DIRTY Dancing

Live Coding Infrastructure for *Distributed Interactive Real-Time Yamming*

Martin Carlé mc@aiguphonie.com

Iannis Zannos zannos@gmail.com

Abstract

DIRTY Dancing is a modular, open-source framework for live-coded animation that integrates real-time sound synthesis and motion capture data exchange using the Open Sound Control (OSC) and Virtual Motion Capture (VMC) protocols. The framework connects SuperCollider (sound synthesis) and Godot (visual rendering) to enable bidirectional, real-time interaction between motion and sound. Unlike conventional animation pipelines that treat motion, sound, and rendering as separate sequential stages, DIRTY Dancing supports concurrent, code-based manipulation of all three within a single networked environment. The paper describes the system architecture, data flow, interaction design, and mapping strategies, and demonstrates the system through three use-case scenarios: solo performance, networked collaboration, and animation authoring. Initial evaluation includes latency measurements from a transatlantic session (Athens–Tokyo, ~100 ms round trip) and qualitative feedback from early participants. Results indicate that the system supports responsive audiovisual interaction and enables new forms of collaborative animation practice, while also revealing current limitations in accessibility for non-technical users and in performance scalability. The framework is implemented and functional; directions for further evaluation and development are discussed.

Keywords: Live coding, Real-time animation, Motion capture, Open Sound Control (OSC), Virtual Motion Capture (VMC), Distributed performance, Embodied media art, Collaborative authoring.

1. Introduction

1.1 Overview

This paper presents DIRTY Dancing (Distributed Interactive Real-Time Yamming), a framework for real-time, networked animation driven by live coding. The system enables artists and developers to manipulate motion capture data and coordinate sound synthesis through code during performance or rehearsal, treating animation as an ongoing process of exploration rather than a fixed output.

The project draws on the VTubing paradigm, in which motion capture and real-time rendering allow performers to animate avatars live for broadcast or recording, and extends it through a custom OSC interface compatible with the VMC protocol. This links the open-source environments of SuperCollider and Godot, allowing creators to connect sound, movement, and animation logic through a shared data layer.

The remainder of this section situates the work in its research context (§1.2), describes the software framework (§1.3), and states the paper's specific contributions (§1.4). Sections 2–4 describe the underlying technologies, interaction design, and system architecture respectively.

Section 5 presents use cases and initial evaluation. Section 6 discusses implications and limitations, and Section 7 concludes.

1.2 Research Context

DIRTY Dancing sits at the intersection of three active research areas: live coding, embodied interactive performance, and real-time animation.

Live coding has been established as both a compositional method and a performative practice in which programs are written and modified in real time, often in front of an audience (Collective, 2004; Collins et al., 2003; Magnusson, 2011; McLean & Wiggins, 2010). Key features of the practice—immediacy, transparency, and continuous feedback between code and perception—translate productively into the temporal domain of animation. Ward (2011) frames live coding as an inherently open-ended practice resisting the closure of conventional software production, a position our framework explicitly extends into the animation context. The TOPLAP manifesto (Collective, 2004) articulates the ethical and aesthetic stakes of this approach, emphasizing audience visibility of the generative process— a principle we adopt in the design of our interaction layer.

Embodied interactive performance and motion-capture art provide a second grounding. Birringer (2008) and Schiphorst (2009) have demonstrated how movement data can serve as a generative substrate for artistic expression, dissolving the boundary between performer body and computational system. More recent work in full-body interaction (Fdili Alaoui, 2019; Peng et al., 2023) has formalized design principles for systems that couple gestural input with real-time audiovisual output, highlighting the importance of low-latency feedback and expressive mapping transparency. Research on sonic interaction design (Rocchesso et al., 2008; Delle Monache & Rocchesso, 2016) further informs our mapping strategies, offering principled frameworks for relating physical motion parameters to auditory output. These frameworks guide the three mapping types implemented in DIRTY Dancing (see §2.3).

Real-time animation and VTubing ecosystems constitute a third lineage. The VMC protocol (Contributors, 2024) has emerged as an open standard for streaming skeletal joint data from motion capture hardware to rendering engines, enabling the live avatar performances that characterize VTuber content (Ferreira et al., 2022; Popescu et al., 2021; Zhao et al., 2025). Khor et al. (2025) survey integration approaches combining motion capture monitoring with extended reality (XR) applications, and Tira (2023) documents practical VMC integration workflows. Our framework reuses this infrastructure while extending it: SuperCollider intercepts VMC data streams and, rather than passing them directly to a renderer, subjects them to programmable transformations and sound mappings before forwarding them to Godot.

Accessibility and interface design for creative coding tools is a fourth area of relevance, and one where limitations must be acknowledged. Comparative work on creative coding environments (Scaffidi et al., 2005; Resnick et al., 2009) distinguishes between tools designed for expert programmers and those targeting broader creative communities. DIRTY Dancing, in its current implementation, requires familiarity with both SuperCollider and Godot scripting; it does not yet provide a high-level graphical interface for non-programmer users. The browser-based configuration partially reduces this barrier by eliminating installation requirements but does not remove the need for coding literacy. This limitation is discussed in §6.

Evaluation of interactive performance systems raises methodological challenges addressed in the broader literature. Barbosa (2003) surveys networked music systems, identifying latency

thresholds and interaction quality as key evaluation dimensions. Wanderley & Depalle (2004) offer a taxonomy of gestural control in music, portions of which transfer to motion-driven animation. O'Modhrain (2011) proposes evaluation criteria for musical interfaces that balance technical performance metrics with perceptual and artistic quality assessments. We adopt a lightweight version of this framework in §5.3.

1.3 Software Framework

The DIRTY Dancing framework was designed as a modular and extensible system enabling the integration of live-coded sound synthesis, real-time animation, and motion capture data exchange across open platforms. Its architecture connects the Godot game engine (Godot Engine Community, 2025) with SuperCollider (McCartney et al., 2011) through the OSC protocol (Wright & Freed, 1997). This setup enables bidirectional communication between audio and visual domains and supports both live streaming of motion data from sensors and playback of pre-recorded motion sequences. Figure 1 shows the splash screen of the Godot application, which runs on Linux, Microsoft Windows, macOS, and in web browsers via WebAssembly.

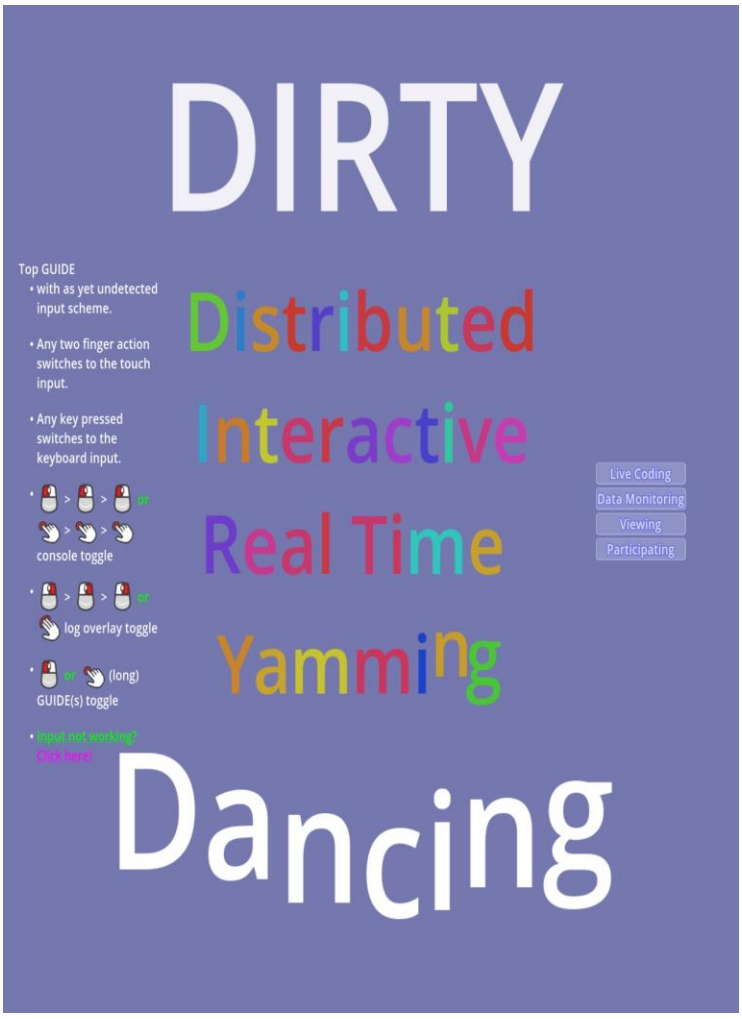


Figure 1: Generic UI Splash Screen on GODOT, showing access to four operation modes

1.4 Contributions

This paper makes the following specific contributions:

1. A **modular architecture** integrating SuperCollider and Godot via OSC and VMC for bidirectional, real-time coupling of motion data and sound synthesis.
2. A **live coding infrastructure for animation**, enabling real-time manipulation of motion capture data through programmable transformations during performance.
3. A **distributed performance model** supporting multi-user co-creation via WebRTC and OSC over wide-area networks.
4. Three **mapping strategies** (direct, derivative, and conditional) for motion–sound interaction, providing a structured vocabulary for expressive
5. Three **demonstrated use-case scenarios** (solo performance, networked collaboration, and animation authoring), supported by initial latency measurements and participant feedback.

2. Underlying Technologies

2.1 VMC and OSC Network Protocols

At the system's core lies a custom extension of the Virtual Motion Capture (VMC) protocol (Contributors, 2024), widely used in VTubing contexts (Ferreira et al., 2022; Khor et al., 2025; Popescu et al., 2021). The VMC protocol transmits 3D joint positions and orientations from motion capture devices such as the Rokoko Smartsuit or camera-based MOCAP systems. In our framework, these data streams are captured by a translation layer that reformats them into OSC-compatible vector arrays. SuperCollider intercepts, transforms, and redistributes these arrays in real time before forwarding them to Godot for visualization. Within the VMC protocol role model, SuperCollider occupies the role of a *Performer* (See Figure 2).

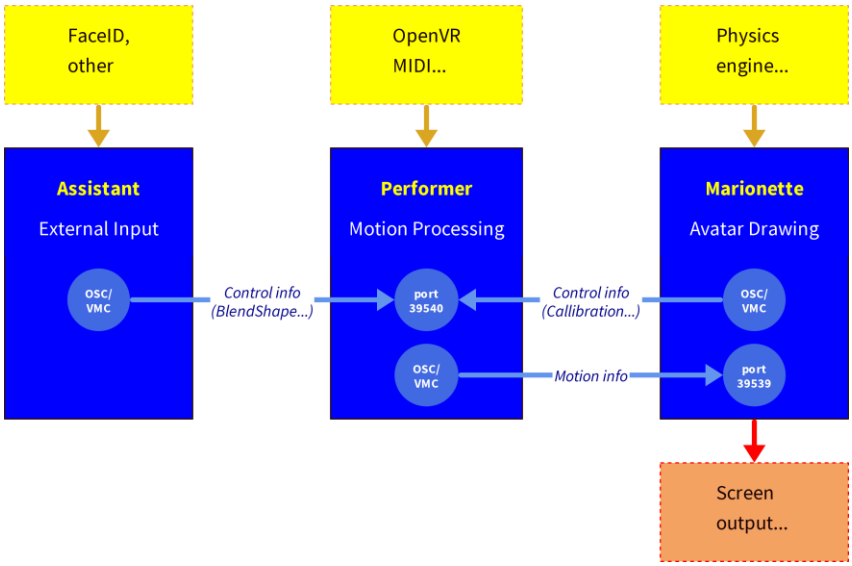


Figure 2: VMC Protocol Architecture, showing the data flow from motion capture hardware through SuperCollider to Godot

This structure supports two principal configurations:

- **Local mode**, where both environments run on a single machine communicating over localhost; and
- **Networked mode**, where data streams traverse a LAN or the Internet, enabling distributed co-performance or remote animation sessions.

In both cases, motion data, animation parameters, and sound synthesis variables share a common OSC namespace, making all parameters addressable through a uniform protocol. This transparency allows artists to live-code animation logic using the same conceptual tools they employ for sound synthesis.

2.2 Data Flow and Transformation

Motion data arrive as time-stamped vectors representing the position, orientation, and rotation of body joints. These can be:

- **Played back** from stored sessions,
- **Streamed live** from a motion capture device, or
- **Generated algorithmically** within SuperCollider through code-based routines.

Before transmission to Godot, each data vector may be intercepted and modified through SuperCollider functions. These transformations may affect:

- The **temporal axis** (e.g., speed, direction, looping, freezing),
- The **spatial axis** (e.g., deformation, offset, scaling, or mirroring), or
- Both simultaneously through dynamic or stochastic mappings.

By manipulating these data interactively, the coder can modulate not only the timing and rhythm of movement but also the expressive coupling between movement and sound.

2.3 Sound–Motion Coupling

SuperCollider uses the same motion vectors to control parameters of sound synthesis in real time. Mappings can target pitch, timbre, dynamics, rhythmic density, or algorithmic parameters within custom synthesis routines. Conversely, sound features such as amplitude envelopes, spectral centroids, or rhythmic structures can influence motion variables, creating a reciprocal feedback loop between sonic and visual layers.

Three mapping strategies are currently implemented:

1. **Direct value mapping** – positional or rotational data directly drive synthesis parameters.
2. **Derivative mapping** – velocities or accelerations modulate synthesis or trigger discrete events.
3. **Conditional mapping** – threshold crossings or pattern detections in movement data trigger specific sound behaviors.

These strategies correspond to well-established categories in the sonic interaction design literature (Delle Monache & Rocchesso, 2016), providing a principled and extensible vocabulary for motion-driven sound. This dual-control structure turns the performer's movement into a live-coded instrument, in which both animation and sound emerge from shared computational operations.

3. Interaction Design

The DIRTY Dancing framework allows artists to work at the intersection of animation, sound, and embodied interaction, treating each data stream as a configurable material open to real-time modification. Unlike conventional animation pipelines that separate modeling, rigging, animation, and rendering into sequential phases, this framework collapses these into a single process of parametric modulation, allowing expressive phenomena—gesture, rhythm, tension, release—to emerge dynamically.

3.1 Motion–Action Coupling

At the heart of the framework's expressive potential lies the motion–action coupling layer: the dynamic interface at which physical movement data, algorithmic transformations, and sound synthesis parameters interact. Every movement captured—whether from a performer's body, a procedural routine, or an input device—can serve simultaneously as an animation driver and a musical control signal. The coupling is continuous rather than symbolic: motion and sound share a common temporal and structural base.

Artists can tune the responsiveness of mappings across a wide range, from tight synchronization (percussive articulations directly coupled to sharp motion changes) to loose correspondence (slow spectral drifts correlated with accumulated motion density).

3.2 Character Structure and Emergent Behavior

The combination of motion capture and procedural animation produces a multi-layered character model. Each avatar is a composite of:

1. An **external skeleton** (kinematic hierarchy controlled by inverse kinematics),
2. An **internal behavior model** (data-driven or algorithmically generated motion), and
3. A s**ound-linked expressive layer** (mapping sonic states to motion or posture).

This layering enables perceptible variation in character behavior arising from feedback dynamics rather than scripted outputs. Small variations in motion data or synthesis parameters can produce shifts in the apparent quality or tempo of the avatar's movement. This is not a claim about simulated emotion or artificial intelligence; rather, it reflects the emergent complexity that arises when multiple coupled, time-varying signals interact in a nonlinear system.

3.3 Live Coding as Interface Design

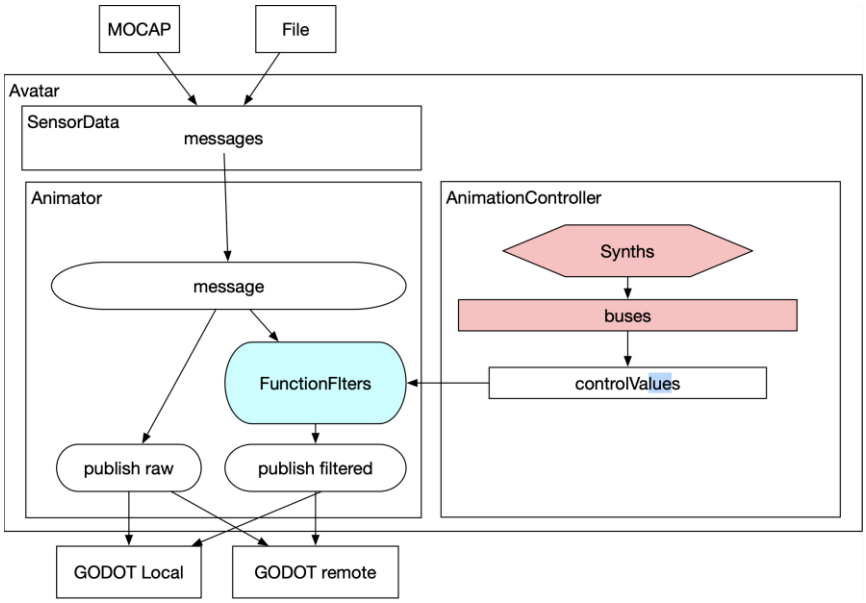
DIRTY Dancing is designed not as a fixed application but as an open improvisational instrument—a system whose behavior can be modified and extended during operation to meet the needs of its users. Live coding serves both as a method of composition and as a mode of interface design, allowing artists to redefine interaction rules while a performance is in progress. This positions the framework within the tradition of "instrument building as performance" described in the live coding literature (Collins et al., 2003; Ward, 2011).

4. System Architecture

4.1 System Components

The system comprises three main modules:

1. **Motion Capture and Tracking Module** – A set of scripts that convert motion capture data from proprietary systems (e.g., Rokoko SmartSuit, VR trackers) into VMC-compatible OSC streams. Raw motion data are streamed as OSC vectors encoding body-joint positions, rotations, and velocities. These vectors can be processed live or recorded for later manipulation.
2. **Data Arbitrator, Interaction and Sound Processing Module** – Implemented in SuperCollider. The core Avatar class manages incoming motion data, distributing them across multiple busses and synths that perform sound synthesis and transformation. Control busses carry real-time modulation signals; audio busses aggregate synthesis output. The modular design allows coders to add or modify sound behaviors during runtime without interrupting audio output. The Avatar class uses 3 instance variables to manage data loading, playback and broadcasting, data filtering and animation logic, and sound processing, as



follows (see Figure 3):

Figure 3: Internal structure and data flow in Avatar class.

   1. **SensorData** loads data from file or receives them in real time, and makes them available to *Animator*
   2. **Animator** applies filters which modify single values of each message vector and broadcasts the resulting data vectors to GODOT both locally and remotely, as required.
   3. **AnimationController** shares message vector data with the sound server, enabling both sound synthesis control and modification of the server data through control-rate algorithms on the server.
3. **Animation and Rendering Module** – Developed in Godot. This layer visualizes motion data as articulated 3D avatars using inverse kinematics and procedural animation. OSC integration within Godot enables real-time synchronization with SuperCollider. Each joint or

scene node can respond dynamically to sonic or motion parameters, enabling tightly coupled audiovisual behavior.

4.2 Live Coding Infrastructure

A live coding infrastructure spans all three modules and includes:

1. **SuperCollider code modules** fosound mapping and signal routing,
2. **Godot live scripting plugins** for real-time animation modification (see Figure 3), and

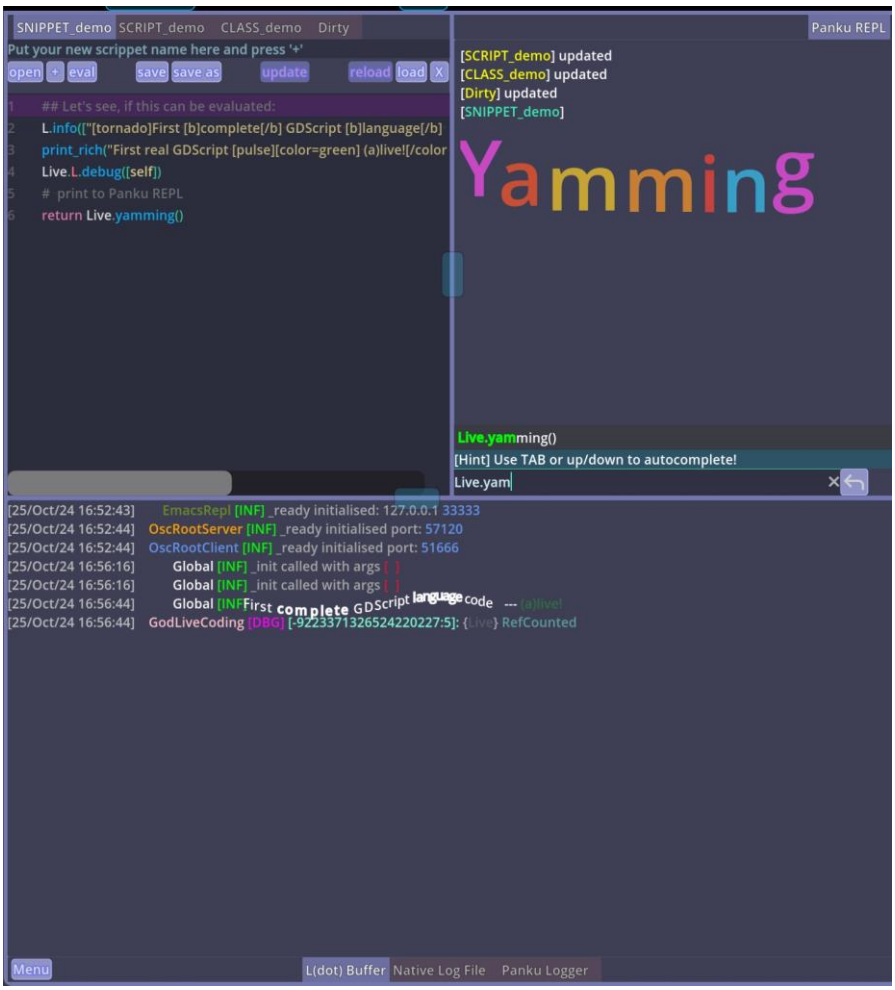


Figure 3: Live coding environment in Godot, showing the scripting interface during an active session

3. **Web-based tools** for distributed control and multi-user participation.

The infrastructure permits incremental code changes during execution. A performer may, for example, redefine the mapping between joint velocity and synthesis frequency while the animation is running, immediately observing the auditory and visual effect. This supports a practice that combines creative coding, performance improvisation, and system prototyping within a single session.

5. Use Cases and Evaluation

5.1 Deployment Configurations

Two principal deployment modes have been developed and tested:

Browser-based configuration (entry level): All components run in a web browser using WebSocket and WebAssembly modules. Pre-recorded motion capture sequences serve as input; sound synthesis runs in an embedded SuperCollider WebAssembly instance. This mode requires no software installation and is intended for workshops, educational settings, and initial exploration. It provides the lowest access threshold but constrains the range of available synthesis processes and limits real-time motion input to pre-recorded data.

Standalone configuration (advanced): The full desktop versions of SuperCollider and Godot are connected via OSC over UDP. Motion data arrive via VMC from a physical sensor system, are processed by SuperCollider, and forwarded to Godot for rendering. Each application can run on separate machines connected through a local or remote network. This mode supports real-time interaction between performer and coder, distributed collaboration in which multiple artists manipulate sound or movement concurrently, and higher-fidelity audiovisual output.

Both modes are interoperable: sessions can begin in the browser and migrate to the standalone configuration for larger-scale performance or production work.

5.2 Use Case Scenarios

Three distinct use scenarios have been developed and tested with the framework:

Scenario A: Improvised Solo Performance. A performer wearing a motion capture suit performs while a coder manipulates sound and animation parameters in real time through SuperCollider. Changes in mapping curves shift the expressive quality of the avatar's movement; sound synthesis responds dynamically to the performer's motion data. This scenario tests the responsiveness of the motion–sound coupling and the legibility of real-time code changes during live performance.

Scenario B: Networked Collaboration. Two performers in separate locations connect peer-to-peer via WebRTC. Each controls a subset of avatar joints or sonic layers, creating an interdependent distributed choreography. A transatlantic test session between Athens and Tokyo measured a round-trip latency of approximately 100 ms. This figure is consistent with accepted thresholds for distributed music performance (Barbosa, 2003; Rottondi et al., 2016) in contexts where strict metric synchrony is not required, and falls within a range compatible with expressive but not latency-critical interaction.

Scenario C: Animation Authoring and Post-Processing. The framework serves as a tool for animation production beyond performance contexts. Artists record movement data, apply temporal filters, spatial re-mappings, or algorithmic distortions through code, and export the modified sequences as animation assets. This scenario demonstrates the framework's utility as a prototyping environment for procedural animation, extending live coding from performance into production workflows.

5.3 Initial Evaluation

The evaluation reported here is preliminary; a systematic user study is identified as priority future work (§6.2).

Technical evaluation. The primary quantitative measurement is end-to-end latency in networked mode. In the Athens–Tokyo transatlantic session described in Scenario B, round-trip latency was measured at approximately 100 ms using OSC timestamp comparison. Frame rate and packet loss were monitored but not systematically recorded across sessions; this is addressed in §6.2. Local (single-machine) operation produced latency below perceptual threshold for motion-sound coupling (< 10 ms), consistent with expectations for OSC communication over localhost.

Qualitative participant feedback. Three early participants—each with experience in live coding or motion capture performance—engaged with Scenarios A and B. Feedback was collected through structured post-session discussion. Participants described the motion–sound coupling as "immediately responsive" and noted that the ability to redefine mappings during performance created a sense of "continuous compositional control." Participants also identified the current SuperCollider coding requirement as a significant learning barrier for non-specialist users, and noted that multi-user coordination in Scenario B required prior agreement on data routing conventions. These observations directly inform the limitations and future directions described in §6.

This feedback should be understood as exploratory and not statistically representative. It establishes a basis for hypothesis formation for future systematic evaluation rather than constituting evidence of general usability or artistic efficacy.

6. Discussion

6.1 Implemented System, Affordances, and Speculative Directions

For clarity, we distinguish three levels of claim in this paper:

Currently implemented: The framework connects SuperCollider and Godot via OSC/VMC, supports bidirectional motion–sound coupling through three mapping strategies, enables real-time code modification during execution, and operates in both local and distributed (WebRTC) configurations. Latency in transatlantic mode has been measured at ~100 ms. The browser-based configuration is functional without software installation. Three use-case scenarios have been demonstrated.

Afforded by the architecture (not yet systematically studied): The modular design allows additional motion sources, synthesis strategies, and rendering backends to be integrated without modifying the core protocol layer. The shared OSC namespace in principle supports arbitrary numbers of simultaneous participants; practical limits have not been determined. The mapping vocabulary (direct, derivative, conditional) provides a structured basis for designing new expressive behaviors.

Speculative and requiring future investigation: Whether the system's interaction model is accessible to users without a background in live coding; whether the approach yields aesthetically distinctive results compared with existing animation tools; whether the distributed configuration supports sustained creative collaboration over time; and how the system performs under production-scale demands.

6.2 Limitations and Future Work

Several significant limitations of the current system must be acknowledged.

Accessibility. The system currently requires coding literacy in both SuperCollider (for sound mapping and data transformation) and Godot (for animation and rendering). While the browser-based configuration removes installation barriers, it does not reduce the programming requirement. The system is therefore currently suited to practitioners with a technical background in live coding or creative coding; it does not yet serve animators, dancers, or performers without programming experience. Developing higher-level interfaces—graphical patch editors, pre-built mapping templates, or guided configuration environments—is the primary direction for broadening access.

Evaluation. The evaluation reported in §5.3 is preliminary. A systematic user study, including structured tasks, standardized questionnaires (e.g., adapted from the System Usability Scale or established creative tool evaluation frameworks), and a larger participant sample across skill levels, is required before claims about usability or expressive utility can be made. Quantitative documentation of frame rate, packet loss, and synchronization error across multiple network conditions is also needed.

Comparative positioning. A full comparative analysis with existing tools—including VTubing platforms (e.g., VSeeFace, VMagicMirror), animation prototyping environments (e.g., Blender's animation nodes, Unity's Timeline), and other live coding systems for visual media (e.g., Hydra, Gibber)—remains to be conducted. Such a comparison would clarify the specific conditions under which DIRTY Dancing offers advantages or presents tradeoffs relative to established workflows.

Scalability. Network performance in multi-participant configurations beyond two simultaneous users has not been tested. The WebRTC peer-to-peer architecture introduces known scalability constraints as participant count increases, which would need to be addressed (e.g., through selective forwarding units or relay servers) for larger ensemble configurations.

6.3 Relation to Open-Source Creative Practice

By building on open-source tools (SuperCollider, Godot) and open protocols (OSC, VMC, WebRTC), DIRTY Dancing is positioned within a broader ecosystem of freely available creative technology, including the GNU/FLOSS software culture and the collaborative coding communities associated with live coding practice. This architectural choice has concrete practical consequences: all components can be inspected, modified, and redistributed by users; no dependency on proprietary platforms or commercial licenses is required; and contributions from the community can extend the framework's capabilities. These are verifiable properties of the implementation, not aspirational claims.

We avoid the claim that this makes the system universally "accessible"—a term that implies the removal of barriers that the system, as currently implemented, does not fully remove. The open-source orientation reduces certain economic and legal barriers (cost, licensing restrictions) while the technical barrier of coding proficiency remains substantial.

6.4 The Meaning of "DIRTY"

The term DIRTY in the project's title stands for Distributed Interactive Real-Time Yamming and also functions as a conceptual marker. "Dirty" coding—improvisation, direct data manipulation, and tolerance of imperfection—contrasts with the smooth, error-concealing outputs of polished production pipelines. This aesthetic stance, which foregrounds process over

product, aligns with practices of creative misuse, hacking, and playful subversion that have driven innovation in both software culture and media art (Magnusson, 2011; Ward, 2011). Network latency, coding errors, and real-time parameter exploration are not failures to be hidden but expressive materials to be engaged.

7. Conclusion

This paper has presented DIRTY Dancing, a distributed, open-source framework for real-time networked animation and sound synthesis. The implemented system connects SuperCollider and Godot via OSC and VMC, supports three motion–sound mapping strategies, enables incremental live coding of animation and synthesis behaviors during execution, and operates in both local and wide-area network configurations. Three use-case scenarios—solo performance, networked collaboration, and animation authoring—have been demonstrated, and initial evaluation including latency measurement and qualitative participant feedback has been reported.

The current implementation demonstrates that motion capture data and sound synthesis parameters can be managed within a shared OSC namespace, enabling a form of animation practice in which design, execution, and performance are not sequential phases but concurrent, mutually influencing processes. Specific limitations—notably the requirement for programming literacy, the absence of systematic evaluation, and the lack of comparative analysis with existing tools—are identified as priorities for future work.

Future development directions include: enhancement of the MOCAP module to include interfaces to camera-based 3D skeleton capture models using OpenPose and related software; design of higher-level graphical interfaces to reduce coding barriers; a systematic user study with structured tasks and standardized questionnaires; quantitative network performance analysis across diverse configurations; comparative evaluation against established animation and VTubing tools; and exploration of AI-driven motion and sound modules as optional extensions to the generative layer.

References

Barbosa, Á. (2003). Displaced soundscapes: A survey of network systems for music and sonic art creation. Leonardo Music Journal, 13, 53–59. https://doi.org/10.1162/096112104322750791

Birringer, J. (2008). Performance, technology, and science. PAJ Publications / MIT Press.

Collective, T. (2004). TOPLAP live coding manifesto. https://toplap.org/manifesto/

Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. Organised Sound, 8(3), 321–330. https://doi.org/10.1017/S135577180300030X

Contributors, V. P. (2024). VMC protocol specification – Virtual Motion Capture (OSC/VMC protocol). https://protocol.vmc.info/english.html

Delle Monache, S., & Rocchesso, D. (2016). Enactive sound design: A process-based approach to shape sound–action relationships. AI & Society, 31(4), 535–549. https://doi.org/10.1007/s00146-015-0621-z

Fdili Alaoui, S. (2019). Making an interactive dance piece: Tensions in the choreographer-technologist collaboration. Proceedings of the 2019 on Designing Interactive Systems Conference, 1195–1207. https://doi.org/10.1145/3322276.3322341

Ferreira, J. C. V., de Souza, J. P., & Silva, F. (2022). VTuber concept review: The new frontier of virtual entertainment. Proceedings of the 24th Symposium on Virtual and Augmented Reality (SVR 2022). https://doi.org/10.1145/3604479.3604523

Godot Engine Community. (2025). Godot Engine: Open source game engine. https://godotengine.org/

Khor, C. Y., Mubin, S. A., & Neo, T.-K. (2025). A multimodal framework for integrating motion capture monitoring into XR applications. Proceedings of the ACM on Human-Computer Interaction. https://doi.org/10.1145/3734435

Magnusson, T. (2011). Algorithms as scores: Coding live music. Leonardo Music Journal, 21, 19–23. https://doi.org/10.1162/LMJ_a_00056

McCartney, J., Wilson, S., Cottle, D., Collins, N., & others. (2011). The SuperCollider book. MIT Press.

McLean, A., & Wiggins, G. (2010). Live coding: Real-time music and video performance. In R. T. Dean & A. McLean (Eds.), The Oxford handbook of algorithmic music (pp. 245–266). Oxford University Press.

O'Modhrain, S. (2011). A framework for the evaluation of digital musical instruments. Computer Music Journal, 35(1), 28–42. https://doi.org/10.1162/COMJ_a_00038

Peng, H., Ma, X., & Wobbrock, J. O. (2023). BodyLogic: A system for body-driven audio and visual performance. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Article 682. https://doi.org/10.1145/3544548.3580700

Popescu, V., Jiang, J., Wang, H., Wei, Z., Chen, Z., & Wang, R. (2021). Loose avatar–streamer coupling for expressive VTubing. 2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 791–800. https://doi.org/10.1109/ISMAR52148.2021.00105

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 60–67. https://doi.org/10.1145/1592761.1592779

Rocchesso, D., Serafin, S., Behrendt, F., Franinoínović, K., Hermann, T., Pauletto, S., Susini, P., & Visell, Y. (2008). Sonic interaction design: Sound, information and experience. CHI '08 Extended Abstracts on Human Factors in Computing Systems, 3969–3972. https://doi.org/10.1145/1358628.1358969

Rottondi, C., Chafe, C., Allocchio, C., & Sarti, A. (2016). An overview on networked music performance technologies. IEEE Access, 4, 8823–8843. https://doi.org/10.1109/ACCESS.2016.2628440

Scaffidi, C., Shaw, M., & Myers, B. (2005). Estimating the numbers of end users and end user programmers. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, 207–214. https://doi.org/10.1109/VLHCC.2005.34

Schiphorst, T. (2009). Soft(n): Toward a somaesthetics of touch. In CHI '09 Extended Abstracts on Human Factors in Computing Systems (pp. 2427–2438). ACM. https://doi.org/10.1145/1520340.1520345

Tira, H. (2023). VMC protocol integration handbook – tracking via VMC protocol. https://docs.warudo.app/docs/mocap/vmc

Wanderley, M. M., & Depalle, P. (2004). Gestural control of sound synthesis. Proceedings of the IEEE, 92(4), 632–644. https://doi.org/10.1109/JPROC.2004.825882

Ward, A. (2011). Live coding: A user's manual. In R. T. Dean (Ed.), The Oxford handbook of computer music (pp. 163–182). Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199792030.013.0008

Wright, M., & Freed, A. (1997). Open Sound Control: A new protocol for communicating with sound synthesizers. Proceedings of the International Computer Music Conference (ICMC), 101–104.

Zhao, R., Yoon, D. W., Wang, Y., Zhang, Z., Zhu, M., Zhu, Y., Kim, B., & Wohn, D. (2025). Who reaps all the superchats? A large-scale analysis of VTuber monetization on YouTube. Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 1–16. https://doi.org/10.1145/3613904.3642441