Binaural synthesis with the Web Audio API

Thibaut Carpentier

Binaural synthesis with the Web Audio API

Thibaut Carpentier
UMR STMS IRCAM-CNRS-UPMC
1, place Igor Stravinsky
75004 Paris, France
thibaut.carpentier@ircam.fr

ABSTRACT

The Web Audio API is a powerful new platform for audio rendering within the browser and it provides a great opportunity for large deployment of audio applications. This paper explores the possibilities and limitations of the API for 3D sound spatialization based on binaural synthesis over headphones. The paper examines different processing structures and presents a new web-based server which allows the user to load individualized Head-Related Transfer Functions from a remote database.

Categories and Subject Descriptors

H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing; D.3.2 [**Programming Languages**]: Language Classifications; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Algorithms

Keywords

Sound spatialization, binaural rendering, 3D audio, Web Audio API, virtual loudspeakers, HRTF

1. INTRODUCTION

The way audiovisual media is consumed and experienced has changed substantially in recent years with the widespread use of mobile devices. The number of audiovisual content sources is multiplying, our exposure to media is increasing and the time devoted to it is rising steadily. Almost all mobile phones and tablet computers are now capable of playing music and/or video and they typically come with a stereo headset. Basically everybody who owns a mobile phone is thus a potential headphone user, and listening habits are evolving accordingly.

At the same time 3D technologies are also developing with the intention of providing the user with increased sensations of space. Concomitantly with the rise of 3D movie productions, 3D audio content for audio and audiovisual consumer applications is emerging.

1st Web Audio Conference (WAC), January 2015, Paris, France.

The expression "binaural technology" covers various methods of sound recording, synthesis and reproduction, which allow for delivering 3D spatial audio content over headphones. For instance, binaural recordings can be made by placing (miniature) microphones in the ear canals of a listener; when played-back over headphones such recordings can produce authentic auditory experience including its spatial aspects. On the other hand, binaural signals can be synthesized by digital signal processors in order to virtually position a sound arbitrarily around a listener. Such spatialization method completely overcomes the limitations of conventional stereophonic techniques.

Along with the fifth revision of the HTML standard, the World Wide Web Consortium (W3C) is developing a new application programming interface (API) for processing and synthesizing audio in web applications: the Web Audio API (WAA) provides specifications and description of a high-level JavaScript (JS) API for audio rendering in the browser [22]. Its goal is to include capabilities for mixing, processing, filtering and other common digital audio tasks. The WAA relies on an audio graph paradigm where a number of node objects are connected together to define the overall signal chain. In addition to the audio nodes natively provided by the API it is also possible to write custom effects directly in JS.

The global context (societal, cultural, technological and economical) thus creates favourable conditions for the development and large deployment of binaural audio applications and content. To that purpose, the Web Audio API appears as a tool to potentially reach broader audience, adapt productions to new listening habits and also to explore new paradigms (e.g. collaborative spatialization, mobile interactive applications). As a matter of fact, several initiatives have already been launched e.g. by major broadcast institutions: in 2013, Radio France presented nouvOson [17], a web platform dedicated to multichannel and binaural audio productions (documentaries, dramas, music, etc. available in 5.1 or binaural format); similarly BBC Radio [2] uses the WAA to broadcast radio drama or live orchestral music also in 5.1 and binaural format; etc.

The intention of this paper is to relate the development of various prototypes for binaural synthesis using the WAA. It is not our aim to release a "ready-to-distribute" binaural application, but to discuss the possibilities and limitations of the current WAA for such kind of signal processing. This work can thus be helpful to developers willing to create their own binaural processors.

This paper is organized as follows: Section 2 briefly presents the principle of binaural synthesis. Section 3 relates the implementation of a binaural panner with HRIR implemented as finite impulse response filters. In Section 4 we investigate the use of parametric models of HRTFs within the Web Audio API. Section 5 presents a HRTFs file format which allows for server/clients usage. Finally Section 6 evokes possible applications or future extensions.

2. BINAURAL TECHNOLOGY

The term "binaural hearing" refers to being able to integrate information that the auditory system and the brain receive from the two ears. Indeed our auditory percepts are essentially built on the basis of two inputs, namely the sound-pressure signals captured at our two eardrums. One remarkable property of humans' binaural hearing is its ability to localize sound in three-dimensional space to an accuracy of a few degrees. It is the direction-dependent characteristics of the sound signals reaching our two ears which enable us to localize the sound sources. Psychophysical studies have shown that various mechanisms are involved in the human auditory system for sound localization [24]. For sounds located in the horizontal plane, the angular direction is predominantly determined from interaural time differences (ITD) and interaural level differences (ILD), whereas sound elevation mainly depends on direction-dependent spectral cues generated by the obstruction of an incoming sound wave by the listener (diffraction and scattering effects of the pinna, head, and torso).

These acoustic interactions of an incoming sound with the listener's anatomy can be described by spatial filters called head-related transfer functions (HRTFs) or equivalently head-related impulse responses (HRIRs). HRTFs completely and uniquely capture all linear properties of the sound transmission and they contain all proposed descriptors of localization cues.

As a consequence, any sound source can be virtually simulated anywhere in the 3D auditory space by filtering an audio signal with the HRTFs corresponding to the desired location and presenting the resulting binaural signals over headphones (cf. Fig 1). Such audio processing is denoted "binaural synthesis".

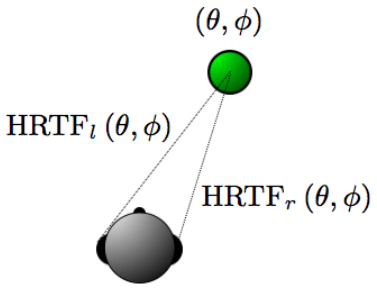

Figure 1: Left and right HRTFs for a source located at (θ, ϕ).

Since they depend on anatomic features such as the size and shape of head and ears, the cues for sound localization (especially the spectral cues) are idiosyncratic and HRTFs differ considerably among individuals. Measuring the HRTFs of a listener is a tedious task and it is yet restricted to a few laboratories (the measurements are often made in anechoic conditions). However databases of HRTFs for several hundreds of human subjects are available (see e.g. [36][26]) and can be used in a binaural synthesizer.

3. BINAURAL SYNTHESIS USING HRIR

After measurement and post-processing, HRTFs are usually stored as a set of finite impulse response (FIR) filters represented in the z-domain:

$$\mathrm{HRTF}_k\,(\theta, \phi) = \sum_{n=0}^{N-1} b_{k,n}\,(\theta, \phi) \cdot z^{-n} \qquad (1)$$

where $k = l$ or r for the left or right ear respectively and $\{b_{k,n}\,(\theta, \phi)\}_{n \in [0..N-1]}$ are the N filter taps for a source located at azimuth θ and elevation ϕ.

Given a source signal $x(t)$, the binaural signals for simulating the source at (θ, ϕ) are obtained by filtering:

$$\begin{aligned} y_l(t) &= \mathrm{HRTF}_l\,(\theta, \phi) * x(t) \\ y_r(t) &= \mathrm{HRTF}_r\,(\theta, \phi) * x(t) \end{aligned} \qquad (2)$$

Such linear convolution is an expensive operation cpu-wise (even though HRTFs are usually only a few milliseconds long). It can however be efficiently implemented thanks to transformation to the spectral domain (complexity $\mathcal{O}(N \log_2 N)$ instead of $\mathcal{O}(N^2)$).

3.1 The PannerNode Interface

3.1.1 *Current implementation*

The Web Audio API provides a *PannerNode* which allows to spatialize an incoming audio stream in three-dimensional space. The *PannerNode* currenlty supports two panning algorithms: an equal-power panning law and a so-called "HRTF algorithm" which performs binaural synthesis (and which turns out to be the default mode of the *PannerNode*). Although the specifications crucially lack documentation on this topic, one can get more insights e.g. by examining the underlying source code for open-source browsers such as Google Chrome or Mozilla Firefox. For this *PannerNode* Google Chrome relies on the Blink engine [7] which is a fork of the WebCore component of WebKit. Similarly Mozilla Firefox uses its own fork of Blink. Google Chrome and Mozilla Firefox are then based on the same rendering engine.

The Blink implementation code synthesizes the binaural signals through FFT-based convolution. The convolution kernel is not partitioned thus introducing an extra latency of half the FFT size. The source code embeds one set of HRTFs which, according to the code comments, is derived from the IRCAM Listen HRTF Database [36] through averaging of the (diffuse-field equalized) impulse responses and truncation to 256 samples at 44.1 kHz sampling rate (i.e. half the length of the original IRCAM HRTFs). For other sampling rates, the truncated responses are resampled. The resulting HRTF dataset is dubbed "IRC_Composite". In order to further reduce the computational cost, the HRIRs are again truncated upon loading in the processing engine: a leading delay is determined (as the average group delay) and

rendered by a delay line while the remaining part of the impulse response is loaded into the convolver(s). Whenever the source position is changed (through the *setPosition* method of the *PannerNode*), the delay lines and the kernels of the convolvers are updated accordingly. The varying delay is implemented by linear interpolation in-between the delay times with a smoothing time of 20 milliseconds. On the output of the convolvers a linear crossfade is applied with a transition time of 45 milliseconds.

Strangely enough, the HRTF panning mode of the *PannerNode* also supports stereo input streams, in which case the output signals are computed as:

$$\begin{aligned} y_l(t) &= \mathrm{HRTF}_l\,(\theta,\phi) * x_l(t) \\ y_r(t) &= \mathrm{HRTF}_r\,(\theta,\phi) * x_r(t) \end{aligned} \tag{3}$$

The interpretation of such formulae is doubtful.

3.1.2 Limitations

It is very much appreciated that the WAA provides binaural synthesis capabilities and the current implementation offers very decent spatialization effects. However we believe that the current API could be slightly improved in order to further widen the field of possibilities and provide an even better user experience.

It is known that non-individualized HRTFs may suffer from many limitations (in-head localization, inaccurate lateralization, poor vertical effects, weak front-back distinction, unwanted coloration effects, etc.). The fact that the WAA is constrained to one set of HRTFs is thus a restriction both for the end-users and for the application developers. Offering the possibility to load other sets of HRTF data would allow i) the user to select HRTFs that "suit" him/her and ii) the developers/researchers to create new applications; for instance the WAA could become a platform for large scale investigation and evaluation of binaural reproduction and for assessing individualization techniques (customization methods, adaptation process, etc.).

In addition to that, we believe that the API should provide further details and documentation on the HRTF set currently in use (which technique was used for averaging the data, what motivated the half-length truncation, etc.) as this may have a great impact on the spatial quality of the renderer.

It is worth noting that the underlying C++ implementation of the *PannerNode* supposedly has the possibility to load other HRTF sets, although this feature is not available in the high-level JS API.

3.2 The BinauralFIRNode

Individualization of HRTFs is a large field of research and one core topic of the BiLi project [3]. In order to overcome the current limitations of the *PannerNode* we developed a new binaural synthesis engine which allows to load custom HRTFs data set and to process them as FIR filters. The implementation is straightforward as it only relies on native nodes: HRIRs are loaded into *AudioBuffers* structures in order to be used as filter kernels of the *ConvolverNodes* which perform the HRIRs convolution. Whenever the source is moved in space the kernels are updated according to the new position and *GainNodes* are used to apply a crossfade (with adjustable duration) for smooth transition. A schematic of the audio graph is depicted in Fig. 2. The resulting audio processor is called *BinauralFIRNode*; the JS source code is made publicly available [5] and released under the BSD-3-Clause license.

The Source Position Manager can be further improved by building a k-d tree of all HRTF positions (upon loading a new HRTF set). This makes all further queries on the data set much faster than a brute force nearest neighbor search (complexity $\mathcal{O}(\log n)$ instead of $\mathcal{O}(n)$). Several JS k-d tree implementations can be found (e.g. [14] has been successfully used).

A minimalistic graphical user interface has been developed (Fig. 3) using jQuery-Knob [15]. It is obviously possible to spatialize multiple sources simultaneously by inserting several *BinauralFIRNodes* in the audio graph. Fig. 4 presents another user interface with multiple sources.

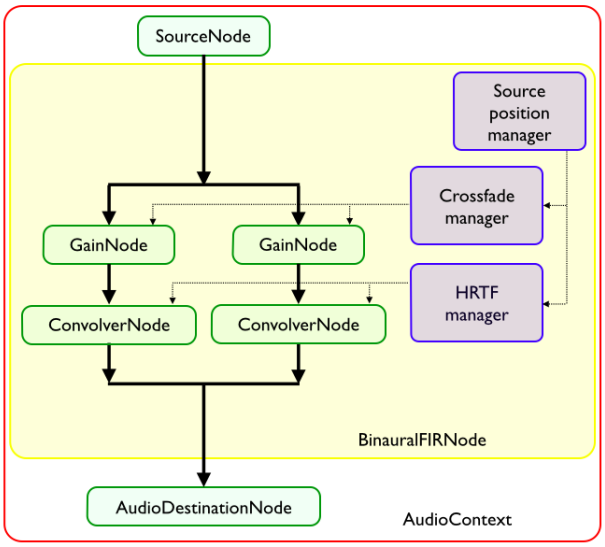

Figure 2: Overview of the *BinauralFIRNode* (for the sake of simplicity, the processing chain is displayed for one ear only; the processing is similar (i.e. two *GainNodes* and two *ConvolverNodes*) for the other ear.

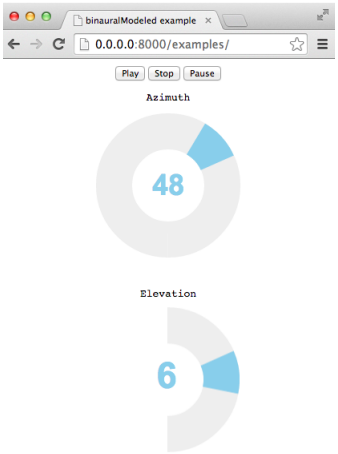

Figure 3: Minimalistic user interface for controlling the azimuth and elevation angle of one sound source.

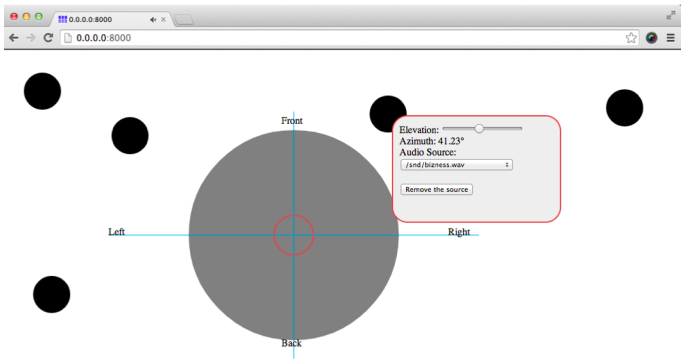

Figure 4: Another user interface (view from above) with multiple sources. The black circles represent sound sources. Clicking on a circle allows to edit the azimuth, elevation and select the associated sound file. It is also possible to dynamically add or remove source elements.

4. BINAURAL SYNTHESIS USING PARAMETRIC MODELS

For practical real-time applications, HRTFs are frequently approximated by a model in order to avoid heavy processing cost. This section presents a commonly used model of minimum-phase + pure-delay HRTF approximation [28].

4.1 Monaural delay and minimum-phase filter

4.1.1 Background

Being a causal and stable filter, an HRTF can be decomposed into a minimum-phase part H_{min} component and an all-pass component H_{exc}:

$$\begin{aligned} \text{HRTF} &= \text{mag} \cdot \exp(\mathbf{i} \cdot \varphi) \\ &= \text{mag} \cdot \exp(\mathbf{i} \cdot mph) \cdot \exp(\mathbf{i} \cdot exc) \\ &= H_{min} \cdot H_{exc} \end{aligned} \tag{4}$$

with $\mathbf{i} = \sqrt{-1}$ the imaginary number and $H_{exc} = \exp(\mathbf{i} \cdot exc)$. The phase φ of each HRTF is thus decomposed into the minimum-phase mph and the phase of the all-pass component i.e. the excess phase denoted exc. The minimum-phase mph is related to the magnitude spectrum through the Hilbert transform $\mathcal{H}$ (see [34], section 11):

$$mph = \Im\{\mathcal{H}(-\log(\text{mag}))\} \tag{5}$$

where $\Im\{\cdot\}$ represents the imaginary part. The excess phase of HRTF is usually almost linear (up to approximately 8 or 10 kHz) [32][28] and the phase information of the higher frequencies is not used by the auditory system to estimate the directions of arrival. It is thus possible to build a simplified model of HRTF where the all-pass component H_{exc} is replaced by a pure delay τ (which of course depends on the direction (θ, ϕ) of the HRTF). This pure delay is referred to as monaural delay.

Ultimately, filter design techniques can be applied in order to approximate the minimum-phase part of HRTF H_{min} with an infinite impulse response (IIR) filter [27][28]. Due to numerical stability reasons (see [34], section 6), the resulting IIR digital filter needs to be decomposed into a cascade of first order or second order sections (also known as biquad filters). For the sake of simplicity, we will consider only second-order sections for the remainder of this paper (anyway first-order sections can be seen as a particular case of second-order sections).

In summary, the parametric model consists in approximating each HRTF with a minimum-phase cascade of second order sections (magnitude spectrum) and a monaural delay. Several perceptual studies (see e.g. [37][27]) have shown the validity of such simplified model.

Besides computational efficiency, this parametric model may also be interesting e.g. in case of non-individualized listening: indeed it would be possible to scale the monaural delays according to the radius of the listener's head, which is a first step towards individualization of HRTFs.

The remainder of this section relates the implementation of such parametric model with the WAA.

4.1.2 SecondOrderSectionsNode

In the WAA, second-order filters (two poles two zeros) are implemented in the *BiquadFilterNode*. The public interface of this node provides access to the filter type ("lowpass", "highpass", etc.) and standard control parameters: cutoff frequency, gain and resonance factor (Q). In the underlying low-level code, the control parameters are mapped to the coefficients of the filters thanks to well-known "cookbook formulae". Unfortunately it is not possible for the JS developer to directly set the filter coefficients, making the *BiquadFilterNode* unusable for filter design techniques.

In order to overcome this limitation, we developed a custom JS biquad node. This custom node offers the possibility to:

- directly set the low-level filter taps (numerator and denominator of the z-domain filter); it is however the caller's responsibility to ensure that the coefficients provided are stable;
- serially-cascade several second-order sections.

The source code of this node has been released [6].

Two main issues were faced during the development of this module:

- at the time of writing, custom effects are handled by the *ScriptProcessorNode* which can process audio directly using JS. The major drawback of this node is that it runs in the main UI thread (rather than in the audio processing thread); this makes it sensitive to UI interactions, such as resizing the browser window, which may break realtime constraints and provoke audio dropouts. Fortunately it seems that forthcoming versions of the WAA will solve this issue by introducing the *AudioWorkerNode*.
- as most recursive process, biquad filters may generate denormalized numbers (depending on the hardware being used). It is known that denormalization processing can cause serious performance issues and cpu-spikes. Unfortunately it is not possible to "flush-to-zero" denormalized numbers within the *ScriptProcessorNode*. We had to use a common workaround consisting in adding a small inaudible DC offset in the feedback loop.

4.1.3 *FractionalDelayNode*

The monaural delays τ (or similarly the ITD) of the parametric model need to be implemented with variable fractional delay lines [29] (indeed, rounding delays to the nearest integer would result in approximately 3 degrees of error in perceived angle of arrival [27]).

The WAA *DelayNode* implements variable delay lines. As stated in the WAA specifications, a mechanism guarantees smooth transition ("without introducing noticeable clicks or glitches to the audio stream") when the delay time is varying. In the current Google Chrome and Mozilla Firefox code, this is achieved by means of a linear interpolation of the delay line. Linear interpolation is a particular case of 1st order FIR Lagrange interpolator. It is known that FIR Lagrange interpolators provide a good approximation of fractional delay for low frequencies [29]: their magnitude response is maximally flat about the DC; however it rolls off very quickly (especially for low-order filters), resulting in a low-pass filtering of the signal. One possible alternative is to approximate the fractional delay with IIR all-pass filters such as Thiran interpolators (unity magnitude response in the whole frequency band and maximally flat group delay at the DC frequency).

Similarly to the custom *SecondOrderSectionsNode* (Section 4.1.2) we developed a custom JS fractional delay node using 1st order IIR Thiran allpass interpolator [9]. The node could easily be extended to other interpolation schemes.

4.1.4 *Overall architecture*

The overall architecture (see Fig. 5) of the parametric processor is quite similar to the *BinauralFIRNode*; when varying the position of the source, the HRTF manager updates the monaural delay (updating the *FractionalDelayNode*) as well as the minimum-phase filters (i.e. the coefficients of the *SecondOrderSectionsNode*); *GainNodes* are used to crossfade during the transition state.

The JS implementation of the complete processor is released on GitHub [4].

4.2 Other parametric models

Several other parametric models have been proposed in the literature. Popular approaches are:

- physical modeling: this relies on analytical models of the HRTFs based on simplified geometrical descriptions of the head (e.g. spherical head model),
- perceptual modeling: this tries to construct perceptually relevant sets of HRTFs based on statistical analysis (e.g. averaging or multidimensional analysis).

Anyway, from a signal processing point of view, most of the proposed parametric models end up with filters structures that can be implemented in the WAA with either *SecondOrderSectionsNode*, *ConvolverNode*, *FractionalDelayNode* or any combination of the latter. These three processing nodes thus constitute a useful toolbox for most developers willing to create a binaural synthesis engine.

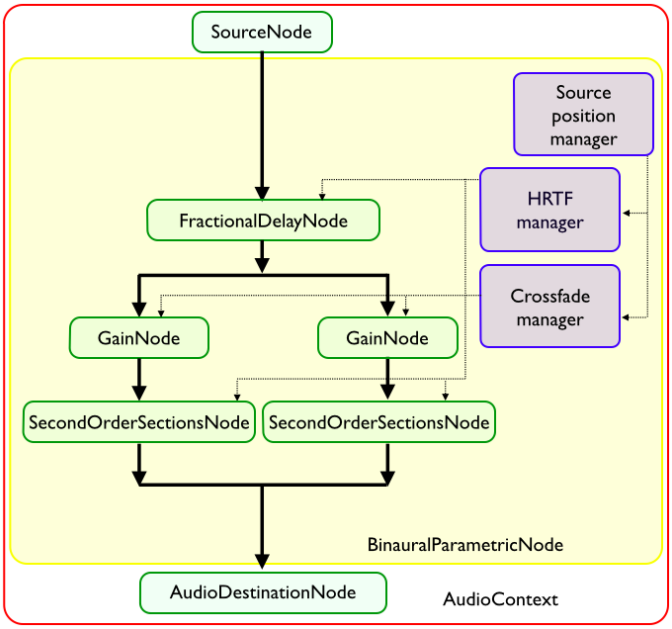

Figure 5: Overview of the *BinauralParametricNode* (for the sake of simplicity, the processing chain is displayed for one ear only; the processing is similar (i.e. one *FractionalDelayNode* two *GainNodes* and two *SecondOrderSectionsNode*) for the other ear.

5. HRTF FILE FORMAT

In 2013 a new data format for storing and exchanging HRTFs was proposed [30]: the Spatially Oriented Format for Acoustics (SOFA) is a self-described, network-transparent, machine-independent data format that supports the creation, access, and sharing of array-oriented spatial acoustic data. SOFA prescribes a set of specifications for consistent description of data and on top of these specifications, field-specific conventions can be elaborated; e.g. the *SimpleFreeFieldHRIR* convention represents HRTFs measured in free-field with an omnidirectional source and stored as impulse responses; the *SimpleFreeFieldSOS* convention (currently under discussion with the SOFA board) stores data with the parametric model described in Section 4. Main HRTFs databases are already available under the SOFA format [19] and APIs are provided for reading/writing data [20]. Furthermore the SOFA format was recently approved by the AES subcommittee SC-02 and assigned to the working group SC-02-08 on audio file interchange; this initiative is referred to as "AES-X212 HRTF file format standardization project" [1].

5.1 SOFA architecture

SOFA builds on top of netCDF [16] in order to re-use and benefit from existing powerful, widely used, data formats. The general architecture of the SOFA library is depicted in Fig. 6. From bottom to top:

- zlib [21] is used for data compression,
- HDF5 [10] is the numerical container,
- HDF5 HL is a high-level API to HDF5,
- curl [8] is used for remote data transfers (see Section 5.2),

- netCDF4 [16] is the data format,
- netCDF C++ is a high-level API to netCDF4,
- the SOFA library itself has two layers: one layer implements the SOFA specifications i.e. the requirements common to all kinds of SOFA files (not only HRIR); another layer implements the different sets of SOFA Conventions (such as e.g. the *SimpleFreeFieldHRIR* convention).

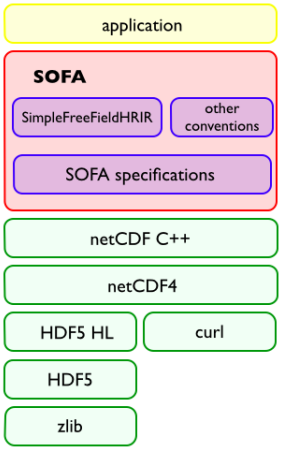

Figure 6: Architecture of the SOFA library.

5.2 SOFA and OpenDAP

As SOFA relies on netCDF, it is compatible with the Open-source Project for a Network Data Access Protocol (OpenDAP, [18]). OpenDAP includes standards for encapsulating structured data and allows for data transport over HTTP. More specifically an OpenDAP server can host a collection of data (typically in HDF or netCDF format) and serve them to remote clients (see Fig. 7) such as web browsers, web applications, graphics programs or any DAP-aware software (e.g. Mathworks Matlab). Hyrax [12] is an OpenDAP server that supports the netCDF format. An Hyrax server has been installed at IRCAM [13] in order to host HRTF databases under the SOFA format.

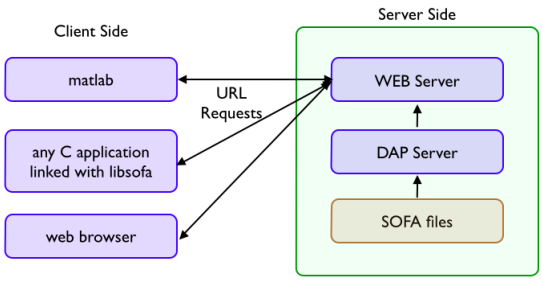

Figure 7: Remote access to SOFA HRTFs files through OpenDAP protocol.

Standard HTTP requests can be sent to the OpenDAP server in order to query the metadata of the files or to retrieve the numerical data (in whole or partially). The queries come in the form of specially formed URLs and the replies consist of MIME documents. All the requests start with a root URL, and they all are in the form of a GET, using a suffix on the root URL and a constraint expression to indicate which service is requested and what the parameters are. For instance the OpenDAP ASCII Service returns an ASCII representation of the requested data. Given a root URL:

`http://hrtf.ircam.fr/listen/irc_1002.sofa`

one can send ASCII requests to the server by appending suffix ".ascii?" to the URL:

`http://hrtf.ircam.fr/listen/irc_1002.sofa.ascii?`

The server returns a simple text message containing all the numerical data. It is further possible to query a single variable within the sofa file, e.g.:

`http://hrtf.ircam.fr/listen/irc_1002.sofa.ascii?SourcePosition`

returns the matrix of source positions (with coordinates given by azimuth, elevation and distance):

```
SourcePosition[0], 0, -45, 1.95
SourcePosition[1], 15, -45, 1.95
SourcePosition[2], 30, -45, 1.95
SourcePosition[3], 45, -45, 1.95
SourcePosition[4], 60, -45, 1.95
...
```

Such HTTP requests can be easily performed from JS code using the *XMLHttpRequest*, then parsed with e.g. regular expressions, such that an HRTFs loader can be directly integrated within a binaural processor in the WAA. In summary Fig. 8 depicts a diagram of a general framework for a binaural synthesizer running in a web browser and fetching HRTFs from a remote server (discussing the possibilities and difficulties of audio/video streaming to the WAA is however beyond the scope of this paper).

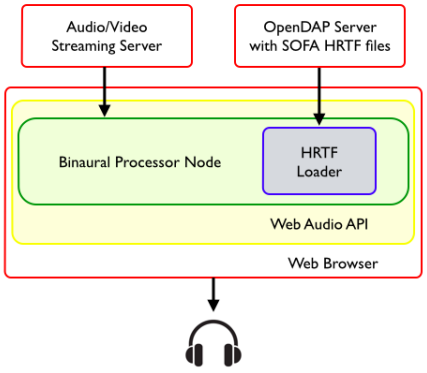

Figure 8: General framework of a binaural processor in the Web Audio API.

6. APPLICATIONS AND POSSIBLE EXTENSIONS

Binaural synthesis is potentially useful in many applications where spatial sound information is of importance. This is the case in various contexts such as virtual reality, gaming, computer music, telecommunications (e.g. audio/video chat), advanced human-computer interfaces, auralization of architectural acoustics [28], etc. It is not the purpose of this paper to detail all of them; Section 6.1 emphasizes one particular example which appears to be valuable to TV or radio broadcasting [17][2].

6.1 Multichannel downmixing

Multichannel audio/visual contents are mostly produced with a channel-based approach: the audio file or audio stream consists of M channels where each channel conveys the signal feeding one loudspeaker of a predefined multichannel layout (such as 5.1 ITU-R BS 775 or other surround configurations). The multichannel stream can be "downmixed" using binaural synthesis based on the "virtual loudspeakers" paradigm ([25][31][33][35]): the HRIRs corresponding to the position of each virtual speaker is convolved with that speaker feed and the convolution products for each of the ears are then summed giving the binaural signal for each ear.

In this approach it might also be helpful to use Binaural Room Impulse Responses (rather than HRIRs measured in anechoic conditions) or to add artificial reverberation with *ConvolverNodes* as room effect is known to positively affect externalization and depth perception (see e.g. [28][24][23]).

With such virtual speakers technique the broadcasting server delivers a unique version of the audio production, and on the client side the user is able to decode the stream according to its listening conditions (either headphones or surround speakers if available).

6.2 Head-tracking

In everyday perception we use head motion along with our aural and visual senses. The introduction of head-tracking device into binaural systems is therefore a crucial step as it introduces interaction of the processor and the listener. The proficiency of dynamic synthesis has been demonstrated (e.g. [28][23]) notably to reduce front/back reversals or to resolve cone-of-confusion errors, even for non-individualized listening conditions.

A basic dynamic processor has been implemented in the WAA using head-tracking via a standard webcam camera. We used the *headtrackr* JS library [11] which was straightforward to integrate; the library regularly generates face tracking events which are used to update the positions of the sources of the *SecondOrderSectionsNode*. Informal listenings revealed satisfactory rendering with acceptable latency; anyway more detailed listening experiments should be done in the future.

6.3 Cross-talk cancellation

Future work may consider the implementation of a cross-talk cancellation (CTC) processor for transaural reproduction [28]. CTC is required to transcode binaural signals for stereo speakers restitution; this enables 3D spatialization effects with only a pair of transducers (however with the constraints that the listener is located at the sweet spot of the system, and that the loudspeakers are far enough from walls or obstacles so that acoustical reflections can be neglected). Coding a CTC node with the WAA is rather straightforward: it only implies *ConvolverNodes* or *SecondOrderSectionsNodes* (if the CTC are implemented with FIR or IIR filters respectively), and no crossfading is involved in the audio graph (as CTC depends only on the loudspeakers span).

7. CONCLUSIONS

This paper discussed the use of the Web Audio API for binaural synthesis applications. The WAA opens up new perspectives and offers great opportunities for deployment of binaural applications in many contexts. However several restrictions of the current API were pinpointed (not possible to load custom HRTFs set or to directly set Biquad filter coefficients, *ScriptProcessorNode* suffering from interferences with the UI thread, etc). Several processing nodes were developed, extending the capabilities of the current *PannerNode*. Finally we presented an OpenDAP server allowing for remote access to HRTF databases.

8. ACKNOWLEDGMENTS

This work was funded by the French FUI project BiLi ("Binaural Listening", [3], FUI- AAP14) with support from "Cap Digital Paris Region", and by the French ANR project WAVE (`http://wave.ircam.fr`, ANR-12-CORD-0027).

The author would like to thank Arnau Julià Collados for developing the JavaScript codes and Samuel Goldszmidt for helpful and fruitful discussions.

9. REFERENCES

[1] AES X212 standard.
`http://www.aes.org/standards`
(accessed February 1, 2015).

[2] BBC binaural website.
`http://www.bbc.co.uk/rd/projects/binaural-broadcasting`
(accessed February 1, 2015).

[3] BiLi project.
`http://www.bili-project.org`
(accessed February 1, 2015).

[4] Binaural Parametric Model JS library.
`http://github.com/Ircam-RnD/binauralModeled`
(accessed February 1, 2015).

[5] BinauralFIR JS library.
`http://github.com/Ircam-RnD/binauralFIR`
(accessed February 1, 2015).

[6] Biquad Filter JS library.
`http://github.com/Ircam-RnD/biquad-filter`
(accessed February 1, 2015).

[7] Blink rendering engine.
`http://www.chromium.org/blink`
(accessed February 1, 2015).

[8] cURL library.
`http://curl.haxx.se`
(accessed February 1, 2015).

[9] Fractional Delay JS library.
`http://github.com/Ircam-RnD/fractional-delay`
(accessed February 1, 2015).

[10] HDF5 library.
`http://www.hdfgroup.org/HDF5`
(accessed February 1, 2015).

[11] Headtracking via webcam.
`http://github.com/auduno/headtrackr`
(accessed February 1, 2015).

[12] Hyrax Data Server.
`http://www.opendap.org/download/hyrax`
(accessed February 1, 2015).

[13] Ircam OpenDAP Server.
`http://www.hrtf.ircam.fr`
(to be published soon).

[14] JavaScript k-d Tree Implementation.
`http://github.com/ubilabs/kd-tree-javascript`
(accessed February 1, 2015).

[15] jQuery Knob.
`http://github.com/aterrien/jQuery-Knob`

(accessed February 1, 2015).

[16] netCDF library.
http://www.unidata.ucar.edu/netcdf
(accessed February 1, 2015).

[17] nouvOson website.
http://nouvoson.radiofrance.fr
(accessed February 1, 2015).

[18] OpenDAP.
http://www.opendap.org
(accessed February 1, 2015).

[19] SOFA Conventions.
http://sofaconventions.org
(accessed February 1, 2015).

[20] SOFA library on SourceForge.
http://sourceforge.net/projects/sofacoustics
(accessed February 1, 2015).

[21] zlib library.
http://www.zlib.net
(accessed February 1, 2015).

[22] P. Adenot, C. Wilson, and C. Rogers. Web Audio API W3C Working Draft – 14 october 2014.
http://webaudio.github.io/web-audio-api/
(accessed February 1, 2015).

[23] D. R. Begault and E. M. Wenzel. Direct comparison of the impact of head tracking, reverberation, and individualized head-related transfer functions on the spatial perception of a virtual speech source. *J. Audio Eng. Soc*, 49(10):904 – 916, 2001.

[24] J. Blauert. *Spatial hearing: The psychophysics of human sound localization.* The MIT Press, Boston, 1997.

[25] J. Blauert and P. Laws. True simulation of loudspeaker sound reproduction while using headphones. *Acta Acustica united with Acustica*, 29(5):273 – 277, 1973.

[26] T. Carpentier, H. Bahu, M. Noisternig, and O. Warusfel. Measurement of a head-related transfer function database with high spatial resolution. In *Proc. of the 7th EAA Forum Acusticum*, Kraków, Sept. 2014.

[27] J. Huopaniemi and N. Zacharov. Objective and subjective evaluation of head-related transfer function filter design. *Journal of the Audio Engineering Society*, 47(4):218–239, 1999.

[28] J.-M. Jot, V. Larcher, and O. Warusfel. Digital signal processing issues in the context of binaural and transaural stereophony. In *Proc. of the 98th Audio Engineering Society Convention*, Paris, Feb. 1995.

[29] T. I. Laakso, V. Välimäki, M. Karjalainen, and U. K. Laine. Splitting the unit delay. *IEEE Signal Processing Magazine*, 13(1):30 – 60, January 1996.

[30] P. Majdak, Y. Iwaya, T. Carpentier, R. Nicol, M. Parmentier, A. Roginska, Y. Suzuki, K. Watanabe, H. Wierstorf, H. Ziegelwanger, and M. Noisternig. Spatially oriented format for acoustics: A data exchange format representing head-related transfer functions. In *Proc. of the 134th Convention of the Audio Engineering Society*, Roma, May 4-7 2013.

[31] A. McKeag and D. S. McGrath. Using auralisation techniques to render 5.1 surround to binaural and transaural playback. In *Proc. 102nd Convention of the Audio Engineering Society*, Munich, March 1997.

[32] S. Mehrgardt and V. Mellert. Transformation characteristics of the external human ear. *J. Acoust. Soc. Am.*, 61(6):1567 – 1576, June 1977.

[33] H. Møller. Fundamentals of binaural technology. *Applied Acoustics*, 36:171 – 218, 1992.

[34] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-Time Signal Processing (2nd Ed.)*. Prentice Hall, New Jersey, 1999.

[35] F. Richter. Bap binaural audio processor. In *Proc. of the 92nd Convention of the Audio Engineering Society*, Vienna, March 1992.

[36] O. Warusfel. LISTEN HRTF database.
http://recherche.ircam.fr/equipes/salles/listen/
(accessed February 1, 2015).

[37] F. L. Wightman and D. J. Kistler. The dominant role of low-frequency interaural time differences in sound localization. *J. Acoust. Soc. Am.*, 91(3):1648 – 1661, March 1992.