Gesture Capture, Processing, and Asynchronous Playback within Web Audio Instruments.

Benjamin Taylor
Louisiana State University
btayl61@lsu.edu

Jesse Allison
Louisiana State University
jtallison@lsu.edu

ABSTRACT

The authors describe tools for recording interactions and gestures within the Nexus User Interface web audio toolkit. More broadly, the authors investigate and implement data processing algorithms which manipulate gestures and treat them developmentally as objets geste*, similar to* objets sonores*. Gesture playback settings are noted, including variable playback rate, noise modulation, and granulation. In addition, the authors note the benefits of implementing this system within JavaScript's event-based asynchronicity.*

1. INTRODUCTION

With the development of the Web Audio API [1], the web browser is emerging as a flexible and expressive platform for computer music performance and electronic instrument design. Several toolkits are making digital signal processing on the web more accessible, including Tone.js [2], Gibberish [3], and Flocking [4]. Popular interfaces for web audio include live coding [5, 6, 7], patching interfaces such as Patchwork [1] and the Web Audio Tool [2], as well as graphical audio controls like WAAX/mui [3], Interface.js [3] and NexusUI [4]. An advantage of live coding web audio is the ability to rapidly generate automated material and dynamic polyphony on-the-fly. On the other hand, advantages of graphical user interfaces (GUIs) include the use of visual metaphors, gestural control, and visual feedback.

We seek to add easy-to-use interface automation tools to our NexusUI web audio interface toolkit in order to encourage gestural computer music improvisation using web audio interfaces. When building interface automation tools for the web, we have the advantage of past computer music research into gesture processing within human-computer interactions. For example, CNMAT's "o." [10] object library for Max [5] applies signal processing techniques and grouping algorithms to streams of OSC data from an interface. We are especially interested in how this type of gesture processing could be applied to compositional and improvisatory concerns, similar to how motives are developed in Hyperscore [11]. Hyperscore lets users define a motive gesture, then compose with the motive through a sketchpad interface.

Dr. Garth Paine [12] provides a methodology in support of these tactics in his description of a morphology of computer music gestures and interfaces. By applying Trevor Wishart's notion of dynamic morphology [13] to the fields of interfaces and gestures, Paine sees gestures, instruments, and mappings as compositional materials which can evolve during performance. Reading Paine, we extract the notion of an *objet geste*, a gesture object similar to Pierre Schaeffer's notion of an *objet sonore*. Like in the "o." tools and Hyperscore, this approach makes a gesture the subject of compositional development through signal processing and manipulation.

With these methods in mind, we share a tool for gesture recording, processing, and playback for web audio instruments, using the NexusUI web interface toolkit. This tool is specifically intended to take advantage of the gestural, visual, and object-oriented nature of graphical interfaces on the web. We focus on approaching an interaction as an *objet geste* and applying signal processing techniques to it such as noise modulation, variable playback rate, and granulation. As we will describe, JavaScript's asynchronous timing and object-orientation afford significant opportunities in this field. This tool is open-source and available as part of the NexusUI toolkit.

1.1 Nexus User Interface

Figure 1. NexusUI widgets.

Nexus User Interface (NexusUI) [14] is a JavaScript library of HTML5 audio interface components previously developed by the authors. NexusUI offers a variety of touch-compatible

[1] http://www.patchwork-synth.com/
[2] http://www.webaudiotool.com/
[3] http://hoch.github.io/WAAX/
[4] http://www.nexusosc.com
[5] http://www.cycling74.com

interfaces for web audio projects or distributed mobile participation projects. Notably, it was used recently in Sébastien Piquemal and Tim Shaw's composition *Fields* [15] which distributes audio to audience cell phones as a sound diffusion technique. NexusUI was first introduced in [14] and can be further explored through tutorials online.

2. RECORDING GESTURES

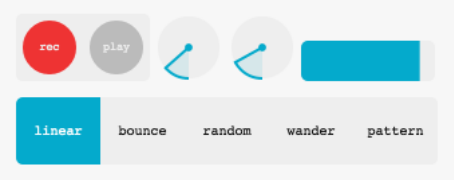

Figure 2. NexusUI gesture recorder control panel.

In order to process and replay gestures within NexusUI, a gesture recorder interface was created and bundled with the toolkit as the NexusUI *ghost* object (the top left widget in Figure 2). *Ghost* provides an interface with *record* and *play* buttons for toggling gesture recording and playback. The ghost is initialized by calling its *.watch()* method within NexusUI's *nx.onload* function:

```
nx.onload = function() {
  ghost1.watch()
}
```

During recording, the *ghost* widget monitors every NexusUI component on its webpage and persistently records the state of that component every 20ms. Some gesture data is lost in this process; this technique will not accurately capture events faster than 20ms. However, it is useful for recording and replaying general motions and gestures.

2.1 Buffer

Recorded interface data is stored in a JSON data structure, visualized in Figure 3. Each interface component constitutes one property in this structure. Each component also contains subproperties for each of that component's data outputs (such as *x* and *y* for a two-dimensional touch position widget). Each property is initialized with an empty array which is written to during the gesture recording process, becoming as long as the gesture length.

While this is not a generalized gesture format as has been proposed [16], it is an object-oriented and flexible gesture data format which can be shared between NexusUI interfaces or sent through a network and reconstructed in the interface of collaborator.

3. GESTURE PLAYBACK

When recorded gestures are replayed, interface components are updated visually so that the user perceives a visual as well

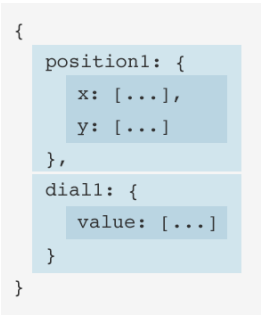

Figure 3. Data structure for recorded gestures.

as aural replication of the recorded action. As opposed to the streaming style of gesture recording, gesture playback is event-based and asynchronous. *Interface components are affected only when and if their value changes in the gesture array buffer*. For example, if during recording a toggle is turned on and five seconds later it is turned off, then the gesture playback will only emit those two events five seconds apart. Between those events, the user can interact with the toggle as they like. Moments of inactivity are not replayed, and gesture playback does not assume streaming control of the entire interface.

This event-based playback allows users to interact freely with other parts of the interface that were not interacted with during the recording. This is vital, as the user does not need to re-route the *ghost* to watch the specific widgets it wishes to record. The *ghost* will watch the whole interface, but only playback the active events that were fired during the recording period. We found this strategy affords a coexistnece between the performer and the *ghost* during playback.

3.1 Playback Modes

Recorded gesture data can be replayed into the interface in several replay modes. These are chosen through a the gesture recorder GUI (see Figure 2) or can be chosen programmatically using NexusUI's API.

3.1.1 Linear

Linear mode (Figure 4B) replays a gesture in its original direction from beginning to end. If looping playback is turned on in the *ghost* object, linear mode will jump back to the beginning when reaching the end of the buffer.

3.1.2 Bounce

Bounce mode alternates forward and reverse gesture playback. This is useful for avoiding sudden jumps in output value, which can occur when moving from the end of a gesture to the beginning in linear mode.

3.1.3 Random

Random mode plays the gesture non-linearly, consistently jumping to a random point in the gesture array buffer. This jump occurs each animation frame (20ms). Figure 4D illustrates

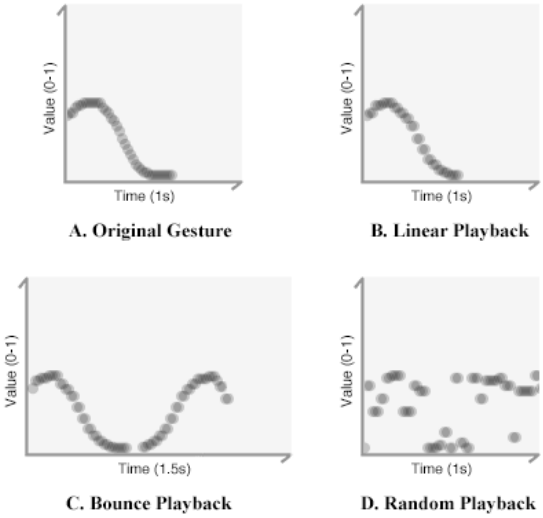

Figure 4. Modes of gesture playback. The source gesture (A) is of a slider interface moving from half value (0.5) to no value (0) over 0.6 seconds.

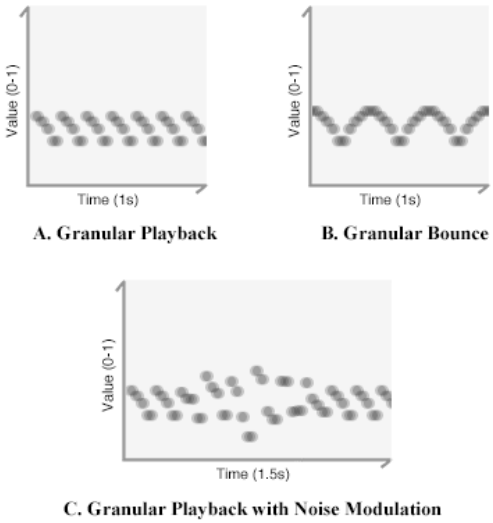

Figure 5. Granular gesture playback in different modes.

how output values are constricted to values from the gesture, which are all in the lower half of the value spectrum in this gesture. So, while the results of random playback are far removed from the original gesture motion, all output values will be picked from the original gesture.

3.1.4 Wander

Wander mode is a form of generative playback which randomly chooses to move forward or backward by a single step each animation frame, therefore reading through the gesture buffer similarly to the *drunk* object in Max. Wander mode might read buffer indeces 9 then 8 then 7, 8, 7, 8, 9, 10, and so on. Like bounce mode, this is a way of achieving generative gesture developent without experiencing large jumps in output value because buffer indeces are always consecutive.

3.2 Granular Playback

Loop points may be added during gesture playback in order to achieve granular playback of a gesture. Loop points can be modified using a range slider in the gesture recorder interface (Figure 2). This functionality may be used in tandem with any playback mode, so that a small section of the gesture may be played back in bounce mode or random mode (Figure 5).

3.3 Gesture Processing

In addition to the aforementioned playback modes, several gesture signal processing techniques are made available through the *ghost* component.

3.3.1 Noise

The gesture signal can be modulated by a variable-amplitude noise signal, introducing an element of variation into the gesture's values upon playback. Figure 6 shows our original gesture played back with noise modulation of 0.1 (left graphic), 0.3 (middle), and 0.5 (right graphic).

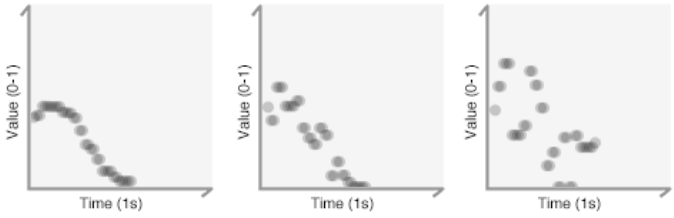

Figure 6. Gesture playback with low, medium, and high noise modulation.

3.3.2 Playback Rate

Gesture playback rate can be changed in real time, including to negative numbers.

3.3.3 Filtering

During replay, gesture data may be sent through a lowpass or highpass filter. As this may be a more specialized concern, this is not part of the general recorder interface, but can be enabled using the NexusUI API.

3.3.4 Transposition and Scaling

Finally, gesture data can be the subject of transposition (addition or subtraction from output values) or amplitude scaling (multiplication or division of output values). Through transposition, the gesture's output values may be universally raised or lowered. Through scaling, the amount of difference between the lowest and highest gesture values may be amplified or reduced, akin to a very basic dynamic compression.

4. CONCLUSIONS AND FUTURE WORK

Gesture capture, processing, and playback within the NexusUI web audio interface toolkit contributes to a need for automation and improvisation tools in browser-based audio GUIs. By building upon prior research into the compositional development of gestures through signal processing, we offer new strategies for computer music performance on the web.

Several notable opportunites present themselves. The processed replay of a gesture may itself be captured as a new gesture, leading to that gesture's evolving development. In addition it is possible, though untested, that *ghost* components could be used to control and automate other *ghost* components, leading to generative and unscripted gesture recording and playback.

The research described in this paper incites further work as well. Investigations should be made into better graphical interfaces for controlling the replay of gestures, possibly through a drawing interface. Visualizing each recorded gesture could also aid improvisation. When web audio interfaces see a greater influx of gestural, generative compositional devices, web audio may transcend its status as a novelty and simply become another exciting way to make music.

Acknowledgments

The authors wish to thank the Lousiana State University Center for Computation and Technology, as well as all contributers to the open-source NexusUI framework.

5. REFERENCES

[1] P. Adenot, C. Wilson, and C. Rogers. Web Audio API, http://webaudio.github.io/web-audio-api/. [Online]. Available: http://webaudio.github.io/web-audio-api/

[2] Y. Mann, "Interactive Music with Tone.js," in *Proceedings of the 1st annual Web Audio Conference*, 2015.

[3] C. Roberts, G. Wakefield, and M. Wright, "The Web Browser as Synthesizer and Interface," in *Proceedings of the New Interfaces for Musical Expression conference*, 2013.

[4] C. Clark and A. Tindale, "Flocking: a framework for declarative music-making on the Web," in *Proceedings of the 2014 International Computer Music Conference*, 2014.

[5] J. K.-M. Charlie Roberts, "Gibber: Live Coding Audio in the Browser," in *Proceedings of the 2012 International Computer Music Conference*, 2012.

[6] K. Stetz, "Lissajous: Performing Music with Javascript," in *Proceedings of the 1st international Web Audio Conference*, 2015.

[7] C. McKinney, "Quick Live Coding Collaboration In The Web Browser," in *Proceedings of the 14th Annual Conference in New Interfaces for Musical Expression (NIME)*, 2014.

[8] W. Hsu, "Managing Gesture and Timbre for Analysis and Instrument Control in an Interactive Environment," in *Proceedings of the 2006 International Conference on New interfaces for Musical Expression*, 2006.

[9] V.-F. Maniatakos and C. Jacquemin, "Towards an Affective Gesture Interface for Expressive Music Performance," in *Proceedings of the 2008 International Conference on New Interfaces for Musical Expression*, 2008.

[10] A. Freed, J. MacCallum, and A. Schmeder, "Composability for Musical Gesture Signal Processing using new OSC-based Object and Functional Programming Extensions to Max/MSP," in *Proceedings of the 2011 International Conference on New Interfaces for Musical Expression*, 2011.

[11] M. Farbood, H. Kaufman, and K. Jennings, "Composing with Hyperscore: An Intuitive Interface of Visualizing Musical Structure," in *Proceedings of the 2007 International Computer Music Conference*, 2007.

[12] G. Paine, "Gesture and Musical Interaction: Interactive Engagement Through Dynamic Morphology," in *Proceedings of the 2004 conference on New Interfaces for Musical Expression*, 2004.

[13] T. Wishart, *On Sonic Art*, S. Emmerson, Ed. Philadelphia, U.S.A.: Harwood, 1996.

[14] B. Taylor, J. Allison, Y. Oh, D. Holmes, and W. Conlin, "Simplified Expressive Mobile Development with NexusUI, NexusUp, and NexusDrop," in *Proceedings of the New Interfaces for Musical Expression conference*, 2014.

[15] S. Piquemal and T. Shaw, "Fields," in *Proceedings of the 1st international Web Audio Conference*.

[16] A. Jensenius, T. Kvifte, and R. I. Godoy, "Towards a Gesture Description Interchange Format," in *Proceedings of the 2006 international conference on New interfaces for musical expression*, 2006.

[17] B. Taylor and J. Allison, "BRAID: A Web Audio Instrument Builder with Embedded Code Blocks," in *Proceedings of the 1st international Web Audio Conference*, 2015.

[18] N. Schnell, V. Saiz, K. Barkati, and S. Goldszmidt, "Of Time Engines and Masters: An API for Scheduling and Synchronizing the Generation and Playback of Event Sequences and Media Streams for the Web Audio API," in *Proceedings of the 1st annual Web Audio Conference*, 2015.

[19] V. Saiz, B. Matuszewki, and S. Goldszmidt, "Audio oriented UI components for the web platform," in *Proceedings of the 1st annual Web Audio Conference*, 2015.