WAAX: Web Audio API eXtension

Hongchan Choi
Center for Computer Research in Music and Acoustics(CCRMA)
Stanford University
Stanford, CA, USA
hongchan@ccrma.stanford.edu

Jonathan Berger
Center for Computer Research in Music and Acoustics(CCRMA)
Stanford University
Stanford, CA, USA
brg@ccrma.stanford.edu

ABSTRACT

The introduction of the Web Audio API[1] in 2011 marked a significant advance for web-based music systems by enabling real-time sound synthesis on web browsers simply by writing JavaScript code. While this powerful functionality has arrived there is a yet unaddressed need for an extension to the API to fully reveal its potential. To meet this need, a JavaScript library dubbed WAAX[2] was created to facilitate music and audio programming based on Web Audio API bypassing underlying tasks and augmenting useful features. In this paper, we describe common issues in web audio programming, illustrate how WAAX can speed up the development, and discuss future developments.

Keywords

Web Audio API, Chrome, JavaScript, web-based music system, collaborative music making, audience participation

1. NEW ERA OF WEB AUDIO

After years of muteness, the web gradually attained the power to integrate sound through plugins such as Flash and QuickTime, albeit without capability of real time sound synthesis. The new `audio` element in HTML5 was remarkable in allowing for common audio needs including streaming of audio playback [6]. The recent introduction of Web Audio API opened a new chapter to the next level of audio application. Currently under the active development, the goal of this API is to incorporate the capabilities found in modern music software such as sound synthesis, mixing, and processing tasks into the web browser without additional third-party components.

Web Audio API opens the path to use a web browser as a full-blown music environment and we believe this transformation brings several benefits to the NIME community: developers can iterate and experiment in a way that never has been possible; distribution or publishing of a work to common users or audiences becomes possible with minimum effort; cutting-edge web technologies such as WebSocket, WebGL, and WebRTC can be easily integrated into the music project simply by writing JavaScript codes [7].

[1] http://www.w3.org/TR/webaudio/
[2] https://github.com/hoch/waax

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NIME'13, May 27 – 30, 2013, KAIST, Daejeon, Korea.
Copyright remains with the author(s).

The other benefit of the Web Audio API is that the W3C Audio Working Group developing it which consists of members from leading software companies including Google, Apple, Mozilla Foundation, and W3C invited experts - with the primary contributor being the Google Chrome team.

2. BREEDS OF MUSIC APPLICATIONS

There have been several breeds of professional music applications designed to be used on specially equipped workstations for serious music production. Over the past decades we have observed their rapid development fueled by industry competition, such that at present it can be difficult to differentiate them from individual features.

Before the advent of new form factors such as smartphones and tablets, most music applications were developed for the professional market. The difference in form factors introduced new paradigms by emphasizing casual, playful and social quality of the application. The success of mobile apps contributed to a newly emerged market for music software by embracing these key elements [16]. It is also worth mentioning that some mobile applications are reportedly used for professional music production. Thus, the boundary between casual and serious production is getting increasingly ill-defined.

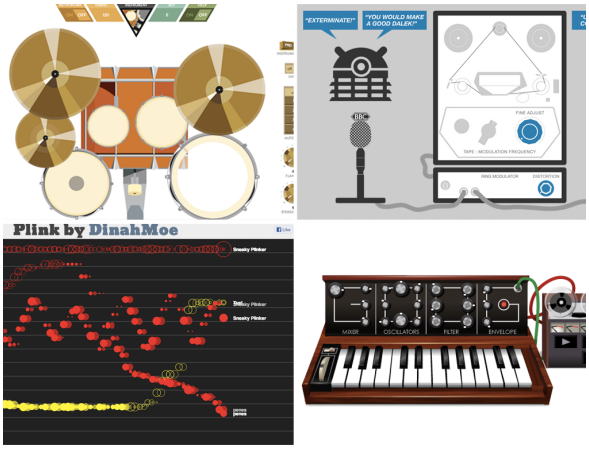

Figure 1: Music Applications built with Web Audio API

Between the two parallel worlds of professional and mobile music applications, web-based music systems have a very unique position. Its presence is not limited by form factors because the web browser exists on various platforms such as workstation, laptop, handheld and even embedded system. With the power of real-time audio processing, the strength in visual interaction, and the freedom from central-

ized regulation (i.e. App Store), the overall user experience of web music system shows a huge potential even though the application may lose some of the advantages of native execution.

As shown in Figure 1, various projects on the web are using Web Audio API to demonstrate its power and flexibility: Jam with Chrome, BBC Radiophonic Workshop, and Plink [4] [2] [8]. The purpose of these demos is to present novel user experience with the implication of performance and versatility of Web Audio API.

Using this API is plain and simple as writing JavaScript codes. However, many web developers who are unfamiliar with programming music or audio face a steep learning curve. Conversely, audio programmers are often unfamiliar with the details of web programming Our goal is to find a sweet spot between two opposite ends of the spectrum accommodating needs from both sides.

3. RELATED WORK

There are numerous JavaScript libraries that simplify and accelerate the web development. Some of them are so innovative and efficient that they change how developers think and work. For example, WebGL is derived from OpenGL ES 2.0 and it is fairly difficult to program OpenGL ES without prior experience [11]. `Three.js`, a JavaScript WebGL library, is specifically designed to solve this problem by hiding OpenGL ES behind the high level (human readable) API [5].

The same approach can be taken to audio programming and several JavaScript libraries have been crafted to serve the purpose. The most prominent one is `audioLib.js`. Its reliance on plain JavaScript for processing audio results in low performance, however, the scope of the project is fairly comprehensive and well-designed inspiring other JavaScript projects in many ways [9]. `Tuna` is another library for guitar effects. It is comparable to WAAX in terms of the implementation due to the fact that it runs on Web Audio API for optimized audio processing, but its primary focus is encapsulating several audio effects with a simplified interface [1]. Gibber also attempts to provide a platform for experimental music performance, but its audio processing depends on `audioLib.js`, which performs audio processing in JavaScript, as well as other web development libraries [15].

WAAX is distinct from the projects mentioned above:

- WAAX has its own abstraction layer called *unit* as an atomic building block. It is derived from the convention of unit generators in common computer music frameworks.
- All sound synthesis or signal processing is performed by native code, not in the script level, ensuring optimum efficiency.
- WAAX introduces *syntactic sugars*[3] making the code more comprehensible with considerably less effort.
- It offers several useful tools for web or audio developers: `analyzer` units for `canvas` element or WebGL visualization, `GUI` units for interactive parameter control, and a simple web-based IDE for rapid development.

As WAAX is in early stages of development, there are several components (which are yet to be released publicly) to be incorporated into this library in the near future. One of missing parts is *WebRTC*, the new web technology that enables peer-to-peer connection between browsers [12]. It realizes the idea of real time collaboration by interconnecting multiple clients without heavy server-side programming. The other interesting working draft is *Web MIDI API* that empowers the browser to access local MIDI devices (i.e. USB-MIDI keyboards and controllers) [10].

[3]Syntax within a programming language that is designed to make things easier to read or to express

4. WAAX: KEY FEATURES AND EXAMPLES

With our progress on WAAX so far, we report three key features in this paper: the WAAX unit framework, the visualizer unit, and the interactive code editor. This section describes the merits of these features and short code snippets to demonstrate basic usages.

4.1 WAAX Unit Framework

A *node* is an atomic object of the Web Audio API and an audio graph (patch) can be generated by connecting multiple nodes in a certain order. Built on top of it, a WAAX *unit* is an abstraction of an audio graph with additional features for a specific purpose. This abstraction is to hide underlying tasks from users and to provide easy access to parameters.

```
1  // creating units
2  var saw = new WX.Oscil({ type:"sawtooth" }),
3      sqr = new WX.Oscil({ type:"square" }),
4      lpf = new WX.ModLPF({ cutoff:2500, Q:12 }),
5      env = new WX.ADSR({ a:0.001, d:0.002 }),
6      vrb = new WX.ConVerb({ source:"ir/hall.wav" });
7  // building an audio graph
8  WX.link(saw, lpf, env, vrb, WX.DAC);
9  // additional connection
10 sqr.to(lpf);
11 // adjust parameters
12 saw.frequency = sqr.frequency * 2;
13 saw.gain = 0.2;
14 rev.mix = 0.3;
15 env.params = { s:0.5, r:0.2, gain:0.5 };
16 // trigger a note
17 env.noteOn();
```

Listing 1: Using WAAX

From line 2 to 5, the code constructs 5 units with initial settings by passing JavaScript object literals. Line 6 implies that loading an impulse response file for reverberation can be done simply by passing a URL. The `WX.ConVerb` unit encapsulates all the necessary steps to fetch data file from the server via `XHR(XMLHttpRequest)`[4]. Line 8 shows the example of syntactic sugar connecting 5 units in one line of code. Note that `WX.Out` is the master fader of the WAAX system. Getters and setters set parameters (line 12-15) and assigning an object literal to the `.params` setter will automatically map values accordingly (line 15). Finally, a developer can trigger a note with the method `.noteOn()` of `WX.ADSR` unit as shown at line 17.

4.2 Visualizer

HTML5 brought 2D and 3D graphics into modern browsers. A stream of audio sample can be displayed by these visual components but it requires a bit of configuration to draw audio into the visualization. This is where the analyzer units

[4]The Web API used to send HTTP requests directly to a web server and load the server response data directly back into the script.

```
1  // create canvas and 2d context
2  var cvs = document.getElementById('wx-viz');
3  var ctx = cvs.getContext('2d');
4  // create an oscillator and a visualizer
5  var osc = new WX.Oscil({ frequency:688 }),
6      viz = new WX.Visualizer({ context:ctx });
7  // connecting units
8  osc.to(WX.DAC);
9  osc.to(viz);
10 // assign user-defined function
11 viz.onDraw = function(buffer, c, w, h) {
12     c.clearRect(0, 0, w, h);
13     c.beginPath();
14     for(var i = 0, b = buffer.length; i < b; ++i)
15         ctx.lineTo(i, buffer[i]*100+100);
16     ctx.stroke();
17 }
18 // rendering loop
19 (function draw() {
20     requestAnimationFrame(draw);
21     viz.draw();
22 })();
```

Listing 2: Using WX.Visualizer

in WAAX come in: `WX.Waveform` and `WX.Spectrum` are designed to visualize waveform and spectrum. `WX.Visualizer` takes care of a user-defined renderer to draw any arbitrary visualization as shown in Listing 2.

Listing 2 demonstrates that the visualizer unit can be treated as a regular audio unit. In line 9, an audio stream goes into the visualizer unit. Line 11 to 17 defines the user-defined renderer by assigning it to `.onDraw` property of the visualizer which is executed every frame by the animation loop in Line 19 to 22. Customizing visualization can be done by modifying various visual properties of the 2D context instance(`viz`).

4.3 Interactive Code Editor

The web-based code editing has become popular because it introduces numerous advantages over offline editing: instant publishing to the web, effortless collaboration, secure cloud storage, and more. Services like `jsfiddle`[5] and `jsbin`[6] take this concept further by providing small IDEs (integrated development environments) for web programming. These environments take care of all the prerequisites for common web projects such as getting a hosting service, setting up servers, and installing libraries.

The most remarkable benefit in this environment is that the code modification can be instantly interpreted by the JavaScript engine in the web browser. This encourages rapid experiments and iterations with minimum effort - our rationale for creating the interactive code editor for WAAX and Web Audio API. (See Figure 2) With this mini IDE, one can swiftly sketch a musical idea by writing code directly on the web and hearing the result on the fly. To finalize their work, a user can export their code as a HTML file that can easily be imported to the other web projects.

5. USE CASES

5.1 Large Scale Audience Participation

We believe the most relevant use case out of this technology for the NIME community is the audience participation and collaborative music making. There have been attempts to realize collective user experiences using mobile platforms [13][14]. However, the distribution of mobile software in a centralized marketplace suffers from regulatory procedures, an obstacle that prohibits fast experimentation and even participation, which makes on-site participation virtually impossible. On the contrary, a web-based solution is perfect to achieve this goal as the following scenario suggests:

- A web server can be established to serve two purposes: software distribution and coordination between participants.
- Users can load pre-made instruments onto their web browser on any platform simply by accessing the web page on the server. No additional installation or authentication is required.
- Without reloading the page, the server can send and receive data to arrange the connected clients by managing rhythm, timbre, instrumentation and etc.
- The boundary of the network is not limited to the local area, bypassing NATs or firewalls because the entire communication is based on WebSocket or WebRTC via ICE(interactive connectivity establishment), unlike the UDP-based protocol OpenSoundControl.

5.2 Audio Programming Hub

Web-based code editors are already a great fit for this task. With their service, developers can share code snippets with others without setting up a web service. Furthermore, version control systems allow anyone to fork the original code to fix, improve or expand it. Collective efforts like this might result in an ever-expanding audio code library that anyone can fetch or contribute.

5.3 Collective Content Creation

With the WAAX framework, developers can easily create a custom unit from a new idea or algorithm. Sound designers can produce presets or a sample library that can be used in conjunction with the new unit. A collection of WAAX units and audio content can be bundled into one package for easy distribution as in the business model we have seen in the music production industry for decades. Needless to say, using such well-crafted audio software will improve sonic experience on the web.

6. CONCLUSION AND FUTURE WORK

For the last few months of development, our primary goal was to build a solid scaffolding with a user-friendly interface. With the recent addition of analyzer units and the interactive code editor, we foresee the potential of this project to transform the web browser into a music platform for creation and performance. Although support for external MIDI devices is still in the drafting phase, we will continue to expand the territory once the Web MIDI API is publicly released.

The Web Audio API is currently supported by cutting-edge browsers such as Chrome and Safari. From the perspective of general web development, these inconsistencies of implementation between various vendors have been a great difficulty in web programming. Polyfills[7] are commonly employed to work around this problem. WAAX will

[5] http://jsfiddle.net/
[6] http://jsbin.com/

[7] JavaScript code which provides facilities that are not built into a web browser.

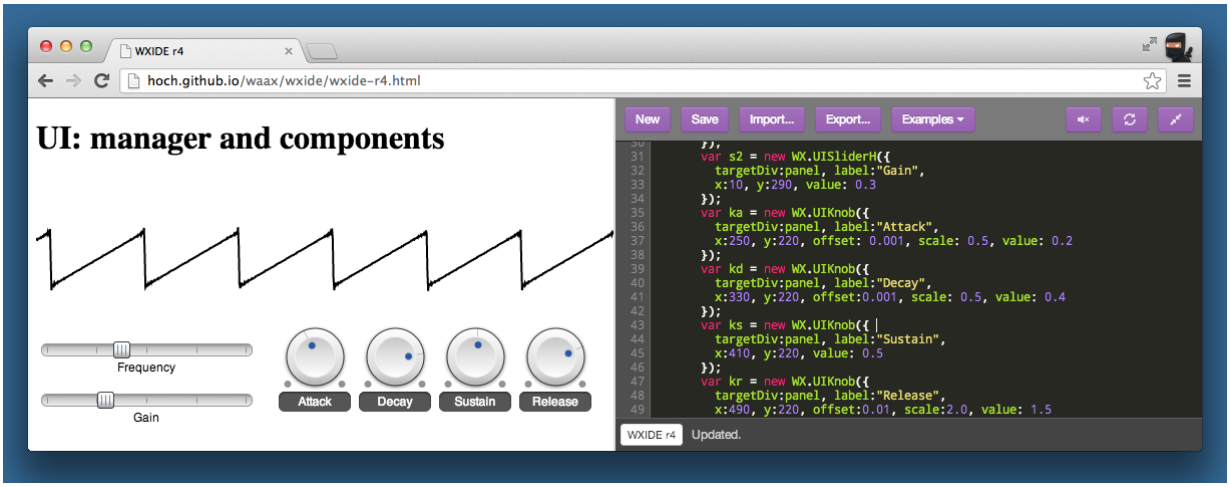

Figure 2: WXIDE: The interactive code editor for WAAX

target Chrome as a primary platform and will try to support Safari and FireFox with polyfills as the project moves forward. As of the time of writing, support on the Web Audio API by Internet Explorer is not planned thus we will not support it [3].

From the standpoint of the NIME community, we expect to see larger scale experiments using the web as a musically expressive media. The seamless integration with rich visual components such as 2D/3D graphics, HTML and CSS will open a whole new level of interactivity that cannot be found in many native audio applications. Instant interconnection between clients will enable a compelling form of collaborative music making. Moreover, we can continue to design mobile music experiences without spending resources developing or publishing of native applications since support for the Web Audio API on mobile web browsers is planned by major vendors in the near future.

7. ACKNOWLEDGMENTS

We would like to thank Chris Rogers(Google) for creating Web Audio API and Boris Smus(Google) for providing invaluable input to our project. We are also grateful to Chris Chafe(CCRMA), Luke Dahl(CCRMA), John Granzow(CCRMA) and Collin Sullivan(CCRMA) who helped develop the academic research and offered technical advices on web development.

8. CODE REPOSITORY

The entire code repository including code examples in this paper is available on GitHub at:

https://github.com/hoch/waax

9. REFERENCES

[1] An audio effects library for web audio. https://github.com/Dinahmoe/tuna. Accessed: 02/01/2013.

[2] Bbc radiophonic workshop. http://webaudio.prototyping.bbc.co.uk. Accessed: 04/23/2013.

[3] Get ready for plug-in free browsing (internet explorer). http://msdn.microsoft.com/en-us/library/ie/hh968248. Accessed: 02/01/2013.

[4] Jam with chrome. http://www.jamwithchrome.com/. Accessed: 04/23/2013.

[5] Javascript 3d library. https://github.com/mrdoob/three.js/. Accessed: 02/01/2013.

[6] Mozilla developer network: <audio>. https://developer.mozilla.org/en-US/docs/HTML/Element/audio. Accessed: 04/23/2013.

[7] Mozilla developer network: Html5. https://developer.mozilla.org/en-US/docs/HTML/HTML5. Accessed: 04/23/2013.

[8] Plink by dynahmoe. http://labs.dinahmoe.com/plink/. Accessed: 04/23/2013.

[9] A powerful audio tools library for javascript. https://github.com/jussi-kalliokoski/audiolib.js. Accessed: 02/01/2013.

[10] Web midi api, w3c editor's draft. http://webaudio.github.com/web-midi-api/. Accessed: 02/01/2013.

[11] Webgl - opengl es 2.0 for the web. http://www.khronos.org/webgl/. Accessed: 04/23/2013.

[12] Webrtc 1.0: Real-time communication between browsers, w3c editor's draft. http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcdatachannel. Accessed: 02/01/2013.

[13] G. Essl and M. Rohs. Interactivity for mobile music-making. *Organised Sound*, 14(02):197–207, 2009.

[14] J. Oh and G. Wang. Audience-participation techniques based on social mobile computing. In *Proceedings of the International Computer Music Conference (ICMC 2011)*, 2011.

[15] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference (ICMC 2012)*, 2012.

[16] G. Wang, G. Essl, J. Smith, S. Salazar, P. Cook, R. Hamilton, R. Fiebrink, J. Berger, D. Zhu, M. Ljungstrom, et al. Smule= sonic media: An intersection of the mobile, musical, and social. In *Proceedings of the International Computer Music Conference (ICMC 2009)*, pages 16–21, 2009.