Meyda: an audio feature extraction library for the Web Audio API *

Hugh Rawlinson Nevo Segal Jakub Fiala
Goldsmiths, University of London
New Cross
London
SE14 6NW
mu202hr@gold.ac.uk mu202ns@gold.ac.uk mu201jf@gold.ac.uk

ABSTRACT
There are many existing native libraries and frameworks for audio feature extraction used in multimedia information retrieval. Many are dependent on highly optimised low level code to cope with the high performance requirements of realtime audio analysis. In this paper, we present a new audio feature extractor library, Meyda[1], for use with the JavaScript Web Audio API, and detail its benchmarking results. Meyda provides the first library for audio feature extraction in the web client, which will enable music information retrieval systems, complex visualisations and a wide variety of technologies and creative projects that previously were relegated to native software. The Meyda project, including source code and documentation is released under an MIT license.

Categories and Subject Descriptors
H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Systems*

General Terms
Algorithms, Performance, Design

Keywords
Audio Feature Extraction, Web Audio, Music Information Retrieval

[1] `github.com/hughrawlinson/meyda/tree/wac-release`

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

1. INTRODUCTION
The Meyda (from Hebrew - "information") library provides a research basis for further development of real-time client-side feature extraction in the context of the Web Audio API. It consists of an initial set of useful features for potential applications including multimedia information retrieval systems, complex audio visualisation, video game audio, and accessible demonstrations of related concepts for education across a variety of low-cost platforms. These initial features were chosen with influence from the set of features implemented in the YAAFE Audio Feature Extractor for Python, filtered by what was feasible in the context of the Web Audio API.

2. BACKGROUND
Audio feature extraction is a very important field of research cultivated mainly in the Multimedia Information Retreival (MIR) community, whose main purpose is to "provide new paradigms and methods for searching through the myriad variety of media all over the world" [1]. Technologies developed within this community include algorithms for computer vision, media query-based search, media classification etc.

With the relatively recent emergence of the "Web APIs" including the Web Audio API [2], we are convinced that the Web is ultimately becoming a complete software platform capable of results comparable to hardware-dependent systems. It was with these facts in mind that we conceived the idea of enabling high-fidelity feature extraction in a Web environment, in particular the JavaScript-driven Web Audio API, whose processing features including the ScriptProcessorNode [2] are crucial for accessing raw audio data and performing analysis.

The applications of such an implementation are numerous: speech recognition, for instance, utilises the *Mel-frequency cepstral coefficients* feature [3]. Despite the fact that there have been efforts to integrate speech recognition in web search interfaces [4], these included an integrated low-level solution built in the browser architecture, not unlike other Web APIs. Other applications may use a server-side extraction library. Using a lightweight JavaScript-based implementation to obtain necessary feature data and apply recognition algorithms may prove a more flexible, customisable, and broadband-independent option for professional and academic use.

3. RELATED RESEARCH

Research in audio feature extraction has, despite being quite an established scientific field, found a great amount of new applications in the past three decades, in particular in the MIR community, with researchers beginning to adopt audio-based algorithms for not only sound media, but also corresponding video analysis, e.g. Rautiainen et al. [5]. Subsequently, after 2000, a substantial amount of audio feature extraction methods have been implemented as software products, even outside of academia. These are utilised in a variety of instances, including speech recognition [6], music identification [7], multi-modal video summarisation [8] etc.

Currently, there are already many native libraries for feature extraction on both Unix- and Windows-based systems. In particular, we were influenced by YAAFE [9], an open-source command line interface (CLI) for offline audio feature extraction [9], with most features adopted from the extensive list released as part of the CUIDADO project [10]. YAAFE is implemented in C++ and provides a Python wrapper for ease of use. We took cues as to which audio features to initially target for support from YAAFE's feature list. What is more, we used this library alongside the Java implementation of jAudio [11] during the testing process as a reference for validating extraction output. As for jAudio, it is an audio feature extraction library implemented in Java [11]. Accessible through a CLI or through a graphical user interface (GUI), it provides a number of statistical feature extractors as well as several semantic, MIR-oriented functions such as beat detection.

4. IMPLEMENTATION DECISIONS

4.1 Design

Perhaps the most significant design goal of Meyda was to provide simple integration into the existing Web Audio API node graph architecture. It integrates directly into the Web Audio API by accepting an AudioContext[2] and any node that implements the AudioNode interface as a source in the constructor of the object to be connected to Meyda's internal ScriptProcessorNode. We chose this architecture to provide the maximum ease of integration for users of the library. Meyda is intended as a terminating "pseudo-node" in the graph of nodes because it does not provide any transformations, rather, it takes data from the audio in the node graph and exposes it for usage in other contexts. As such, it does not implement the AudioNode interface because it should not be connected to a DestinationNode or any other further nodes.

There is significant continuing discussion as to how best to provide frequency domain data in the Web Audio API working group, in particular a split [12] between providing a set of Fourier Transforms in the web audio, or waiting for the Web Array Math API[13] specification, which contains Fourier Transform functionality, to be formally submitted, reviewed, and implemented. Since the AnalyserNode containing an FFT implementation has a 'black box'-style interface with few options and seemingly logarithmically scaled output, we used the existing open source *jsfft* implementation of the Fast Fourier Transform (FFT) in JavaScript[14] which was released under an MIT License. Prior to the FFT, the time domain data is passed through a Hanning windowing function to improve the accuracy of the resulting frequency spectrum data.

Meyda was intended to provide a real-time Audio Feature Extraction API, and as such we placed great import on the optimisation of the feature extraction implementations. At a minimum, they were required to operate faster than real-time on what we decided were 'reasonable' buffer sizes based on research into typical usage implementations of feature extraction on native platforms.

Concerning data output design, we found it important to provide some level of flexibility to the user, in order to enable both instantaneous (buffer-independent), and buffer-specific extraction. This is to ensure that, although the user can specify a custom extraction time or frequency, each 'frame' of the incoming audio stream can be analysed specifically. Therefore we decided to implement two output systems:

- **Immediate,** with a Meyda.get(features) method returning instantaneous data from the currently analysed real-time buffer.
- **Synchronized,** whereby a callback is specified in the Meyda constructor, executing desired code in sync with the internal ScriptProcessorNode. This is useful in instances where output is required for every buffer that is analysed by Meyda.

In order to simplify interaction with the API and to enable development of richer application structures, we decided to include a featureInfo object as a property of Meyda, carrying useful information about each of the implemented feature extractor functions, such as output type ("number", "array", "multipleArrays") and, for some of the features, output range for automated visualisation.

4.2 Implemented Features

Our chosen implemented features fit into three distinct categories: perceptual features, time domain features, and spectral domain features. Time domain features are generally related to volume, while frequency statistics describe the shape of the spectrum and imply certain timbral characteristics. Perceptual features are features that are strongly correlated with human perception. As Meyda is a real-time audio analysis framework, the values are calculated on a per-frame basis, where the user of the library can specify the frame-length in the constructor of the Meyda instance. Meyda provides a callback to its enclosed ScriptProcessorNode to access each consecutive buffer of time and frequency domain that passes into the node.

The following is a list of features, and their definitions as implemented in Meyda[2]. These are identical in definition to their corresponding features in YAAFE, unless indicated otherwise.

[2]A more detailed description including implementation details, output information and sources is to be distributed as a text file in Meyda's production package

4.2.1 Time-Domain

RMS
The root mean square of the waveform calculated in the time domain to indicate its loudness. Corresponds to the 'Energy' feature in YAAFE, adapted from Loy's *Musimathics* [15].

Energy
The infinite integral of the squared signal. According to Lathi [16].

Zero Crossing Rate
The number of times that the signal crosses the zero value in the buffer. Corresponds to ZCR in jAudio.

4.2.2 Frequency-Domain

Amplitude Ratio Spectrum (transformation)
Frequency domain amplitude spectrum, a transformation from the complex FFT.

Power Ratio Spectrum (transformation)
Frequency domain power spectrum, a transformation from the complex FFT.

Spectral Slope
A measure of how 'inclined' the shape of the spectrum is. Calculated by performing linear regression on the amplitude spectrum.

Spectral Rolloff
The frequency below which is contained 99% of the energy of the spectrum.

Spectral Flatness
The flatness of the spectrum as represented by the ratio between the geometric and arithmetic means. It is an indicator of the 'noisiness' of a sound.

Spectral Centroid
An indicator of the brightness of a given spectrum, represents the spectral centre of gravity.

Spectral Spread
Indicates the 'fullness' of the spectrum.

Spectral Skewness
Indicates whether or not the spectrum is skewed towards a particular range of values.

Spectral Kurtosis
The 'pointedness' of a spectrum, can be used to indicate 'pitchiness'.

Mel Frequency Cepstral Coefficients
A widely used metric for describing timbral characteristics based on the Mel scale. Implemented according to Huang [17], Davis [18], Grierson [19] and the *librosa*[3] library

4.2.3 Perceptual

Perceptual Loudness
The loudness of the spectrum as perceived by a human, using Bark bands. Outputs an object consisting of Specific Loudness (calculated for each Bark band) and Total Loudness (a sum of the specific loudness coefficients).

Perceptual Spread
How 'full' a human will perceive the sound to be.

Perceptual Sharpness
Perceived sharpness of the loudness Bark coefficients.

4.2.4 Utility extractors

- **Buffer**: a simple Float32Array of sample values
- **Complex Spectrum:** a ComplexArray object carrying both real and imaginary parts of the FFT.

4.3 Example implementation

We have written a basic example of a webpage implementation using Meyda[4] to illustrate our design decisions with regards to how Meyda integrates with the Web Audio API. First, the Meyda library is included in the HTML source. An AudioContext is then instantiated, from which the source node for Meyda is created. The source node can be any of the Audio Nodes in the Web Audio API specification, or a node at the end of a chain in the context graph. At this point, the programmer instantiates a new Meyda object, passing the AudioContext, the source node, and the buffer size for analysis. Once Meyda's instantiation is complete, the user can then query the Meyda instance for the results of its feature extractors. This can be done as a one off query using Meyda's 'get' function, or repeatedly by passing a callback as an additional argument to Meyda constructor, and calling 'start', which will pass data back to the program for every buffer that is analysed.

5. TESTING AND BENCHMARKING

We ran some testing and benchmarking against Meyda on a mid-2010 Macbook Pro with a 2.53GHz Intel Core i5 processor with 4GB of RAM, using a Mozilla Firefox 32.0.3 browser. We feel that this machine represents a relatively standard modern hardware setup for desktop browsing.

5.1 Testing

To ensure accuracy of the implemented feature extractors, we tested the results of Meyda against those of YAAFE and jAudio. The latter was used initially: since jAudio analyses files on a frame average basis, we employed a rather tedious procedure to ensure the same buffer was being analysed:

1. Log time domain buffer and feature set from Meyda in the browser
2. Save the time domain data to a PCM audio file (using a Puredata patch)
3. Load the file into jAudio, set buffer size and feature extractors, and run extraction
4. Compare the results

This approach was partially useful for validating the output of some features. However, we found the above workflow to be rather costly in terms of time. Therefore, we decided to rely on YAAFE, which has a more similar set of extractors and outputs frame-by-frame results. The procedure was thus largely simplified, using a 10-second recording of pure white noise as the analysis input:

1. Log feature set from Meyda in the browser

[3]https://github.com/bmcfee/librosa

[4]http://hughrawlinson.github.io/meyda/examples/basic.html

2. Run YAAFE extraction with the same buffer size from the command line

3. Compare the results from YAAFE CSV and Meyda's console output

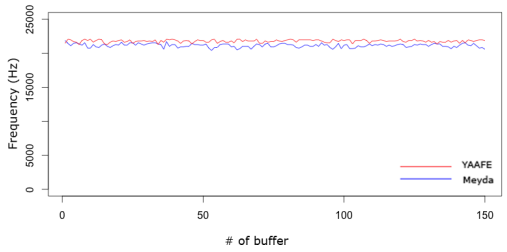

Figure 1: Spectral Rolloff comparison

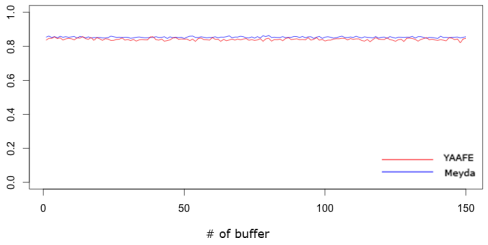

Figure 2: Perceptual Spread comparison

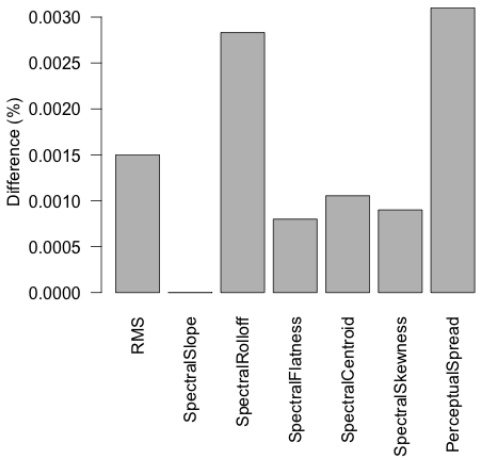

Figure 3: Percentage Differences between YAAFE and Meyda

5.1.1 Results

In figs. 1 to 3, we include plots demonstrating comparisons between the output of several feature extractors in Meyda and YAAFE. These, among others, were used to determine the validity ("sanity-check") of Meyda's results. In particular, they were applied to a recording of pure white noise (in 150 consecutive buffers), which is by definition bound to produce certain logically inferrable values, even when the analysed buffers are not exactly the same. Therefore the values do not match exactly, but rest in the same ranges. The maximum percentage difference between Yaafe and Meyda results was 0.003%(on features where we could calculate[5] the difference), as shown in Figure 3. It is necessary to note that we are still in the process of optimizing and modifying certain algorithms, e.g. the MFCC implementation.

The 'online' extraction paradigm within which Meyda works is reponsible for the slight error in comparison to YAAFE, which works on non-realtime audio. The discrepancy in our testing results is explained by the different start-points of buffers between YAAFE and Meyda. Issues with the 'phasing' of buffers in relation to the time domain samples of the audio result in miniscule differences in the numerical results. Our testing was intended to show that our feature extraction algorithms produce 'sane' results, as Meyda is not intended to precisely match YAAFE. When we start providing parameterisable extractors there will be greater discrepancies between existing libraries as users start to tailor audio features to meet their use case requirements.

5.2 Benchmarking

Assessing the performance of Meyda was necessary to determine that the feature extractors could run comfortably in real-time, and therefore were appropriate for the set of applications that audio feature extraction is widely used in. Despite the fact that there are no comparable implementations of feature extraction libraries specifically targeting the Web Audio API, we ran benchmarking on Meyda to confirm that it was running faster than real-time audio on a relatively standard device. We used benchmark.js to determine running times for each of the extractors separately, then all of them simultaneously.

5.2.1 Results

Benchmarking indicated that Meyda can comfortably run real-time feature extraction in the browser. Computing all of the features simultaneously on one buffer of 512 samples can be run at 288 operations per second, approximately 3.34 times realtime. Furthermore, specific features such as RMS and spectral rolloff were computed on a much quicker time frame, with RMS calculated approximately 700,000 times per second and spectral rolloff slightly above 1,000,000 times per second.

5.3 Mobile browsers

Meyda was tested on two mobile browsers; Firefox 33.0 running on a OnePlus One Android Smartphone (CynanogenMod version 11.0-XNPH38R), and a GeeksPhone Peak (Firefox OS Boot2Gecko 1.3.9.9-prerelease). We were able to simultaneously run all feature extractors except ComplexSpectrum and Loudness on both devices. We aim to make ComplexSpectrum and Loudness both compatible with mobile devices as part of our efforts towards cross-browser compatibility. The fact that these extractors run on mobile devices is very encouraging, as it indicates the potential for applica-

[5]The percentage differences were calculated by taking the absolute value of the individual differences, divided by the feature's output range

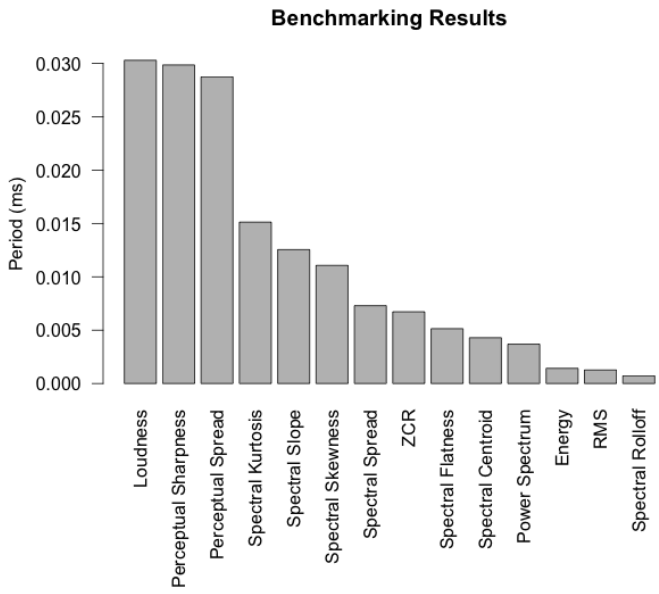

Figure 4: Feature extraction times from a buffer of 512 samples

tions using feature extraction to be used on smaller, cheaper devices, and broaden the potential uses of this technology.

6. FUTURE WORK

It is important to bear in mind that the Web Audio API is an emerging specification, and while it is generally quite feature complete, there are certain features that are important for MIR applications and are not in the current version of the specification. When a best practice method of using custom FFT procedures emerges, whether in the Web Audio API or as part of the proposed Web Array Math API, we intend to add extra feature extractors that depend on it, that are currently unfeasible in real-time. Furthermore, plans have emerged within our team to implement an API for higher-level, semantic sound description, using Meyda. This would encompass tools for musical information retrieval, such as beat detection, harmonic functions detection and timbral and temporal segmentation. Additionally, speech recognition based on Meyda's MFCC extraction may be incorporated within this API.

Some features use constants that have been taken from other papers. For example, spectral rolloff is the frequency below which N% of the energy content is contained, where N is 95[10], 85[20] or 99[9] depending on the source. One way to deal with this would be to add paramaterisation to the feature extractors to allow the user to define the constants that they wish to use, while also providing suitable values.

We intend to improve testing by writing automated unit and integration tests as well as improving our benchmarking suite to more comprehensively test the various combinations of different features at different buffer sizes. There are various tools for this including Travis-CI[21] and Jenkins[22].

Meyda is designed as an open source project, and as such is very much open to contributions from the community in the hopes that we will be left with a sustainable community project. We are particularly interested in contributions related to the optimisation of Meyda, as well as implementations of further feature extractors as listed in *A large set of audio features for sound description (similarity and classification) in the CUIDADO project* [10].

7. CONCLUSIONS

Meyda provides a research basis on which to build further technologies for live music information retrieval systems in the browser. As demonstrated, we achieved our aims of meeting the performance levels required for real-time querying of a music information retrieval system, and anticipate that Meyda will be useful for further applications in the web, including but not limited to information retrieval, visualisation, and in-game use in conjunction with the HTML5 canvas element and WebGL. In order to ensure that Meyda provides accurate and reliable results, we have tested it against several popular feature extraction libraries. In praticular, we have performed extensive cross-checking of Meyda against YAAFE, and proved that we meet the quality standards on various types of audio content. We have also used YAAFE in order to construct Meyda's list of features for it to contain the most relevant and useful features. Meyda is still an ongoing project and the Web Audio API specification is under active development, therefore, there is still further work that would be implemented to Meyda in the near future.

8. ACKNOWLEDGMENTS

The authors would like to thank Dr. Rebecca Fiebrink of Goldsmiths, University of London for providing essential feedback during the development of Meyda. We would also like to thank Dr. Marco Gillies, who along with Rebecca Fiebrink provided feedback on this paper.

9. REFERENCES

[1] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 2, pp. 1–19, Feb. 2006.

[2] P. Adenot, C. Wilson, and C. Rogers, "Web audio api | w3c working draft 10 october 2013." `http://www.w3.org/TR/webaudio/`. Work in progress, accessed: 2014-10-23.

[3] T. Ganchev, N. Fakotakis, and G. Kokkinakis, "Comparative evaluation of various mfcc implementations on the speaker verification task," in *10th International Conference on Speech and Computer (SPECOM 2005), Vol. 1*, pp. 191–194, 2005.

[4] G. Shires and H. Wennborg, "Web speech api specification." `https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html`. W3C Community Group Report, accessed: 2014-10-23.

[5] M. Rautiainen, T. Seppanen, J. Penttila, and J. Peltola, "Detecting semantic concepts from video using temporal gradients and audio classification," in *Proceedings of the 3rd International Conference on Image and Video Retrieval*, pp. 260–270, 2003.

[6] D. O'Shaughnessy, "Invited paper: Automatic speech recognition: History, methods and challenges," *Pattern Recogn.*, vol. 41, pp. 2965–2979, Oct. 2008.

[7] A. Wang, "The shazam music recognition service," *Commun. ACM*, vol. 49, pp. 44–48, Aug. 2006.

[8] J.-Y. Pan, H. Yang, and C. Faloutsos, "Mmss: Multi-modal story-oriented video summarization," in *Proceedings of the Fourth IEEE International Conference on Data Mining*, ICDM '04, (Washington, DC, USA), pp. 491–494, IEEE Computer Society, 2004.

[9] B. Mathieu, S. Essid, T. Fillon, J. Prado, and G. Richard, "Yaafe, an easy to use and efficient audio feature extraction software.," in *ISMIR*, pp. 441–446, 2010.

[10] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the cuidado project," 2004.

[11] C. McKay, I. Fujinaga, and P. Depalle, "jaudio: A feature extraction library," in *Proceedings of the International Conference on Music Information Retrieval*, pp. 600–3, 2005.

[12] "Proper fft / ifft missing." `https://github.com/WebAudio/web-audio-api/issues/248`. Accessed: 2014-10-09.

[13] "Web array math api specification | unofficial draft." `http://opera-mage.github.io/webarraymath/`. Accessed: 2014-10-09.

[14] "jsfft | small, efficient javascript fft implementation for node or the browser.," 2014.

[15] G. Loy, *Musimathics: The Mathematical Foundations of Music, Volume 1*. The MIT Press, 2006.

[16] B. P. Lathi, *Modern Digital and Analog Communication Systems 3e Osece*. Oxford University Press, 3rd ed., 1998.

[17] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[18] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, pp. 357–366, Aug 1980.

[19] M. Grierson, "Maximilian: A cross platform c++ audio synthesis library for artists learning to program.," in *Proceedings of International Computer Music Conference*, 2010.

[20] "jaudio 1.0 feature appendix | spectral rolloff," 2006.

[21] "Travis ci - free hosted continuous integration platform for the open source community," 2014.

[22] "Jenkins - an extendable open source continuous integration server," 2014.

[23] O. Gillet and G. Richard, "Automatic transcription of drum loops," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, vol. 4, pp. iv–269, IEEE, 2004.