

An Audio DSP Toolkit for Rapid Application Development in Flash

Travis M. Doll, Raymond Migneco, Jeff J. Scott, and Youngmoo E. Kim

Music and Entertainment Technology Lab

Department of Electrical and Computer Engineering, Drexel University

3141 Chestnut Street, Philadelphia, PA 19104, U.S.A.

{tdoll, rmigneco, jjscott, ykim}@drexel.edu

Abstract—The Adobe Flash platform has become the de facto standard for developing and deploying media rich web applications and games. The relative ease-of-development and cross-platform architecture of Flash enables designers to rapidly prototype graphically rich interactive applications, but comprehensive support for audio and signal processing has been lacking. ActionScript, the primary development language used for Flash, is poorly suited for DSP algorithms. To address the inherent challenges in the integration of interactive audio processing into Flash-based applications, we have developed the *DSP Audio Toolkit for Flash*, which offers significant performance improvements over algorithms implemented in Java or ActionScript. By developing this toolkit, we hope to open up new possibilities for Flash applications and games, enabling them to utilize real-time audio processing as a means to drive gameplay and improve the experience of the end user.

I. INTRODUCTION

Use of the web as a platform for games and interactive media content has grown dramatically in recent years. Although this is due in part to the widespread availability of broadband connections, faster processors, and more capable browsers and web standards, the Adobe Flash platform has been a (perhaps the most) significant factor. The capabilities and cross-platform architecture of Flash enable graphically rich interactive applications, but comprehensive support for audio and signal processing has been lacking. ActionScript, the primary development language used for Flash, was never intended for the implementation of heavy computation processes and is poorly suited for DSP algorithms. These limitations have posed a challenge for developers seeking to deploy audio-focused applications on the web, since no other platform offers the combination of graphics, animation, user interface tools, and the rapid development environment of Flash.

To address the inherent challenges in the integration of interactive audio processing into Flash-based applications, we have developed the *DSP Audio Toolkit for Flash* (DATF). This toolkit makes use of Adobe's *Alchemy* framework for compiling C code for execution on the ActionScript Virtual Machine offering significantly improved performance over algorithms implemented in Java or ActionScript. By including DATF into an existing Flash project, a developer is afforded access to several common audio processing algorithms including spectral feature analysis, autoregressive modeling and acoustic

room imaging. In developing this toolkit, we hope to open up new possibilities for developers of Flash applications and games, enabling them to utilize real-time audio processing as a means to drive gameplay and improve the end user experience.

There are many potential applications for this toolkit. Music-based video games, such as *Guitar Hero* and *Rock Band*, have experienced a rise in popularity due in part to the integration of interactive audio processing in the gameplay experience. In these games, players respond directly to the music in order to achieve a high score, requiring the game to provide tight synchronization between user interaction and music, which has not been previously possible in Flash applications.

Development of DATF is the result of our efforts to develop collaborative, web-based games to collect perceptual data to help solve computationally difficult problems in music information retrieval, [1], [2]. While such collaborative games [1], [3], [4] consist of measuring the player's response to a stimulus (e.g. music or images), some of our games have an educational focus, requiring players to generate content that drives gameplay [1]. These activities, which focus on the creation of content to test audio perception, have an increased level of user interaction and necessitate significant signal processing computation for real-time audio feedback.

The remainder of the paper is structured as follows: Section II presents a brief background on audio application development using Flash and our prior efforts to develop an architecture to support the requirements of our collaborative data collection games. Section III explains the architecture of the DSP Audio Toolkit for Flash, using the Adobe Alchemy framework. In Section IV, we describe some of the audio DSP functionality implemented in DATF. Section V presents simulations and results demonstrating the computational advantages of DATF over other implementations, and in Section VI we demonstrate how DATF is used in applications we and our collaborators have developed. Section VII discusses our conclusions and future work.

II. BACKGROUND

Prior to version 10, Flash provided modest support for audio playback, but very limited support for sound manipulation and no functionality for dynamic, buffer-based audio control. The primary audio functionality contained in Flash to date

has focused on playback of short sound clips or entire music files. Flash 9 allowed external audio clips to be used within a project by embedding mp3 files into the project's SWF (binary) file (for sound effects) or loading them over a network connection (for music streaming applications). The built-in sound transformation objects made it possible to change the volume of a sound or adjust panning across stereo channels. While the `computeSpectrum` function is provided for the incorporation of a spectrum view during audio playback, no other signal processing functions are available from ActionScript. Furthermore, compatibility with audio files in Flash 9 is limited to the mp3 format and only supports sampling frequencies of 44100, 22050 and 11025Hz [5].

Although ActionScript is not well-suited for complex mathematical processing, `as3mathlib` [6] is a math library fully implemented in ActionScript 3 (for Flash 9). This library includes a wide array of functions, including the Fast Fourier Transform (FFT). It is based on an earlier implementation called AS2.0 Library [7], developed using ActionScript 2 for earlier versions of Flash.

A. Hybrid Architecture for Audio DSP in Web Games

The primary goal of this project is to develop a suite of web-based collaborative games to collect data on aspects of human auditory perception, which requires highly specific manipulation of speech and musical sounds [1]. This objective demands a rapid development platform capable of providing a graphically-rich user interface with real-time audio output along with support for signal processing tasks such as filtering and spectral analysis. Adobe Flash is a mature platform (on version 9 when we began our development and now on version 10) offering easily customizable interfaces with rich graphics and animations. Flash offers cross-platform support (Windows, Linux, and Mac OS X) and has been used for the development of similar collaborative games for data collection [3], [4]. Also, many of our games are designed with an educational component to be used in K-12 classrooms, and the ability to run Flash games within a standard browser makes for easier deployment in such settings where administrator access to install software on computers is severely limited.

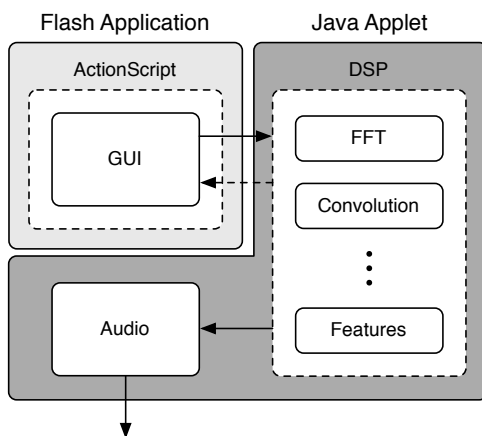


Fig. 1. Hybrid Flash and Java architecture for audio signal processing. In spite of its attractiveness for GUI design, the limitations

of Flash's native scripting language, ActionScript, make computationally intense DSP algorithms impractical to implement. As a cross-platform scripting language, it is not optimal for efficiently implementing computationally intensive DSP algorithms. Some optimized library functions are available for primitive signal processing (e.g., `computeSpectrum`), but these functions are not comprehensive, and they do not offer sufficient parametric control of the functions. This guided our initial approach towards a hybrid architecture consisting of a GUI implemented in Flash, which communicates with a Java helper applet to handle intensive audio processing. We initially chose Java for several reasons, including cross-platform compatibility, support for dynamic audio processing, and the availability of fast, efficient external libraries for DSP (e.g. Fast Fourier Transform (FFT)) [8]. In our hybrid architecture, audio processing and playback is initiated via a call in Flash that sends parameters to the Java applet via a JavaScript bridge. Although this architecture satisfies our audio processing requirements with reasonable performance, several inherent problems remain when using the JavaScript bridge, including:

- Error handling between Flash and Java function calls is limited
- String parameters sent between Flash and Java are limited in length
- Function calls to Java can not be precisely synchronized with interaction in the Flash GUI

These issues significantly limit the responsiveness of the application, reduce the quality of audio feedback, and generally lessen the user experience. In particular, the lack of error handling when calling Java from Flash leaves the user uninformed regarding common issues such as poor network connectivity (inability to contact the game server or loss of a network connection). This requires players to self-diagnose issues, usually leading to a refresh of the browser window, thus losing their current position in the game. The limitations on length of string parameters passed to Java increases the dependency on communication between Flash and Java, since audio files are loaded from the game server by Java. The lack of synchrony between Flash and Java makes this architecture incapable of real-time audio control and manipulation since Flash makes an external request to Java and blocks until the audio playback is complete, creating a "hanging" effect in the browser that halts all user interaction.

III. TOOLKIT ARCHITECTURE

Although we employed the hybrid architecture successfully in deploying and collecting data from two web-based games [1], the limitations led us to seek alternatives to improve the user experience by adding uninterrupted, interactive audio processing directly in the native game environment. The complexity of dealing with two platforms (Flash and Java) also hindered development time, and we desired a more streamlined process to facilitate rapid game development. The release of Adobe Flash version 10 in October, 2008 [9] and public

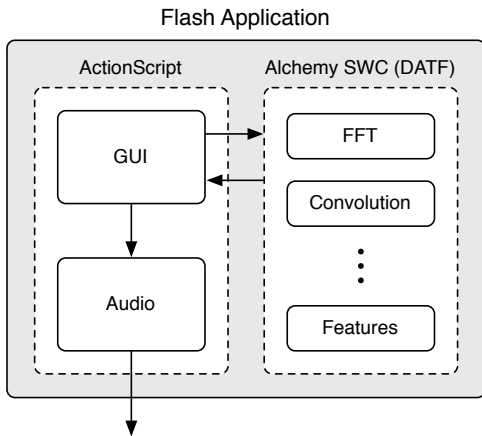


Fig. 2. Flash and Alchemy architecture incorporating DATF.

preview of the Adobe Alchemy project in December, 2008 [10], provided solutions to these limitations.

Unlike previous versions, Flash 10 makes it possible to dynamically generate and output audio within the Flash framework. This functionality is asynchronous, allowing sound to play without blocking the main application thread. The Adobe Alchemy project provides the ability for C/C++ code to be directly compiled for the ActionScript Virtual Machine (AVM2), greatly increasing performance for computationally intensive processes. Adobe claims that Alchemy compiled C/C++ code may be 2-10 times slower than natively compiled C/C++ code, but the performance will be considerably faster than a pure ActionScript implementation, providing an ideal solution for the DSP computation needs of our games. With these tools, it is now possible to develop Flash-based applications that incorporate dynamic audio generation and playback capabilities without the need for an external interface for computation-intensive signal processing applications.

The Alchemy framework enables a relatively straightforward implementation of standard C code into Flash projects, thus existing signal processing libraries written in C can be incorporated as well. C code is compiled by the Alchemy-supplied GNU Compiler Collection resulting in an SWC file, an archive containing a library of C functions, which is accessible in Flash via ActionScript function calls. An integrated application is created by simply including the SWC archive within the Flash project, producing a standard SWF (Flash executable) file when built. The DSP Audio Toolkit for Flash we developed consists of a library of C methods in a SWC file that developers can use to add signal processing functionality to their projects.

IV. AUDIO DSP FUNCTIONS

In DATF, we offer two general categories of audio processing functions for developers to incorporate into their Flash-based projects. The first category consists of spectral analysis functions for extracting and modeling the frequency-domain characteristics of signals. The second category consists of synthesis functions used for filtering and generating audio signals.

A. Audio Analysis Functions

1) *Fast Fourier Transform*: The Discrete Fourier Transform (DFT) is at the core of most signal processing applications and the Fast Fourier Transform (FFT) algorithm provides the standard implementation. We have included forward and inverse FFT methods for both real and complex data, which are based on a well-known C implementation [11]. In Section V, we compare the computational speed of the Alchemy-compiled FFT for real data to ActionScript and Java implementations.

2) *Linear Prediction*: In certain applications, it is useful to analyze the audio spectrum by approximating its spectral envelope. For this purpose, we have included an implementation of Linear Prediction (LP), an autoregressive technique that predicts a sample $x[n]$ as a linear combination of previous samples $x[n-p]$. The associated all-pole transfer function is shown below [12]:

$$H(z) = \frac{X(z)}{G(z)} = \frac{1}{1 - \sum_{p=1}^P \alpha_p z^{-p}}. \quad (1)$$

$$\hat{x}[n] = \sum_{p=1}^P \alpha_p x[n-p] \quad (2)$$

The solution is based on the Levinson recursion and the order, P , of the all-pole filter is a parameter to the function.

3) *Hanning Window*: For short-time audio analysis tasks, we have included the standard Hanning window to alleviate frame edge-effects and sidelobes in the frequency domain.

4) *Sinusoidal Analysis*: In the interest of modeling and manipulating music signals, the sinusoidal analysis function identifies harmonic components within audio signals. The algorithm performs short-time analysis on the audio spectrum by dividing the signal into Hanning windowed frames with 50% overlap. For each windowed frame, LP analysis is used to approximate the spectral contour and the FFT provides the overall frequency response. Slightly lowering the spectral contour provides a threshold function, which is used to isolate harmonics in the frequency spectrum.

In the field of music information retrieval (MIR), spectrum-derived features are often used for audio classification. We have included methods to extract several such features, which may be useful for audio-driven games.

5) *Intensity*: Spectral intensity is a measure of the total energy in the audio spectrum (and thus, the signal) and is measured by summing the magnitudes of each FFT bin, $X[k]$ over K total bins for a given frame n .

$$I_n = \sum_{k=0}^{K-1} X[k] \quad (3)$$

6) *Spectral Centroid*: The “brightness” of a sound is correlated with the spectral centroid. The centroid is obtained by summing the magnitudes of each FFT bin, $X[k]$, weighted by its frequency value, $F[k]$, and dividing by the overall energy

in the signal (intensity).

$$C_n = \frac{\sum_{k=0}^{K-1} F[k]X[k]}{\sum_{k=0}^{K-1} X[k]} \quad (4)$$

7) *Spectral Rolloff*: Spectral rolloff indicates the frequency, R below which 85% of the total spectral energy lies. The rolloff value can provide an indication of the types of sources contained within the sound.

$$R_n = \arg \sum_{k=0}^R |X_n[k]| = 0.85 \sum_{k=0}^{K-1} |X_n[k]| \quad (5)$$

8) *Spectral Flux*: When performing short-time analysis on an audio signal, the change in the spectrum over time can be measured by the spectral flux. The flux measures the Euclidean distance between consecutive spectral frames.

$$F_n = \left(\sum_{k=0}^{K-1} (X_n[k] - X_{n-1}[k])^2 \right)^{\frac{1}{2}} \quad (6)$$

9) *Spectral Bandwidth*: The bandwidth of an audio signal describes the range of the frequencies it contains. The bandwidth is calculated by summing the distance from the center frequency of each bin, $F[k]$, to the spectral centroid, C , weighted by the magnitude of the frequency bin, $X[k]$.

$$B_n = \frac{1}{K} \sum_{k=0}^{K-1} X[k] |F[k] - C_n| \quad (7)$$

B. Audio Synthesis Functions

To complement our audio analysis functions, we have included several FFT-based functions for operations related to audio synthesis and manipulation.

1) *Fast Frequency-Domain Convolution*: The fast convolution method enables a signal to be modified by a finite-length filter through an FFT-based implementation. In order to reduce computational costs, the convolution is performed by taking the FFT of the audio signal and finite impulse response (FIR) filter and performing a point-wise multiplication in the frequency domain. The inverse FFT is used to transform the result back to the time-domain.

2) *Overlap-Add*: The overlap-add method enables block convolution between a long audio signal and a FIR filter. The function divides the input signal into non-overlapping segments of manageable length. The algorithm then convolves the first segment with the filter using the fast convolution function. Since the resulting sequence is longer than the original block size due to the convolution operation, the overlapping segment is added to the next block and the process is repeated for all audio segments.

3) *Room Impulse Response*: For applications requiring realistic modeling of acoustic environments, we have included a room impulse response (RIR) generation function. The RIR is based on the well-known *image model* [13], which is capable of incorporating the dimensions of a room, the locations of the sound sources and listener, and the energy

absorption characteristics of the walls. The RIR function yields a FIR filter that can be used in conjunction with the fast convolution and overlap-add functions to simulate an acoustic room environment by convolving the audio source(s) with a RIR filter.

4) *Additive Sinusoid Synthesis*: To accompany the sinusoidal analysis function, DATF offers a method for generating complex sounds based on a sum of sinusoidal components and a time-varying amplitude distribution. The function requires the user to supply the desired length of the audio signal, the component sinusoid frequencies and time/amplitude points that define the amplitude envelope. This function has applications in generating music and/or sound effects for games.

V. SIMULATION RESULTS

In order to determine the performance benefits afforded by DATF, we measured the average computation times of the Fast Fourier Transform (FFT) algorithm implemented in Flash using ActionScript, Java, and Flash using Alchemy-compiled C code. All of the tested implementations were developed for real-valued input signals, appropriate for audio and for minimizing computation. The FFT was chosen for comparison because of its wide applicability in DSP operations, including several of the analysis and synthesis functions in DATF. The ActionScript FFT (from as3mathlib [6]) is written purely in ActionScript 3, while the Java FFT uses the JTransforms library, which claims to be the fastest Java FFT available [8]. The Alchemy-compiled version of the FFT for real data is based on a well-known implementation in C [11]. All simulations were performed on a 2.4 GHz Intel Core 2 Duo machine running Mac OS X.

The average required computation time for an FFT using each development platform was calculated for FFT lengths at each power of 2 from 256 to 16,384. Timings for each FFT size were based on the elapsed time for 10,000 iterations of the FFT using real-valued data divided by the number of iterations. The average computation time for each development language is shown in Table I.

From the performance results, it is clear that our DSP Toolkit for Flash FFT significantly outperforms the Java and ActionScript implementations of the of the FFT. In particular, our toolkit yields computation times that are approximately 30 times faster than ActionScript and 10-15 times faster than Java implementations. These results provide compelling evidence that our DATF compiled using Alchemy is well-suited for signal processing operations and can be used to support applications with real-time audio processing requirements.

VI. EXAMPLES OF DATF IN DEVELOPED GAMES

To demonstrate the potential for interactive audio-intensive applications, we provide several examples that make use of DATF. Two of these are educational games initially developed using the hybrid Flash-Java architecture [1], which have been rewritten to take advantage of the signal processing afforded by DATF and the dynamic audio capabilities of Flash 10. The third example is a game that incorporates analysis of songs from a user's music library to drive gameplay in real-time.

TABLE I
COMPARISON OF FFT COMPUTATION TIME FOR WEB-BASED PLATFORMS IN MILLISECONDS.

Target Platform	FFT Size						
	16384	8192	4096	2048	1024	512	256
ActionScript (as3mathlib)	96.377	45.157	20.818	9.276	4.460	2.041	0.925
Java (JTransforms)	29.517	20.703	9.393	4.345	1.956	0.901	0.385
Alchemy-C (DATF)	3.009	1.371	0.628	0.297	0.139	0.067	0.034

A. Tone Bender

Tone Bender was developed in order to explore perceptually salient factors in musical instrument identification [1]. The game consists of two interfaces which allow players to create and evaluate modified musical instrument sounds. In the creation interface, the player's objective is to modify the timbre, in terms of the distribution of an instrument's energy over time and frequency, as much as possible while still maintaining the identity of the instrument. A player can maximize their score by creating sounds near the boundaries of correct perception for that instrument, but are still correctly identified by other players. Their score is based on the signal-to-noise ratio (SNR) calculated by the deviation between the original and their modified instrument. In the listening component of the game, players are asked to listen to and identify the instruments produced by other players.

To implement the DSP required for timbre analysis and additive synthesis, Tone Bender makes use of several functions included in the DATF. In particular, the sinusoidal analysis function is needed to extract the harmonic components from the audio in order to generate a visual representation of timbre. Tone Bender uses the sinusoidal synthesis function to dynamically generate sounds for playback based on the parameters chosen by the modified instrument's creator. Since Flash Player 10 permits the playing of buffer-based audio, the player can manipulate parameters in real-time during playback, thus providing immediate feedback and enhancing interactivity.

B. Hide & Speak

Hide & Speak simulates an acoustic room environment to demonstrate the well-known "cocktail party" phenomenon while collecting evaluation data on the effects

that source/listener positions and room reverberation have on speaker identification and speech intelligibility. The cocktail party phenomenon is our ability to isolate a voice of interest from other sounds, essentially filtering out all other sounds in an audio mixture [14]. This game also consists of two components where the players have a creation and listening/identification objective. In the creation activity (Hide the Spy), the player starts with a target voice and is instructed to modify the mixture of voices until the target voice is barely intelligible within the mixture. The player can accomplish this task by adding more speakers to the room, increasing the reverberation, and changing the positions of the sources (including the listener position). As in Tone Bender, players are encouraged to maximize their score by designing difficult rooms, where score is assessed in terms of the signal-to-interferers plus noise ratio (SINR) of the room. The listening component, Find the Spy, requires a player to determine whether an individual is present within a simulated room from a given audio sample drawn from configurations submitted from the creation component.

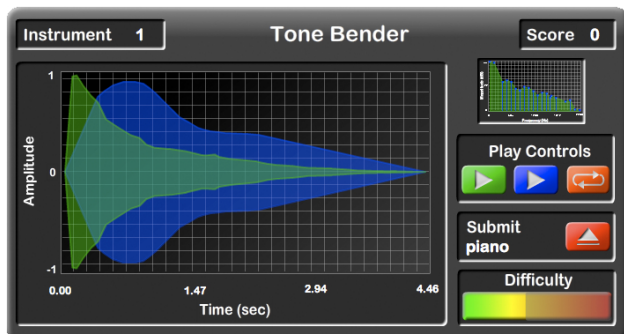


Fig. 3. The creation interface of Tone Bender.

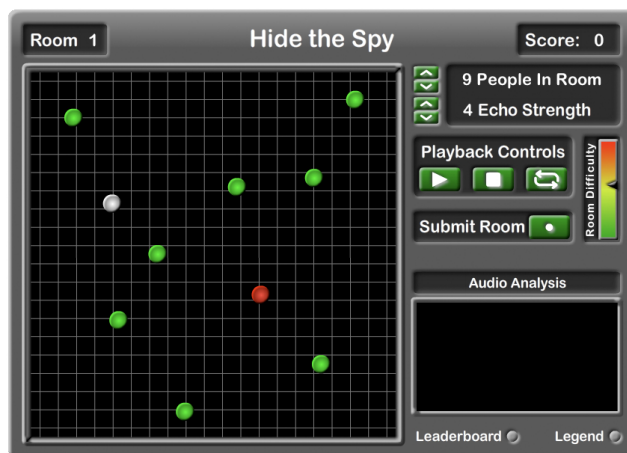


Fig. 4. Hide the Spy interface of Hide & Speak.

Hide & Speak utilizes multiple DATF functions to generate the audio for the simulated acoustic room environment in Hide the Spy, namely room impulse response (RIR) generation, FFT and fast convolution via the frequency domain. The audio for the listener is generated by calculating the RIR for each source based on source and listener positions and subsequently convolving each RIR with the respective source audio using the FFT and fast convolution. This computation is implemented on a per-frame (4096 samples) basis using the overlap-add method. The process is implemented separately

for each ear, taking into account the small difference in position, resulting in stereo audio in order to simulate a realistic acoustic environment.

C. Pulse

Pulse is a Flash-based, side scrolling platform game developed as a collaborative project between the Music and Entertainment Technology Lab and the RePlay Lab at Drexel University. Unlike many music-based games, which are primarily based on rhythm, Pulse generates a dynamic game environment based on the audio features of songs supplied by the user, which are used as the game's sound track. To illustrate the dynamic visual effects in Pulse, Figure 5 depicts how the game environment differs with and without audio.

The player's objective in Pulse is to advance through a level as far as possible before the conclusion of the song, while collecting objects, navigating through obstacles and defeating enemies. The terrain, obstacles, enemies, and background graphics are explicitly determined by the features extracted from the current moment in the song. By integrating the dynamics of the user's music with the visual aspects of the gameplay, Pulse provides a kinetic and responsive gaming environment unseen in other music-based games.

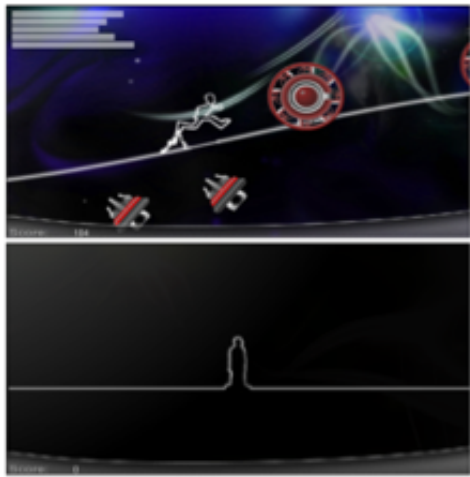


Fig. 5. Top: Depicts the gaming environment when music is present. Bottom: Depicts the gaming environment with a lack of music.

In order to generate the dynamic gaming environment, Pulse makes heavy use of the functions available in DATF, which are needed to map spectral features from the audio to the game's graphical output. In particular, Pulse uses the energy of the audio to determine the slope of the surface that the character traverses, as well as the visual intensity of the background graphics. Since spectral centroid is correlated with the brightness of the sound, Pulse uses centroid values to determine the hue of the background colors, such that higher centroid frequencies correspond to lighter hues. The spectral flux, or spectral deviation over time, is used to determine the vertical position, size, and point value of the player's obstacles. The incorporation of spectral features into the game creates a unique synchrony between the visuals, music, and

gameplay. Furthermore, the player's ability to upload a song from their personal collection as well as switch songs during gameplay while maintaining synchronicity between the audio and graphics potentially represents a new gaming and game development paradigm.

VII. CONCLUSION

We have presented a DSP Audio Toolkit for Flash that facilitates the rapid development of applications requiring signal processing computation. On the Adobe Flash platform, DATF demonstrates significant performance gains over alternatives for implementing signal processing algorithms. We have provided several examples of games that incorporate DATF to enable real-time audio processing and enhance interaction. It is our hope that the toolkit presents new possibilities for Flash developers to utilize real-time signal processing in applications and improve the rich media experience for end users.

We plan to release DATF as an open-source research project, which may be freely used by game developers and the research community. The current status of the project, including relevant documentation and source code, may be found at <http://schubert.ece.drexel.edu/research/DATF>.

ACKNOWLEDGMENT

This work is supported by NSF grants IIS-0644151, DRL-0733284, and DGE-0538476. The authors also thank the Drexel RePlay Lab and students in the Game Design Studio for incorporating the DSP Audio Toolkit for Flash in the development of Pulse.

REFERENCES

- [1] Y. E. Kim, T. M. Doll, and R. V. Migneco, "Collaborative online activities for acoustics education and psychoacoustic data collection," *IEEE transactions on learning technologies*, 2009, preprint.
- [2] Y. E. Kim, E. Schmidt, and L. Emelle, "Moodswings: A collaborative game for music mood label collection," in *ISMIR*, 2008.
- [3] L. von Ahn, "Games with a purpose," *Computer*, vol. 39, no. 6, pp. 92–94, 2006.
- [4] E. L. M. Law, L. von Ahn, R. B. Dannenberg, and M. Crawford, "TagATune: a game for music and sound annotation," in *Proceedings of the 8th International Conference on Music Information Retrieval*, Vienna, Austria, 2007.
- [5] Adobe, "Flash player 9." [Online]. Available: <http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/>
- [6] "as3mathlib." [Online]. Available: <http://code.google.com/p/as3mathlib/>
- [7] R. Wright, "As2 library." [Online]. Available: <http://members.shaw.ca/flashprogramming/wisASLibrary/wis/index.html>
- [8] P. Wendykier, "Jtransforms." [Online]. Available: <http://piotr.wendykier.googlepages.com/jtransforms>
- [9] Adobe, "Flash player 10." [Online]. Available: <http://labs.adobe.com/technologies/flashplayer10/>
- [10] Adobe Labs, "Alchemy." [Online]. Available: <http://labs.adobe.com/technologies/alchemy/>
- [11] H. P. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.
- [12] J. Makhoul, "Linear prediction: A tutorial review," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 561–580, 1975.
- [13] J. Allen and D. Berkley, "Image method for efficiently simulating small room acoustics," in *Journal of Acoustic Society of America*, April 1979, pp. 912–915.
- [14] S. Haykin and Z. Chen, "The Cocktail Party Problem," *Neural Computation*, vol. 17, no. 9, pp. 1875–1902, Sep. 2005.