficult. One performer concentrated on the editor to develop an interesting algorithm first, then spent more time observing the world while repeatedly triggering small variations of this algorithm.

6. DISCUSSION

The way in which an algorithm can be represented impacts its utility in live performance. Although Brown and Sorensen suggest that agent-based systems are inappropriate for live coding due to description complexity, we propose that our multi-paradigm system can overcome this barrier, satisfying many of the beneficial characteristics of live coding languages they identified, including wide applicability of primitives, efficiency, dynamism, and event scheduling, while tersely expressed behaviors remain open to modification under human direction.[11]

Our agents and tags differ from HTML styles in that they do not support nested hierarchical structures. Grouping structures and hierarchies are conceivable, such as scenegraphs and prototype inheritance schema, or perhaps less hierarchical agent-assemblages with more open-ended intermodulation or mutual influence.

Editing a virtual world while it runs occurs frequently in the realm of game development [8], and recently the use of Just-In-Time compilation has broadened the scope of in-game code editing.[12] The authors have been exploring similar techniques [18] and are certain that the future of live-coding audio-visual immersive worlds has fascinating unexplored potential.

Acknowledgements

Graciously supported by The AlloSphere Research Group, the R. W. Deutsch Foundation, and NSF grant IIS-0855279.

7. REFERENCES

[1] A. R. Brown and A. Sorensen. Interacting with generative music through live coding. *Contemporary Music Review*, 28(1):17–29, 2009.

[2] N. Collins, A. Mclean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(03):10, Dec. 2003.

[3] G. Cox, A. Mclean, and A. Ward. Coding Praxis: Reconsidering the Aesthetics of Code. *Goriunova, Olga and Shulgin, Alexei (ed.): readme–Software Arts and Cultures, Edition*, pages 160–175, 2004.

[4] A. de Campo. Republic: Collaborative live coding 2003–2013. In A. Blackwell, A. McLean, J. Noble, and J. Rohrhuber, editors, *Collaboration and learning through live coding*, Report from Dagstuhl Seminar 13382, pages 152–153. Dagstuhl, Germany, 2013.

[5] M. Dolinsky and E. Dambik. Ygdrasil—a framework for composing shared virtual worlds. *Future Generation Computer Systems*, 2003.

[6] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Lecture Notes In Computer Science*, 1997.

[7] J. Freeman and A. Troyer. Collaborative textual improvisation in a laptop ensemble. *Computer Music Journal*, 35(2):8–21, 2011.

[8] N. Frykholm. Cutting the pipe. Presented at GDC Game Developers Conference 2012, 2012.

[9] A. Goldberg. Programmer as reader. *Software, IEEE*, 4(5):62 –70, sept. 1987.

[10] D. Griffiths. Fluxus. In A. Blackwell, A. McLean, J. Noble, and J. Rohrhuber, editors, *Collaboration and learning through live coding*, Report from Dagstuhl Seminar 13382, pages 149–150. Dagstuhl, Germany, 2013.

[11] H. Ji and G. Wakefield. Virtual world-making in an interactive art installation: Time of Doubles. In S. Bornhofen, J.-C. Heudin, A. Lioret, and J.-C. Torrel, editors, *Virtual Worlds*. Science eBook, 2013.

[12] J. Kuchera-Morin, M. Wright, G. Wakefield, C. Roberts, D. Adderton, B. Sajadi, T. Höllerer, and A. Majumder. Immersive full-surround multi-user system design. *Computers & Graphics*, 2014.

[13] M. Mathews. An acoustic compiler for music and psychological stimuli. *Bell System Technical Journal*, 1961.

[14] M. Naef. Spatialized Audio Rendering for Immersive Virtual Environments. In *VRST 02 Proceedings of the ACM symposium on Virtual Reality software and technology*, pages 1–8, May 2002.

[15] R. Potter. Just-in-time programming. In *Watch What I Do: Programming by Demonstration*, pages 513–526. MIT Press, Cambridge, MA, USA, 1993.

[16] M. Rittenbruch, A. Sorensen, J. Donovan, D. Polson, M. Docherty, and J. Jones. The Cube: A very large-scale interactive engagement space. In *Proceedings of the 2013 ACM intl. conf. on Interactive tabletops and surfaces*, pages 1–10. ACM, 2013.

[17] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proc. ICMC*, 2012.

[18] G. Wakefield. *Real-Time Meta-Programming for Open-Ended Computational Arts.* PhD thesis, University of California Santa Barbara, 2012.

[19] G. Wakefield and W. Smith. Cosm: A toolkit for composing immersive audio-visual worlds of agency and autonomy. *Proc. ICMC*, 2011.

[20] G. Wakefield, W. Smith, and C. Roberts. LuaAV: Extensibility and Heterogeneity for Audiovisual Computing. *Proceedings of Linux Audio Conference*, 2010.

[21] P. Wegner. Why interaction is more powerful than algorithms. *Commun. ACM*, 40(5):80–91, May 1997.

[22] I. Whalley. Software agents in music and sound art research/creative work: current state and a possible direction. *Organised Sound*, 14(2):156–167, 2009.

[23] M. Wooldridge and N. Jennings. Agent Theories, Architectures, and Languages: A Survey. *Intelligent Agents*, 1995.

[11]However, our use of the term *agent* does not encompass all the possible meanings of software agents for music and sound art [22].

[12]Such as `github.com/RuntimeCompiledCPlusPlus/RuntimeCompiledCPlusPlus` and the hot reload feature of `unrealengine.com/unreal_engine_4`