Game/Music Interaction - An Aural Interface for Immersive Interactive Environments

Chris Nelson[1] **and Burkhard C. Wünsche**[2]

[1]Boston University, Dept. of Computer Science, 111 Cummington St, Boston, MA
[2]University of Auckland, Dept. of Computer Science, Private Bag 92019, Auckland, New Zealand
Email: chrisn1@bu.edu, burkhard@cs.auckland.ac.nz

Abstract

Game music has the potential to be much more than a passive element of the background. The music can and should affect the game play. The game play can and should affect the music. The player's actions can and should influence the direction and evolution of the music. By tightly linking game play and music, the player becomes much more immersed in the experience, and new creative possibilities abound for the developer. This paper presents a framework for linking game play to music by analysing music on a perceptual and a physical level. Real-time processing is achieved by using the GPU as an APU. The usefulness of the framework is demonstrated by two examples of game play synchronised with auditory perceptions. It is our hope that this paper will enable and stimulate the reader to create musically interactive games, and discovering entirely new ideas in this field.

Keywords: game technology, music synthesis, audio analysis, multi-sensory interfaces, perceptual rendering

1 Introduction

Music has always been a major entertainment medium and sound is used in computer games in the form of background music, speech, sound effects (e.g., gunshots), and game event signals (e.g., "Ka-ching!" sound after a successful player move). It has been shown that audio improves the narrative experience and can be used to guide the player through the game (Salen & Zimmerman 2004). In most cases the sound is controlled by the game play and consists of replaying pre-recorded sound files. Popular tools in game development on PCs are DirectMusic in combination with the MIDI music format which uses the Microsoft Software Synthesiser and Roland sound fonts (Holland 2002). In this paper we explore the symbiosis of music and game play, i.e., the game play should be able to influence the music (e.g., faster music for high action sequences) and vice versa the music should be able to influence the game play (e.g., if the music becomes louder enemy attacks become more violent). We present and evaluate a framework which enables a deeper integration of music and game play.

2 A Short Overview of the History of Game Music

One of the oldest examples of this symbiosis is in the 8-bit Nintendo game, Super Mario Brothers (SMBHQ.com n.d.). When the player is low on time to complete a level, the game doubles the speed of the music. This innovation was as effective as it was simple. By making the music sound rushed, the developers instill a great sense of urgency in the player, as they hurry to beat the clock. From the perspective of the audio library, merely adding a bit of code to allow for the doubling of a song's speed gives each and every song in the game that much more added depth and variance. As an audio library allows for more and more modifications to the music, the music becomes much more than just an element of the background.

Another example comes from a more recent game, .Hack (Bandai Inc. n.d.), which was first released in 2002. When exploring a dungeon, the music is ambient and atmospheric. If an enemy is encountered, a percussive track is layered on top of the first track, since combat is about to take place. Both tracks are from the same logical song, and thus in sync with one another, but the percussive track fades in and out as the current mood of the player, or perhaps the character they're controlling, goes from relaxed to intense, depending on whether combat is occurring. The important aspect of this example is that music isn't simply a single waveform. It's a combination of waveforms, which can be divided into categories of varying granularity, from broad categories like percussion, all the way down to specific instruments. Since the audio library for .Hack respected this fact, the designers were able to eliminate the auditory discontinuity of switching between one song for exploration, and another for combat. Instead, continuity was preserved, and the music flows and morphs itself to fit the scenario of the moment, without interruption.

A third example is a game called Rez (SonicTeam/Sega n.d.), which is the prime inspiration for this paper. As the player flies through each level, the music loop evolves as each way-point is reached. The real magic, however, is that every last sound effect is quantised to the beat of the music. A new sound effect set is loaded for each song / level, and each sound effect set is designed to be the building blocks that fit on top of the foundation that is the current level's music. Furthermore, as the player fires missiles at enemies, they explode in sync with the beat. The game instills an amount of synaesthesia in the player, as they learn to associate the visual aspect of an enemy with the audio element it produces when destroyed. Everything in this game happens in sync with the beat, and a great deal of the music is generated by the player's and the enemies' actions. The final musical output is a collaboration between professional musicians, game designers, the player, and enemy AI. And each gaming session generates a new version of the music.

In all of the above instances the game play influences the music. A very different example is Konami's Dance Dance Revolution series (KONAMI n.d.). Players stand on a specially constructed dance pad, and then must step on specific parts of it, in sync with the music, to perform a unique dance for each song. In contrast to Rez, where the game maps the player's input to the beat, the major game play element is the player mapping their own input

to the beat of the game. The more one is in sync with the rhythm, the higher one's score. Konami has released countless other rhythm games, varying the input device to instruments, such as guitars or drums. The game play, however, remains fairly constant. The success and popularity of this franchise is definitive evidence that players find the creative linking of music and game play to be very compelling.

3 Music Synthesis in DJing and Sonification

An important source of inspiration providing us with many key features for a musically interactive audio library is the art of DJing (Frederiks & Slowly 2003, Andersen 2005). DJs have developed techniques for modifying sound and influencing the audience' mood which are imminently applicable to video games. To solve the problem of the discontinuity that arises between two songs, DJs use beatmatching and frequency equalisation to create hours of music which sounds completely continuous, as songs are carefully weaved together.

Beatmatching is the process of modifying the playback speed of the next song (via pitchbending), so that each measure of the next song is as long as a measure in the song that's currently playing. Once two songs are beatmatched and synchronised, the DJ will crossfade from the current song to the next song, not just by varying the volume, but also by varying the amplitude of coarse frequency bands, usually 3 or 4. The goal, during a transition from one song to the next, is often to keep the combined volume and the amplitude of each frequency band more or less constant, to give the perception of continuity. A skillful DJ can preserve continuity to the point where the audience doesn't know when distinct songs are beginning and ending. Equalisation can also be used to emphasise or remove certain elements of a song, such as the bass line.

Another electronic music technique is granular synthesis, which is the process of creating sound or modifying music by repeating the most recent 1-100ms of audio many times per second. Pick up a CD by Venetian Snares (Venetian Snares 2004) if you desire to hear examples of this effect. Less exotic modifiers such as reverb (echo) and resonance (spiking the amplitude of specific frequencies) are also valid targets for inclusion in an audio library.

DJs techniques to control and modify audio have been maturing for well over 20 years, and game designers should approach their game's music with the mind set and tools of a DJ.

Another source of inspiration comes from the field of scientific visualisation. Various researchers have tried to employ other sensory organs than the eye to perceptualise data in order to avoid visual overload. Grinstein et al. experimented with the sonification of scientific data employing the loudness, pitch and orchestration of sound as additional output dimension (Grinstein & Smith 1990, Grinstein & Levkowitz 1995). Sonification has also been employed to perceptualise uncertainty (Lodha, Wilson & Sheehan 1996) and volume images (Rossiter & Ng 1996). The techniques utilised to visualise data can be translated into the game environment (Kot, Wünsche, Grundy & Hosking 2005) and be used the synthesise sound in correspondence with the game play.

4 A Musically Interactive Audio Library

The examples given in the previous sections constitute merely a slice of the history of game/music interaction. This slice, however, gives us motivation. In each example, a game has been improved by a unique set of features made available in its dependent audio library. Popular audio libraries, such as SDL_mixer (Lantinga, Peter & Gordon n.d.) and OpenAL (*OpenAL - homepage* n.d.), provide only basic capabilities, and thus atrophy the musical creativity of the developers who use them.

The lack of suitable tools for game-music synchronisation motivated this research and resulted in the novel musically interactive audio library discussed in the following sections.

4.1 High Level Design

Our musically interactive audio library was implemented in C/C++ because of it's ability to abstract away complexity into objects with comparatively simple interfaces. Graphics applications can be linked to the audio library via `SoundObjs` and `MusicObjs` objects, which are used for sound effects and music, respectively. SoundObjs are loaded entirely to memory in the constructor (possibly in a forked thread), but MusicObjs are streamed from the disk as chunks are needed. The reason for this is that sounds tend to be small, and played back several times, so we only want to load them once. Music is usually only played back once, so by distributing the decoding over the length of a song, we minimise the amount of work and memory required per frame, to deal with a song. At the moment, music is streamed from disk. Loading an mp3 or otherwise compressed song entirely into memory would decrease the delay associated with accessing the data, at the cost of increased memory use.

Both `SoundObjs` and `MusicObjs` share several methods. Basic functionality, such as `Play()`, `Stop()`, `Pause()`, `Resume()`, `Fade[In|Out]()`, `[Get|Set]Volume()`, and `[Get|Set]Position()`, is provided for both types of audio objects. More advanced methods include `SetSpeed()`, `SetGlitchAttributes()`, `SetSpeedInterpolationFactor()`, and a plethora of `MetaSync_*()` functions explained below.

The `SetSpeed*()` methods deal with pitchbend, playing the audio back at a different rate, and smoothly interpolating between two different playback rates. These methods are essential in providing DJ-style control of audio. `SetGlitchAttributes()` deals with granular synthesis, and allows for varying the number of samples repeated, the speed at which they're played back, and whether the granular synthesis replaces or augments the audio currently being played. Frequency equalisation is implemented, but not yet exposed through the API.

The `MetaSync_*()` functions are for use with music and allow the designer of a game to get answers about every aspect of a piece of music such as "Was a bass drum hit since the last frame?", "Have we reached a new half-beat since the last frame?", "What is the current BPM?" etc. This music meta data is stored with a song's *MetaSync* file and is loaded when a `MusicObj` is initialised. Each frame, the audio library updates the song's *MetaSync* state, as music is pushed to the soundscard. Exposing the song's musical structure to the programmer allows for game play elements to be linked to the beat, the percussion's rhythm, or even specific instruments. Furthermore, players could drop in different music with different *MetaSync* data to affect the game play in unexpected and unplanned ways. Each song would make the game play differently, and as a result add value to both the music and the game.

4.2 Generation of *MetaSync* Data

Generating *MetaSync* data can be difficult and ideally should be performed automatically. We have developed a tool which enables human users to quickly describe a song in detail, i.e., what qualifies as a "snare drum" or how to differentiate a bassline from a bass drum. The program is a modified version of the first author's DJing software. A five minute track was described completely in approximately five hours.

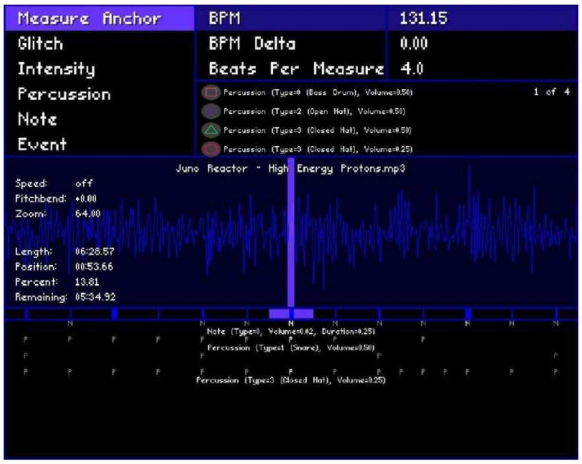

Figure 1: *MetaSync* data generation software that uses a Playstation2 controller as the input device.

The process of using the tool is quite simple and is illustrated in figure 1. A song is loaded into a waveform (the middle row in the above image), which can be played back at various speeds. As the song plays back, it is the user's job to press buttons which will indicate the presence of a certain piece of metadata as they audibly and visibly pass under the metaphorical record needle (the vertical bar in the middle of the waveform). The timestamp of these button presses is quantised to a user-definable degree. One useful mode of playback allows a measure to be looped until the user decides they've completely described it. Specific instrument groups can be copied and pasted into and from multiple clipboards, which capitalises on the repetitive nature of many types of music. The resulting *MetaSync* file format is a series of plain text directives, each directive describing a musical element at a specific timestamp.

Currently the generation of *MetaSync* data is performed by hand as described above. However, we plan to use the Fast Fourier Transform to convert small chunks of music, tagged by the hand-generated *MetaSync* data, into the spatial domain. We could then use Neural Networks in order to identify such chunks of music in the untagged part of a music file and as such complete *MetaSync* files semi-automatically. Once enough *MetaSync* data exists for different types of music such an algorithm might even be able to process an entirely new piece of music.

4.3 Frequency Querying / Modification / Reaction

The audio library also allows a game to query and modify the amplitude of a range of frequencies, via Fast Fourier Transforms and equalisation. This gives designers another set of information about the structure of the music. Aspects of the game can react to aspects of the music. As bass frequencies become increasingly prominent, the environment's lighting could become darker. The absence of midrange frequencies could make certain enemies change behaviour. High frequencies, when present, could make a helpful platform appear, only to slowly fade away if the frequencies don't remain. The audio library gives developers opportunities to use the music's structure creatively, and to make their world's behaviour dependent on the music, which in turn could be player-modifiable, allowing for increasingly varied game play. The player's actions could also affect the music, via equalisation. If you destroy a crystal, perhaps the high frequencies of the music are muted for 20 seconds, which, in turn, affects another element of the world.

4.4 Low Level Implementation

The heart of the audio library is the audio callback function. This function is called in a separate thread every time the sound card's output buffer needs refilling. Linux allows for smaller output buffer, typically 1024 samples, providing lower audio latency. Windows typically requires a buffer size of at least 2048 samples. Each iteration of the audio callback function loops through every `SoundObj` and `MusicObj` that's currently playing. A nested loop then proceeds to mix data from each object's audio buffer into the sound card's output buffer, processing the effects on a per-sample basis.

Harware acceleration is achieved by treating audio signals as graphics primitives, i.e. audio signals are loaded, a chunk at a time, onto a texture. Once there, we can draw these textures to an off-screen buffer in specific ways to obtain the results of operations such as multi-track mixing, amplitude scaling, pitchbend, reverb, equalisation, granular synthesis, and more. This allows the vast majority of work to be offloaded to the graphics hardware, saving valuable CPU time, performing the calculations in a shorter amount of actual time, and potentially allowing for increased audio responsiveness and decreased latency.

5 Results

A motivation for writing the audio library was the desire to be able to write musically interactive games. One of the first author's works-in-progress was chosen to be extended to use this new audio library. The game in question is a 2d spaceship shooter, similar in game play to Space Invaders. The game was modified such that enemies fire bullets when the bass drum kicks, making certain sections of the song safe, and others exceedingly dangerous. Different weapons that the player uses are triggered by different instruments, so if the player uses the right weapon during a snare roll, more damage is output. Only a small fraction of the audio library has been used thus far, but already the game has a definite link to the music.

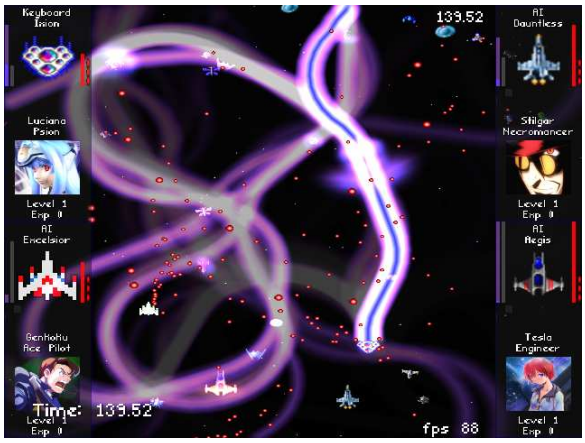

Figure 2: Red enemy bullets pulsate with the beat of the music, as a homing laser fires in sync with the music's bass line.

Another application developed for use with the audio library is *DJing* software. The audio library provides all the functionality necessary to simulate a DJ setup, with equalisation, beatmatching via pitchbend, crossfading, and scratching (via the `SetSpeed*()` functions). By combining the DJ capabilities with the *MetaSync* data, it's theoretically possible for a game to beatmatch two tracks, and crossfade between them, as the player progresses from one level to the next, preserving the music's

continuity. This feature will prove useful for the aforementioned spaceship shooter.

An example of beatmatching is illustrated in figure 3. The high-amplitude sections of the lower two waveforms are bass drums, which are lined up with one another, so the user knows everything is in sync. The bottom waveform has a pitchbend of -1.37%, so it plays back at the same speed as the top track.

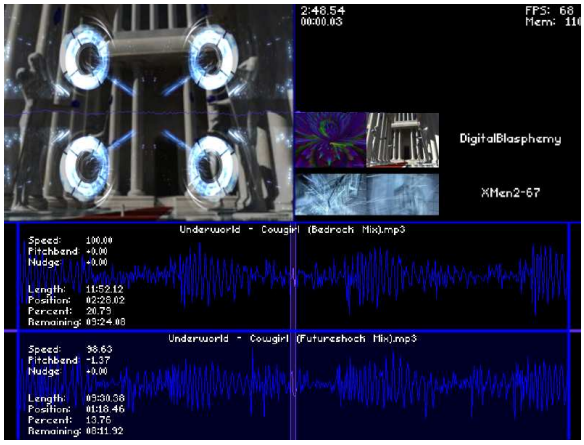

Figure 3: Two remixes of an Underworld track are visibly beatmatched.

This audio library is still in development, but it currently provides all the major features of SDL_mixer, as well as some important music / game interaction technology. Future improvements will include multi-track music, as found in .Hack, enhanced GPU acceleration, and a revamped API. Once this is complete, it will be released to the public, under the GNU LGPL license, and hopefully give rise to some creative games, as a result.

6 Conclusions

Gaming history shows us the important effects that music can have on the game play experience, when it is actively linked to the world, rather than a passive element of the background. Super Mario Brothers, .Hack, Rez, and Dance Dance Revolution all illustrate different ways music can interact with a game.

Game designers and programmers are bound by the tools they use. The most popular publicly available audio libraries don't give designers the tools they need to exercise their creativity in the field of game / music symbiosis. We have presented a musically interactive audio library which gives game designers read/write access to the underlying structure of the music used in a game. The library provides a framework for linking game play to music by analysing music on a perceptual and a physical level. The usefulness of the framework was demonstrated with two examples of game play synchronised with auditory perceptions.

7 Acknowledgements

Thanks are due to all game developers whose creative risks advance the state of gaming as a whole. Specifically, thanks is owed to the designers and programmers whose methods are referenced in this paper. Sam Lantinga's SDL library deserves credit as well. By releasing such a powerful tool under an open source license, countless programmers are empowered to become game developers. Everyone who has contributed to Simple Direct Media Layer (SDL) deserve thanks.

References

Andersen, T. H. (2005), In the mixxx: novel digital DJ interfaces, *in* 'CHI '05: extended abstracts on Human factors in computing systems', ACM Press, pp. 1136–1137.

Bandai Inc. (n.d.), '.Hack - homepage'. URL: `htpp://dothack.com`.

Frederiks, T. & Slowly, D. (2003), *Dj Techniques*, Sanctuary Publishing.

Grinstein, G. G. & Levkowitz, H., eds (1995), *Perceptual Issues in Visualization*, Springer Verlag, Berlin.

Grinstein, G. G. & Smith, S. (1990), Perceptualization of scientific data, *in* E. Farrell, ed., 'Proceedings of the 1990 SPIE/SPSE Conference #1259: Extracting Meaning form Complex Data', The Society for Image Science and Technology, pp. 190–199.

Holland, H. (2002), *Game Programming - Tricks of the Trade*, Premier Press Inc., chapter 8 - Sound and Music. (Lorenzo D. Phillips Jr., ed.).

KONAMI (n.d.), 'Dance dance revolution gateway'. URL: `http://www.konami.jp/gs/game/ddrgateway/`.

Kot, B., Wünsche, B. C., Grundy, J. & Hosking, J. (2005), Information visualisation utilising 3d computer game engines - case study: A source code comprehension tool, *in* 'Proceedings of the 6th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction (CHINZ 2005)', pp. 53–60. Auckland, New Zealand, 7-8 July 2005, URL: `http://www.cs.auckland.ac.nz/~burkhard/Publications/CHINZ05_KotWuenscheGrundyHosking.pdf`.

Lantinga, S., Peter, S. & Gordon, R. (n.d.), 'SDL_mixer 1.2'. URL: `http://www.libsdl.org/projects/SDL_mixer/`.

Lodha, S. K., Wilson, C. M. & Sheehan, R. E. (1996), LISTEN: Sounding uncertainty visualization, *in* R. Yagel & G. M. Nielson, eds, 'Proceedings of Visualization '96', IEEE, pp. 189 – 195.

OpenAL - homepage (n.d.), URL: `http://www.openal.org/`.

Rossiter, D. & Ng, W.-Y. (1996), A system for the complementary visualization of 3D volume images using 2D and 3D binaurally processed sonification representations, *in* R. Yagel & G. M. Nielson, eds, 'Proceedings of Visualization '96', IEEE, pp. 351 – 354.

Salen, K. & Zimmerman, E. (2004), *Rules of Play*, MIT Press, Cambridge, Massachusetts.

SMBHQ.com (n.d.), 'Super mario bros headquarters homepage'. URL: `htpp://smbhq.com/smb.htm`.

SonicTeam/Sega (n.d.), 'Rez - homepage'. URL: `http://www.sonicteam.com/rez`.

Venetian Snares (2004), 'Huge chrome cylinder box unfolding'. Planet Mu.