Adapting Personal Music for Synesthetic Game Play

Sam Rossoff
Department of Computer Science
University of Victoria
Canada

George Tzanetakis
Department of Computer Science
University of Victoria
Canada

Bruce Gooch
Department of Computer Science
University of Victoria
Canada

Abstract

Music can significantly effect game play and help players understand underlying patterns in the game, or the effects of their actions on the characters. Conversely, inappropriate music can have a negative effect on players by creating additional difficulties. While game makers recognize the effects of music on game play, solutions that provide users with a choice in personal music are not forthcoming. We design, implement and evaluate an algorithm for automatically adapting an arbitrary music track from a personal library and synchronizing play back to the user, without requiring any access to the video game source code.

1 Introduction

Much of what we expect from video games is a function of our experience with movies and other visual media. When we talk about music in movies, we talk about it as a contributing factor to the artistic medium. As a result, music in video games, much like movies, is left to the discretion of artistic narrators, who choose tracks to advance the story line of a game or to contribute emotional depth. Unlike movies, which are a passive medium; in games the player takes a more active role. Often the musical score is added without examining the effect it has on this more active role that the player is having. Most games have sound effects associated with them that do influence player action or communicate information back to the player. Game makers are often overlooking an opportunity to help players both enjoy and understand their games. Tuning the parameters of the musical score of a game to influence player actions provides an exciting possibility for game creation.

A popular approach to this issue is to allow players to choose the music they want to hear during the course of a game from a personal music collection. This allows for a degree of customization to help players better immerse themselves in the game. However, in terms of game play, this results in a "chicken and egg" problem; players cannot appropriately select music without having already experienced the game. Players tend not to be experienced composers and often lack a conceptual understanding of what it is they are trying to achieve. Additionally, nor does their music library have the flexibility of adapting to the game dynamics.

This paper reports on an algorithm to automatically adapt soundtracks from personal music specified by the user. By examining user input during game play we can compute an optimal rhythm for a level of a platform game. We determine the underlying structure of a sound track by automatically estimating the tempo and dominant periodicities of the music. Finally, we adjust the music to the rhythm of the game level to achieve synesthetic game play. Our method is evaluated in three ways: user input is shown to be predictable; user input is identified as an aproximation of the rate of gameplay; and input types between different genres of video games can be distinguished using a naive bayes classifier.

FDG 2010, June 19-21, Monterey, CA, USA

Smith et al.[2008] posit that two dimensional platform games can be broken down into subcomponents or "rhythm groups". While this solution could provide for extremely accurate assessment of the underlying "rhythm" of a level, currently no implementations exist. Additionally, we have to consider the possiblity of proprietary code being used for the game in question. Because we cannot necessarily know the underlying structure of the game, we are forced instead to look at what data we do have available: user input. If we assume user input is a close approximation of the underlying structure of the level, then an understanding of the user input is sufficient. The accuracy of such an approach is evaluated in section 3.

Our algorithm analyzes the player's actions as input and translates it into a meaningful beat sequence which can be synchronized with a predetermined music track from a personal library with an automatically annotated beat structure. Because the beat sequence of the music is deterministic, it does not need be calculated at run time. The resulting implementation of the system, can thus be broken into three major parts:

I Extracting meaningful user input

II Identifying the tempo of the music

III Syncronizing the user input and music

2 Background and Previous Work

Music has been shown to affect mood and deal with feelings of monotony and boredom [Karageorghis and Terry 1997]. People were able to engage more thoroughly in a dull task if they had music to listen to as well. Matesic and Cromartie [2002] were able to show that listening to music decreases lap pace and increases overall performance of untrained runners. While music had an influence on the performance of both trained and untrained athletes, the effect was much larger on the untrained group. Matestic posits that the music may provide a pacing advantage, as the cause for this effect. Cram and Duser [2000], were able to show that music not only have a marked affect on performance but also affects heart rate. In their study, users who exercised to music with faster tempo, showed higher heart rates on average.

Thayer [Minami et al. 1998] demonstrated that modifying the musical scores for the same visual response can heavily affect the electrodermal response in subjects. Participants were shown a safety film, a known stressor, with either a documentary score, a horror film score, or no music. Not only did participants show significantly more electrodermal response to the horror score, compared to the control, but less reaction when shown the documentary control. The implication of this study is that music can directly affect player's "mood" in a measurable fashion.

Konecni [1982] argued that because music requires cognitive processing, listening to music may impair performance

as the additional cognitive load acts as a distractor. McKelvie and Low [2002] demonstrate that in the presence of music there is little or no effect on spatial IQ scores and reading comprehension. By comparison Rauscher et al. [1993] show significant improvement from exposure to Mozart. Rauscher argues that music contributes as a neuropsychological primer for the children, and that playing the music before testing increases their performance. Cassidy and MacDonald [2007], by contrast show that high arousal music has a demonstrably negative effect, while low arousal music has a smaller, but still negitive, effect. While overall music does tend to inhibit performance, the music in these studies is unrelated to the task at hand. The study does not compare the effect of music that has been chosen to augment preformance in the task. Additionally, on certain cognitive tasks (eg. Stroop) the music has a positive effect on performance, suggesting that the right music can be beneficial.

The idea of using audio sounds to convey additional information in Computer Science has been around for some time. Early work attempted to simulate audio icons [Blattner et al. 1989]. Synthetic sounds that are often created to express an underlying quality, as in the previous example, are known as Sonification. A good example of this is Chafe and Leistikow [2001] who created a sonification technique for understanding internet latency. However, instead of using time repetition to display data, a direct mapping between tones is established. Kilander and L'onnqvist [2002] attempted to extend peripheral awareness of users through audio-based techniques in a similar manner to our work. Much of their implementation is concerned with the human interactive component; they made important attempts to streamline incoming sounds through traffic shaping. While traffic shaping allowed for the separation of sounds (unlike previous work), it did not do so in an intelligent manner. The sounds became spaced, but spacing was unrelated to user interaction.

Alternatively, there have been a number of attempts to modify game play to musical scores. Some of the more famous techniques are the *RockBand* and *GuitarHero* franchises. Each "level" in a game like this is generated from an original sound track. This design of game has become very popular recently, however, it is limited by the media which it will accept. Instead of being able to supply your own sound tracks, tracks must be created and then distributed to users by the official licenser. To combat this, a number of grass roots projects such as Dancing Monkeys have sprung up [O'Keefe and Haines 2009]. Holm et al. [2005] provide a very good survey of tools of this nature. In his work he explores the usage of not only Audio but Visual images as well. Finally, he implements such an approach in a mobile phone context [Holm et al. 2006]. While this work is tangentially related, it is mainly concerned with adapting games to music, instead of music to games which is our goal.

One of the major problems with adapting music to an interactive setting is that a lot of music, especially western, is linear in nature. Griffin [1998] implies that traditional solutions to this problem tend to resemble using segments of music to represent specific events and handling interaction between them. This is highly flawed, as handling the interaction often times ruins the intent. For example, when cutting off a segment that is too long, the result is "un-musical." Likewise, reducing the length of segments (to prevent interaction), has less musical value than truncated longer segments. Griffin suggests a solution to this problem by treating the music as a number of layers, some of which are constant, while others fade in and out based on specific events.

Figure 1: System Diagram

While Griffin's solution does produce fairly good music, it still makes many poor choices based on the fact that individual MIDI files have to be both simple and universal in nature to compete with the underlying music. Additionally, it can only run on scenarios where a musical score is composed of multiple separate tracks with a predefined underlying layer. It does not, therefor, handle audio files such as the ones in personal audio music collections. A more advanced implimentation of this algorithm can be found in Left 4 Dead's director [Larkin 2008].

3 User Event Extraction

Ideally, an implementation would would take place in game source code where interaction can be observed directly. Source code for most games is generally unavailable; as such, the goal for our implementation is to interface with an existing game without access to the source code. There are a number of possible alternatives to direct access, and they can be observed by looking at the input structure of an operating system. While an ideal implementation would be cross platform, the input structure often depends on the design of the system in question. Given the target of adapting sound to video games, it would make logical sense to target a platform with high proliferation of said media. Although we could consider modern seventh generation consoles as a viable platform, support for running code in parallel is often non-existent. The resulting choice is the Windows platform, specifically Vista for this implementation.

3.1 Intercepting Events

The lack of source code for the target application will mean that we will have to access the DirectInput to capture user events. Fortunately, Windows, as of 3.2, implements hooks for identifying keyboard and mouse input [Marsh 1993]. A hook is a mechanism for intercepting events and passing them to a filter function which can be designed by the user. This filter function can then analyze the data before sending it on. In our implementation, this analysis will consist of time stamping and pushing the data to a seperate application via a pipe.

Because of the nature of hooks, these filter functions must exist in a seperate dynamically linked library (DLL) which is then accessed by the client application. The net effect is that the filter function can be called from the DLL for every user event without tedious paging of memory. It is important to note that on a system with multiple cores, all hooks will be called in order for a single message, but multiple messages can have their hooks processed asynchronously. To handle this, events need to keep track of their timestamps and not their interarrival time. Interarrival time needs to be processed elsewhere, outside of the hook. Finally, to guarantee the accuracy of captured keyboard events passing through Direct Input we must use the low level version of keyboard and mouse hooks (WH_KEYBOARD_LL and not WH_KEYBOARD). Our hooks have a sampling frequency of 5.0ms, which is an order of magnitude faster than the standard $60th$ of a second often used in video games for frame updates.

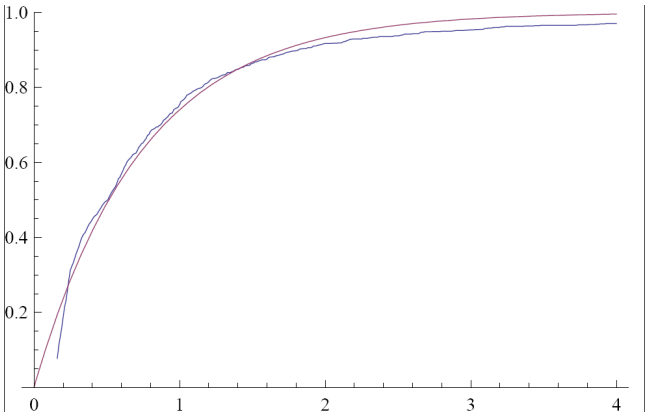

Figure 2: Plot of Estimated Expontential Distribution against Real Data

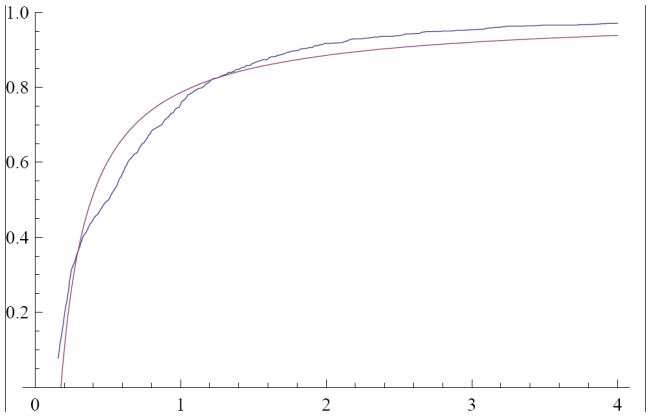

Figure 3: Plot of Estimated Power-law Distribution against Real Data

3.2 Establishing a Model

Once user data has been properly identified, statistical information is gathered about the periodicity of user actions. The rate of incoming input is characterized as a distribution with a mean and standard deviation. This is done for each type of input. As a result, new interarrival times can be identified as being part of the current model, or unlikely. If a number of incoming interarrival times are classified as unlikely ($P < .05$), the model is considered "broken," as the player has established a new beat rate. The median beat rate of this model is then compared to a precomputed beat rate.

3.3 Experimental Model

There are two candidate distributions for examination: Power Law and Exponential. In order to determine which of these curves is correct we need to compare the best candidates of both possible models. We will use Maximum Likelihood to estimate the parameters of both candidates from our observed data. The basis for a maximum likelihood estimation is the probability of a given observed value for a parameter space. We can perform this estimate for all data points, thus producing a selection that is most likely given our observed data. We then can use a Least Squares method to fit our estimate to the data.

Neither equation provides a very good fit to the data. Much of the instability is an attempt to match the shorter duration interarrivals. If, instead, we look at only events greater than .5 seconds the Power Law Distribution provides an excellent fit to the data, while the Exponential Distribution continues to provide a less than ideal fit. It is important to note that events less than .5 seconds constitute a significant portion of all events (45%). The most likely cause of this instability is a similar source as that of shorter events. While events below $\frac{1}{6}$th of a second are completely uniform, those from $\frac{1}{6}$th to .5 of a second are a combination of uniform and the later model (figures 2 and 3). As the later model provides a high degree of accuracy past this point, we are able to determine that it does represent, atleast partially, the underlying structure.

The mean of the estimated distribution for the period that the model is valid is used as the estimate periodicity.

4 Beat Extraction

The identification of tempo and beat structure is a well defined problem in the field of music information retrieval. Earliest studies involved beat extraction by using subjects tapping or clapping in time [Drake et al. 2000]. Tzanetakis et al. [2002], compare two methods of beat extraction based on beat histograms. These histograms are assembled by identifying the amplitude envelope periodicities of multiple frequency bands. This is accomplished with a standard Discrete Wavelet Transform filter bank and a multiple channel envelope extraction. While this technique does give an accurate representation, some level of information is lost due to imprecision on the part of the performer. To expand on this technique, our algorithm utilizes graduated non-convexity to smooth the extracted histograms while retaining their peak structure.

4.1 Multiresolution Analysis

To preform beat extraction we are going to need to understand which sounds occur at which times. Because sounds are characterized by their frequency, it becomes necessary to use frequency analysis to identify the sounds that are contained within the signal. We can consider a signal as existing in the **Time-Domain** where the independent variable is time and the dependent variable is the amplitude at that time. To obtain which sounds are occuring, we must necessarily use a frequency transform to preform analysis in frequency. This frequency transform will move us from the **Time-Domain** to the **Frequency-Domain**. Additionally, to identify the periodicity or beats of a given sound, we need to know not only which sounds are occuring, but when they are occuring temporally. As a result, our frequency analysis will have to occur at multiple resolutions. This multiresolution analysis will allow us to identify when frequencies occur at different times. Envelope extraction is used to obtain the periodicity.

The human auditory system does not have perfect resolution for both frequency and time either. Instead, it has good time resolution for high frequencies, and good frequency resolution for low frequencies. This implied tradeoff would require multiple windows in a Short Time Fourier Transform application. Instead, our application utilizes a Wavelet Transform which encompases these characteristics. A wavelet can be viewed in terms of a scaling function ϕ and a mother wavelet ψ, where the resolution is obtained by updating the scaling function and then scaling the wavelet

by the scaling function.

As our application occurs on discrete data to which we wish to apply a continuous function, it is necessary that we apply concrete mathematics [Graham et al. 1989] to rectify this situation. Instead of applying a continous function over our signal, we will instead define a set of matrices which perform the wavelet transform in a discrete context. We can achieve this by utilizing the matrices which allow us to go from the father to son scaling functions and mother to daughter wavelets.

For a given scaling function ϕ(x) there must exist some matrix P^j such that we can produce the scaling function at the next level of resolution $j-1$ by taking the product, or:

$$\phi^{j-1}(x) = \phi^j(x)P^j$$

Likewise there must be some matrix Q^j such that, given a scaling function $\phi(x)$, the product produces the wavelet function $\psi(x)$ at the next level of resolution $j-1$, or:

$$\psi^{j-1}(x) = \phi^j(x)Q^j$$

These matricies P^j and Q^j have transforms h^j and g^j expressed formally:

$$A^j = (P^j)^T$$

$$B^j = (Q^j)^T$$

We can use the rows of these matricies A^j and B^j as our filters $\mathbf{g}^j$ and $\mathbf{h}^j$ to analyze the signal data and produce the coeffecients $\mathbf{c}^j$ and $\mathbf{d}^j$ for the wavelet transform through a processes called subband coding. These coeffecients will give us translation and scaling information about our original signal. In the case of sound, this will correspond to location in time and period of a given set of sounds

Daubechies' wavelets have the property of being orthonormal and compact for the infinite real line, which satisfies our requirements for our filterbank. The coeffecients of the DAUB4 wavelet family are expressed as follows:

$$p = a = \frac{1}{4\sqrt{2}}(1+\sqrt{3}, 3+\sqrt{3}, 3-\sqrt{3}, 1-\sqrt{3})$$

$$q = b = \frac{1}{4\sqrt{2}}(1-\sqrt{3}, -3+\sqrt{3}, 3+\sqrt{3}, -1-\sqrt{3})$$

Where the p sequence represents the nonzero entries of the columns in our vector $\mathbf{P}$ and q represents the same entries of the columns in the vector $\mathbf{Q}$. Similarly, a and b provide the same service for vectors $\mathbf{h}$ and $\mathbf{g}$ as rows to maintain the relationship between $\mathbf{P}$ and $\mathbf{h}$. Furthermore, these sequences have the quadrature mirror filter property, which allows us to create the wavelet sequence from the scaling function sequence by reversing the order of the entries and alternating their signs [Stollnitz et al. 1995].

4.2 Envelope Extraction

Once the data has been seperated, the relevant time domain amplitude envelope can be extracted for each band. Initially, the bands are run through a low pass filter to identify dominant frequencies. Full wave rectification occurs to move the data into the positive domain. Next the data is down sampled to a common range and each band is normalized via mean removal. Finally, the data is run through an autocorrelation function

$$y(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)x(n+k)$$

and the top five periodicities are added to the histogram The music analysis system Marsyas [http://marsyas.sness.net/] was used to implement the algorithms because the framework is naturally designed for synchronous signal processing.

While this beat histogram implementation does exhibit peaks for beats with greater strength, it does not provide more in-depth insight into ranges of beats. If the underlying music has a constant tempo, then the corresponding tempos and dominant periodicities (beats) would show up as impulses spanning single bins of the histogram. Frequently, music contains expressive changes in rhythm; therefore, several neighboring histogram bins are affected reducing the performance of the original algorithm.

4.3 Graduated Non-Convexity

To distill this range of periodicities into a single value while preserving their strength, we expand on the original code by applying a technique known as Graduated Non-Convexity. Analyticaly, we can express this as an energy minimization function of the form:

$$\sum_{i=1}^{m} \chi_i (f_i - d_i)^2 + \lambda \sum_{i=1}^{m} \sum_{i' \in N} g_\gamma (f_i - f_{i'})$$

Where d_i is the smoothing term and g_γ here represents our blur level going from $g_\gamma^{(n)}$ to $g_\gamma^{(0)}$ as our target, and $g_\gamma^{(n)}$ is sufficiently large to be strictly convex[Blake and Zisserman 1987] .

The beat histogram is first blurred by a 1 dimensional Gaussian kernel to establish a pyramid. For the purposes of this implementation, kernels of size 3 were applied starting at a $\gamma^{(4)}$. Major peaks were then identified on the blurred images. The peaks were then related to other peaks on histograms of higher level granularity until the bottom of the pyramid was reached. After being identified, the areas around an identified beat rate were then modified with a Haar transform to simulate lateral inhibition. Lateral inhibition is necessary for this algorithm to give each group of peaks a deterministic result, and to prevent identified peaks from providing secondary influence.

This algorithm was applied on a series of music with different beat rates and tempos. Both standard and syncopated beats were successfully identified in all cases. A sampling of the results can be seen in Figure 4. Figure 5 shows a blur factor of $\gamma^{(4)}$. These graphs show that while some beats appear very strong, once we factor in neighbors they do not retain their absolute strength. It should be noted that this algorithm does not run in real time, and is necessarily precomputed for all tracks in a personal music collection.

5 Syncronization

Given the user input beat rate, the nearest beat rate with the highest strength, in the music, can be identified by our model. In addition we can give greater importance to different kinds of input at this stage (stronger beats to keyboard or mouse as necessary). A ratio is then constructed between the target beat and the user rate. This ratio is used to modify the rate at which the music is fed to the audio driver, and eventually output. The method for time stretching and shrinking the music is a phasevocoder which allows for changes in the time domain while preserving frequencies.

5.1 Phasevocoder

A vocoder is a time-scaling algorithm which stretches audio samples over a larger or smaller window without causing a shift in the frequencies. The most popular vocoders

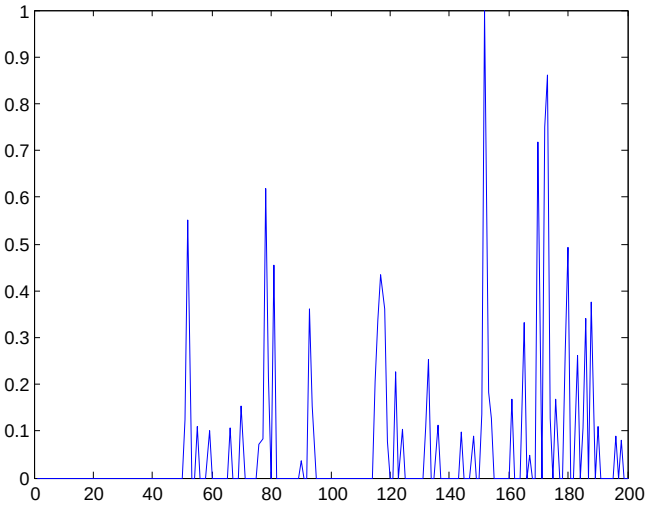

Figure 4: Plot of Beat Histogram

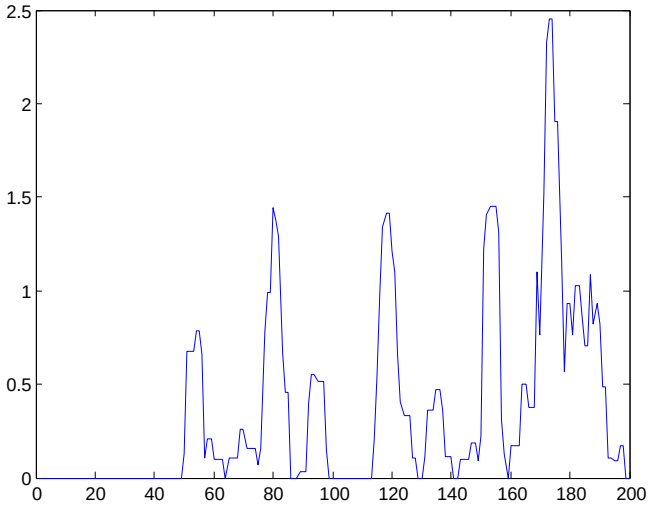

Figure 5: Plot of Beat Histogram with a blurring factor of 4

achieve this dilation by considering overlapping time windows and aligning the "phases" between them. The result is that phase consistency within a given frequency channel is consistent over time, but that phase consistency is not maintained across all the channels in a given time slice. The phasevocoder in question is based on the work of Jean Laroche and Mark Dolson's [1999].

During the analysis of the signal using a STFT we define an analysis hop factor R^a such that given a time-instant t_u^a for sucessive integer values u, $t_u^a = R^a u$. We can then calculate the Fourier transform of this instant t_u^a by taking a windowed portion of the original signal centered about t_u^a. This STFT, denoted by $X_{t_u^a,\Omega_k}$ can be expressed:

$$X_{t_u^a,\Omega_k} = \sum_{n=-\infty}^{\infty} h(n) x_{t_u^a+n} e^{-j\Omega_k n}$$

with respect to the original signal x. In this case $h(n)$ is our windowing function which is necessarily compact. Here $\Omega_k = \frac{2\pi k}{N}$ is the center of the frequency channel k where N is the size of the discrete Fourier transform.

As time dilation is the desired result of this process we will use a different synthesis hop factor R^s, giving us different time instants $t_u^s = R^s u$. We can thus obtain a short-time signal $y_u(n)$ for each of these time-instance via the inverse Fourier transform, denoted $Y_{t_u^s,\Omega_k}$. We can then multiply each short-time signal with a synthesis window $w(n)$ and sum to produce the output signal y.

$$y_n = \sum_{u=-\infty}^{\infty} w(n - t_u^s) y_{n-t_u^s}^u$$

where:

$$y_n^u = \frac{1}{N} \sum_{k=0}^{N-1} Y_{t_u^s,\Omega_k} e^{j\Omega_k n}$$

5.2 Phase Alignment

While the STFT will produce values which dictate the frequency at a given timeslice, these frequencies do not necessarily have the correct phase. Due to the influence different waves have on each other if they are "in" or "out" of phase, a complete algorithm must identify and correct phases between time slices and frequency channels.

To identify the phase of the STFT $Y_{t_u^a,\omega_k}$ we must first perform phase unwrapping. Phase unwrapping is a technique that gives us the phase increment between consecutive frames. This is important because the phase between one time slice and the next needs to be consistent. The phase increment can then be used to estimate the instantaneous frequency of sinusoids in a given channel. The instantaneous frequency $\hat{w_k}(t_u^a)$ of the closest sinusoid is determined by the phase increment $\delta\phi_u^k$ and the frequency channel given by Ω_k:

$$\hat{w}_k(t_u^a) = \Omega_k + \frac{1}{R^a}\delta\phi_u^k$$

where the principal determination of the increment is taken between $\pm\pi$ because of the cyclical nature of sinusoids. We can think of this as the increment $\delta\phi_u^k$ being the phase shift between the instantaneous frequency $w^k(t_u^a)$ and the frequency channel Ω_k. As phases need to be aligned between time slices, the value $\hat{w}_k(t_u^a)$ can be used to propagate the correct phase alignment between time slices.

$$\angle Y(t_u^s, \Omega_k) = \angle Y(t_{u-1}^s, \Omega_k) + R^s \hat{w}_k(t_u^a)$$

5.3 Implementation of Time-Stretching

The actual implementation of the phase vocoder occurs once again in the Marsyas [http://marsyas.sness.net/] framework. Information about the user interaction rate comes in through a pipe and is modeled in the manner described in section 3 . Information received from the user model is a ratio representing R^s/R^a.

As beat detection will, on occasion, determine the harmonic of the underlying beat structure, ratios greater than 2 or less than $\frac{1}{2}$ will be scaled by the inverse amount. We can then take this ratio and multiply it with the sampling rate to achieve the correct output rate. This is then used as input into our phasevocoder.

While the phase vocoder will run in real time, because the spacing between the currently playing time slice and the next slice being processed is not taken into account there is a potential loss of time. This actual change in the output is perceptible if it occurs too often, regardless of phase locking. To compensate for this the output is modified gradually at a rate of 1 to 64 every 8 slices of the current processing rate of music.

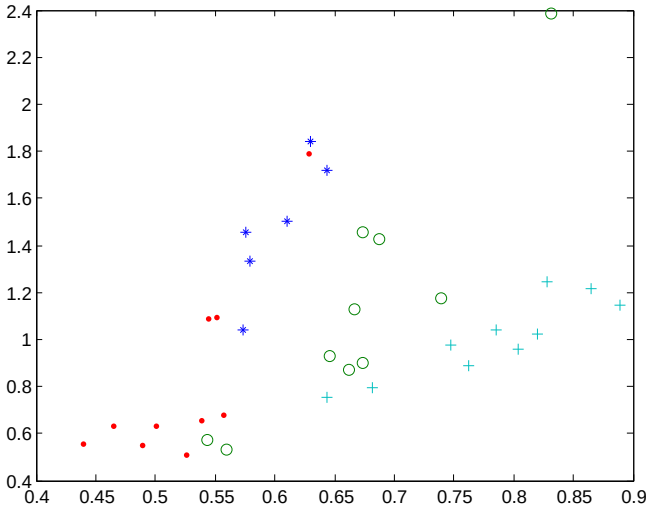

Figure 6: Plot of Std vs Mean for 4 Different Genres, with Green Circle Representing the Platformer

6 Results and Discussion

While we have discussed and implemented a fully functioning system it is also necessary to evaluate our system both as a complete piece of software, as well as individual pieces to demonstrate our underlying assumptions are correct. One of the major assumptions of this work is that video games have an underlying structure. While this has been concluded in previous work, in this paper we report on our evaluation of the accuracy of this assumption. In the earlier section 3.3 we discussed how we can model the underlying structure, thus implying that there is a structure. In this section we will evaluate if this structure is a) correlated with user experience and b) unique to specific games. Finally, we will report on the evaluation of the fully functioning system.

6.1 Associating with User Experience

While we have shown that the data shows specific characterstics associated with two overlapping distributions, there is no guarantee that this is actually due to the video game. For all we know users always interact in this way. If user interaction with a video game is unique to the game in question (or even games in general), then we should be able to verify this by changing the time domain (speeding up and slowing down) of the game. One would expect that if the user is attempting to approximate this model then the resulting interarrival time would likewise change. If, on the other hand, the user merely interacts with the computing devices consistently in this way, we should see no variation.

Users were asked to play a game (in this case Starcraft^{tm}) at two seperate speeds (denoted to the user as Slow and Fast). These speeds were the equivalent of 75% and 100% standard gameplay (thus constituting a 33% speed up). Users were then asked to replay the same level at both speeds and the data was recorded in the previously discussed manner. Two data sets were generated per user and the data was analyzed for comparison using a T-test on the means of the trials.

Results indicated a significant difference between the two groups (P(T) $<$ 1%), and that users interacted with the game, on average, 17% faster while maintaining a similar standard deviation. This shift is statistically significant enough for us to indicate that increasing the speed does affect user interaction in the anticipated method. However, this shift is still smaller than the expected 33% increase. This suggests that there is a second factor at play here. While the user does speed up to accommodate the faster pace, there are limits to the speed of human interaction. If we examine only events shorter than .5 seconds (as in the previous subsection) we notice that in addition to having similar distributions, these make up half of all events in both cases, which explains why we see about half the speed up we might expect.

Cassification Matrix				
Genre	RTS	Plat	MMO	FPS
RTS	80%	20%	0%	0%
Platformer	30%	50%	20%	0%
MMO	0%	0%	90%	10%
FPS	0%	17%	17%	67%

Table 1: The Results of Genre Classification

Cassification Matrix			
Genre	RTS	MMO	FPS
RTS	90%	10%	0%
MMO	0%	90%	10%
FPS	0%	17%	83%

Table 2: The Results Without Platformer Data

6.2 Game Genre Classification based on User Input

The idea that games have an underlying structure, or design, is not an original idea[Smith et al. 2008]. This idea has yet to be evaluated, and a demonstration is necessary for this work. Because we have established that user input is predictable to a degree, and that it is a close approximation of whatever structure exists, it becomes necessary to show that this model is not identical across different games.

To demonstrate this we conducted a pilot study where ten users were asked to play four different games. User input was recorded over the course of normal play. To guarantee the most noticable results, games were chosen from four different genre: Real Time Strategy (Starcraft^{tm}), First Person Shooter (Left4Dead^{tm}), Massive Multiplayer Online Role Playing Game (World of Warcraft^{tm}), and Platform Leveler (Super Mario World^{tm}). Each genre was run for 10 trials, with the exception of the first person shooter, which received only 6. Data was gathered for each user, and mean and standard deviation of interarrival time of the user's actions was calculated for each data sample.

To evaluate this data we used a Naive Baysian classifier on the average interarrival time and standard deviation. The implementation used was Weka[http://www.cs.waikato.ac.nz/ml/weka/]. Using a 10 fold cross-validation 72% accuracy was achieved; individual results are tabulated in table 1. The most notable standout was the Platformer with 50% accuracy. If we remove that from the data we achieve 88.5% accuracy as shown in table 2. This is most likely due to rapid changes in pace between levels, whereas the other genres tend to be consistent across levels. If we examine the data directly we see that the platformer has a standard deviation from .5 seconds to 2.3 seconds. No other data set has a standard deviation spanning more than a second; moreover it spans the range of all three others (see figure 6).

While more data might still provide higher accuracy, this clearly shows that between these four games, the game being

Preformance	
Condition	Avg. Progress
Silence	24.15%
Unmodified	15.40%
Adapted	26.95%

Table 3: A Comparison of the Preformance between Our System and Control

played can be determined purely from the input data. This, in turn, provides strong evidence that the underlying models being approximate by user input are different across these different video games.

6.3 User Feedback

Finally, there is the concern that our system may have a negative effect on performance as a distractor. Research in the past has demonstrated that music, especially music characterized as "High Arousal" [Cassidy and MacDonald 2007], can have a negative effect. As our system is designed to adapt music to gameplay to reduce negitive effects, we evaluate it in the course of regular usage. For this purpose we conducted a pilot study to observe how well a user preformed under our system. Participants were recruited from the Faculty and Student Body of the University of Victoria Computer Science Department.

Participants were asked to play Super Mario World^{tm} in the manner in which they felt appropriate. If they were unfamiliar with the game they were given instructions on the controls. Participants were randomly assigned to play the game with no music first (Negative Control), with music which was not adapted (Positive Control, or our system (Experimental); this order was randomized for each user. Participants were allowed 5 trials with each condition for a total of 15 trials, where a trial consisted of a death or completion of the level. Data was recorded on how far into the level the users were able to progress in a single life. The data on table 3 summerizes the average distance into the level for each of the condidtions. The average distance of all trials was 22.2% which corresponded to approximately a minute and a half of game play.

If we look at the data from this study we can see that Cassidy's[2007] findings are reaffirmed. In the presence of an additional distractor (Positive Control) performance is inhibited. However, when we examine the application of our current system vs. the negative control we notice that this performance difference is reduced as well as being reversed. Although the sample size is not sufficient to say this trend is certain ($P > .1$) it is evidence that our system is not as significant a distractor ($P < .1$).

7 Future Work

While the majority of this paper is concerned with adapting music to user data, a large portion of our evaluation is concerned with that user data. Being able to draw information out of user interaction is a useful tool; our identifying genre based on user interaction is evidence of this. Future application of this system could includ identifying the specific game the user is playing, the competency of the user playing, or the specific user that is playing, all of which could be of great benefit to our system or the state of the art. Finally, being able to identify which musical track might fit best to the user interaction is a future goal of this system.

8 Conclusion

In 2009, over 750 video games were released. Top sellers moved over 5 million units during 2009 in US alone[Wikipedia]. Despite these impressive sales, only four games had this high level of success. All of them came with sound tracks. More importantly, many of them had specialized sound tracks, where the actions of the players influenced the sound in more ways than just sound effects. This new area of relevant non-diegetic music has farther reaching implications than just assisting as a story telling tool. By changing the nature of where sound comes from, its uses also change.

The purpose of our work is to establish a method for adapting generic audio data to human input. To that end, our algorithm and the implementation here described accomplishes this goal. However, there still exists the possibility for errors. Because the algorithm is based upon a lower level technique for identifying beat strength, we inherit all the possible inaccuracies that lie therein. For example, music with irregular beat patterns is not adapted effectively using this approach. It will be interesting to see how improvements in understanding tempo and rhythm in audio files affects this work.

More information about our system and a video can be found at http://leffe.cs.uvic.ca/smr624/Syncronization/

References

Blake, A., and Zisserman, A. 1987. *Visual Reconstruction.* MIT Press.

Blattner, M. M., Sumikawa, D. A., and Greenberg, R. M. 1989. Earcons and icons: Their structure and common design principles (abstract only). *SIGCHI Bull. 21*, 1, 123–124.

Cassidy, G., and MacDonald, R. A. 2007. The effect of background music and background noise on the task performance of introverts and extraverts. *Psychology of Music 35*, 3, 517–537.

Chafe, C., and Leistikow, R. 2001. Levels of temporal resolution in sonification of network performance. In *Auditory Display*, International Community for Auditory Display.

Cram, S. M. J. H. D., and Duser, B. L. V. 2000. Effect of music tempo on heart rate and perceived exertion during rest, exercise, and recovery. *Research Quarterly for Exercise and Sport*.

Drake, C., Penel, A., and Bigand, E. 2000. Tapping in time with mechanically and expressively performed music. *Music Perception 1*, 18, 1–23.

Graham, R. L., Knuth, D. E., and Patashnik, O. 1989. *Concrete mathematics: a foundation for computer science.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Griffin, D. S. 1998. Musical techniques for interactivity. *Gamasutra 2*, 18, 3–5.

Holm, J., Havukainen, K., and Arrasvuori, J. 2005. Personalizing game content using audio-visual media. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACM, New York, NY, USA, 298–301.

Holm, J., Arrasvuori, J., and Havukainen, K. 2006. Modifying game content with personal media. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, ACM, New York, NY, USA, 86.

http://marsyas.sness.net/. Marsyas: Musical analysis, retrieval and synthesis for audio signals.

http://www.cs.waikato.ac.nz/ml/weka/. Weka 3: Data mining software in java.

Karageorghis, C. I., and Terry, P. C. 1997. The psychophysical effects of music in sport and exercise: a review. *Journal of Sport Behavior (JSB) 20*, 1, 54 – 68.

Kilander, F., and L'onnqvist, P. 2002. A whisper in the woods an ambient soundscape for peripheral awareness of remote processes. *International Conference on Auditory Display*.

Konecni, V. J. 1982. Social interaction and musical preference. *The Psychology of Music*, 497–516.

Larkin, T., 2008. Left 4 dead developer commentary. No Mercy (developer commentary).

Laroche, J., and Dolson, M. 1999. Improved phase vocoder time-scale modification of audio. *IEEE Transactions on Speach and Audio Processing 7*, 3.

Marsh, K. 1993. Win 32 hooks. *MSDN*.

Matesic, B. C., and Cromartie, F. 2002. Effects music has on lap pace, heart rate and perceived exertion rate during a 20-minute self-paced run. *The Sport Journal*.

McKelvie, P., and Low, J. 2002. Listening to mozart does not improve children's spatial ability: Final curtains for the mozart effect. *British journal of developmental psychology 20*, 2, 241–258.

Minami, K., Akutsu, A., Hamada, H., and Tonomura, Y. 1998. Video handling with music and speech detection. *IEEE MultiMedia 5*, 3, 17–25.

O'Keefe, K., and Haines, E., 2009. Dancing monkeys. http://monket.net/dancing-monkeys-v2/Main_Page.

Rauscher, F. H., Shaw, G. L., and Ky, C. N. 1993. Music and spatial task-performance. *Nature 365*, 611.

Smith, G., Cha, M., and Whitehead, J. 2008. A framework for analysis of 2d platform levels. In *Sandbox Symposium*, ACM SIGGRAPH.

Stollnitz, E. J., DeRose, T. D., and Salesin, D. H. 1995. Wavelets for computer graphics: a primer. 2. *Computer Graphics and Applications, IEEE 15*, 4, 75–85.

Tzanetakis, G., Essl, G., and Cook, P. 2002. Human perception and computer extraction of musical beat strength. *5th Int. Conference on Digital Audio Effects*.

Wikipedia. 2009 video games. Wikipedia.org.