

Pulse 2

An Experiment in Music-Reactive Gaming

Introduction:

Today's popular music-based games, with titles such as *Guitar Hero* and *Rock Band*, are very rhythmically oriented. The games depart from typical character-based adventures, and the player often pretends they are simply playing an instrument rather than exploring some vast fictional world. Game-play depends heavily on the player to react to the music in a strict and pre-determined fashion. Furthermore, the music in these games is pre-processed, and the stages or levels, for each song, are hard-coded beforehand to provide a strong connection to the music. This also means that whatever the game comes with is the final set-list of music, unless further downloads become available after the release.

The original *Project Pulse*, and subsequently *Pulse 2*, have explored breaking many of these boundaries by integrating real-time music analysis software into different settings than your typical rhythm-game, such as a platformer or side-scrolling action game. What this means for the game is that it essentially becomes an extension of the analysis software, reacting and adapting to the music, and allowing the player to insert any track from their own music library: an open-ended experience. It allows the player the choice to play with music they already have a connection with, while still being able to sample new music that might work well for the game. On the flip-side, this also means that a lot of planning has to go into a game that will support vastly different styles and qualities of music once it is programmed and released. Creating a completely dynamic system of game-world generation will always lose to manual human pre-coding, but as analysis techniques get stronger, we hope this gap will be bridged. This series seeks to provide users with a fresh way to experience their music, as we now look ahead and begin further work on new areas of application for this method of game-design.

Technical Breakdown:

Development Platform

Pulse 2 was developed using Adobe Flash CS4, and deployed using the Adobe Air1.5 framework. The Air run-time environment allows the user to download the game and play it as a desktop application. This allows the user free and easy access to their music library, where a website based application would require long upload times and obstructions from software security issues. The Air framework also integrates the latest additions to Actionscript3, providing functionality for dynamic music applications, as well as many system based tasks, such as automatic-updating and file-browsing methods.

Real-Time Analysis

The music analysis software, developed by the Drexel University MetLab, was written and compiled as an SWC library which can be imported into flash, and called to run in the background while the player runs the game. It utilizes the Actionscript "Sample Data Event" to continually scan and playback sections of bytes from a streaming audio file. It first establishes a play-back offset, so that a window of data can be collected from at least a couple seconds worth of the music before it starts to play. From there, the music maintains a perpetual offset, so the flash application will always have a window of data from the upcoming audio. The software is sent bytes from the song file at a steady frame-rate, and performs spectral audio analysis, returning values for various features. The features of main interest in this application have been: the intensity of the audio, the bandwidth of the signal, and the centroid value of the spectrum, most closely related to tone brightness.

Use of the Feature Data

Pulse 2 utilizes the data returned by this analysis to perform three main functions: draw a continually unfolding graph of audio visualization for the game world, monitor thresholds and detect peaks for the spawning of various obstacles, and finally detect timeslots in which the player can perform moves 'on the beat' to earn points.

The creation of the playing environment takes in values from all three of the spectral features used in the game. First, a 'velocity' of the rise and fall of the solid white line, which makes up the ground plane, is computed by continually adding the difference of subsequent intensity values. This results in pattern of what can be called different 'elevations levels' of the line as the music changes in intensity. A soft or silent part section of music will produce a line that stays very low in vertical coordinates. Once the song starts to pick up, the player will be presented with a rising mountain, which will level off once the music settles into the high intensity section. This elevated playing field will be populated by different forms of obstacles than the lower fields. The playing line itself is derived from an averaging of several of these intensity values, while a second background line is also generated which correlates more directly with straight intensity values from the music as they are played back.

In addition to these lines, the playing field is also comprised of a cloud of color, which fluctuates in size with the change in bandwidth of the audio signal. The color of this cloud is additionally adjusted by the centroid values. Each song is given a base color, determined by genre, and then the fluctuation of the centroid value above or below mid-level results in either a red-shift or blue-shift of the color value, for low and high centroid values respectively.

The various enemies and obstacles in the game are divided up into different 'casts' which appear when the music occupies different ranges in spectral intensity. For example, the 'Ripple-snake' obstacle which hits the player into the air is only generated when the musical intensity is very low, thus creating somewhat of an obstacle usually at the beginning of a song until it picks up. If they hang over into a high intensity section, the player has the chance to use them to the advantage of getting shot very high into the air.

As the music picks up, various other obstacles start to appear such as air-blasts, and flying hexagons which hit or explode on noticeable peaks in the music. This is accomplished by analyzing the music data ahead for areas where one intensity value is larger by a certain threshold than the two adjacent values to it. Thus, if a string of 3 intensity samples reads: '100,1000,120' we can be pretty sure something significant happened on the 1000 mark to effect such a fluctuation in intensity. Many of the enemies will be called at the moment the prediction is made, giving them time to play out for a second or two before they pop or hit the playing field at the predicted peak in the music. The high-intensity triangle enemies on the other hand, are generated when peaks in the music actually occur, creating a different kind of obstacle, rather than needing to be avoided on the beat.

The scoring system in the game is based on the challenge of performing certain moves 'on the beat'. These windows of opportunity are determined in a similar fashion as determining peaks for spawning enemies, only we give the player extra padding to make it a little easier: instead of allowing them only a window of about $1/30^{\text{th}}$ of a second, it gives them about $1/10^{\text{th}}$. The closer the player is to hitting exactly on the detected peak, the more points they will receive for that move. Some attacks available in the game will additionally be stronger if used on high intensity moments of the music, further challenging the player to really listen for good areas in the music to spring.