Overview and Status of Binaural Rendering in Browsers

Michael Weitnauer[1]

[1] *Institut für Rundfunktechnik, 80939 München, E-Mail: weitnauer@irt.de*

Introduction

Simple audio playback and more sophisticated operations used not to be natively possible with Internet Browsers until a few years ago, hence Web developers had to use embedded Flash applications or Browser Plugins like QuickTime for such implementations. Due to security issues and platform dependencies both of the mentioned possibilities are nowadays deprecated and even disabled in the default configuration of some Browsers.

The standardisation and the implementation of HTML5 made it for the first time possible to embed and control audio or video elements as files or streams natively. However, the conduction of signal processing operations was still not possible. Hence, the W3C Audio Working Group started in 2011 an activity to standardise an API for sophisticated audio signal processing in Browsers.[1] The resulting *Web Audio API* (WAA) specification [2][3] evolved since then and has been widely adopted in Browsers.

Web Audio API

The WAA enables handling basic audio operations in an audio *context* and has been designed to allow modular routing. Basic audio operations are performed with audio *nodes*, which are linked together to form an audio routing graph, see Figure 1. Several sources also with different types of channel layout are supported. This modular design provides the flexibility to create complex audio functions with dynamic effects. The WAA *nodes* are controlled by JavaScript Code which is embedded in the HTML application and thus conducted on the end user device. The JavaScript Code is steering typically C/C++ implementations of the audio *nodes* in the Browser. The WAA nodes are then applied in a real-time process onto the input or generated signals.

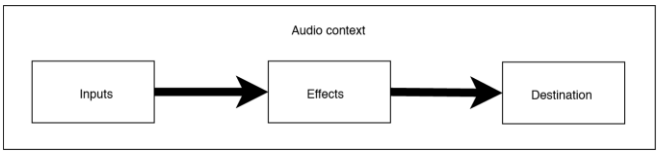

Figure 1: Modular routing concept of the Web Audio API (Source: https://developer.mozilla.org)

The Web Audio API specification defines currently, amongst others, these *nodes*:

- *ChannelSplitter* and *ChannelMerger* may be used for de-multiplexing and multiplexing of audio channels from and to one stream
- *BiquadFilter* & *IIRFilter* represent modules for low-order filter creation but also for sophisticated IIR filter design using coefficients
- *Gain nodes* are used to apply a gain value
- *DynamicsCompressor* provides a compression effect module
- *Delay nodes* may be used to apply a delay on the signals
- *Convolver* represents a module which can be fed with any impulse response to be convoluted with the input signal
- *Panner node* interfaces represent the position and behaviour of an audio source signal in space. They produce either a Stereo panning or a binaural output
- *Oscillator* and *WaveShaper* are used for signal generation and shaping
- *ScriptProcessor* is a very flexible module to apply any custom signal processing written in JavaScript and offers an interface to each audio sample, block by block

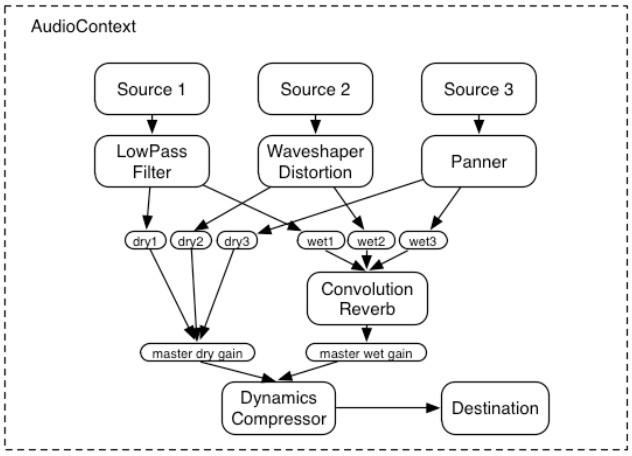

Figure 2: Example of a Web Audio routing with different *nodes* (Source: https://webaudio.github.io/web-audio-api/)

An arbitrary example of a combination of these *nodes* is depicted in Figure 2.

One of the key features of the Web Audio API is sample-accurate scheduling of sound playback with low latency [3] as it is also intended to be used for musical applications which require a very high degree of rhythmic precision. This also includes the possibility of dynamic creation of effects or *nodes*. Very useful for sophisticated web audio applications is also the possibility for an automation of audio parameters to be used for envelopes, fade-ins / fade-outs, granular effects, filter sweeps, etc.

Web Audio API Support

Even though the Web Audio API has not been released as a stable version 1.0 yet, the majority of Browser manufacturers are usually implementing the latest draft of

the specification. Figure 3 illustrates the current support in the major Browsers. All current versions of relevant Browsers support nowadays the Web Audio API. The only major exceptions are the Internet Explorer, whose development has been discontinued by Microsoft, and the Opera Mini. However, it has to be noted that not all Browser support the Web Audio API to the full or same extent.

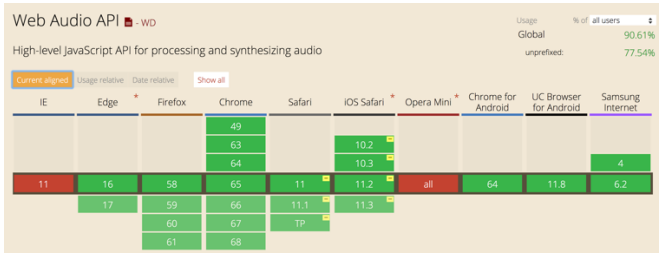

Figure 3: Web Audio API support in Browser versions. The current versions is grey highlighted. (Source: https://caniuse.com/#feat=audio-api)

Considering the market share of Browsers worldwide and in Germany (Figure 4), the latest Internet traffic user agent statistics show that the coverage of Browsers that do support the WAA is approximately 90% (worldwide) and 89% (Germany).

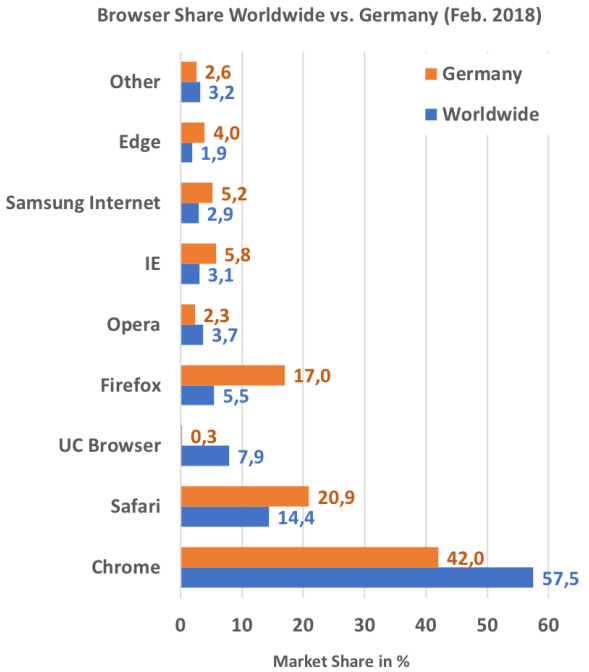

Figure 4: Browser market share worldwide vs. Germany in February 2018, based on Internet traffic and user agent detection (data source: http://gs.statcounter.com/)

This makes the Web Audio API to an attractive possibility to distribute sophisticated audio applications including binaural renderings.

Binaural aspects of Web Audio

There are basically three options to apply binaural renderings with the WAA.

(1) The *ConvolverNode* can be used along with an Head Related Transfer Function (HRTF) or a Binaural Room Impulse Response (BRIR). The convolution of the impulse response(s) with the dry input signal(s) is then applied in real-time.

(2) The *PannerNode* provides already an *HRTF* mode which applies a convolution with HRTFs from an internal database with the input signals that are positioned in a three-dimensional space.

(3) The *ScriptProcessorNode* can be used to implement and apply a custom binaural synthetisation, written in JavaScript. It has to be noted that the *ScriptProcessorNode* is already declared as deprecated in the latest draft of the specification.

(4) The recently introduced *AudioWorkletNode* will replace the *ScriptProcessorNode* as soon as the implementation in Browsers is complete and offers also the possibility to apply custom signal processing. The main difference to the *ScriptProcessorNode* will be that it is no more executed in the Browser UI thread but within its own Web Audio thread. This will reduce the risk of audible glitches and other artefacts due to performance issues.

Each of the mentioned possibilities has advantages and disadvantages and the choice of the appropriate solution depends on the use case. If the computational complexity has to be low, only the *ConvolverNode* and the *PannerNode* should be used as the conduction of the custom code for each block requires much performance of the CPU. Depending on the number of nodes, too many custom binaural synthetisations will cause audible artefacts. While the usage of own impulse responses with the *ConvolverNode* requires less CPU performance than a complete custom binaural synthetisation using a *ScriptProcessorNode*, the complexity is still higher than a binaural convolution with the *PannerNode* which uses HRTFs from an internal database.

Although the specification crucially lacks a documentation on defined HRTFs for the *PannerNode*, one can get more insights e.g. by researching the underlying source code for open-source browsers such as Google Chrome or Mozilla Firefox. According to T. Carpentier in [5], both Browsers Chrome and Firefox use the same collection of HRTFs which is adopted from the IRCAM Listen HRTF database [4]. The HRTFs are available in a 15° resolution for both azimuth and elevation and were created through averaging of the (diffuse-field equalized) impulse responses and a truncation to 256 samples at 44.1 kHz sampling rate (i.e. half the length of the original IRCAM HRTFs). For a one degree resolution of the HRTFs used by the *PannerNode* for azimuth and elevation, the existing HRTFs from the internal database are interpolated by the Browsers.

If performance is not an issue, the *ScriptProcessorNode* or *AudioWorkletNode* offer great flexibility for custom implementations. Especially the *AudioWorkletNode* seems to be promising for complex operations.

The *HRTF* mode of the *PannerNode* offers additional settings for the binaural rendering such as the distance model ("exponential", "linear" and "inverse"), the orientation and the extent of sound sources in terms of width, height and depth [3]. The listener's perception of the binaural *PannerNode* output can be modified with the WAA *listener*

interface. Additional to the three-dimensional position, the orientation of the *listener* can be changed on the fly by changing two vectors. The *forward* vector for the position at which the nose is pointing and the *up* vector representing the direction the top of a person's head is pointing. Any modifications to the *PannerNode* and the *listener* must be applied sample-accurate by the Browser.

Examples

The following sections describe examples of implementations which use the Web Audio API in the context of binaural rendering.

Custom HRTF usage with BinauralFIR

A library for the usage of own HRTF or BRIR databases in Web Audio applications has been released by IRCAM in 2015 [7][5]. It creates basically a custom WAA *node* that offers an API to control the position of sound sources. It may be mainly used for a binaural synthesis without changing head rotations.

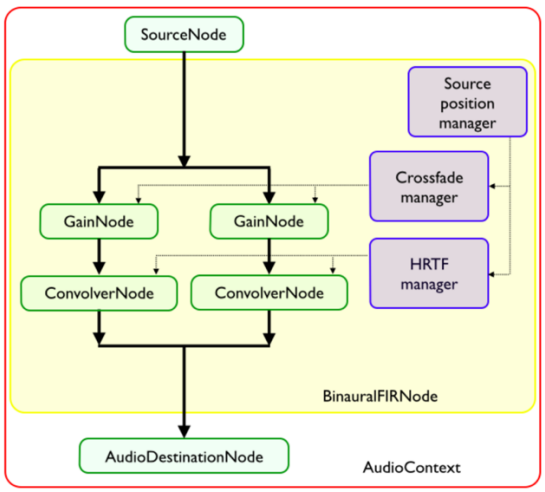

Figure 5: Basic concept of the binauralFIR library [5]

The binauralFIR library uses the *ConvolverNode* to apply the convolution of the impulse responses with the input signal. Figure 5 illustrates the basic concept behind. While the library is useful for own HRTF datasets, it requires compared to the *PannerNode* a higher CPU performance and is thus not recommended for a large number of sound sources. An example of the library can be found under [6].

Object-based audio rendering with bogJS

A new way to distribute audio content is the so-called object-based audio approach where the individual elements are only mixed together in the end device. This requires a rendering stage and offers the opportunity to render a special version for headphone listening. The IRT published a JS framework for this approach in 2016. [8][9]

Due to performance reasons, it uses only the *PannerNode* for binaural synthesis of sound sources. It offers multiple options to use audio input signals, either connecting to the HTML5 media elements which is useful for streams or longer files, or by requesting shorter files via a XMLHttpRequest and decoding them with the WAA *decodeAudioData()* method.

Example demos of the framework are published under [10], which demonstrates a user personalisation, and [11]. The latter uses an interface of the Browser to device sensors such as the accelerometer and magnetometer to detect the device orientation. If the user rotates the (mobile) device, the 360° video view and the WAA *listener* orientation is changed accordingly. By combining the WAA and the DeviceOrientation API [12], a head-tracked binaural application can be realised.

Ambisonics encoding and decoding

Google published recently libraries for Ambisonics encoding [13] and decoding [14], including a binaural synthesis of the decoded Ambisonics channels. Its rendering process is powered by the *GainNode* and *ConvolverNode*, ensuring the optimum performance. It supports currently Ambisonics decoding up to 3rd order and uses the SADIE HRTF database from the University of York [15].

The basic concept of the Ambisonics decoder library Omnitone for HOA is illustrated in Figure 6. A few examples of the Omnitone library can be found under [16].

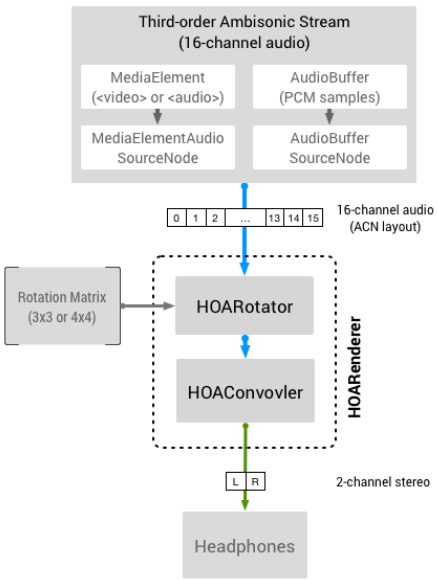

Figure 6: Basic concept of the Omnitone Ambisonics decoder library [14]

The Resonance Audio SDK from Google [13] provides not only a binaural Ambisonics rendering, but more importantly it offers also a room model for the auralisation of sound sources with their direct sound, early reflections and late reverb for a customizable room. The room model supports a large number of surfaces and the cuboid room dimensions can be chosen freely. The auralisation is realised by using a combination of many WAA *nodes* such as *ConvolverNode*, *BiquadFilterNode* and *GainNode*. A demo application of the room model web application using binaural rendering is published under [17].

Issues of the Web Audio API

As already mentioned, the WAA has not yet been released in a stable version 1.0. Although the Browser manufacturers are usually implementing the latest draft in their releases,

there are differences in the support of WAA features and the implementation of the features itself across browsers. Usually the Browser releases for Desktop devices are shipped with a more reliable and stable WAA implementation, while the releases for mobile devices such as Smartphones and Tablets differ sometimes in the implementation. It may be assumed that this situation will be improved once a stable version 1.0 has been published by the W3C WG.

Since the usage of uncompressed audio signals in Web applications is still not a realistic use case due to the large bandwidth needed for only a few audio channels, the distribution of encoded audio files and streams is common. Due to various reasons, the Browsers have different decoders integrated and also different implementations of those. This means in practice that no reliable decoding behaviour can be expected and thus should be tested. An implementation of such a test was published in [9].

Although Chrome and Firefox are using the same HRTF database, there are clearly large audible differences between the databases of other Browsers as the HRTFs for the *PannerNode* are not defined in the specification.

Future Development

A stable version 1.0 release is expected to be published by the end of 2018. The Audio WG of W3C will then start the work on the next major release of the specification, which is aimed to be published in 2020.

In version 2.0, there will likely be an interesting new feature for binaural rendering: the usage of custom HRTFs along with the *PannerNode,* potentially represented in the SOFA [18] format. This would allow a performance effective binaural synthetisation of other HRTFs.

The implementations of the already mentioned *AudioWorkletNode* in Browsers started and first implementation may be tested with a special flag, see [19].

Summary

This paper introduced the use of the Web Audio API for binaural rendering in Browsers. The WAA opens up new perspectives and offers great opportunities for deployment of binaural applications in many contexts as it is already widely supported in current versions of Internet Browsers. However, several restrictions of the current API were described, too. A selected list of examples for the usage of the WAA in the context of binaural rendering has been presented as well.

References

[1] W3C Audio WG Charter, URL: https://www.w3.org/2011/audio/charter/audio-2016.html

[2] Web Audio API, URL: https://github.com/WebAudio/web-audio-api

[3] Web Audio API Specification, URL: https://webaudio.github.io/web-audio-api/

[4] IRCAM Listen HRTF database, URL: http://recherche.ircam.fr/equipes/salles/listen/

[5] T. Carpentier, Binaural Synthesis with the Web Audio API, 1st Web Audio Conference (WAC15), Paris 2015

[6] BinauralFIR Example, URL: http://ircam-rnd.github.io/binauralFIR/examples/

[7] BinauralFIR Library, URL: https://github.com/Ircam-RnD/binauralFIR

[8] M. Weitnauer and M. Meier, bogJS – A JavaScript framework for object-based rendering in browsers, 2nd Web Audio Conference (WAC16), Atlanta 2016

[9] bogJS Library, URL: https://github.com/IRT-Open-Source/bogJS

[10] IRT Lab, URL: https://lab.irt.de/demos/object-based-audio/RadioDrama/

[11] IRT Lab, URL: https://lab.irt.de/demos/object-based-audio/360/

[12] DeviceOrientation Event Specification, URL: https://w3c.github.io/deviceorientation/spec-source-orientation.html

[13] Resonance Audio, URL: https://developers.google.com/resonance-audio/

[14] Omnitone Documentation, URL: https://googlechrome.github.io/omnitone/

[15] SADIE HRTF database, URL: https://www.york.ac.uk/sadie-project/GoogleVRSADIE.html

[16] Omnitone Examples, URL: https://rawgit.com/GoogleChrome/omnitone/master/examples/hoa-renderer.html

[17] Resonance Audio Room Model Examples, URL: https://cdn.rawgit.com/resonance-audio/resonance-audio-web-sdk/master/examples/room-models.html

[18] AES Spatial acoustic data file format, AES69-2015

[19] AudioWorkletNode implementation in Chrome, URL: https://developers.google.com/web/updates/2017/12/audio-worklet