Creating a DJ-ready Web Player for Interactive Music

Attila Haraszti
Independent Author

attila@haywirez.com

ABSTRACT
The talk will explore the motivation, challenges and solutions devised in a year-long quest to build a DJ-ready web player for songsling.studio, a tool I'm building for publishing and presenting interactive music on the Internet. The player itself utilizes an HLS-like solution as a streaming mechanism, and allows arbitrary changes to the playback speed, direction and content. I will dive into the history and legacy of computerized music playback systems, the dangers of leaving music presentation in the hands of a few dominant platforms and highlight the unrealized potential of our everyday computing devices.

1. INTRODUCTION
In a short introduction story chronicling my background in electronic music production and DJing, I will illustrate how the format of interaction influences the content of music and the process of creation. I will point out that by 2020, an entire generation will have grown up experiencing music overwhelmingly through dominant web streaming platforms such as YouTube, Spotify or Soundcloud.

2. THE POWER AND LEGACY OF PLAYBACK SYSTEMS
By a combination of earlier technical limitations and conventions related to digital playback, the web players provided by these platforms promote an extremely limited way of interacting with music, even when compared to previous analog systems of inferior quality. The end result is an encouragement of passive listening in order to steer the listeners towards the monetary goals of the platforms, instead of nurturing what should be a sacrosanct, intimate relationship between people and music [1].

2.1 Examination of Current Playback Systems
The first surprising feature of typical present-day media players is how little of the visual user interface is dedicated to any kind of meaningful control over the sonics themselves. Control elements such as a seekbar, playback and volume control occupy typically below 3-5% of the visual field. They are also limited in their essence – the seekbar quickly jumps to the correct position in a recording, but limited auditory information is conveyed when compared to cueing music using a fast-forward or rewind action on a classical turntable or tape machine.

2.2 Desired design
In my approach to designing a player from scratch, I set a hard design constraint of allowing the user full control and explorability of the sound played. That means that at any time, listeners should have the ability to near-instantly reverse playback, speed up or slow down. Provided that the given portion of a composition has been fetched from the network, they should also be able to near-instantly jump to that arbitrary point in the timeline.

3. THE TRIALS & TRIBULATIONS OF BUILDING A PLAYER USING THE WEB AUDIO API
Most players on the web make use of the <audio> HTML element, even if it might be piped into a Web Audio API graph via a MediaElementSourceNode to provide more pleasant fade-ins and fade-outs. This approach is unsuitable for our use case given the inability to control playback speed in a satisfactory manner. Besides the limited playback rate range of 0.5-4x, most browsers implement pitch-preserving and time-stretching algorithms that were likely designed with one use case in mind — listening to human speech at faster or slower speeds. The proposed preservesPitch attribute that would allow developers to switch off this processing step is not implemented in any browser except Mozilla Firefox. Backwards playback is also not possible, therefore I had to explore a more low-level solution using the Web Audio API.

3.1 The Naïve Approach
An AudioBufferSourceNode has a k-rate playbackRate parameter that has a sufficient resolution for controlling playback speed for our needs. However, compositions are typically much longer than a single buffer is designed to hold, and it also would not be efficient to fetch them from a network resource using a single, large request. Therefore, the use of streaming segmentation techniques such as HLS[1] is well-advised. The Web Audio API provides sufficient precision for scheduling so that individual segments of audio can be played one after another in a seamless, gap-free manner — colloquially known as "buffer stitching". We can therefore use this technique to pre-schedule a few segments in front of the playhead as needed. The problem arises from the fact

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

[1] HTTP Live Streaming — a protocol for transferring unbounded streams of multimedia data

that the AudioBufferSourceNodes are fire-and-forget, single use only — they can be stopped before they start, but not rescheduled. As the playback speed should be allowed to change at any point and in an unpredictable manner, all scheduled nodes that have not yet started playback have to be canceled. Even if the same AudioBuffer can be reused and referenced in several AudioBufferSourceNodes, creating the new nodes takes too long, which leads to problems when our current playhead position is close to the tail edge of the node that is currently playing.

3.2 Buffers of Buffers

As described above, it is not possible to create and reschedule the AudioBuffers sufficiently fast. Luckily, this can be circumvented by taking inspiration from multiple buffering techniques used in computer graphics rendering for preventing visual tearing or stuttering. Thus, we can simply create a sufficiently large cyclic buffer of AudioBufferSourceNodes in advance and backfill them as needed. For a use case involving audio, we need to have significantly more nodes ready than the double or triple buffered approach typically used for graphics. I have found that an acceptable perceptual responsiveness can be achieved by approximating the number of common frames per second refresh rates.

3.3 Controlling state

The implementation of the necessary scheduling mechanism quickly leads to complex state management issues that proved cumbersome to debug. In order to make it easier to reason about the code and improve code maintainability, I have opted for formalizing the business logic using statecharts, a well-known application modeling technique [2]. This allowed the decoupling of the implementation from the description of behavior, with several substates representing particular components of the code.

3.4 Custom resampling via AudioWorklet

Unfortunately, the buffer-stitching approach has another problem that materializes even in the best-controlled scenarios: Interpolation between sample frames at certain sample and playback rates might lead to inconsistent frame boundaries between neighboring buffers. Most of the times, this is not disturbingly perceptible, but it does become apparent if we test the player using a perfectly segmented sine wave. As a workaround with more added complexity, overlapping buffers can be created with precisely scheduled volume switchovers [3]. An alternative, theoretically perfect solution can be achieved by implementing a custom resampling algorithm and playhead tracking mechanism inside an AudioWorklet process. However, as popular browser support is still lagging, we have to weigh the benefits and tradeoffs.

4. A VISION OF THE FUTURE: PRESENTING MUSIC VIA WEB APPS INSTEAD OF RECORDINGS

My goal was to preserve, or at least imitate, the tactile feedback and interaction allowed by analog playback mechanisms. However, it is just as important to examine what future types of musical interaction could come as a result of a networked, distributed online playback system.

In my vision, artists should be concentrating on interactive music that supersedes the current focus on simple, static recordings. This means that each web player becomes a networked interaction portal, which can influence what other listeners hear and see. Using the latest iteration of the player engine, I will demonstrate an implementation of horizontal re-sequencing [4, 5] as a composition method of interactive music and what possibilities this might offer to the wider community of music producers.

5. ACKNOWLEDGMENTS

I would like to give my thanks to Stephan Hesse for the suggestion on researching computer graphics processing approaches, Chris Wilson for open-sourcing his DJ deck implementation using the Web Audio API [6] and Christoph Guttandin for creating and maintaining the standardized-audio-context library [7]. I would also like to express my gratitude for all current and previous organizers of the Web Audio Conference for inspiration and support, as well as electronic music pioneers such as Laurie Spiegel and Jeff Mills for making me continually rethink my approaches to music.

6. REFERENCES

[1] Spiegel, L. 1992. Music: Who Makes it? Who just takes it? Electronic Musician #8, 1 (January 1992), 114. Retrieved October 6, 2019 from http://retiary.org/ls/writings/em_back_page_slashed.html

[2] Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.* 8, 3 (June 1987), 231-274. DOI= https://doi.org/10.1016/0167-6423(87)90035-9.

[3] Harris, R. 2016. Phonograph.js: Tolerable mobile web audio. (August 2016). Retrieved October 6, 2019 from https://medium.com/@Rich_Harris/phonograph-js-tolerable-mobile-web-audio-55286bd5e567

[4] Sweet, M. 2015. Writing interactive music for video games: a composers guide, Upper Saddle River, NJ: Addison-Wesley. (September 2014), 278-279

[5] Phillips, W. 2017. A Composer's guide to game music, Mit Press Ltd. (2017), 188-193

[6] Wilson, C. wubwubwub (2014), GitHub repository, https://github.com/cwilso/wubwubwub

[7] Guttandin, C. standardized-audio-context (2019), GitHub repository, https://github.com/chrisguttandin/standardized-audio-context