AN AUDIO PROCESSING LIBRARY FOR MIR APPLICATION DEVELOPMENT IN FLASH

Jeffrey Scott[†], **Raymond Migneco**[†], **Brandon Morton**[†],**Christian M. Hahn**[‡]
Paul Diefenbach[‡], **Youngmoo E. Kim**[†]
Electrical and Computer Engineering, Drexel University[†]
Media Arts and Design, Drexel University[‡]
{jjscott, rmigneco, bmorton, cmhahn, pjdief, ykim }@drexel.edu

ABSTRACT

In recent years, the Adobe *Flash* platform has risen as a credible and universal platform for rapid development and deployment of interactive web-based applications. It is also the accepted standard for delivery of streaming media, and many web applications related to music information retrieval, such as Pandora, Last.fm and Musicovery, are built using Flash. The limitations of Flash, however, have made it difficult for music-IR researchers and developers to utilize complex sound and music signal processing within their web applications. Furthermore, the real-time audio processing and synchronization required for some music-IR-related activities demands significant computational power and specialized audio algorithms, far beyond what is possible to implement using Flash scripting. By taking advantage of features recently added to the platform, including dynamic audio control and C cross-compilation for near-native performance, we have developed the *Audio-processing Library for Flash* (ALF), providing developers with a library of common audio processing routines and affording Flash developers a degree of sound interaction previously unavailable through web-based platforms. We present several music-IR-driven applications that incorporate ALF to demonstrate its utility.

1. INTRODUCTION

The use of web applications is now commonplace due to the widespread availability of broadband connections, improved client processing power, and the capabilities afforded by Adobe Flash. Flash has become the dominant platform for the development of web-based interactive media applications by providing tools for easily implementing rich graphics, animation and user interface controls as well as cross-platform deployment. Despite its popularity, however, Flash's support for sound and music processing has historically been limited. ActionScript, Flash's native development language, was never intended to accommodate computationally intensive algorithms, such as the signal processing required for real-time audio feature extraction and analysis.

Recognizing the potential for developing audio- and music-centric applications on the web, we have developed the *Audio processing Library for Flash* (ALF), which addresses the audio limitations of the Flash platform. ALF is based on Flash version 10 and capitalizes on the the recently introduced Adobe *Alchemy* framework, which allows existing algorithms written in C/C++ to be compiled into byte code optimized for the ActionScript Virtual Machine for significantly improved performance [1, 2]. By utilizing the dynamic audio capabilities recently added to Flash 10 and the computational benefits of Alchemy, ALF provides Flash developers with a library of common audio processing routines that can be incorporated into applications, such as spectral feature extraction and analysis, filtering and reverberation.

By including real-time audio processing capabilities to Flash, ALF provides web applications with an additional degree of sound interaction that has previously only been available on native PC platforms. For example, ALF is capable of supporting music-based games in Flash requiring responses from the player precisely timed to music. Although ALF can be used to enhance the audio of any Flash application, our goal is to enable a new form of web apps that can be driven by user-supplied audio. This potentially allows a user to choose a wide range of customized musical inputs, such as selections from their personal collection or completely user-generated music content (song remixes and mashups, which are becoming increasingly commonplace). As we will demonstrate, ALF facilitates the development of games that are dynamically driven by the acoustic features of songs from a user's music library, thus creating unique game play experiences depending on the provided audio content.

2. RELATED WORK

There are many software packages available that provide libraries for feature extraction and audio synthesis that exist as open source projects for research and development. While many provide similar functionality, each library was developed to address particular implementation issues, such as cross-platform support, computational efficiency and ease of implementation. In this section, we provide a brief description of some existing libraries.

Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals) is an audio processing and MIR framework built in C++ with a GUI based on Qt4 [3]. The project includes a wide variety of functions for analysis and synthesis as well as audio features and classification algorithms. Being one of the first such projects, the scope of Marsyas is significant and it has been used in many research projects as well as commercial endeavors.

jAudio was developed to be an easy to use Java-based system for feature extraction. The cross-platform nature of Java and GUI tools were the motivating factors for the choice of development language. The creators attempted to make the system as easily extensible as possible, avoid redundant computation, and ensure the algorithms were separate from other functionality to increase ease of portability [4].

M2K is a project under the International Music Information Retrieval System Evaluation Laboratory which is based off of the Data to Knowledge (D2K) machine learning and data mining environment [5]. D2K is employs a visual programming environment in which users connect modules together to prototype algorithms. The M2K project has taken this framework and built in an array of MIR tools for rapid development and testing of MIR systems.

The *MIRToolbox* is an audio feature extraction library built in MATLAB that emphasizes a modular, parameterizable framework [6]. The project offers a wide range of low-level and high-level features as well as tools for statistical analysis, segmentation and clustering.

CLAM is an analysis/synthesis system written in C++ designed to be entirely object-oriented to allow for significant re-usability of code and functionality [7]. It provides audio and MIDI input/output, supports XML and provides tools for data visualization.

FEAPI is a platform-independent programming application interface for low-level feature extraction written in C [8]. In contrast to the previously described systems, FEAPI allows developers to create their own applications using C/C++ without being required to use the interfaces designed to work with the above libraries.

3. IMPLEMENTATION

The driving force behind the development of ALF was to provide developers with an efficient, cross-platform and open source MIR and audio synthesis library. By choosing Flash as the development platform, we target developers seeking to rapidly develop and deploy web-based and/or cross-platform desktop applications. As we will discuss, the multi-layered and open source architecture of ALF also permits ease of development for programmers with various expertise and does not require prior knowledge or experience in audio programming.

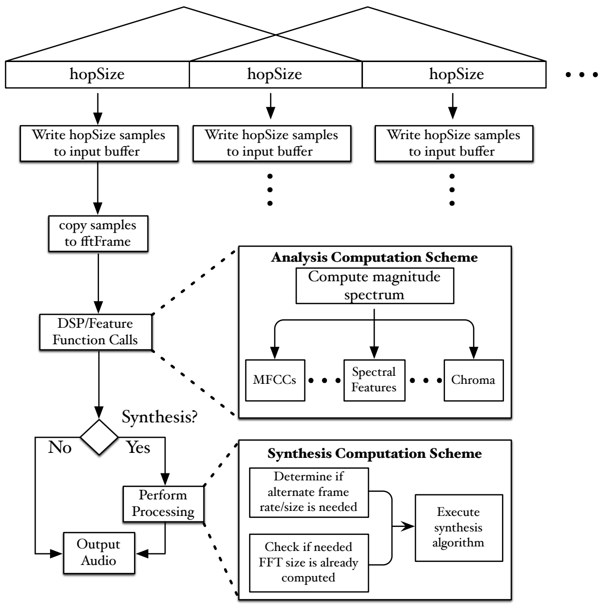

hopSize
hopSize
hopSize
Write hopSize samples to input buffer
Write hopSize samples to input buffer
Write hopSize samples to input buffer
copy samples to fftFrame
DSP/Feature Function Calls
Analysis Computation Scheme
Compute magnitude spectrum
MFCCs
Spectral Features
Chroma
Synthesis?
No
Yes
Perform Processing
Output Audio
Synthesis Computation Scheme
Determine if alternate frame rate/size is needed
Check if needed FFT size is already computed
Execute synthesis algorithm

Figure 1. Frame-based computation and processing flow in ALF.

3.1 Architecture

The dynamic audio functionality in the current version of Flash is somewhat asynchronous, allowing sound to be processed outside of the main application thread. Thus the DSP routines can execute independently of the Flash script rather than having to wait for C/C++ functions to finish executing, allowing front-end UI and other operations to continue if they are not dependent on data computed using ALF functions.

There are several layers of abstraction in ALF providing a flexible framework with various levels of control depending on the needs of the developer. The heavy computation is executed by the C/C++ functions which are compiled using the Adobe supplied Alchemy compiler for use with ActionScript (AS). We provide a basic AS wrapper to properly handle the shared memory management between C/C++ and ActionScript for those wishing to have basic access to the C functionality. The top layer streamlines audio input/output and provides simple calls to perform feature extraction and analysis-synthesis tasks. The entire project is open source so that a developer may customize the architecture to meet application-specific needs. ALF is fully documented and is currently available via a subversion repository online [1] .

To ensure tight synchrony between the video and audio output in Flash, the processing flow was developed according to the diagram shown in Figure 1. The frame size is set by the the video frame rate since ALF is designed with graphical oriented applications in mind, thus the time-frequency resolution of the system is also determined by this parameter. Whenever possible, a single FFT is used in computing the features returned to the user, however, certain algorithms require transforms of sizes other than the

[1] http://music.ece.drexel.edu/ALF

Table 1. ALF Functions

	Function Name	Description
Analysis	Spectrum	Computes the magnitude spectrum using the FFT algorithm
	Harmonics	Finds the harmonics of the frequency spectrum
	MFCC	Calculates the Mel-Frequency Cepstral Coefficients
	LPC	Performs Linear Prediction and returns the coefficients and gain
	Bandwidth	The frequency range present in the signal
	Centroid	The center of gravity of the frequency spectrum
	Flux	The change in energy from the previous frame
	Intensity	Calculates the energy of the spectrum
	Rolloff	The frequency below which %85 of the spectral energy lies
	Autocorrelation	Computes the autocorrelation via the FFT
	Chroma	An representation of the spectral energy present in the 12 individual semitones
	Beat Tracking	Returns whether a beat occurs on each frame (based on bandwise autocorrelation)
Synthesis	Filter	Filters the audio signal - FIR and IIR implementation
	Reverb	Applies reverb by using a room impulse response (RIR) as an FIR filter
	Phase Vocoder	Alters the tempo and/or pitch of the audio

default size. A shared buffer system is also used so that we can perform operations at variable frame rates and overlap lengths without having to read in the data again using different frame sizes.

3.2 Performance

As previously mentioned, the computationally intensive routines in ALF are implemented in the Alchemy-optimized C code to avoid the limitations of ActionScript. While slightly slower than native C code, the Alchemy-optimized code provides significant performance gains over identical algorithms implemented with ActionScript. In a related paper, we performed a benchmark analysis of the FFT algorithm using the ActionScript *as3mathlib* implementation versus our Alchemy-compiled C implementation as well as Java's JTransforms. Averaging the computation speed over 10,000 iterations, we showed our implementation to be nearly 30 times faster than the ActionScript version [1]. The results of this performance comparison are outlined in Table 2. These computational gains open up myriad possibilities for developing interactive music-IR driven applications in the Flash framework.

Table 2. Comparison of FFT Computation Time for ActionScript and Alchemy-compiled C code in milliseconds.

Target	**FFT Size**					
Platform	**8192**	**4096**	**2048**	**1024**	**512**	**256**
ActionScript	45.157	20.818	9.276	4.460	2.041	0.925
Java	20.703	9.393	4.345	1.956	0.901	0.385
Alchemy-C	1.371	0.628	0.297	0.139	0.067	0.034

3.3 ALF Functions

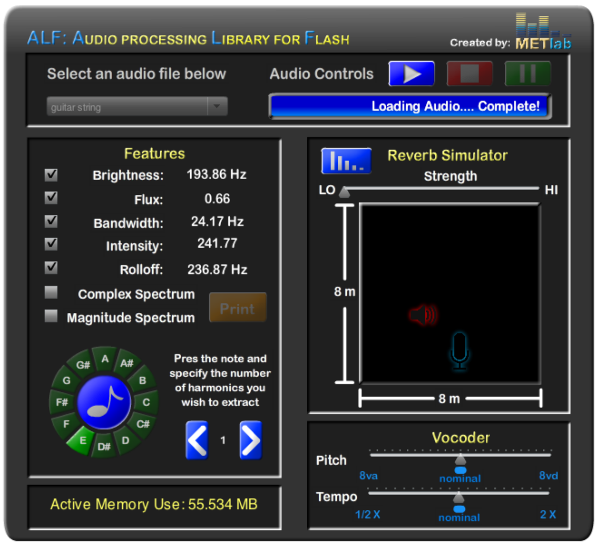
The functions available in ALF are categorized as either "analysis" or "synthesis" and are outlined in Table 1. The analysis functions include several spectral processing routines and features, such as partial extraction and MFCCs, that are useful in many MIR tasks [9]. Synthesis functions are also available so that the developer can dynamically modify the audio output stream to achieve a desired effect. In a related paper, we discuss the implementation and algorithms used for the reverb and filter functions [2]. The remainder of this section will briefly discuss the implementation of two additional synthesis functions added to ALF: phase vocoding and beat tracking.

The most important consideration in developing the beat tracking algorithm was the stipulation that it run in real-time. Our beat tracking algorithm is based off of that proposed by Klapuri but uses an autocorrelation as opposed to a bank of comb filters for computational efficiency [10]. We first compute the power spectrum and separate it into six octave-based sub-bands. The energy envelope in each sub-band is calculated and the bandwise autocorrelation of these vectors is computed. Summing the resulting six autocorrelations and finding the highest peak after the zeroth lag yields an estimate of the tempo.

The phase vocoder is based on a popular, FFT-based implementation in which overlapping frames (specified by ALF's frame rate) are analyzed and re-synthesized using overlap-add in order to perform pitch and/or time-scale modification in real-time [11]. Each frame is processed by a FFT, which is used to determine the phase offset for each frequency bin and thus the estimated, true bin frequency. Pitch modification is achieved by multiplying the bin frequencies by a pitch shift factor, which shifts the audio's pitch in the desired manner after performing the IFFT. Time stretching is achieved by first applying the appropriate pitch-shift factor, performing an IFFT and and re-sampling the audio frame in the time domain to achieve the desired speed.

4. DEVELOPING WITH ALF

Many of the applications developed with ALF thus far have followed the same basic program structure, which is detailed in Figure 3. Input audio is analyzed on a per frame basis and feature values are returned in real-time for the developer to incorporate into their application. Any additional processing required for synthesis functions is executed in a separate processing chain, which eliminates any computational overhead when synthesis functions are not required.


ALF: Audio Processing Library for Flash
Created by: METlab
Select an audio file below
Audio Controls
Features
Print
Reverb Simulator
Strength
LO
HI
8 m
8 m
Pres the note and specify the number of harmonics you wish to extract
Vocoder
Pitch
8va
nominal
8vd
Tempo
1/2 X
nominal
2 X

guitar string

Loading Audio.... Complete!

Brightness:	193.86 Hz
Flux:	0.66
Bandwidth:	24.17 Hz
Intensity:	241.77
Rolloff:	236.87 Hz

Complex Spectrum
Magnitude Spectrum

Active Memory Use: 55.534 MB

Figure 2. Application demonstrating ALF functionality.

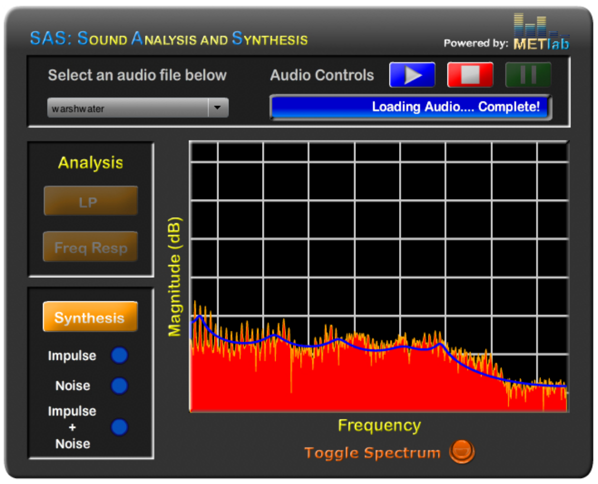
The flexible nature of the architecture shown in Figure 3 combined with the low learning curve of Flash allows developers to rapidly create audio and music-based applications to serve a variety of target audiences and purposes. Possible applications include:

- Music-centric games requiring real-time feature extraction to drive the game environment
- Music exploration interfaces that group user libraries into categories (emotional, genre, etc.) based on extraction and comparison of audio features
- Educational activities for enhancing K-12 curricula in natural science and/or mathematics [12]

Currently, we have several applications developed using ALF for the purpose of audio-based experimentation, analysis/synthesis and music-driven games for entertainment, which we will discuss in the subsequent sections.

4.1 ALF Workbench

Figure 2 demonstrates the ALF Workbench, which allows developers to interactively experiment with different audio files and some of the functions available in ALF. The left panel of the interface showcases the spectral features, which are updated during audio playback and can be exported in a CSV file when the file completes. A pitch wheel is also shown, which allows the user to determine the chromatic notes present in the spectrum of tonal audio. The right panel of the work bench features the room reverb and phase vocoding functions. The reverb interface allows the user to manipulate the positions of the source and listener in a virtual room to simulate immersive environments.



warshwater

Loading Audio.... Complete!

Figure 4. Sound analysis-synthesis app showing linear predictive analysis and magnitude spectrum of speech.

4.2 Beat-Sync-Mash-Coder

Recently, so-called artist "mashups", blending two or more songs in a creative way, have emerged as a popular form of expression for musicians and hobbyists. To this end, the *Beat-Sync-Mash-Coder* is a tool developed for semi-automated, real-time creation of beat-synchronous mashups [13]. This application utilizes the beat-tracking and phase vocoding functions available in ALF along with an intuitive, Flash-based GUI to help automate the task of synchronizing various clips without the complexities incurred with traditional digital audio workstations. The Beat-Sync-Mash-Coder is capable of sustaining real-time phase vocoding on 5-9 audio tracks, depending on the available hardware, thus allowing the user to create dynamic, intricate and musically coherent soundscapes.

4.3 Sound Analysis and Synthesis Application

The application depicted in Figure 4 uses ALF's analysis and synthesis capabilities to perform linear-predictive analysis on speech signals in order to re-synthesize it using different excitation signals. Linear prediction coefficients are extracted at each frame using the Levinson-Durbin recursion to obtain a time-varying model of the vocal tract [14]. The user can then simulate the effect of various excitation sources by using ALF's filtering function to sample the vocal tract with impulsive, noisy or mixed-spectra signals.

4.4 Applications For Music-Driven Gameplay

We present two novel music-driven games which resulted from a collaboration between departments at our university. Both games harness MIR functionality in ALF to create unique and immersive gameplay experiences.

4.4.1 Pulse

Pulse is a musically reactive, side-scrolling platform game that utilizes a player's personal music collection to drive the gameplay. Unlike other music games, which rely on

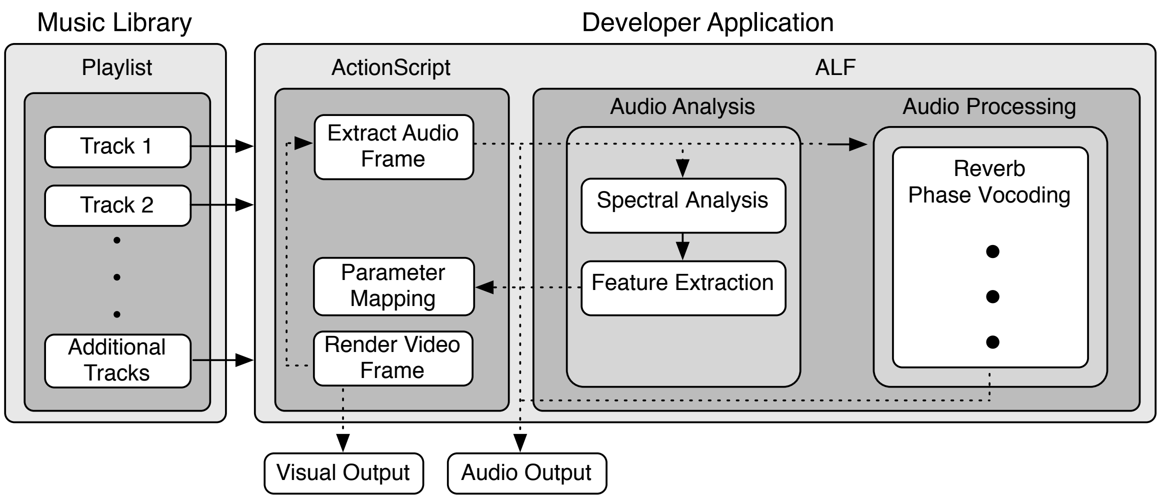


Figure 3. Typical implementation of an application using ALF.

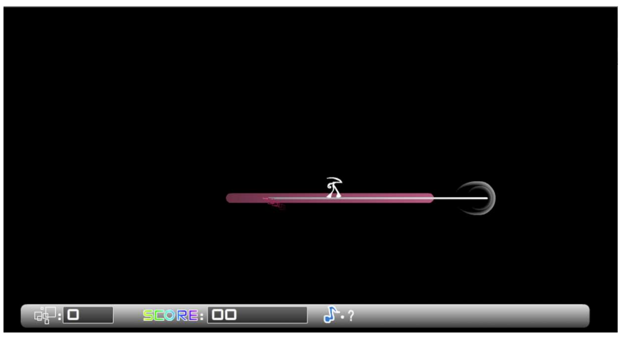
off-line audio analysis to determine the gaming environment, Pulse utilizes ALF functionality to update the game environment in real-time, mapping the quantitative features extracted from the audio to changes in the game's environment variables. This concept increases the replay value of Pulse, since the gamer's experience is limited only by the number of tracks in their music library.

By employing ALF's frame-based processing structure, ALF maps features extracted from the user-selected audio to environment parameters so they are updated in sync with the user-specified frame rate. To permit ample rendering time for the graphics, a "frame-look-ahead" parameter is specified which delays audio playback while features are accumulated from ALF functions. Game environment variables that react to changes in the game's audio include the background scenery, enemies and obstacles of the Pulse character as well as the slope of platform supporting the character. The effect of the audio on the gameplay is evident in Figure 5 where (a) shows the game screen when there is no audio playing and (b) is typical realization of the parameter mapping to game output.

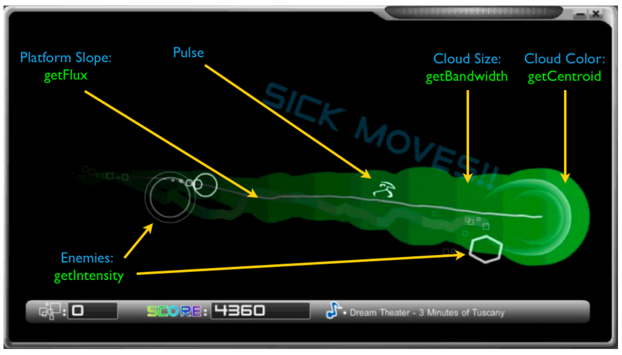
4.4.2 Surge

The concept behind *Surge* is to facilitate exploration of one's own music library though an interactive, DJ-style beat matching game. This expands the concept of audio feature-based gaming environments to include tempo analysis and modification of the game's music. Whereas gameplay in Pulse depends on audio features to dictate the environment, Surge uses game environment parameters to alter the audio in real-time.

The Surge game environment, shown in Figure 6, consists of planets that represent songs the player has provided from their music library. Each song is analyzed with ALF's beat tracker function so that the planet is associated with a song tempo. The game audio depends on which planet the player is on and their proximity to nearby planets. As the player nears a new planet, they will hear the music associated with the new planet. In order to move from planet-



(a)



(b)

Figure 5. Pulse game environment during static (a) and dynamic (b) moments in the game's music.

to-planet, the player (by moving their character) must adjust the rotation of their current planet (altering tempo and beats of the song) to match that of the target planet. That is, the music tempo is adjusted using ALF's phase vocoder according to the planet's rotation, which is dependent upon the player's actions in the game environment.

5. FUTURE WORK

There are several features we would still like to add to ALF including spectral contrast features and other less com-

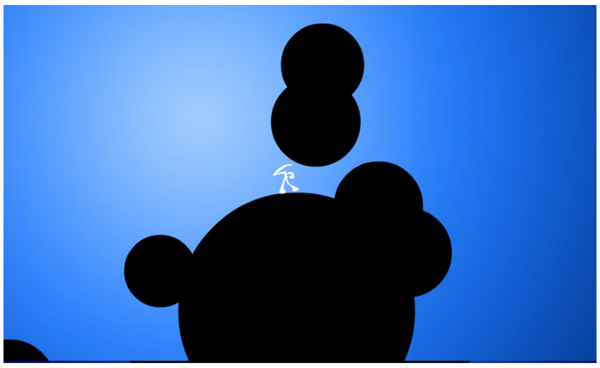


Figure 6. *Surge* game environment.

monly used statistical spectrum descriptors. The most significant component that would augment the usefulness of ALF for the music-IR community would be classification. There are many open source classification libraries available to perform common classification methods such as GMM, SVM, and naive Bayes classification that can be integrated into the current framework.

We will continue to emphasize the real-time capabilities of ALF and optimize the algorithms and architecture to ensure additional algorithms operate in real-time. The newest version of the Flash Player (10.1) will allow byte level access to the audio input creating potential for even further user interaction via real-time analysis and processing of voice/music input to a microphone or other audio device connected to a computer.

6. CONCLUSIONS

The Audio processing Library for Flash affords music-IR researchers the opportunity to generate rich, interactive, real-time music-IR driven applications. The various levels of complexity and control as well as the capability to execute analysis and synthesis simultaneously provide a means to generate unique programs that integrate content based retrieval of audio features. We have demonstrated the versatility and usefulness of ALF through the variety of applications described in this paper. As interest in music driven applications intensifies, it is our goal to enable the community of developers and researchers in music-IR and related fields to generate interactive web-based media.

7. REFERENCES

[1] T. M. Doll, R. Migneco, J. J. Scott, and Y. Kim, "An audio DSP toolkit for rapid application development in flash," in *IEEE International Workshop on Multimedia Signal Processing*, 2009.

[2] R. Migneco, T. Doll, J. Scott, C. Hahn, P. Diefenbach, and Y. Kim, "An audio processing library for game development in Flash," in *Proc. of the IEEE Games Innovations Conference (ICE-GIC 2009)*, Aug. 2009, pp. 201 –209.

[3] G. Tzanetakis and K. Lemstrom, "Marsyas-0.2: A case study in implementing music information retrieval systems," in *Intelligent Music Information Systems: Tools and Methodologies*. Information Science Reference, 2008, pp. 31–49.

[4] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle, "jAudio: A feature extraction library," in *Proc. of the 6th International Conference on Music Information Retrieval*. London, U.K.: ISMIR, 2005.

[5] J. S. Downie, A. F. Ehmann, and X. Hu, "Music-to-knowledge (M2K): a prototyping and evaluation environment for music digital library research," in *Proc. of the 5th ACM/IEEE-CS Joint Conf. on Digital Libraries*. New York, NY, USA: ACM, 2005, pp. 376–376.

[6] O. Lartillot, P. Toiviainen, and T. Eerola, *A Matlab Toolbox for Music Information Retrieval.*, ser. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, 2007, pp. 261–268.

[7] X. Amatriain, M. De Boer, and E. Robledo, "CLAM: An OO framework for developing audio and music applications," in *Proc. of the 17th Annual Conference on Object-Oriented Programming, Systems, Languages and Applications*, 2002.

[8] A. Lerch, G. Eisenberg, and K. Tanghe, "FEAPI: A low level feature extraction plugin api," in *In Proc. of 8th Int. Conference on Digital Audio Effects (DaFX '05)*, 2005.

[9] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[10] A. P. Klapuri, A. J. Eronen, and J. T. Astola, "Analysis of the meter of acoustic musical signals," in *IEEE Transactions Speech and Audio Processing*, 2004, pp. 342–355.

[11] M. Dolson, "The phase vocoder: A tutorial," in *Computer Music Journal*, vol. 10, no. 4. MIT Press, 1986, pp. 14–27.

[12] T. M. Doll, R. V. Migneco, and Y. E. Kim, "Online activities for music informatioin and acoustics education and psychoacoustics data collection," in *Proc. of the International Conference on Music Information Retrieval*. Philadelphia, PA: ISMIR, 2008.

[13] G. Griffin, Y. E. Kim, and D. Turnbull, "Beat-sync-mash-coder: A web application for real-time creation of beat-synchronous music mashups," in *Proc. of the IEEE Conf. on Acoustics, Speech, and Signal Processing*, 2010.

[14] T. F. Quatieri, *Discrete-Time Speech Signal Processing*, A. V. Oppenheim, Ed. Prentice Hall Signal Processing Series, 2002.