Design of a binaural synthesis processor in the browser using Web Audio API

Author:
Arnau Julià Collados

Supervisors:
T. Carpentier, S. Goldszmidt
and F. Vallverdú

A thesis submitted in fulfilment of the requirements for the degree of Audiovisual Systems Engineering in the IRCAM-Espaces Acoustiques et Cognitifs Team and Analyse des Pratiques Musicales Team

Paris and Barcelona, June 2014
Abstract

Design of a binaural synthesis processor in the browser using Web Audio API

by Arnau Julià Collados

English:

Binaural technology intends to recreate a 3D sound experience over headphones. This can be achieved by means of binaural recordings (using microphones inserted in the ear canal) or by binaural synthesis. The latter simulates virtual sources and room effect using dedicated filters, namely HRTFs.

The Web Audio API is a W3C standard API for processing and synthesizing audio in web applications. As it is still in a process of development and the documentation is scarce, a deep study of the Web Audio API has been done.

The aim of this project is to develop a real-time binaural synthesis processor in the browser using the Web Audio API. For the binaural synthesis approach, two different processing structures have been developed: synthesis of measured HRTF filters and synthesis of measured HRTF filters modeled as IIR filters and a delay.

Català:

La tecnologia binaural té com a objectiu recrear una experiència de so 3D a través d'auriculars. Això s'aconsegueix a través de gravacions binaurals (utilitzant micròfons inserits al conducte auditiu extern) o a través de síntesi binaural. La síntesi binaural simula fonts virtuals i l'efecte de sala utilitzant filtres anomenats filtres HRTF.

La Web Audio API és una API de processament i síntesi d’àudio per aplicacions web estandarditzada pel consorci W3C. Com aquesta tecnologia està encara en procés de desenvolupament i la documentació és escassa, s’ha fet un estudi profund de la tecnologia.

L’objectiu d’aquest treball és la realització d’un sistema de síntesi binaural al navegador utilitzant la Web Audio API. En aquest treball s’han implementat dos mètodes de síntesi binaural: la síntesi de filtres HRTF mesurats i la síntesi de filtres HRTF mesurats modelats a través de filtres IIR i un delay.
Castellano:

La tecnología binaural tiene como objetivo recrear una experiencia de sonido 3D a través de auriculares. Esto se consigue a través de grabaciones binaurales (usando micrófonos insertados en el conducto auditivo externo) o a través de la síntesis binaural. La síntesis binaural simula fuentes virtuales i el efecto de la sala usando filtros llamados filtros HRTF.

La Web Audio API es una API de procesamiento y síntesis de audio para aplicaciones web estandarizada por el consorcio W3C. Como esta tecnología está aún en proceso de desarrollo y la documentación es escasa, se ha hecho un estudio profundo de la tecnología.

El objetivo de este trabajo es la realización de un sistema de síntesis binaural en el navegador usando la Web Audio API. En este trabajo se han implementado dos métodos de síntesis binaural: la síntesis de filtros HRTF medidos y la síntesis de filtros HRTF medidos modelados a través de filtros IIR y un delay.
Acknowledgements

Thanks to Samuel Goldszmidt and Thibaut Carpentier for hosting me at IRCAM, and to help me developing this project. *Merci beaucoup!* Donar gràcies a en Sisco Vallverdú per fer el seguiment del meu projecte des de Barcelona durant aquests mesos.

Finalment, agraïr a la meva parella, amics i família, ja que m’han fet sentir molt estimat en tot moment.
# Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>09/06/2014</td>
<td>Document Creation</td>
</tr>
<tr>
<td>1</td>
<td>20/06/2014</td>
<td>Document Revision</td>
</tr>
<tr>
<td>2</td>
<td>06/07/2014</td>
<td>Document Revision</td>
</tr>
</tbody>
</table>

Document distribution list:

<table>
<thead>
<tr>
<th>Name</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arnau Julià Collados</td>
<td><a href="mailto:arnau.julia@gmail.com">arnau.julia@gmail.com</a></td>
</tr>
<tr>
<td>Thibaut Carpentier</td>
<td><a href="mailto:Thibaut.Carpentier@ircam.fr">Thibaut.Carpentier@ircam.fr</a></td>
</tr>
<tr>
<td>Samuel Goldszmidt</td>
<td><a href="mailto:Samuel.Goldszmidt@ircam.fr">Samuel.Goldszmidt@ircam.fr</a></td>
</tr>
<tr>
<td>Francesc Vallverdú</td>
<td><a href="mailto:francesc.vallverdu@upc.edu">francesc.vallverdu@upc.edu</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Written by</th>
<th>Reviewed and approved by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: 07/07/2014</td>
<td>Date: 08/07/2014</td>
</tr>
<tr>
<td>Name: Arnau Julià Collados</td>
<td>Name: Francesc Vallverdú Bayes</td>
</tr>
<tr>
<td>Position: Project Author</td>
<td>Position: Project Supervisor</td>
</tr>
</tbody>
</table>
# Contents

Abstract i  
Acknowledgements iii  
Revision history and approval record iv  
Contents v  
List of Figures vii  
List of Tables viii  
Abbreviations ix  

1 Introduction 1  
1.1 Introduction 1  
1.2 Objectives 1  
1.3 Requirements and specifications 2  
1.4 Methods and procedures 2  
1.5 Workplan 3  

2 State of the art 4  
2.1 Head-Related Transfer Function 4  
  2.1.1 Localization cues 5  
  2.1.2 Measuring HRTF 7  
2.2 Binaural Synthesis 8  
  2.2.1 Modelling HRTF by a minimum-phase filter and a pure delay 9  
2.3 Transaural Audio 11  

3 Project Development 13  
3.1 Basic experiments with the Web Audio API 14  
  3.1.1 What is the Web Audio API 14  
  3.1.2 How the Web Audio API works 15  
  3.1.3 The AudioNode Interface 17  
  3.1.4 The AudioParam Interface 19  
  3.1.5 The DelayNode Interface 20  
  3.1.6 The ConvolverNode Interface 22  
  3.1.7 The ScriptProcessorNode Interface 23  
  3.1.8 The BiquadFilterNode Interface 26
# List of Figures

2.1 Cone of confusion ............................................. 7  
2.2 Process of binaural synthesis ................................. 9  
2.3 Implementation of HRTF as minimum-phase and pure delay 10  
2.4 Transaural synthesis ......................................... 11  

3.1 Binaural synthesis processors ................................. 13  
3.2 Communication of threads for the update of currentTime property 16  
3.3 Start of a buffer with AudioBufferSourceNode .................. 17  
3.4 Difference between the a-rate and k-rate parameters .......... 19  
3.5 DelayNode in a loop ...................................... 22  
3.6 General diagram Binaural in Browser Application ............. 28  
3.7 First schema of BinauralFIR node ............................ 29  
3.8 Final schema of BinauralFIR node ........................... 30  
3.9 Flow diagram of setPosition method .......................... 32  
3.10 Allpass-interpolated delay line. .............................. 35  
3.11 Final schema of BinauralBiquad .............................. 37  
3.12 Module structure .......................................... 39  
3.13 User interface ........................................... 40  

4.1 Comparison of BinauralFIR node and PannerNode ............. 42  
4.2 Different sizes of ScriptProcessorNode of BinauralBiquad node 43  
4.3 Comparison of BinauralBiquad and BinauralBiquad with the native WAA BiquadFilterNode ........................................... 43  

A.1 Gantt Diagram ............................................. 55
### List of Tables

A.1 Work Package 1 ........................................... 47  
A.2 Work Package 2 ........................................... 48  
A.3 Work Package 3 ........................................... 49  
A.4 Work Package 4 ........................................... 50  
A.5 Work Package 5 ........................................... 51  
A.6 Work Package 6 ........................................... 51  
A.7 Work Package 7 ........................................... 52  
A.8 Work Package 8 ........................................... 52  
A.9 Work Package 9 ........................................... 53  
A.10 Milestones .............................................. 54  
C.1 Results iMac 2009 ........................................ 58
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRCAM</td>
<td>Institut de Recherche et Coordination Acoustique/Musique</td>
</tr>
<tr>
<td>WAVE</td>
<td>Web Audio View Edit</td>
</tr>
<tr>
<td>EAC</td>
<td>Espaces Acoustiques Cognitifs</td>
</tr>
<tr>
<td>HRTF</td>
<td>Head-Related Transfer Function</td>
</tr>
<tr>
<td>HRIR</td>
<td>Head-Related Impulse Response</td>
</tr>
<tr>
<td>ITD</td>
<td>Interaural Time Differences</td>
</tr>
<tr>
<td>ILD</td>
<td>Interaural Level Differences</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite Impulse Response</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>WAA</td>
<td>Web Audio API</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Introduction

The first browser was created in 1990 as a software application for retrieving static documents to the user. In less than 25 years, the browser has become the most important application on all our devices and it is becoming the new operating system. Now, with the Web Audio API, a World Wide Web Consortium (W3C) standard API for processing and synthesizing audio in web applications, the browser goes one step further and it becomes a platform to create audio rich web applications.

This degree’s thesis is carried out at IRCAM between the Espaces Acoustiques et Cognitifs (EAC) team and the Analyse des Pratiques Musicales (APM) team. It takes part in the WAVE Project under the supervision of Samuel Goldszmidt (research and development engineer of the APM team) and Thibaut Carpentier (research and development engineer of the EAC team).

1.2 Objectives

The purpose of this project is to develop a real-time binaural synthesis processor in the browser using the Web Audio API. Two different synthesis techniques will be developed: synthesis of measured HRTF filters and synthesis of measured HRTF filters modeled as IIR filters and pure delay. For this purpose, a deep study of the current state of the Web Audio API will be done.
1.3 Requirements and specifications

The requirements and specifications of this project are the following:

Requirements:

- To develop an application able to synthesize binaural audio signals in real-time in the browser by using two different synthesis techniques.
  - Ability to load custom hrtfs
  - Ability to do a smooth transition when changing the position
- To formalize use of the API for the designed software.
- To document the developed functionalities and components.
- To design and implement a demo with the developed software.

Specifications:

- To develop all the software in JavaScript and the Web Audio API following the W3C standards.
- To prevent audio glitches.

1.4 Methods and procedures

The EAC team from IRCAM is the creator of Spat Software [1], a software suite for spatialization of sound signals in real-time intended for musical creation, postproduction, and live performances with advanced spatialization algorithms.

The EAC team is constantly improving their software and extending it to new platforms. A new platform always implies new users and thus, possible new uses of the spatialization technologies. One of the bigger changes in the last decade has been the explosive growth in the Internet use. Thanks to the Internet, a lot of new technologies have been developed. The interest of EAC team in the browser is manifold:

- Reach a broader audience.
• Adapt to current listening habits (e.g., everybody listens to music on mobile devices, with headphones).

• Explore new paradigms (e.g., collaborative spatialization).

Thus, it is natural that the EAC team starts using web technologies. To be able to develop processing audio software on the web it is necessary to have audio processing tools in the browser. Thanks to the development of the Web Audio API now it is possible to develop web audio rich applications.

All that lead the EAC team to develop their first software in the browser, a binaural synthesis software. Thus, this project can be summarized in three main blocks:

• To test the current state of the Web Audio API and web technologies to develop complex audio software in the browser.

• To develop a binaural synthesis processor by using measured HRTF filters.

• To develop a binaural synthesis processor by using measured HRTF filters modeled as IIR filters and pure delay.

Therefore, this project starts a new line of development on the team by using the Web Audio API as audio processing tool.

In order to develop a binaural synthesis processor, HRTF data is needed. This project will use the IRCAM measured HRTF filters database and the measured HRTF filters modeled as IIR filters and pure delay database developed by the EAC team researchers.

1.5 Workplan

The work packages, the milestones and the Gantt diagram can be found in the Appendix A.
Chapter 2

State of the art

One purpose of binaural technology is to create a 3D stereo sound sensation for the listener by simulating to be in a room with virtual sources. This technology relies on a set of tools for recording or synthesize and render binaural signals to the listener ears. Binaural technology uses the eardrum signals, through which it is possible to perceive and locate the sound sources. Therefore, the binaural signals are formed by a pair of signals related to the left and right ears.

The binaural technology can be summarized in two different strategies:

- Binaural recording: Ability to record a sound scene at the entrance of ear canal. Thus, the recorded sound scene can be reproduced.
- Binaural synthesis: Ability to synthesize the signals at the entrance of ear canal, by means of dedicated filters, namely HRTFs. Thus, any sound scene can be synthesized.

This project will be focused on the binaural synthesis strategy.

2.1 Head-Related Transfer Function

Measurements and models of Head-related impulse response (HRIR) and the corresponding frequency domain functions (Head-related transfer function, HRTF) describe
the acoustic propagation between one source and the listener ears. Researches revealed that HRTFs comprise all the localization cues. The transfer functions evaluated at one specific azimuth and elevation are enough for the synthesis of realistic three-dimensional sound events for headphone or loudspeaker reproduction since HRTF collects all the information related to the physical propagation and diffraction around the head, body and ears of a listener. Consequently, as the acoustic wave is affected by the listener morphology, the HRTF is unique for each individual. The studies of Algazi, Avendano, and Duda [2], and Fels and Vorländer [3] compare several HRTF recordings of a dummy-head with various morphological setups to find the relative roles of head, pinna and torso. All these studies revealed that, although several elements modify the HRTF spectrum, one of the key elements is the pinna, which is responsible for the most distinguishable characteristics of the HRTF signal. The complexity of the pinna is a benefit from the point of view of localization of sounds in the space because it creates spectral features that change as a function of sound direction. Nevertheless, the fact that is highly dependent on the individual’s morphology is a challenge because the acquisition of individualized HRTFs is a tedious operation.

2.1.1 Localization cues

The HRTF contains information about cues for localizing a sound source. Two of the most important cues for localizing the angular position of a sound source are the Interaural Time Difference (ITD) and the Interaural Level Difference (ILD) [4]). These cues are based on the relative difference of the wavefront at the two ears. The ITD is the difference in arrival time of a sound between two ears. This time difference between the sound arrivals at the ears helps to identify the lateralization of the sound source. As HRTF includes this cue, it is possible to estimate it from the HRTF. The extraction of the ITD is based on the delays difference of the left and right HRTF. There are several methods to estimate the HRTF delay and one of them is the phase difference [5]. The phase can be expressed as:

\[
\tau_{\text{phase}}(f) = \frac{\psi(f)}{2\pi f}
\]
where $f$ refers to frequency, and $\psi(f)$ to the phase spectrum of a specific HRTF. Therefore, the ITD can be expressed as the phase difference:

$$ITD(f) = |\tau_{L-phase}(f) - \tau_{R-phase}(f)|$$

where $\tau_{L-phase}$ is the left HRTF phase spectrum and $\tau_{R-phase}$ is the right HRTF phase spectrum.

On the other hand, the ILD is the difference in sound pressure level reaching the two ears. Due to the shadowing of the sound wave by the head, a sound coming from a source located to one side of the head will have a louder intensity at the ear nearest the sound source. ILD is effective above approximately 1 kHz because longer wavelengths will diffract around the obstructing surface of the head [6]. ILD is defined as the difference magnitude spectrum of the left and right HRTFs. Larcher introduces a formula [5] to derive the ILD from the HRTF:

$$ILD = 10\log_{10} \frac{\int_{f_1}^{f_2} |H_L(f)|^2 df}{\int_{f_1}^{f_2} |H_R(f)|^2 df}$$

where $|H_{L,R}(f)|$ refers to the HRTF magnitude spectrum. As explained before, the ILD is more effective for frequencies above 1 kHz, therefore the frequency boundaries are usually evaluated between 1 kHz and 5 kHz [7].

When ITD and ILD cues are equal for two different positions, a confusion is created. Referring to Figure 2.1, note that a sound source at position A produces an identical ITD and ILD that a sound source at the B position. This is called the cone of confusion [8].

The duplex theory revealed the importance of interaural cues (ITD, ILD) for the lateralization of sound sources. However, as these cues are ambiguous (cone of confusion) they cannot fully explain how humans localize sounds. Thus, more information is needed to localize sounds. It has been proven that monaural cues also exist and contribute to sound localization. These cues are called spectral cues.

The spectral cues can help to disambiguate the cone of confusion. In other words they are especially important for localization in elevation. A spectral cue (SC) is defined as
a salient feature of the HRTF magnitude spectrum which may be detected, analyzed and interpreted by the auditory system to localize sounds. [7]. The HRTF spectrum magnitude embeds a lot of information thus, there a lot of possible SC candidates. That is why it is difficult to detect a spectral cue. Currently, two main theories about which are the important points in a HRTF for localization exist. The first establishes that the spectrum irregularities, such as peaks or notches, are the responsible of the localization of the sources. The second is based on the use of global features, which suggest that the localization judgment results from a comparison between the perceived spectrum and the HRTF spectra stored in the brain memory [9]. A differential point between the SC and the ITD and ILD cues is that the SC provides information about localization by using monaural processing (left and right ears separately) and binaural processing, while the ITD and ILD only use binaural processing.

2.1.2 Measuring HRTF

The traditional and more reliable method to obtain a HRTF set is through a real acoustical measurement. The procedure consists in recording a sound stimulus with two microphones inserted into the ears of an individual or a dummy-head. The subject sits in a chair, ideally inside an anechoic chamber, and a loudspeaker situated in a specific azimuth and elevation emits a signal that is recorded for the microphones. Several test signals can be used, but most often it consists of a swept-sine signal [10]. The position of the loudspeaker must have an angular resolution higher than 2° and the relative location between the loudspeaker and the individual has to be tracked [7]. These kind of
measurements need dedicated equipment and it is for these reasons that there are not too many places in the world where is possible to measure HRTF. There are two main measurement protocols:

- Blocked ear canal: the microphone is situated at the ear entrance.
- Open ear canal: the microphone is situated in the ear canal, close to the eardrum.

This method has a worst signal to noise ratio due to frequency resonances of the ear canal.

HRTF data can also be obtained by analytic processes, using computer models based on wave propagation and diffraction around a sphere or a replica of a human head. These methods are very promising, but they are not yet used in production environments.

After the measurement, a post-processing is applied, essentially to remove the contributions of the measuring equipment (speaker, microphone, amplifier, etc.). After this post-processing, we have clean individualized HRTFs. If these HRTFs are used in a non-individualized context (i.e. by someone else), the spectral cues may be perceived as timbre degradation rather than spatial cues. So, some spectral smoothing (or other techniques) have to be applied to somehow minimize the spectral effects.

## 2.2 Binaural Synthesis

Binaural synthesis is the simulation of binaural recording. Thus, its purpose is to synthesize the signals at the entrance of ear canal.

Binaural Synthesis consists in convolving a monophonic source signal with two filters that model the acoustic transfer function between one source location and the listener ears, that is to say, the Head Related Transfer Function for a specific azimuth and elevation. This methodology can be applied to measured HRTF, that can be measured as described in the 2.1 section, or modeled. The digital transfer function can be expressed as a Finite Impulse Response (FIR) filter:

\[
H(z) = h[0] + h[1]z^{-1} + h[2]z^{-2} + ... + h[N − 1]z^{−N−1}
\]
where $N$ is the number of samples of the Head-related impulse response, HRIR.

\[ x_m \]

\[ H_L \]

\[ H_R \]

\[ y_L \]

\[ y_R \]

**Figure 2.2:** Process of binaural synthesis where $H_L$ and $H_R$ are the HRTF filters, and $x_M$ the input mono signal. Extracted from [12]

### 2.2.1 Modelling HRTF by a minimum-phase filter and a pure delay

One of the problems in 3-D sound synthesis is the computational cost of the use of real HRTF data for the synthesis. However, measured HRTFs can be approximated by a simplified model. One commonly used HRTF model is composed of:

- A minimum-phase filter: accounts for the magnitude spectrum of HRTF.
- A pure delay: represents the temporal information contained in the HRTF.

The HRTF can be represented as,

\[ H(\omega) = |H(\omega)|e^{j\varphi(\omega)} \]

where $H(\omega)$ is the original HRTF, $|H(\omega)|$ is the magnitude response and $\varphi(\omega)$ is the phase response, which may be decomposed into the minimum-phase response $\mu(\omega)$ and the excess-phase response $\eta(\omega)$ [13].

\[ H(\omega) = |H(\omega)|e^{j\mu(\omega)}e^{j\eta(\omega)} = H_{mp}(\omega)e^{j\eta(\omega)} \]
where $H_{mp}(\omega)$ is the minimum-phase transfer function. The frequency dependent excess-phase component can also be decomposed into linear-phase and all-pass components.

$$\eta(\omega) = \eta(\omega)_{\text{linear}} \eta(\omega)_{\text{allpass}}$$

Plogsties et al. [14] showed that the allpass components can be removed without audible consequences. Thus, the all-pass component can be neglected without disturbing the spatial perception. Moreover, the linear-phase component $\eta(\omega)_{\text{linear}}$ can be replaced by a time delay without audible consequences, $\eta(\omega) = \omega \tau$ [13]. Therefore, the HRTF can be modeled as a minimum-phase transfer function (representing the ILD and SC cues) and a linear delay modeling the excess-phase response as it is showed in the Figure 2.3.

![Figure 2.3: Implementation of HRTFs using minimum-phase HRTF approximation and pure delays. Extracted from [13]](image)

The most direct way to model HRTF minimum phase transfer functions is to use an FIR filter design. A filter of the desired order is obtained by windowing the measured impulse response with a rectangular window. Another technique is to model the HRTF minimum phase transfer function filters with infinite impulse response (IIR) filters (for further information see [12]). The IIR HRTF model can be expressed as:

$$H_{mp}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-P}}{1 + a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_M z^{-M}}$$

where $H_{mp}$ is the minimum-phase transfer function, and the filter order is the maximum of M and P. Usually, high-order recursive filters (IIR) are highly sensitive to quantization of their coefficients and their practical implementation may lead unstable results. It is for that that they are usually implemented as a serially-cascade of biquad filters sections as:
Chapter 2. State of the art

\[ H_{mp}(z) = \frac{c_o^{(1)} + c_1^{(1)} z^{-1} + c_2^{(1)} z^{-2}}{1 + d_1^{(1)} z^{-1} + d_2^{(1)} z^{-2}} \cdot \frac{c_o^{(2)} + c_1^{(2)} z^{-1} + c_2^{(2)} z^{-2}}{1 + d_1^{(2)} z^{-1} + d_2^{(2)} z^{-2}} \cdot \cdots \cdot \frac{c_o^{(F)} + c_1^{(F)} z^{-1} + c_2^{(F)} z^{-2}}{1 + d_1^{(F)} z^{-1} + d_2^{(F)} z^{-2}} \]

With this cascade of biquads, it is possible to implement any high-order IIR filter.

2.3 Transaural Audio

Nowadays, binaural synthesis reaches very good level of virtual sound sources simulation. Binaural synthesis, by essence, is limited to headphones rendering. On the other hand, transaural synthesis allows the recreation of binaural signals over loudspeakers. Owing the nature of the loudspeakers transmission, acoustic crosstalk occurs. Nevertheless, with carefully chosen speakers feed signals, one can cancel out this crosstalk (at the ear level).

As showed in the figure 2.4, the signals at the listener ears for transaural synthesis are:

\[ Z_L = H_{LL}\hat{Y}_L + H_{RL}\hat{Y}_R \]
\[ Z_R = H_{LR}\hat{Y}_L + H_{RR}\hat{Y}_R \]
where $Z_L$ is the sound at listener left ear, $Z_R$ is the sound at listener right ear, $H_{LL}$ is the left ear HRTF filter for a virtual source at left loudspeaker position, $H_{RR}$ is the right ear HRTF filter for a virtual source at right loudspeaker position, $H_{RL}$ is the left ear HRTF for a virtual source at right loudspeaker position, $H_{LR}$ is the right ear HRTF for a virtual source at left loudspeaker position, $\hat{Y}_L$ is the signal emitted by the left loudspeaker and $\hat{Y}_R$ is the signal emitted by the right loudspeaker.

The solution to cancel the crosstalk is based on pre-processing the binaural signals $\hat{Y}_L$ and $\hat{Y}_R$ to guarantee that $Z_L = Y_L$ and $Z_R = Y_R$ where $Y_L$ and $Y_R$ are the left and right binaural signals:

$$\hat{Y}_L = \frac{Y_L H_{RR} - Y_R H_{RL}}{H_{RR} H_{LL} - H_{LR} H_{RL}}$$

$$\hat{Y}_R = \frac{Y_R H_{LL} - Y_L H_{LR}}{H_{RR} H_{LL} - H_{LR} H_{RL}}$$

With the crosstalk cancellation, the ears perceive the clean binaural signals.
Chapter 3

Project Development

Currently, the Web Audio API has a binaural synthesis processor called *PannerNode*. The figure 3.1 exhibits the different processing blocks that have been developed in this project and that will be further presented in the remainder of this chapter. The red block is the current binaural node in the WAA. The green blocks are all the developed software in this project. It can be summarized in two new binaural synthesis processors:

- **binauralFIR**: Binaural synthesis processor using measured HRTFs.
- **binauralBiquad**: Binaural synthesis processor using measured HRTFs modeled as IIR filters and pure delay. For the remainder of this report, we will refer to this node as *BinauralBiquad*.

![Figure 3.1: Binaural synthesis processors: the native WAA PannerNode and the developed BinauralFIR and BinauralBiquad](image_url)
3.1 Basic experiments with the Web Audio API

The first part of this project consists in exploring the possibilities and limitations of the Web Audio API to know how to focus our project. The purpose of this section is not to reproduce the Web Audio API specifications [15], but to explain the tricks and underdocumented parts found during the development of this project. This section aims to help people eager to develop audio applications with this exciting tool called Web Audio API.

3.1.1 What is the Web Audio API

The Web Audio API (WAA) is a high-level JavaScript (JS) Web API from the Audio Working Group of W3C for processing and synthesizing audio in web applications. This API was first drafted by Chris Rogers in December 2011 [16] and it is currently edited by Chris Wilson (Google) and Paul Adenot (Mozilla) [15]. The WAA is based on an audio routing graph paradigm, where AudioNode objects are connected together to create the audio scenario, similar to other audio applications such as e.g., Max/MSP [17] or Pure Data [18]. Unlike these two latest softwares, WAA is not a visual programming language.

Prior to the creation of the Web Audio API, the use of the audio in the browser was very basic. Thanks to the introduction of the audio HTML element in HTML5, it was possible to play audio files on the browser through a standard API. Nevertheless, the audio element only allows basic streaming audio playback. Thus, the next step was to create a processing audio API, the Web Audio API.

One of the powerful points of Web Audio API is that the actual audio processing is done in the underlying implementation of the browser, that is to say, in Assembly/C/C++ code. This low-level implementation allows to achieve low latency values -considering latency the time between the user execute an action and the sound is available at the output-, high CPU performance and, in conclusion, it allows to have good audio processing tools in the browser. An Audio Data API based on JS processing had been previously developed by the Mozilla project. This API is now deprecated, but it formed the basis of the ScriptProcessorNode. The ScriptProcessorNode is a special node of the WAA that supports JS processing. This node will be deeply explained in the 3.1.7 section.
Currently, the WAA is available in the latest versions of Chrome, Firefox, Safari, Opera, Chrome for Android, Firefox for Android, Opera Mobile and iOS Safari and it is being implemented in Internet Explorer [19]. Therefore, it is widely available, and developers have access to a free powerful tool for creating audio applications in web browsers. In addition, as usual when talking about web applications, it is possible to design an application by using the same code for different devices and operating systems.

1 //create the context
2 var context = new AudioContext();
3 //create an oscillator
4 var oscillator = context.createOscillator();
5 oscillator.type = 'square';
6 oscillator.frequency.value = 440;
7 //create a gain
8 gainNode = context.createGain();
9 gainNode.gain.value = 0.5;
10 //Connect Web Audio API graph nodes
11 oscillator.connect(gainNode);
12 gainNode.connect(context.destination);
13 //Start the oscillator at currentTime = 0
14 oscillator.start(0);

Listing 3.1: First example of Web Audio API code

To start working with the Web Audio API it is necessary to create an AudioContext on which all the AudioNode objects will be created. In the 3.1 code example we create an OscillatorNode -working as a source-, a GainNode, and we connect the oscillator to the gain, and the latter to the destination.

3.1.2 How the Web Audio API works

Web rich applications are written using HTML5, CSS and JS among others technologies. JS was born as a simple script language for the browser and it historically suffers an important limitation: all its execution process remains inside a unique thread, the main thread. The intensive use of JS in modern web applications and the fact that the real-time audio processing has special temporal requirements -and it does not have any priority inside the main thread- lead us to the conclusion that it is not a good idea to
execute audio-processing operations in the main-thread. The WAA solves this problem by doing most of the actual processing in the underlying implementation of the browser, and therefore, in a separate thread called the audio thread, and by exposing an interface in JavaScript to interact with the audio-processing tools.

One of the most important things when working with real-time audio is to have a very precise clock for the use of audio events such as drum machines, games or sequencers. The Web Audio API exposes access to the audio subsystem’s hardware clock as a property of AudioContext called currentTime. This property is a time in fractions of seconds which starts at zero when the AudioContext is created and it increases in real-time. This time corresponds an ever-increasing hardware timestamp.

![Diagram of Communication of threads for the update of currentTime property](image)

**Figure 3.2:** Communication of threads for the update of currentTime property

The audio clock works in the audio thread, but the user has access to it through the AudioContext currentTime property exposed in the main thread. Therefore, the audio clock is updated inside the audio thread and it is read in the main thread. This read is done every 128 audio samples [20], thus the period of update of the currentTime with a typical sampling rate of 44100 Hz leads to an update rate of approximately 3 ms.

$$\text{periodOfUpdateCurrentTime} = \frac{\text{blockSize}}{\text{sampleRate}} = \frac{128\text{samples}}{44100\text{Hz}} \approx 3\text{ms}$$

This update period could be considered accurate enough for the majority of audio applications. The decision of do not having a sample accurate update of the currentTime property in the main thread is based on that it is not possible to guarantee that the main thread will be enough accurate between acquiring the currentTime’s value from the audio thread and employing that value for scheduling process. This is due to the main thread can yield the CPU time to other interrupt-driven processes. Also, the
fact that the clock is read every 128 audio samples by the main thread is related to the block-processing implementation, as will be discussed in the AudioNode section (see 3.1.3).

With this precise clock, it is possible to schedule audio events, such as e.g., the start and stop of an AudioBufferSourceNodes at specific times [21]. When the start of an AudioBufferSourceNode is scheduled at specific time, an event is thrown to the audio thread as it can be seen in the figure 3.2. When the buffer source is finished, an event of type Event [22] will be dispatched to the onended event handler attribute of the AudioBufferSourceNode.

![Diagram of communication of threads for the playing of a buffer with the AudioBufferSourceNode](image)

**Figure 3.3:** Communication of threads for the playing of a buffer with the AudioBufferSourceNode

The WAA also provides methods for scheduling individual aspects of an AudioNode at specific moments. This is deeply explored in the 3.1.4 section. Chris Wilson (Google) explains how to schedule events for complex applications like change the tempo in middle of two bars by mixing the use of the Web Audio clock and the JavaScript clock in a HTML5 Rocks publication [23].

### 3.1.3 The AudioNode Interface

The AudioNode Interface represents three types of nodes with its distinctive features:

- Audio Source: It supports connect and disconnect methods for inserting/removing the node from the audio graph. Currently, two source nodes are provided: the OscillatorNode and the AudioBufferSourceNode.
• Intermediate processing modules: An intermediate processing module (e.g., GainNode, BiquadFilterNode) has inputs and outputs, and connect and disconnect methods.

• Audio Destination: The AudioDestinationNode has one input and no outputs and it represents the final destination to the audio hardware -what the user will ultimately hear-. As the AudioDestinationNode represents the audio hardware, the input can have more than two channels if the audio hardware is multi-channel. This feature is not tested in this work because it is centered in binaural and transaural processing where only two channels are necessary, but it is interesting for implementations of multi-channel spatialization systems.

For performance reasons, practical implementations will need to use block processing, with each AudioNode processing a fixed number of sample of size block-size. Block-size is defined to be 128 sample which corresponds to roughly 3ms at a sample-rate of 44.1KHz.

An important point is that AudioNodes are EventTargets, as described in DOM [24]. EventTarget is a DOM interface implemented by objects that can receive DOM events and have listeners for them. This means that it is possible to dispatch Events to AudioNodes the same way that other EventTargets accept Events as is showed in the following example.

```javascript
1   // Create an AudioContext
2   var context = new AudioContext();
3   // Create an AudioNode (A ConvolverNode, for instance)
4   var audioNode = context.createConvolver();
5   // Add an Event Listener of Events type 'test'
6   audioNode.addEventListener('test', function(){
7       console.log("AudioNode is an EventTarget");
8   });
9   // Dispatch an event of type 'test' to the audioNode
10  audioNode.dispatchEvent(new Event('test'));
```

Listing 3.2: An AudioNode is an EventTarget
3.1.4 The AudioParam Interface

An AudioParam controls an individual aspect of an AudioNode such as volume. A parameter of an AudioNode can be set immediately to a particular value using the value attribute, or the value changes can be scheduled to happen at precise times by using automation methods [25]. The AudioParams can be divided into two types:

- **a-rate** parameters: the parameters are taken into account on per-audio-sample.
- **k-rate** parameters: the parameters are sampled at the time of the first sample of the audio block, and that value is used for the entire block.

The AudioParam interface provides several self-explanatory automation methods to schedule the values: setValueAtTime, linearRampToValueAtTime, exponentialRampToValueAtTime, setTargetAtTime, setValueCurveAtTime and cancelScheduledValues. It is possible to find the rules to call these methods in the WAA specification [25].

In the development of the modules of this project, the use of automation of gain values is needed for the creation of crossfading effects. For that purpose it is necessary to use GainNodes. The GainNode has one AudioParam attribute, the gain and it is a-rate. The gain default value is 1, which is the maximum value it can reach and the minimum is 0. As the gain attribute is an AudioParam, it is possible to use the listed AudioParam methods to control it. Therefore, to do a linear crossfading between two audio sources, it is necessary to use two GainNodes and to sync a fade-in and a fadeout by using the linearRampToValueAtTime method.

This method schedules a linear continuous change from the previous scheduled parameter value to the given value and it has two parameters: the value parameter, where the
parameter will linearly ramp, and the endTime parameter, which indicates when the ramp will finish in the currentTime coordinate system. Its output is calculated as:

\[ v(t) = V_0 + (V_1 - V_0) \times \frac{t - T_0}{T_1 - T_0} \]

where \( T_0 \) is the time of the previous event, \( T_1 \) is the endTime parameter passed into this method, \( t \) is the audio clock value, \( V_0 \) is the value at time \( T_0 \) and \( V_1 \) is the value parameter passed into this method.

Thus, if you want to schedule a ramp, you have to reproduce these two steps:

1. Indicate the starting point by setting the gain value by using an AudioParam method (e.g., SetValueAtTime)
2. Start the ramp by setting the final gain value and the \( \text{endTime} \) [26]

```javascript
var now = audioContext.currentTime;
var endTime = now + 1;
// Indicate the start point of the ramp
gain1.gain.setValueAtTime(1, now);
gain2.gain.setValueAtTime(0, now);
// Start the ramp
gain1.gain.linearRampToValueAtTime(0, endTime);
gain2.gain.linearRampToValueAtTime(1, endTime);
```

Listing 3.3: Crossfading between gain1 and gain2 by using gain AudioParam methods

### 3.1.5 The DelayNode Interface

A delay is one of the fundamental operations in real-time audio processing. WAA implements a delay-line as a processing AudioNode called DelayNode. This node delays the incoming audio by a certain amount of time where \( \text{output}(t) = \text{input}(t - \text{delayTime}(t)) \). The DelayNode has one \( a \)-rate AudioParam attribute called delayTime which specifies the delay.
In this section, two experiments will be showed: the use of very small delays, and the use of the `DelayNode` inside in a loop. In the following example, a delay of 0.5 samples (sub-sampling delay) is tested:

```javascript
// Creation of the buffer
var buffer = audioContext.createBuffer(1, 512, 44100);

// Get the Array of Data
var bufData = buffer.getChannelData(0);

// Modify the Array input = [0, 1, 0.5, 0,...., 0]
bufData[1] = 1;
bufData[2] = 0.5;

// Creation of the source node
source = audioContext.createBufferSource();

// Put the created buffer to the source node
source.buffer = buffer;

// Creation of the delay
delay = audioContext.createDelay();

// 0.5 sample delay
delay.delayTime.value = 0.5/44100;

// Web Audio graph connections
source.connect(delay);
delay.connect(audioContext.destination);
```

Listing 3.4: Testing a subdelay sample of a DelayNode

The output can be analyzed by using an `AnalyserNode` and adjusting it to the appropriate scale, by using a `ScriptProcessorNode` -it is analyzed in the 3.1.7 section-, by using a recorder [27] and an offline software analysis such as Octave [28] or by using an `OfflineAudioContext`.

In this experiment, for an input of [0, 1, 0.5, 0] the output is [0, 0.5, 0.75, 0,25, 0], thus, as expected, a linear interpolation of the fractional delay is used to calculate the output. In the scope of this project, another fractional delay node will be developed, which provides different interpolation schemes (not only linear interpolation) [29]. This will be deeply explained in the 3.4 section.

The second test studies the comportment of the `DelayNode` inside one loop. This case can be useful to implement, for example, Infinite Impulse Response (IIR) filters by using
delay-lines and gains. The WAA specification does not mention any limitation on using a DelayNode inside one loop.

![Figure 3.5: DelayNode in a loop](image)

When used inside a feedback loop, the delay value shall be greater than 128 samples (i.e. the processing block-size). Looking for an explanation, it can be found in the public-audio list that it is a limitation due to the block processing implementation of the delay module [30] [31]. Therefore, if a delay inside a loop is necessary for values smaller than 128 samples, a handmade delay implementation has to be done.

### 3.1.6 The ConvolverNode Interface

This interface represents a processing node which applies a linear convolution effect given an impulse response. The ConvolverNode has two attributes: the buffer, and a boolean indicating the normalization called normalize.

The convolution is a high-intensive math operation used in many applications in science, engineering and mathematics. In acoustics is e.g., used for creating reverberation effects based on room impulse responses. Indeed, filtering an anechoic input signal with an FIR allows for reproducing the room reverberation in a realistic way. In binaural technology, the convolution is used to filter the input audio stream with a HRTF.

As the convolution is the main operation for the first of our modules -the BinauralFIR node-, it is important to test if the convolution works as expected because a perfect fidelity of the operation is needed for binaural technology. We did some tests by putting a delta as the input stream, and comparing the output of the system with the buffer set at the ConvolverNode with good results. In fact, there are tests done by Google and Mozilla to test their implementation of the WAA [32] [33]. Also, the official tests -W3C’s official test suite [34]- of Web Audio API are currently being started to write. Chris Lowis wrote a blog entry which explain how to proceed to write official tests [35].
3.1.7 The ScriptProcessorNode Interface

The ScriptProcessorNode is an AudioNode which can generate, process, or analyze audio directly using JavaScript. It can be initialized by indicating three parameters: the number of input channels, the number of output channels and the buffer size. The buffer size is adjustable and it represents the size (in samples) of the input/output processing block. It must be one of these values: 256, 512, 1024, 2048, 4096, 8192, 16384.

As the ScriptProcessorNode is a node that processes the audio in JavaScript -in the main thread-, it implies that the audio has to be moved between the audio thread and the main thread. As explained in the 3.1.3 section, the AudioNodes are EventTargets and the ScriptProcessorNode is an AudioNode. It means that the ScriptProcessorNode accepts Event objects. When an event occurs, an Event object is created and passed sequentially to the event listeners. The Event object is accessible from within the handler function, via the Event object passed as the first argument.

The Event responsible of the audio transport between the audio thread and the main thread is the AudioProcessingEvent. This is an Event object which is dispatched to the onaudioprocess event handler [22] of ScriptProcessorNodes at a specified frequency. This frequency is related to the ScriptProcessorNode buffer size and it can be calculated as:

\[ \text{frequencyAudioProcessingEvent} = \frac{\text{sampleRate}}{\text{bufferSize}} \]

The AudioProcessingEvent has an inputBuffer attribute, where the input audio data is stored, and an outputBuffer where the processed audio data has to be saved as it is showed in the following example:

1 //create the context
2 var context = new AudioContext();
3 //create an oscillator
4 var oscillator = context.createOscillator();
5 oscillator.type = 'sine';
6 oscillator.frequency.value = 440;
7 //create a ScriptProcessorNode
8 var ScriptProcessorNode = context.createScriptProcessor(1024, 1, 1);
9 //The event dispatched to the event handler
10 ScriptProcessorNode.onaudioprocess = function(event){

```javascript
// get the input and the output
var input = event.inputBuffer.getChannelData(0);
var output = event.outputBuffer.getChannelData(0);

// process the data: simple case where output = input/2
for(var i = 0; i < input.length; i = i + 1){
    output[i] = input[i]/2;
}

// Connect Web Audio API graph nodes
oscillator.connect(ScriptProcessorNode);
ScriptProcessorNode.connect(context.destination);
// Start the oscillator at currentTime = 0
oscillator.start(0);
```

Listing 3.5: Using the ScriptProcessorNode

Currently, the `ScriptProcessorNode` is the most controversial point of the current API because it is considered a second class citizen inside the Web Audio API. The current problematic can be summarized in three points:

1. The process of real-time audio in JavaScript implies that the audio data is processed in the main thread, therefore, in the browser user interface thread. This implies that if you are not rendering the audio at a speed that is at least as fast as real-time, then there will be some unexpected behavior like audio glitches and clicking. It means that all the other JS code must run fast enough to allow the audio to run at the speed that it needs. The main thread has a lot of operations to do, and the audio does not have any priority inside it. Therefore, if the main thread is busy doing some operation and the processing of the audio is not finished at the expected time, when the audio thread recover the `outputBuffer` to continue their audio operations -doing more processing or communicating the scriptProcessor with the destination-, will be audio dropouts and glitches because the `outputBuffer` will not be totally filled. This is one of the main problems that is worsen when resizing the window or doing intensive JavaScript operations.

2. The synchronization of the ScriptProcessorNode and the native nodes is possible thanks to the `playbackTime` attribute of the `AudioProcessingEvent`. The `playbackTime` indicates when the processed audio in each event will be played in the same
coordinate system as `AudioContext currentTime`. Theoretically, this time allows a very tight synchronization between processing directly in JavaScript and the other events in the `AudioContext`'s rendering graph. However, currently it is not correctly implemented in the browsers due to the difficulty calculating the latency introduced by the `ScriptProcessorNode`. [36].

3. The third problem is that due the nature of the `ScriptProcessorNode` it is not possible to modify parameters -global variables- during the processing of one buffer. When the `onaudioprocess` event handler receives an `AudioProcessingEvent`, the callback function is called by using the same mechanism as any other event -e.g., a click event-. If a global variable is trying to be updated -for instance, via a user interaction- while the event is being processed, nothing would happen until its finalization. When finished, the JavaScript thread can update the variable. Therefore, you will see this update in the next `AudioProcessingEvent`. The global variables used as parameters can be considered a kind of 'k-rate' parameters with an update frequency of `frequencyAudioProcessingEvent`.

The web audio community is hardly working on solving these issues and it seems that it is possible to solve some of these problems by using web workers [37].

A web worker is a JavaScript script executed from an HTML page that runs in the background, independently of other user-interface script, that is to say, it runs in a separate thread. The worker-based `ScriptProcessorNode` was proposed as a solution in June 2012 [38] and it is still under discussion [39]. Although the use of workers could be a perfect solution for the first point exposed, the worker-based `ScriptProcessorNode` have two main drawbacks:

1. Latency: the fact that the workers work with asynchronous operations could introduce a long latency as Chris Wilson exposes in this message [40].

2. Access to data: the web workers work in a separate thread, therefore, they cannot access to the complete JavaScript state (e.g., global variables). It is possible to pass all the needed information to the worker thread, but it can imply an increase of the latency value.
3.1.8 The BiquadFilterNode Interface

A digital biquad filter is a second-order recursive linear filter, containing two poles and two zeros. [41].

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \]

The biquad filter is implemented in the WAA as a node called BiquadFilterNode. It provides several configurations of biquad filters commonly used in audio processing such as e.g., lowpass, highpass, bandpass, lowshelf, derived from the cookbook formulae from Robert Bristow-Johnson [41]. They are more detailed in the DelayNode section of the API specification [42]. The BiquadFilterNode also provides AudioParams to control the typical parameters of one filter like frequency, detune, Q and gain. All these parameters are k-rate.

The basic limitation of the BiquadFilterNode is that the filters have to be designed by using the frequency, detune, Q and gain parameters [41] and it is not possible to design the filters by using directly the coefficients of the formula. Although the current API covers a lot of use cases, this design imposes a restriction for other use cases. In our case we have some FIR filters representing the HRTF filters. For CPU considerations, we model these FIR filters into a series of biquads filters. The modeling operation produces a list of coefficients and these coefficients are not compatible with the cookbook formulas, thus, with the frequency, detune, Q and gain parameters. This is an important drawback for our project, but it is solved by designing our own BiquadFilter library (3.3).

However, as the design of the Web Audio API follows the open source philosophy where everybody can take part and contribute on the design of the API, we decided to participate on it. We think that this feature could be useful for the whole web audio community and especially for filter designers.

The first step to write an API proposal is to check if there is some current open issue on the Web Audio API issues list on GitHub [43] with a similar proposition. A second optional -but good- step is to take a look at the code implementation. The web audio code can be found in open source browsers like Chromium and Firefox [44] [45]. In this case, we took a look at Blink (Chromium) implementation of BiquadFilterNode [46] and we saw that currently exist an internal method called setNormalizedCoefficients where you can indicate the coefficients of the biquad formula. Thus, it would be easy to
expose this method to the JavaScript interface -the Web Audio API-. The last step is to publish the issue, to expose clearly your proposal and to indicate a clear use case. In our case, we published our first participation in the standard Possibility to set the filter coefficients and not just Q/\text{freq}/\text{gain} \cite{47} and it is currently in discussion.

### 3.1.9 The PannerNode Interface

This interface represents a processing node which spatializes an incoming audio stream in three-dimensional space for binaural listening. The PannerNode is a sound-advanced library added to the Web Audio API for the requirements of modern 3D HTML5 games \cite{48} \cite{49}.

Using a PannerNode, an audio stream can be spatialized or positioned in space relatively to an AudioListener. An AudioContext contains only one AudioListener. Both AudioContext and AudioListeners have a position in the 3D space using a right-handed cartesian coordinate system \cite{50}. Generally, the binaural synthesis systems use spherical coordinate system (azimuth, elevation and distance) but the PannerNode uses a cartesian coordinate system to ease the use of this node in 3D games. With that, it is easier to integrate the binaural synthesis with JavaScript 3D graphic libraries like Three.js \cite{51}.

The PannerNode has several methods related to the position and directivity of the source and three basic distance model -linear, inverse and exponential- to reduce the level of an audio source as it moves away from the listener. The PannerNode has two models for spatializing the sound: the basic model, equalpower, and the higher quality spatialization model, the HRTF model.

This node introduces a very advanced sound module for binaural listening on the web, but it has some limitations. The PannerNode implementation of Chromium and Mozilla uses one HRTF set that is created by averaging HRTF impulses responses from human subjects from the IRCAM Listen HRTF Database \cite{52} \cite{53}. As stated before in the State Of The Art section \cite{2}, the HRTF is highly dependent on the listener morphology. Therefore, if it is pretended to achieve good spatialization results it is necessary to use a different HRTF set for each person and the PannerNode does not allow it. Looking backward in the mailing list of the WAA it is possible to find a request from the BBC
R&D team of apply their own HRTF set in the PannerNode [54] but currently not seems to be movement on this area.

The fact that it is not possible to apply different HRTF sets in the PannerNode and the fact that the API does not have advanced distance and directivity algorithms has lead the Espaces Acoustiques et Cognitives team from IRCAM to design and create their own binaural nodes.

### 3.2 The BinauralFIR Node

The BinauralFIR node is the first node developed in this project. It provides binaural listening to the Web Audio API users. The main novelty of this node is that it permits to use different HRTF datasets. This library can be used as a regular node inside the Web Audio API. It means that you can connect native nodes to the BinauralFIR node by using the connect method.

![Figure 3.6: General diagram of a binaural in the browser application](image)

The BinauralFIR node can be useful to create e.g, an audio streaming application with binaural listening controlled by the user as it is showed in the figure 3.6. In this example, an audio streaming such as e.g., a live radio concert emitted in multi-track is get from a server. The HRTF dataset is get from another HRTF datasets server and the binaural application is executed in the mobile browser. The binaural signal is created in the mobile browser with the parameters chosen by the user.
3.2.1 How it works

The BinauralFIR node is based on the more basic approach explained in the Binaural Synthesis section (2.2). It consists in convolving a mono input signal with two HRTF filters, the left ear filter and the right ear filter. Thus, a priori, this node can be constructed by only using one node: the ConvolverNode.

When we started developing this node, we designed a first schema formed by a source connected to a ConvolverNode, and this latter connected to the destination.

With this basic structure and an HRTF dataset it is possible to place one virtual source in one specific virtual position, as it can be seen in the following example:

```javascript
// Create the context
var context = new AudioContext();

// Create a oscillator that will work as a source
var source = context.createOscillator();

// Create the convolver
var convolver = context.createConvolver();

// Web Audio graph connections
source.connect(convolver);
convolver.connect(context.destination);

// Here, we will assume that we have the coefficients of both filters -left and right- for a specific position in two arrays called leftFilter and rightFilter
// First, we create an AudioBuffer with 2 channels, length of 512 samples (length of our filters) and sample rate of 44100Hz
var convolverBuffer = context.createBuffer(2, 512, 44100);

// Get the real buffer of left channel and right channel to fill them.
var leftBufferChannel = buffer.getChannelData(0);
```
var rightBufferChannel = buffer.getChannelData(1);

// Fill the buffers with the filters data
for (var i = 0; i < 512; i++) {
    leftBufferChannel[i] = leftFilter[i];
    rightBufferChannel[i] = rightFilter[i];
}

// Set to false the normalization of the ConvolverNode before to set the buffer
convolver.normalize = false;

// Set the buffer to the ConvolverNode
convolver.buffer = convolverBuffer;

// Start the source
source.start(0);

Listing 3.6: Basic structure of BinauralFIR

However, this design has a big limitation: it is only useful for static sources because if the position of the source -the buffer of the ConvolverNode- has to be changed, there will be audio glitches. Static sources can be useful to create some scenarios (e.g., 5.1 surround to binaural sound in a streaming scenario) but our goal is to design a binaural module with dynamic sources.

If we want to create dynamic virtual sources, a smooth transition -crossfading- is needed when changing the filter. When a ConvolverNode buffer is changed, there is a transient state until the signal is stable, and during this transient state, there are audio glitches. Therefore, another solution has to be used. Our proposal is to use two ConvolverNodes and do a crossfading to change between them.

Figure 3.8: Final schema of BinauralFIR node
As it can be seen in the figure 3.8, we create a ConvolverAudioGraph object formed by a ConvolverNode and a GainNode and two instances of it are created:

- mainConvolver: Represents the current state.
- secondaryConvolver: Represents the potential target state when moving a source to a new position.

Therefore, a smooth transition between these two audiographs has to be created as explained in the 3.1.4 section. When a user wants to change the current position, it must call the public setPosition method and this method manage the process. In the figure 3.9 it is possible to find a flow diagram that shows the working process of this method. As explained before, when a ConvolverNode buffer is changed, there are small audio glitches during the transient state. It is for this reason that the only possible solution to totally avoid the audio glitches is to wait a guard interval between change the ConvolverNode buffer and the start of the crossfading.

### 3.2.2 HRTF dataset format

As our node allows to set different HRTF datasets, we must define an HRTF dataset format. Our coordinate system is based on a spherical coordinate system where the position of a point is specified by three numbers:

- Azimuth: From 0° to -180° (degrees) for sources on your left, and from 0 to 180 for sources on your right.
- Distance: Radial distance of that point from a fixed origin expressed in meters.
- Elevation: From 0° to 90° (degrees) for sources above your head, 0 for source in front of your head, and from 0 to -90 for sources below your head)

The HRTF dataset format is an array of objects containing four properties for each input position: azimuth, distance, elevation and a stereo AudioBuffer with the coefficients of the left and right filters.

A list of free HRTFs databases is available in the Appendix B. It is possible to use all these HRTF datasets in this node by adapting them to the HRTF dataset format of this node.
3.2.3 Locate the closest position

The HRTF dataset only provides filters for discrete points in the 3D space. Whenever the source position is set (by the user), the processor has to search for the nearest position available in the dataset. In our node, when a new HRTF dataset is set, the spherical coordinates are transformed to cartesian coordinates and a 3-dimensional tree [55] is created by using the k-d Tree JavaScript Library [56], a JavaScript implementation of a k-dimensional tree data structure. When the tree is created, the list of points, the distance function and the list of dimensions have to be set. In our case, the list of cartesian points for all the filters, a euclidean distance function and the cartesian
coordinates \((x, y, z)\). Thus, when a virtual source is moved to a new position, the *nearest* function of the created *tree* is called and this function returns the nearest neighbor to the input position.

### 3.2.4 API

The BinauralFIR node is published as open-source with a BSD-3-Clause license [57] and it is available on the IRCAM GitHub account [58]. All the code, a readme file, a test, and an example are available in the page. This library exposes six public methods for their use:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BinauralFIR.connect()</code></td>
<td>Connects the BinauralFIRNode to the Web Audio graph</td>
</tr>
<tr>
<td><code>BinauralFIR.disconnect()</code></td>
<td>Disconnects the BinauralFIRNode from the Web Audio graph</td>
</tr>
<tr>
<td><code>BinauralFIR.HRTFDataset</code></td>
<td>Sets an HRTF Dataset to be used with the virtual source.</td>
</tr>
<tr>
<td><code>BinauralFIR.setPosition(azimuth, elevation, distance)</code></td>
<td>Sets the position of the virtual source.</td>
</tr>
<tr>
<td><code>BinauralFIR.getPosition()</code></td>
<td>Gets the current position of the virtual source.</td>
</tr>
<tr>
<td><code>BinauralFIR.setCrossfadeDuration(duration)</code></td>
<td>Sets the duration of crossfading in milliseconds.</td>
</tr>
<tr>
<td><code>BinauralFIR.getCrossfadeDuration()</code></td>
<td>Gets the duration of crossfading in milliseconds.</td>
</tr>
</tbody>
</table>

### 3.3 The BiquadFilter Library

Although the current limitation of the WAA *BiquadFilterNode* explained in the 3.1.8 section is being discussed by the W3C Audio Working Group, we decided to create our own *BiquadFilter* library to solve the current limitation. The *BiquadFilter* library is a
JavaScript library that implements a cascade of biquad filters. With this library it is possible to design a filter by using directly the coefficients of the following formula:

\[ H(z) = g \cdot \frac{1 + a_1^{(1)} z^{-1} + a_2^{(1)} z^{-2}}{1 + b_1^{(1)} z^{-1} + b_2^{(1)} z^{-2}} \cdot \frac{1 + a_1^{(2)} z^{-1} + a_2^{(2)} z^{-2}}{1 + b_1^{(2)} z^{-1} + b_2^{(2)} z^{-2}} \cdot \ldots \cdot \frac{1 + a_1^{(N)} z^{-1} + a_2^{(N)} z^{-2}}{1 + b_1^{(N)} z^{-1} + b_2^{(N)} z^{-2}} \]

where \( g \) is the global gain of the cascade of biquads and \( N \) is the number of biquad filters. Thus, four coefficients have to be indicated for each biquad filter.

This library allows to design one biquad filter by setting the filter coefficients and not the Q/freq/gain parameters. Also, it implements a cascade of biquads filters. With this implementation, it is possible to save \( N \) multiplications, compared to an implementation with \( N \) biquads processed independently.

This module is implemented as a JavaScript library and not as a Web Audio API node (by using a `ScriptProcessorNode`) because we consider that it provides more flexibility. It does not mean that it is not possible to use it on the Web Audio API. With this implementation, it is possible to use different libraries inside the same `ScriptProcessorNode`. Thus, a more efficient module is created.

### 3.3.1 API

The `BiquadFilter` library is published as open-source with a BSD-3-Clause license [57] and it is available at IRCAM GitHub account [59]. All the code, a readme file, a test, and an example are available in the page. The details of their use can be found in the GitHub readme. This library exposes two public methods for their use:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BiquadFilter.setCoefficients(coef)</code></td>
<td>Sets the coefficients of the filter. Depending on the number of coefficients, more or less biquad filters are used.</td>
</tr>
<tr>
<td><code>BiquadFilter.process(inputBuffer, outputBuffer)</code></td>
<td>Calculates the output of the cascade biquad filter for an inputBuffer. The inputBuffer and the outputBuffer are Arrays with the same length.</td>
</tr>
</tbody>
</table>
3.4 The FractionalDelay Library

A fractional delay filter is a device for bandlimited interpolation between samples [29]. It is used in numerous fields of signal processing, including binaural synthesis.

The delay used when modeling HRTFs with a minimum-phase and pure delay corresponds to the ITD estimated from the HRIR. According to Mills [60], the least noticeable difference of ITD changes can be as low as 10 µs - 0.44 samples for a sample rate of 44100Hz-. Thus, the ITD has to be capable of delivering finer resolutions, that is to say, fractional delays.

In Splitting the unit delay (Välimäki et al. [29]), the fractional delay concepts are explained. Based on the comparison of several techniques for bandlimited approximation of a fractional digital delay, we decided to use a First-Order Allpass Interpolation (also called Thiran Approximation). Our decision is based on obtain the maximum plain frequency response and obtain a good performance. An IIR digital filter can meet the same frequency-domain specification with a smaller number of multiplications than an FIR filter.

A delay line interpolated by a first-order allpass filter is drawn in Figure 3.10.

\[
x[n - \Delta] = y[n] = \eta \cdot x[n] + x[n - 1] - \eta \cdot y[n - 1]
\]

where \( \Delta \) is the desired sub-sampling delay and \( \eta \) is:
\[ \eta = \frac{1 - \Delta}{1 + \Delta} \]

### 3.4.1 API

The *FractionalDelay* library is published as open-source with a BSD-3-Clause license [57] and it is available at IRCAM GitHub account [62]. All the code, a readme file, a test, and an example are available in the page. The details of their use can be found in the GitHub readme. This library exposes three public methods for their use:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FractionalDelay.setDelay(delayTime)</code></td>
<td>Set the delay time.</td>
</tr>
<tr>
<td><code>FractionalDelay.getDelay()</code></td>
<td>Get the delay time.</td>
</tr>
<tr>
<td><code>FractionalDelay.process(inputBuffer)</code></td>
<td>Process the data for an input buffer.</td>
</tr>
</tbody>
</table>

### 3.5 The BinauralBiquad Node

The *BinauralBiquad* node is the second node developed in this project. The final goal of this node is the same as the *BinauralFIR* node, to provide binaural listening to the Web Audio API users, but by using a different approach for the synthesis of the binaural signal. This node synthesize the binaural signals by using measured HRTFs modeled as IIR filters and pure delay.

The fact to use a modeling of HRTFs has several intentions:

- Lower CPU cost.
- Opens the door towards some individualization (e.g., allowing to customize the ITD).
- It helps smoothing out some individual cues by smoothing out the spectral magnitude.
3.5.1 How it works

As stated before, in the 2.2.1 section, measured HRTFs can be approximated by a simplified model such as minimum-phase and pure delay. The HRTF FIR filters used in this project have 512 coefficients while the minimum equivalent modeled of cascade biquads version has 24 coefficients. Thus, the difference on CPU load -and in bandwidth network when downloading an HRTF dataset- is significant.

This node implements the minimum-phase and pure delay strategy. The minimum-phase is implemented by using a cascade of biquad filters and the delay is implemented by using a fractional delay. As explained before in the BiquadFilterNode (3.1.8) and DelayNode (3.1.5) sections, the limitations of the Web Audio API in these two nodes lead us to develop our own BiquadFilter library (3.3) and FractionalDelay library (3.4).

The BinauralBiquad node is designed by using the learned strategies used in the BinauralFIR node. Thus, we use the same strategy for the smooth transition when changing the virtual source position. We create a ProcessingAudioGraph object formed by two BiquadFilter libraries, two FractionalDelay libraries and a GainNode and two instances of it are created. In this node, two BiquadFilter libraries are needed because each BiquadFilter implement only one filter, and each ear has their own filter. The same happens with the FractionalDelay library where each ear has a different delay and two delay-lines are needed.

Figure 3.11: Final schema of BinauralBiquad
As it can be seen in the figure 3.11, a `ScriptProcessorNode` is used to process the audio of each object. By using only one `ScriptProcessorNode` for each `ProcessingAudioGraph`, smaller CPU load and latency values are achieved compared to using a `ScriptProcessorNode` for each library. This is the main reason why the `BiquadFilter` library and `FractionalDelay` library are not designed as a Web Audio Node - by using a `ScriptProcessorNode`.

### 3.5.2 HRTF dataset format

As this node is similar to the `BinauralFIR`, we use a similar format. The coordinate system is exactly the same, thus, for more information see HRTF dataset format of `BinauralFIR` (3.2.2).

The HRTF dataset format is an array of objects containing six properties for each input position: azimuth, distance, elevation, itd, array of left filter coefficients, and an array of right filter coefficients.

### 3.5.3 Locate the closest position

As this node uses the same coordinate system of the `BinauralFIR` node, it uses exactly the same system to find the closest position for an input position. For further information, see 3.2.3.

### 3.5.4 API

The `BinauralBiquad` node is published as open-source with a BSD-3-Clause license [57] and it is available at IRCAM GitHub account [63]. All the code, a readme file, a test, and an example are available in the page. As the final goal of this node is the same as the `BinauralFIR` node, the public methods are exactly the same, thus, the API can be found in the 3.2.4 section.
3.6 Packaging and Test

Currently it does not exist an agreement on what is the best solution for the client-side JavaScript modularization. As this project takes part of the WAVE project [64], the modules developed on this project follow the way to package decided by the WAVE team. The WAVE team has decided to go for the CommonJS [65], that it is de facto standard to package the libs in JavaScript. Also, all the code is written following the ECMAScript5 specification. For a more detailed information, it is possible to read the WAVE modules publication in the project blog [66].

All the modules have the final structure of the figure 3.12 where `index.js` is the module file, `package.json` contains relevant information about the module -name, the current version, dependencies, authors, etc.-, `<module-name>.js` is a ready-to-use browser compatible version of the lib , `<module-name>.min.js` is the minified companion, the `README` file contains the relevant instructions, `LICENCE` file contains the chosen license, and the `Gruntfile` contains means to compile your own version of the lib, and launch the test suite.

Another important point when developing open source software is to provide a readable code. Thus, the open source community can extend and improve the software. Also, we provide a test suite to test the integrity of our software. A software test checks if the software works as expected ((e.g., the set return the input of the get). This is very important to easily test that the changes done in the code do not break the software.
3.7 Demos

A small demo was created for each developed module to demonstrate how to use it. They can be found in the examples folder of each GitHub repository [59] [58] [62] [63].

On the other hand, a user interface to easily interact with the virtual sources was created. Currently it does not exist an agreement on what is the best solution to graphically interact with a binaural synthesis system. Our user interface is a 2D graphical representation (XY plane) of the spatial sound scene where you can easily create virtual sources, assign them a sound and move them around the listener. Also, it is possible to change their elevation by using a source’s menu as it is showed in the figure 3.13.

This interface is created by using the Snap.svg JavaScript library [67]. With this library it is possible to create Scalable Vector Graphics (SVG) DOM objects [68] and interact with them. It means that each drawn object corresponds to a DOM element and thus, it is easy to interact with it.

Figure 3.13: User interface
Chapter 4

Results

In this section, the CPU performance of the three current binaural synthesis systems in the browser is compared: the native WAA PannerNode, and the binauralFIR and BinauralBiquad nodes developed in this project. Also, different configurations of BinauralBiquad node are tested. The complete results can be found in the Appendix C table.

The tests are done in an iMac 2009, 2.93Ghz Intel Core 2 Duo, 4GB 1067Mhz DDR3, OS X 10.8.5 with Google Chrome 35.0.1916.153 as browser.

4.1 BinauralFIR and PannerNode

The PannerNode (see 3.1.9) is totally implemented in C++ and it can be only used with a specific HRTF dataset compiled with the browser. Google Chrome uses HRTF filters of length of 256 samples with a sample rate of 44.1KHz [69]. On the other hand, the BinauralFIR node (see 3.2) can be used with any HRTF dataset and it is implemented in C++ (the WAA processing nodes) and JavaScript.

In this experiment, we compare the performance of this two nodes by using a 256 HRTF filter length in both systems.

As it is showed in the figure 4.1, the PannerNode has lower CPU cost. We expected that the results will be more similar due to the static binauralFIR sources only uses native WAA nodes, that is to say, C++ processing. However, the measured results do
not agree with this. This would deserve further investigations. Nevertheless, the CPU performance of the BinauralFIR is very good and it can be perfectly usable. Also, the BinauralFIR node is more configurable because it is capable of using:

- Different HRTF datasets: it is possible to use any HRTF dataset.
- Different spatialization algorithms: it is possible to implement different spatialization algorithms directly in JavaScript such as e.g., directivity, distance.

### 4.2 Different configurations of BinauralBiquad

The BinauralBiquad node (see 3.5) is totally implemented in JavaScript. It means that it uses the controversial ScriptProcessorNode to do the audio processing (see 3.1.7). As explained in the 3.1.7 section, the ScriptProcessorNode buffer size is adjustable. Lower buffer size result in a lower (better) latency, and higher numbers are necessary to avoid audio breakup and glitches. Therefore, a compromise has to be made.

In this experiment, buffer sizes of 1024, 4096 and 16384 samples are tested. As it is showed in the figure 4.2, the ‘BinauralBiquad 1024’ configuration obtains low CPU performance results. The ‘BinauralBiquad 4096’ configuration obtains the better compromise; enough CPU good performance and an acceptable latency value. This low CPU performance results were expected due to the ScriptProcessorNode problematic explained in the section 3.1.7.
We expected that the CPU load would be linear with respect to the number of sources. However, the measured results do not agree with this. This would deserve further investigations and more measurements.

As previously explained (see 3.1.8), nowadays, the BiquadFilterNode can only be designed by using the frequency, detune, Q and gain parameters. As the possibility to directly set the filter’s coefficients to the BiquadFilterNode is being discussed on the Web Audio Community [47], a test was done to compare the performance of our BiquadFilter library -JavaScript processing- with the BiquadFilterNode.

Thus, the BiquadFilter library is replaced by the native WAA BiquadFilterNode. Therefore, the ScriptProcessorNode is still being used by the FractionalDelay library, but the more intensive operation -biquad- is done in the native implementation. As it is showed in the figure 4.3, the CPU consumption has drastically decreased.
Chapter 5

Budget

This economic budget is only an academic exercise which not necessarily needs to correspond to the real case.

This project is developed by using open-source tools and all the developed software is published as open-source with a BSD-3-Clause license [57]. The direct costs derived from the researcher and the engineer are derived by the coordination tasks and technical support of this project. The indirect costs are related with the office, amortization of the computer and servers, electricity and office and building personal. The indirect costs are approximated to:

\[ \text{IndirectCosts} = 5 \text{months} \cdot 50E/\text{month} = 250E \]

The direct costs are:

<table>
<thead>
<tr>
<th>Status</th>
<th>Workers</th>
<th>Price/Hour</th>
<th>Hours/Week</th>
<th>Monthly salary</th>
<th>Months</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intern Student</td>
<td>1</td>
<td>2.73E</td>
<td>40</td>
<td>436.05E</td>
<td>5</td>
<td>2180.25E</td>
</tr>
<tr>
<td>Researcher</td>
<td>1</td>
<td>30E</td>
<td>2</td>
<td>240E</td>
<td>5</td>
<td>1200E</td>
</tr>
<tr>
<td>Engineer</td>
<td>1</td>
<td>30E</td>
<td>3</td>
<td>360E</td>
<td>5</td>
<td>1800E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TOTAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5180.25E</td>
</tr>
</tbody>
</table>

Therefore, the total cost is: 5180.25E + 250E = 5430.25E
Chapter 6

Conclusions and future development

In this project, two binaural synthesis systems in the browser have been developed. Both systems provide a novelty respect the current binaural system in the browser -the WAA PannerNode-, that is to provide the possibility to use your own HRTF datasets and thus, to provide better perceptual results to the final user. For this purpose, a deep study of the current state of the Web Audio API has been done.

All our objectives were accomplished because we developed the synthesis software with all the functionalities that we expected at the start of the project. The unique limitation has been that the current approach of the ScriptProcessorNode has limited the CPU performance of the BinauralBiquad node. Nevertheless, we trust that the ScriptProcessorNode will be improved in the near future by removing the JavaScript audio processing from the main thread.

In this project, a first stable version of the synthesis nodes has been developed, but in the future, a lot of new features can be added. The novelties can be summarized in three sections:

- Radiation models: To apply distance and directivity models.
- Perceptual factors: To apply perceptual factors such as e.g., source presence and source brillance.
• Room response: To apply room response models.

Another interesting and easy module to develop is a transaural node. The transaural node allows to use the binaural signals through the loudspeakers.
Appendix A

Work Packages, Tasks, Milestones and Time Plan

<table>
<thead>
<tr>
<th><strong>Project:</strong></th>
<th>Project proposal and workplan</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WP ref:</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Major constituent:</strong></td>
<td>documentation</td>
</tr>
<tr>
<td><strong>Sheet:</strong></td>
<td>1 of 9</td>
</tr>
<tr>
<td><strong>Short description:</strong></td>
<td>Description and planning of the project. Gantt diagram.</td>
</tr>
<tr>
<td><strong>Planned start date:</strong></td>
<td>03/03/2014</td>
</tr>
<tr>
<td><strong>Planned end date:</strong></td>
<td>14/03/2014</td>
</tr>
<tr>
<td><strong>Start event:</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>End event:</strong></td>
<td>T4</td>
</tr>
</tbody>
</table>

| **Internal task T1:** | Description and planning of the project. |
| **Internal task T2:** | Writing of the work plan. |
| **Internal task T3:** | Revision and modification of the work plan. |
| **Internal task T4:** | Delivery of the document. |

| **Deliverables:** | ArnauJulia_Proj Proposal.pdf (14/03/2014) |

**Table A.1: Work Package 1**
**Project**: Study of the bibliography

**Major constituent**: documentation

**Short description**: Study of the state of the art of spatialization, Web Audio API and ECMAScript.

<table>
<thead>
<tr>
<th>WP ref: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sheet: 2 of 9</td>
</tr>
<tr>
<td>Planned start date: 15/01/2014</td>
</tr>
<tr>
<td>Planned end date: 10/02/2014</td>
</tr>
<tr>
<td>Start event: T1</td>
</tr>
<tr>
<td>End event: T6</td>
</tr>
</tbody>
</table>

**Internal task T1**: Reading of Web Audio API specification.

**Internal task T2**: Reading of “JavaScript: The Good Parts”

**Internal task T3**: Reading of book “3D sound for virtual reality and multimedia” (Begault).

**Internal task T4**: Reading of paper “Review of Digital Filter Design and Implementation Methods for 3-D Sound”

**Internal task T5**: Reading of “Communication Acoustics (Signals and Communication Technology)” (Blanert)

**Internal task T6**: Reading of presentations and articles of JavaScript and HTML5.

**Deliverables**: 

<table>
<thead>
<tr>
<th>Table A.2: Work Package 2</th>
<th></th>
</tr>
</thead>
</table>
**Project**: Testing the spatialization nodes of Web Audio API

**WP ref**: 3

**Major constituent**: software

**Sheet**: 3 of 9

**Short description**: Creation of examples using spatialization tools of Web Audio API and creation of tools for test the performance.

**Planned start date**: 05/02/2014

**Planned end date**: 18/02/2014

**Start event**: T1

**End event**: T5

<table>
<thead>
<tr>
<th>Internal task T1:</th>
<th>Get used to the work environment (sublime text + WAVE server).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal task T2:</td>
<td>Creation of examples with pannerNode of Web Audio API.</td>
</tr>
<tr>
<td>Internal task T3:</td>
<td>Search and implementation of tools for test the examples.</td>
</tr>
<tr>
<td>Internal task T4:</td>
<td>Tests on Matlab comparing the performance of browsers.</td>
</tr>
<tr>
<td>Internal task T5:</td>
<td>Writing of the documentation on the wiki of the project.</td>
</tr>
</tbody>
</table>

**Deliverables**: Table A.3: Work Package 3
**Project:** Software development  
**WP ref:** 4  
**Major constituent:** software  
**Sheet:** 4 of 9

**Short description:** Development of the own spatialization software based on the algorithms and filters of IRCAM on the browser and test of the Spat Software.

**Planned start date:** 19/02/2014  
**Planned end date:** 25/05/2014  
**Start event:** T1  
**End event:** T9

<table>
<thead>
<tr>
<th>Internal task T1</th>
<th>Internal task T2</th>
<th>Internal task T3</th>
<th>Internal task T4</th>
<th>Internal task T5</th>
<th>Internal task T6</th>
<th>Internal task T7</th>
<th>Internal task T8</th>
<th>Internal task T9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of convolverNode and creation of static virtual sources with HRIR FIR filters.</td>
<td>Test of the Spat Software.</td>
<td>Creation of visual interfaces with bindings of WAVE libs.</td>
<td>Crossfading of the filters and creation of dynamic virtual sources.</td>
<td>Study of biquadFilterNode.</td>
<td>Write documentation on the wiki of the project.</td>
<td>Development of own biquadFilter Library</td>
<td>Development of own delay (fractional delay)</td>
<td>Development of binaural synthesis node with HRIR IIR filters</td>
</tr>
</tbody>
</table>

**Deliverables:**

<p>| Table A.4: Work Package 4 |</p>
<table>
<thead>
<tr>
<th>Project: Creation of public API</th>
<th>WP ref: 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: software</td>
<td>Sheet: 5 of 9</td>
</tr>
<tr>
<td>Short description: Creation of public API for ease the use of the software developed.</td>
<td>Planned start date: 05/05/2014</td>
</tr>
<tr>
<td></td>
<td>Planned end date: 20/06/2014</td>
</tr>
<tr>
<td></td>
<td>Start event: T1</td>
</tr>
<tr>
<td></td>
<td>End event: T4</td>
</tr>
<tr>
<td><strong>Internal task T1:</strong> Planning of the public methods and attributes.</td>
<td>Deliverables:</td>
</tr>
<tr>
<td><strong>Internal task T2:</strong> Isolating the configuration files.</td>
<td></td>
</tr>
<tr>
<td><strong>Internal task T3:</strong> Implementation.</td>
<td></td>
</tr>
<tr>
<td><strong>Internal task T4:</strong> Writing of the documentation of the API.</td>
<td></td>
</tr>
</tbody>
</table>

Table A.5: Work Package 5

<table>
<thead>
<tr>
<th>Project: Critical Review</th>
<th>WP ref: 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: document</td>
<td>Sheet: 6 of 9</td>
</tr>
<tr>
<td>Short description: Review of the work plan and restructuring of the work.</td>
<td>Planned start date: 21/04/2014</td>
</tr>
<tr>
<td></td>
<td>Planned end date: 30/04/2014</td>
</tr>
<tr>
<td></td>
<td>Start event: T1</td>
</tr>
<tr>
<td></td>
<td>End event: T4</td>
</tr>
<tr>
<td><strong>Internal task T1:</strong> Review of the work done.</td>
<td></td>
</tr>
<tr>
<td><strong>Internal task T2:</strong> Restructuring the work plan.</td>
<td></td>
</tr>
<tr>
<td><strong>Internal task T3:</strong> Writing the document.</td>
<td></td>
</tr>
<tr>
<td><strong>Internal task T4:</strong> Delivery of the document</td>
<td>ArnauJulia_Critical Review.pdf (30/04/2014)</td>
</tr>
</tbody>
</table>

Table A.6: Work Package 6
### Appendix A. Work Packages, Tasks, Milestones and Time Plan

<table>
<thead>
<tr>
<th><strong>Project</strong></th>
<th><strong>WP ref:</strong> 7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Major constituent:</strong> software</td>
<td><strong>Sheet:</strong> 7 of 9</td>
</tr>
<tr>
<td><strong>Short description:</strong> Evaluation of the final software.</td>
<td><strong>Planned start date:</strong> 11/06/2014</td>
</tr>
<tr>
<td></td>
<td><strong>Planned end date:</strong> 16/06/2014</td>
</tr>
<tr>
<td></td>
<td><strong>Start event:</strong> T1</td>
</tr>
<tr>
<td></td>
<td><strong>End event:</strong> T2</td>
</tr>
</tbody>
</table>

**Internal task T1:** Carry out perceptual experimentation.

**Internal task T2:** Compare the performance of final software in different browsers

| **Deliverables:** |

---

### Table A.7: Work Package 7

<table>
<thead>
<tr>
<th><strong>Project:</strong> Demos</th>
<th><strong>WP ref:</strong> 8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Major constituent:</strong> software</td>
<td><strong>Sheet:</strong> 8 of 9</td>
</tr>
<tr>
<td><strong>Short description:</strong> Design and develop mockups for demonstrate the use of the software.</td>
<td><strong>Planned start date:</strong> 11/06/2014</td>
</tr>
<tr>
<td></td>
<td><strong>Planned end date:</strong> 20/06/2014</td>
</tr>
<tr>
<td></td>
<td><strong>Start event:</strong> T1</td>
</tr>
<tr>
<td></td>
<td><strong>End event:</strong> T3</td>
</tr>
</tbody>
</table>

**Internal task T1:** Design of mockups.

**Internal task T2:** Development of mockups.

**Internal task T3:** Test of mockups.

| **Deliverables:** |

---

### Table A.8: Work Package 8
<table>
<thead>
<tr>
<th><strong>Project:</strong> Final Report</th>
<th><strong>WP ref:</strong> 9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Major constituent:</strong> documentation</td>
<td><strong>Sheet:</strong> 9 of 9</td>
</tr>
</tbody>
</table>

**Short description:** Writing of the final report of the project.

**Planned start date:** 09/06/2014

**Planned end date:** 27/06/2014

**Start event:** T1

**End event:** T3

**Internal task T1:** Writing the document.

**Internal task T2:** Design the presentation.

**Internal task T3:** Delivery of the document.

**Deliverables:**

ArnauJuliia_TFGFinalReport.pdf (27/06/2014)

| **Table A.9: Work Package 9** |
### A.0.1 Milestones

<table>
<thead>
<tr>
<th>#WP</th>
<th>#Task</th>
<th>Short Title</th>
<th>Milestone</th>
<th>Date (week)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>Work Plan</td>
<td>ArnauJulia_workplan.pdf</td>
<td>10/03/2014</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Web Audio API.</td>
<td>Understand Web Audio API.</td>
<td>20/01/2014</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Spatialization bases.</td>
<td>Understand Spatialization.</td>
<td>27/01/2014</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>First example.</td>
<td>Creation of first 3D sound experiment.</td>
<td>17/02/2014</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>ConvolverNode.</td>
<td>Use of convolverNode with HRIR IRCAM filters.</td>
<td>24/02/2014</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Visual Interface.</td>
<td>Creation of visual interface with binding libraries.</td>
<td>03/03/2014</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Crossfading filters in realtime.</td>
<td>Creation of crossfading in real time for obtain smooth transition between different filters.</td>
<td>17/03/2014</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>binauralIIR.</td>
<td>Binaural synthesis with model of HRTF filters.</td>
<td>25/05/2014</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Public API</td>
<td>API implementation.</td>
<td>18/06/2014</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>Critical Review</td>
<td>ArnauJulia_CriticalReview.pdf</td>
<td>28/04/2014</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>Demos</td>
<td>Creation of demos with the developed software.</td>
<td>16/06/2014</td>
</tr>
</tbody>
</table>

*Table A.10: Milestones*
A.0.2  Time Plan (Gantt diagram)

Figure A.1: Gantt Diagram
Appendix B

Free HRTFs databases

A list of free measured HRTF databases is attached:

- IRCAM, France
  
  Web http://recherche.ircam.fr/equipes/salles/listen/index.html
  
  Number of subjects 51
  Number of directions 187

- CIPIC, University of California Davis, USA
  
  Web http://interface.cipic.ucdavis.edu/sound/hrtf.html
  
  Number of subjects 45
  Number of directions 1250

- Acoustic Information Systems Laboratory, Tohoku University, Japan
  
  Web http://www.ais.rie.tohoku.ac.jp/lab/db-hrtf/index.html
  
  Number of subjects 3
  Number of directions 454

- Bill Gardner and Keith Martin, MIT Media Lab, Massachusetts Institute of Technology, USA
  
  Web http://sound.media.mit.edu/resources/KEMAR.html
  
  Number of subjects KEMAR dummy head
  Number of directions 710
Appendix C

Performance tables

Legend:

- **binauralFIR**: binauralFIR node with a HRTF database of length of 256 sample @44.1KHz

- **binaural Biquad 1024**: binauralBiquad node with 6 biquads (24 coefficients) @44.1KHz with a ScriptProcessorNode bufferSize of 1024 samples.

- **binaural Biquad 4096**: binauralBiquad node with 6 biquads (24 coefficients) @44.1KHz with a ScriptProcessorNode bufferSize of 4096 samples.

- **binaural Biquad 16384**: binauralBiquad node with 6 biquads (24 coefficients) @44.1KHz with a ScriptProcessorNode bufferSize of 16384 samples.

- **binaural Biquad native biquad 1024**: modified binauralBiquad node by using native biquadFilterNode instead of biquadFilter library @44.1KHz with a ScriptProcessorNode bufferSize of 1024 samples.

- **Panner Node**: native PannerNode. It uses a HRTF database of length of 256 sample @44.1KHz
<table>
<thead>
<tr>
<th>Number of sources</th>
<th>binaural FIR</th>
<th>binaural Biquad 1024</th>
<th>binaural Biquad 4096</th>
<th>binaural Biquad 16384</th>
<th>binaural Biquad native biquad 1024</th>
<th>Panner Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2%</td>
<td>5.5%</td>
<td>5.2%</td>
<td>5%</td>
<td>2.7%</td>
<td>1.4%</td>
</tr>
<tr>
<td>2</td>
<td>3%</td>
<td>17%</td>
<td>10%</td>
<td>10%</td>
<td>4.6%</td>
<td>2%</td>
</tr>
<tr>
<td>3</td>
<td>4.2%</td>
<td>36%</td>
<td>13.5%</td>
<td>17%</td>
<td>6.6%</td>
<td>2.3%</td>
</tr>
<tr>
<td>4</td>
<td>5.2%</td>
<td>73%</td>
<td>20.6%</td>
<td>25%</td>
<td>9.5%</td>
<td>3%</td>
</tr>
<tr>
<td>5</td>
<td>6.3%</td>
<td>100.2%</td>
<td>27%</td>
<td>29.5%</td>
<td>14%</td>
<td>3.1%</td>
</tr>
<tr>
<td>6</td>
<td>7%</td>
<td>-</td>
<td>36%</td>
<td>35%</td>
<td>17%</td>
<td>3.5%</td>
</tr>
<tr>
<td>7</td>
<td>8.1%</td>
<td>-</td>
<td>42%</td>
<td>45%</td>
<td>20%</td>
<td>3.8%</td>
</tr>
<tr>
<td>8</td>
<td>9.1%</td>
<td>-</td>
<td>47%</td>
<td>47%</td>
<td>23%</td>
<td>4.4%</td>
</tr>
<tr>
<td>9</td>
<td>10.1%</td>
<td>-</td>
<td>76%</td>
<td>61%</td>
<td>28%</td>
<td>4.7%</td>
</tr>
<tr>
<td>10</td>
<td>11.2%</td>
<td>-</td>
<td>100%</td>
<td>84%</td>
<td>32%</td>
<td>5.3%</td>
</tr>
<tr>
<td>11</td>
<td>12.2%</td>
<td>-</td>
<td>-</td>
<td>87%</td>
<td>36%</td>
<td>5.6%</td>
</tr>
<tr>
<td>12</td>
<td>13.2%</td>
<td>-</td>
<td>-</td>
<td>100%</td>
<td>41%</td>
<td>6%</td>
</tr>
<tr>
<td>13</td>
<td>14.2%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>46%</td>
<td>6.4%</td>
</tr>
<tr>
<td>14</td>
<td>15.2%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>52%</td>
<td>6.9%</td>
</tr>
<tr>
<td>15</td>
<td>16.3%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>57%</td>
<td>7.5%</td>
</tr>
<tr>
<td>20</td>
<td>21.2%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>84%</td>
<td>9.5%</td>
</tr>
<tr>
<td>25</td>
<td>26.3%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>100%</td>
<td>12%</td>
</tr>
<tr>
<td>30</td>
<td>31.4%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>14%</td>
</tr>
<tr>
<td>35</td>
<td>36.9%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>16%</td>
</tr>
<tr>
<td>40</td>
<td>42.3%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>18.3%</td>
</tr>
<tr>
<td>45</td>
<td>49%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>20.3%</td>
</tr>
<tr>
<td>50</td>
<td>57%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>23%</td>
</tr>
<tr>
<td>55</td>
<td>65%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>25%</td>
</tr>
<tr>
<td>60</td>
<td>75%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>27%</td>
</tr>
<tr>
<td>65</td>
<td>84%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>29%</td>
</tr>
</tbody>
</table>

Table C.1: Results iMac 2009
Bibliography


