Audio Engineering Society
Convention e-Brief

Presented at the 140th Convention
2016 June 4–7 Paris, France

Implementation of Faster than Real Time Audio Analysis for use with Web Audio API: An FFT Case Study

Luis Joglar-Ongay[1], Christopher Dewey[2], and Dr. Jonathan P. Wakefield[3]

[1] University of Huddersfield, Queensgate, Huddersfield, West Yorkshire, HD1 3DH, UK
L.JoglarOngay@hud.ac.uk

[2] University of Huddersfield, Queensgate, Huddersfield, West Yorkshire, HD1 3DH, UK
C.Dewey@hud.ac.uk

[3] University of Huddersfield, Queensgate, Huddersfield, West Yorkshire, HD1 3DH, UK
J.P.Wakefield@hud.ac.uk

ABSTRACT

There is significant interest in the audio community in developing web-based applications using HTML5 and Web Audio API. Whilst this newly emerging API goes some way to provide audio analysis in the web browser it is limited to a relatively basic FFT with fixed Blackman windowing and no overlap facility. Most previously documented solutions to this issue operate in real time. This paper demonstrates how to perform more sophisticated, faster than real time FFT analysis for use within Web Audio applications. It makes use of the Web Audio API and the dsp.js library. Academics and researchers can use this paper as a tutorial to develop similar solutions within their own web based audio applications.

1. INTRODUCTION

The Web Audio API is defined by the WC3 as a "high-level JavaScript API for processing and synthesizing audio in web applications" [1] and is increasingly being harnessed by developers, creative technologists and musicians alike to develop interactive online applications [2]. In its most simplistic form the Web Audio API uses the concept of nodes, enabling the user to connect a source node (audio input) to a destination node (audio output) via a range of sub-nodes. As part of our research, which focuses on

developing novel assistive tools for music production, we have experimented with various aspects of the API to develop applications. This tutorial focuses on our exploration of performing Fast Fourier Transforms (FFTs) on pre-recorded audio material in the web browser and aims to provide a guide for researchers wanting to explore these features of the API. The paper starts by critiquing the features of API's analyser node before discussing other ways of performing FFTs in the web browser. We then outline an alternative, more sophisticated method for performing faster than real-time (offline) audio analysis for researchers interested in audio analysis. All our experiments were conducted using Google Chrome [3].

2. THE ANALYSER NODE

Real time FFTs can be performed natively in the Web Audio API by connecting a source node to a destination node via an analyser node (named AnalyserNode in the API) allowing the user to extract the FFT data in real time. Unlike MaxMSP's pfft~ object, the analyser node is not capable of processing the audio in response to the FFT analysis or performing an inverse FFT after frequency domain manipulation. It's also worth noting that the analyser node is unique in that it will work with or without a destination node connected [4].

The AnalyserNode has several properties that can be defined. The FFT size can be set using the AnalyserNode.fftSize property and the corresponding number of FFT bins can be queried using the read only property AnalyserNode.frequencyBinCount. The AnalyserNode.smoothingTimeConstant property can be set between 0 and 1 to provide a smoothing between consecutive FFT frames. There are no properties provided however to control the windowing function or overlap which is fixed to a non-overlapped Blackman window. This makes it arguably unsuitable for audio processing.

The purpose of the analyser node (and arguably the Web Audio API in general) has been subject to some debate in online forums with regard to these limitations [5]. It is apparent from these discussions that this node has not been designed for DSP or audio analysis/processing [5]. Rather, it is designed mainly to be able to visually display the audio source in the time and frequency domain in real time [6] and has been widely used in the creation of music visualisations [7].

The Web Audio API features an offline audio context [8]. In contrast to the well documented online context the offline context renders the audio input's data as fast as possible to an audio buffer (named AudioBuffer) using a callback. The PCM data can then be extracted using the getChannelData method on the AudioBuffer.

In order to access and process the audio data the now deprecated ScriptProcessorNode must be created in the used audio context. This node provides the capability of accessing every new buffer through the event audioProcessingEvent to apply any Javascript algorithm to the audio data [1].

This is scheduled to be replaced by the Audio Workers for streamlined audio processing in the Web Audio API. The Audio Workers are similar to the Web Workers which bring multi-threading capabilities to this traditionally single-threaded Javascript scripting language. Although Audio Workers are defined in the latest draft specification [1] at the time of writing no browser supports their use.

We had hoped to perform faster than realtime audio analysis this way, however, unfortunately there are known issues surrounding using the AnalyserNode and ScriptProcessorNode in the Offline Context with rendered audio buffers [9, 10].

This presents audio developers with two problems. Firstly, there is no means of performing faster than real time audio analysis solely using the Web Audio API. Secondly, the AnalyserNode is not appropriate for performing serious DSP/audio analysis given the limitations discussed above. The next part of this paper outlines one solution to these issues.

3. FASTER THAN REAL TIME FFTS IN JAVASCRIPT

Figure 1 below provides an overview of the algorithm with each step described in this section. It is important to note that the DSP.js [11] library has been used to perform the windowing and FFT functions and is required in order for this solution to work.

Firstly the XMLHttpRequest() is called to upload the audio file into an ArrayBuffer. Once the file has been loaded the AudioContext.decodeAudioData() function is called to decode the raw data from the ArrayBuffer to an AudioBuffer that contains the audio information. The PCM samples can then be stored in the browsers memory using AudioBuffer.getChannelData(0). Now the data is appropriately stored it is possible to loop through the audio buffer and perform any operation as fast as computationally possible.

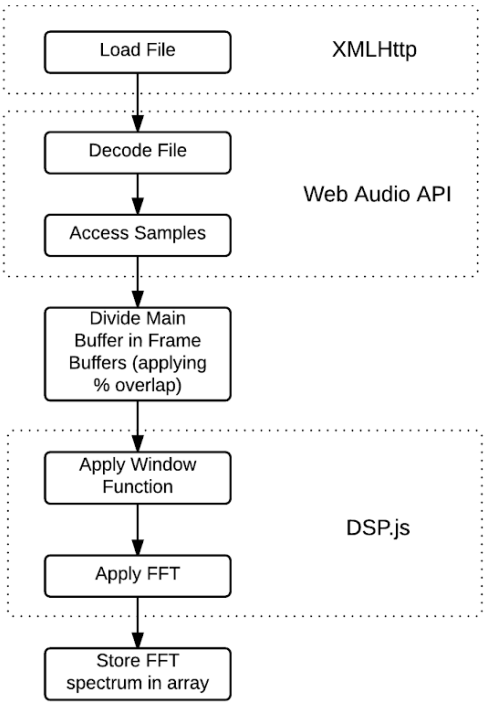

Figure 1 Faster than real time FFT implementation

The following steps follow the standard windowed, overlapped FFT process:

1. The audio buffer is split into a series of smaller buffers of the desired FFT length and taking into account the required overlap in the extraction process.

2. Each of these smaller buffers is then windowed and an FFT performed using functions provided in the DSP.js library.

3. The FFTs from step 2 are stored in an array for further use.

The JavaScript implementation of the above is available for download from:

https://github.com/ljoglar/FasterThanRealTimeFFT

4. CONCLUSIONS

The Web Audio API was critiqued with regard to performing FFT analysis. An alternative approach was outlined that uses DSP.js in conjunction with the Web Audio API for performing faster than real time FFT analysis.

It is hoped that academics and researchers can use this paper as a tutorial to develop similar solutions within their own web based audio applications.

5. REFERENCES

[1] Web Audio Working Draft. Retrieved from https://www.w3.org/TR/webaudio/ Date Accessed 23.03.16

[2] Chris Lowis Web Audio Weekly. Retrieved from http://blog.chrislowis.co.uk/waw.html Date Accessed 23.03.16

[3] https://github.com/GoogleChrome/web-audio-samples Date Accessed 23.03.16

[4] AnalyserNode. Retrieved from https://developer.mozilla.org/en/docs/Web/API/AnalyserNode Date Accessed 23.03.16

[5] https://github.com/WebAudio/web-audio-api/issues/468 Date Accessed 23.03.16

[6] Visualizing Audio #1 Time Domain. Retrieved from http://apprentice.craic.com/tutorials/30 Date Accessed 23.03.16

[7] Exploring the HTML5 Web Audio: visualizing sound. Retrieved from http://www.smartjava.org/content/exploring-html5-

web-audio-visualizing-sound Date Accessed 23.03.16

[8] Offline Audio Context. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/OfflineAudioContext

[9] https://bugzilla.mozilla.org/show_bug.cgi?id=1031851 Date Accessed 23.03.16

[10] https://bugs.chromium.org/p/chromium/issues/detail?id=595032 Date Accessed 23.03.16

[11] dsp.js. Retrieved from https://github.com/corbanbrook/dsp.js/ Date Accessed 23.03.16