TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

M A S T E R A R B E I T

Foundations for Music-Based Games

Ausgeführt am Institut für

Gestaltungs- und Wirkungsforschung

der Technischen Universität Wien

unter der Anleitung von Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Purgathofer
und Univ.Ass. Dipl.-Ing. Dr.techn. Martin Pichlmair

durch

Marc-Oliver Marschner
Arndtstrasse 60/5a, A-1120 WIEN

01.02.2008

Abstract

The goal of this document is to establish a foundation for the creation of music-based computer and video games.

The first part is intended to give an overview of sound in video and computer games. It starts with a summary of the history of game sound, beginning with the arguably first documented game, *Tennis for Two*, and leading up to current developments in the field. Next I present a short introduction to audio, including descriptions of the basic properties of sound waves, as well as of the special characteristics of digital audio.

I continue with a presentation of the possibilities of storing digital audio and a summary of the methods used to play back sound with an emphasis on the recreation of realistic environments and the positioning of sound sources in three dimensional space. The chapter is concluded with an overview of possible categorizations of game audio including a method to differentiate between music-based games. These classifications are then illustrated by means of an example.

In the second part of the thesis (technical foundations) I present two enhancements of the Torque Game Builder (TGB) engine and present a prototype of music-based game taking advantage of these improvements, *Radiolaris*. Chapter one of this section deals with the technical issues of implementing the support of gamepads and joysticks for the Mac OS X version of the Torque Game Builder engine. I summarize a technique for communication with HID (Human Interface Device) compliant USB devices by using libraries present in the operating system and show a possible way of handling the output of these devices within the game engine.

In the second chapter of part two, I portray a way to integrate the *FMOD Ex* audio library into the TGB engine. The TGB engine already includes support for loading and playing music data, *OpenAL*, but unfortunately this library is geared towards using three dimensional sound sources, an approach not suitable for the use with the chosen game engine (the TGB engine is used exclusively for two dimensional games). In comparison, the *FMOD Ex* library incorporates functions to control sound sources directly in two dimensions as well as offering a selection of different predefined audio effects. My approach includes this functionality in the TGB Engine to substitute the *OpenAL* library.

I conclude the second part with a presentation of *Radiolaris*, a prototype developed by Martin Pichlmair and Fares Kayali. The music-based game incorporates many of the new functions for the TGB engine as presented in the thesis.

Zusammenfassung

Diese Arbeit ist der Versuch, eine Grundlage für das Erstellen von musikbasierten Spielen zu geben.

Das erste Teil beschäftigt sich mit den theoretischen Grundlagen von Klang in Computer- und Videospielen. Er bietet eine Übersicht über die Geschichte von Klang in Spielen, beginnend mit dem wohl ersten dokumentierten Spiel, *Tennis for Two*, bis hin zu den aktuellen Entwicklungen auf dem Gebiet. Dabei wird auch auf die zu der jeweiligen Zeit verfügbare Hardware eingegangen.

Des Weiteren bietet das Kapitel eine Einführung in die Entstehung und die Eigenschaften von Klang, sowie die elektronische Erzeugung desselbigen. Ferner wird kurz auf die verschiedenen Möglichkeiten eingegangen, Klang elektronisch zu speichern und wiederzugeben, wobei das Hauptaugenmerk bei der Beschreibung der Wiedergabe auf der Positionierung und realistischen Darstellung von Klangquellen liegt.

Abschließend wird eine Übersicht über die Möglichkeiten der Klassifizierung von Klang in Computer- und Videospielen geliefert. Zusätzlich wird eine Möglichkeit beschrieben, musikbasierte Computer – und Videospiele zu kategorisieren. Anhand eines Beispiels werden diese Einteilungen dann anschaulich gemacht.

Der zweite Teil bietet eine technische Beschreibung der Implementierung einer Gamepad- und Joystickunterstützung für die Torque Game Builder (TGB) Engine in der Mac OS X Version und eine Beschreibung der Einpassung der *FMOD Ex* Bibliothek in die Engine. Im ersten Kapitel dieses Teils wird zusammengefasst, wie ein Gerät, dass der HID (Human Interface Device) Geräteklasse, einer Untergruppe des USB-Standards, angehörig ist, in Mac OS X mittels systemeigenen Bibliotheken angesprochen werden kann. Des Weiteren wird ein Weg beschrieben, die Ausgabe des Gerätes in der TGB Engine zu verarbeiten.

Das zweite Kapitel von Teil zwei beschäftigt sich mit der Integrierung einer alternativen Audio Bibliothek in der TGB Engine. *FMOD Ex* ist eine Bibliothek, die es dem Nutzer ermöglicht, Musikdateien zu laden und wiederzugeben. Zusätzlich enthält *FMOD Ex* diverse vorgefertigte Effekte. Zwar bietet die TGB Engine mit *OpenAL* schon eine solche Bibliothek an, leider ist diese aber auf die Nutzung dreidimensionaler Klangquellen ausgerichtet und somit für die zweidimensionale Darstellung in der TGB Engine ungeeignet. Meine Arbeit zeigt eine Möglichkeit auf, wie *FMOD Ex* anstelle der *OpenAL* Bibliothek verwendet werden kann.

Als Abschluss des technischen Teils präsentiere ich einen Prototyp für ein musikbasiertes Spiel. *Radiolaris*, entwickelt von Martin Pichlmair und Fares Kayali verwendet viele der neu in der TGB Engine integrierten Funktionen, die in diesem Teil vorgestellt wurden.

Acknowledgements

I would like to express my gratitude to the following people helping me in the creation of this thesis.

Univ.Ass. Dipl.-Ing. Dr.techn. Martin Pichlmair for his constant input while writing this thesis. No matter how small the question, I always promptly received an answer to my problem.

Dipl.-Ing. Fares Kayali, for all the input I received from him during the time of writing – especially in the hunt for bugs in the technical part of my thesis.

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Purgathofer, whose lessons sparked my interest in Media and Computer Science.

Last but definitely not least my parents, Grazyna and Frank Marschner who patiently supported me in my studies and assisted me in widening my horizon.

Content

List of Figures

Introduction

From its humble beginnings in the seventies the game industry has risen to become an important sector of today's entertainment business. According to data collected by the Entertainment Software Association (ESA), the U.S. computer and video game industry more than doubled its sales in dollars between the years 1996 and 2006, while nearly tripling the amount of games sold in the same timeframe[1]. [Entertainment Software Association, 2008]

Over the years the industry has seen great leaps in the quality and production values of video games. Graphics have improved tremendously since the industry's early days; starting from a nearly unrecognizable accumulation of pixels to highly detailed three dimensional environments and characters in high resolutions. The possibilities of game play have also expanded greatly, introducing players to new genres like first person shooters, role playing games or real time strategy games over the years. Of course the field of game sound production did not stand still either with current titles delivering more than one hundred different sound sources positioned throughout the three dimensional space and delivered to the player through sound systems using up to eight speakers at once.

Throughout game history, several companies have tried to combine audio and game play to convey a new gaming experience to the player. Music-based games entered the mainstream with *Parappa the Rapper* on the *PlayStation* and *Beatmania* in the (Japanese) arcades in 1997. Since then, a steady stream of music-based games has been released. With the release of *SingStar*, *Karaoke Revolution* and especially *Guitar Hero* the genre has really started surging.

In my thesis I give a foundation for the creation of prototypes for music-based games. For this I use the first part (theoretical foundations) to give an introduction to music in video games in general, offering an overview of the history of sound in games, and then describing the properties of sound in general, as well as digital sound in particular. This is followed by an overview of the most common ways of storing audio for games and a summary of the methods used to play back sound in games. In this summary I put a strong emphasis on the approaches used to create realistic sound (in three dimensional environments).

The chapter is concluded with an overview of means to categorize sounds, including an approach for the classification of music-based games. These categorizations are then illustrated by employing an example, the game *Phase*, released by Harmonix Music Systems in 2007.

In the next two chapters (part 2, the technical foundations) I present methods to extend a game engine with functions necessary to allow rapid prototyping of music-based games. The engine chosen, the Torque Game Builder (TGB) engine, is designed for two-dimensional games. It offers easy accessibility for users through its built-in editor while maintaining the possibility to create more advanced procedures by including support for a scripting language.

Unfortunately the TGB engine has two disadvantages. One of them is only present in the Mac OS X version used in this thesis while the other one is inherent to all versions of the engine.

[1] The terms "computer games" and "video games" are normally used to describe different kinds of software, with "computer games" referring to games played on a PC and "video games" describing games normally played on a console. In this thesis, I will use the term "video games" to describe both groups of games.

The drawback of the Mac OS X version of the TGB engine is the missing joystick and gamepad support. I present a method of finding and using HID (Human Interface Device) devices described by Apple and I offer a way of reading and handling the output of game control devices found this way within the game engine. Further I present an approach to use Apple's Force Feedback Framework to induce Force Feedback effects within the game device, usable with the script language included.

The TGB engine uses the *OpenAL* Library to handle sound, a feature inherited from the Torque Game Engine which it is based upon. While *OpenAL* is suitable for the Torque Game Engine, a framework designed for three-dimensional games, it is rather complicated to work with it in a two-dimensional environment – the prime discipline of the TGB engine. To remedy this, I propose a method to allow the user to utilize another sound library, *FMOD Ex*. This library allows an uncomplicated handling of two-dimensional sound sources without burdening the user with unnecessary calculations. Further the library includes a selection of predefined effects. My approach makes these functions accessible through the script language of the TGB engine.

Both of these TGB engine modifications are built to make their inclusion within the game engine as easy as possible by not requiring any modifications of existing files. The extension of the engine with these capacities should make it easier for users to create music-based games.

As a conclusion of the second part I present *Radiolaris*, a prototype of a music-based game created by Martin Pichlmair and Fares Kayali. Many of the new functions integrated into the TGB engine are a direct result of requirements that appeared in the development of this prototype. This makes the game a prime example to illustrate the capabilities of the additions to the TGB engine.

Part 1: Theoretical Foundations

The History of Sound in Video Games

The Beginning (1958) - 1979

"Video game music has come a long way, baby. " [McDonald, 2004]

The arguably first video game ever created, *Tennis for Two*, does not have any sound at all. It was designed by William Higinbotham in 1958 as a presentation for visitors at his workplace, the Brookhaven National Laboratory. The same is true of Steve Russell's *Spacewar*, created in 1963 – again, the game features no sound effects, as well as no soundtrack.

1972 sees the release of the Magnavox *Odyssey* gaming console, the first gaming system created for the home market. This console also does not feature any sound output. In the same year, Nolan Bushnell's *Pong* is released to a small audience. This game features the first sound effect ever presented in any game – a "sonar-blip" played as the ball hits one of the two paddles. [McDonald, 2004]

Until 1977 sound in video games is restricted to arcade machines. Atari releases the *Video Computer System* (*VCS*, called the Atari *2600* after the introduction of the *5200*) and brings sound effects to the home user. The console features two independent audio channels with three registers, each controlling "a noise-tone generator (what kind of sound), a frequency selection (high or low pitch of the sound), and a volume control." [Wright, 1979] It still takes one more year until the first game featuring what could be considered a sound track is released. *Space Invaders* ,developed by Taito and released first in the arcades, includes a pulsing sound that increases in speed as the aliens attacking the player move closer to the bottom of the screen and thereby the player's position. By changing pace depending on the games state, the soundtrack of *Space Invaders* can be considered the first interactive/adaptive soundtrack used in a video game. [Pidkameny, 2002]

1980 - 1984

The next important steps in terms of music are taken in 1980. The arcade game *Berzerk* is among the first games to feature voice synthesizers, being the first to include talking enemies taunting the player. The same year *Pac-Man*, arguably one of the most popular games of all time, is released. [McDonald, 2004] *Pac-Man's* opening can be considered "the first instance of a melody used to complement the mood of a game." [Pidkameny, 2002]

IBM starts selling the IBM *PC*, specifically the model *5150*, in October of 1981. [Polsson, 2007] In order to save on production costs, cuts are made when possible, resulting in a minimum of audio playback capabilities: As the IBM *PC* is intended to be an office machine, only the PC Speaker is included. The hardware can output a rectangle signal at fixed volume and possesses one sound channel. [Goehler, 2003] Chords can be emulated through alternating between different notes quickly (also called arpeggio), but results are poor. Sample playback is made possible later in the same way it is created on the Commodore *C64* – by quickly turning on and off the speaker. [Weske, 2000]

In 1982 Atari releases the Atari *5200* system that includes the POKEY chip as sound generator. POKEY (POtentiometer and KEYboard Integrated Circuit) provides four semi-independent audio channels

which can be configured in three ways: four channels with 8-bit resolution output, two channels with 16-bit, or one channel with 16-bit and two channels with 8-bit output. Again, all channels feature frequency, noise and volume controls. [Atari, 1982] Already in 1981 *Tempest* is released with two POKEYs, allowing eight voices at the same time. In 1983, *Dragon's Lair* features real human voices and stereo sound [McDonald, 2004], but such sound quality comes at a price: As it is dependant on laser-disc technology the game does not feature a lot of interactive elements.

Commodore presents the *C64* in 1982. The so called home computer is released with the first self developed sound-chip in a home computer, the MOS Technology 6581 Sound Interface Device (SID). Created by Bob Yannes between 1981 and 1982 [Kubarth, 2006], the sound hardware can arguably be considered to be among the best of the 8-bit generation. The 6851 chip features three synthesizer channels that can be used independently. Each channel consists of a waveform generator, an envelope generator (enabling the use of Attack, Decay, Sustain and Release), and an amplitude modulator. Later models of the *C64* include a new version of the MOS Technology 8580 SID sound-chip. This version had slightly different waveforms and reduces the volume of sample playback[2]. [Kuphaldt, 2007]

In 1983 IBM releases a competitor on the home computer market, the *PCJr*. [Weske, 2000] This machine expands the audio capabilities of the original IBM PC, featuring a chip by Texas Instruments providing 3 sound channels and one noise channel. Tandy produces a clone of this machine under the name of *Tandy 1000*. A later model, the Tandy *TL/SL* allows direct playback of samples while using the synthesizer at the same time. [Goehler, 2003]

Automata UK Ltd, a British based publisher uses a novel concept for music in games: The game *Deus Ex Machine*, published in 1984 for the Sinclair *ZX Spectrum* and later released for the *C64*, includes an audio tape intended to be played along with the game. The soundtrack, sound effects, and voice narration are stored on the tape. [Bridgett, 2005]

1985 - 1989

The first home video-game system to really become popular in the mainstream is released in 1985. [Pidkameny, 2002] The *Nintendo Entertainment System* creates its voices (one sine, one noise and two pulse-wave voices and one sample channel) digitally and converts them to analog output with 4-bit DACs [Tätilä, 2007]. The system is able to output four sounds at once. While early games use three channels for music and one for sound effects, later games employ all four channels for music playback by applying certain tricks like switching less important voices off for the duration of a sound effect and playing the effect on that channel instead. [Belinkie, 1999] The included sample channel's main purpose is outputting heavy compressed drum sounds, but some game developers use it to output speech instead. [Tätilä, 2007]

Shortly after the appearance of the *Nintendo Entertainment System*, *Super Mario Bros.* is released in 1985. *Super Mario Bros.* is the first to include constant background music written by a professional composer. The game "established many conventions for game music, which survive to the present day. " [Belinkie, 1999] "With the Super Mario Bros. soundtrack, video game sound design begins to

[2] The original chip emits a crackling noise as the sound volume is turned up and down in fast succession. This error is used to generate sample playback. The later revision reduces the volume and makes sample playback nearly inaudible.

move in a new direction, away from cinematic conventions and toward something altogether new. “ [McDonald, 2004]

The same year the first *Amiga* computer by Commodore, renamed *Amiga 1000* following the release of the *Amiga 500* and *Amiga 2000*, is presented at the Lincoln Centre in New York. [The Purple Owl, 2004] The sound hardware of the *Amiga* is able to play four independent channels. Each of these is capable of playing high quality samples back at different sampling rates. Samples are used for game music and sound effects. A new type of file is created for this kind of playback, the module file, commonly referred to as MOD. “Mod files are music files that have the note and other control data stored in the file but more importantly mods allow you to store pieces of digitized sound (samples) and use them as instruments. “ [Tätilä, 2007] This marks the difference to the MIDI file format, as MIDI files only contain instructions on how to play a sound which can make the same piece of music sound different on different hardware (it is up to the sound hardware manufacturer to decide how to implement instruments). [Velden, 1998]

Sega releases the *Master System* in 1986, a machine with graphic- and sound-chips superior to the ones used in the *Nintendo Entertainment System*. The console uses the Texas Instruments SN76489 chip for sound generation, allowing three channels for music and one for noise generation. Later versions of the console in Japan also include the Yamaha YM2413 FM sound-chip. This chip uses frequency modulation (FM) to create sound, a kind of synthesis that is good for metallic and bell like sound and resembles real instruments more closely than analog synthesizers. [SegaStuff, 2007] [Tätilä, 2007] Atari also releases a new console, the Atari *7800*. The sound hardware does not change from the Atari *2600*, but games can be outfitted with POKEY chips residing on the cartridges themselves to provide appropriate sound. [AtariAge, 2006]

In the same year the first part of the *Dragon Quest* series is released. The composer Koichi Sugiyama recreates the sound of classical music on the *Nintendo Entertainment System*, a feat everyone had previously considered impossible. “For the first time, game music aspired to be musical, and not just bearable. “ [Belinkie, 1999]

In the PC world, 1986 marks the year of the release of the Covox *Speech Thing* and the Disney *Sound Source*. [Goehler, 2003] Instead of using an extension slot, both devices are connected to the machine through the parallel port. The Covox *Speech Thing* even gets released in different versions, allowing recording or stereo playback. The sound hardware only receives lukewarm support by game developers. [Weske, 2000]

In 1987, the first part of *Final Fantasy*, another classic franchise, is developed and released by Square. The composer of the game music is Nobuo Uematsu, the most famous and successful game music composer of all time. [Belinkie, 1999] Uematsu combines melodies with classic sounding background parts, breaking entirely new ground in the process. [McDonald, 2004]

Also in 1987, the first part of the *Zelda* series is released. Created by Shigeru Miyamoto, the creator of Mario and many other Nintendo franchises, the game features a well-received soundtrack by Koji Kondo (also the composer of the *Super Mario Bros.* soundtrack) and went on to incorporate music strongly into its game-play in the later parts. [Nintendo, 2005]

AdLib releases its sound card for the PC in the same year. Up to this point “other systems sounded more or less like the multiplicity of interconnected PC-Speakers” [Goehler, 2003] The Yamaha

YM3812 (more commonly known as OPL2) located on the card, used in low-end-keyboards before, employs frequency modulation to either allow six voices and five percussion instruments or playback of nine voices. IBM introduces the IBM *Music Feature Card* which basically represents a Yamaha FB-01 synthesizer. The card features 8 FM voices, stereo with pan and voices controlled by four operators (as opposed to AdLib's two). Additionally it offers a library of over 300 synthesized high-quality instruments. Further, two of these cards can be used at the same time to offer 16 simultaneous voices. [Goehler, 2003]

The *Famicom Disk System*, a peripheral for the *Nintendo Entertainment System* only released in Japan, sees the release of *Otocky*, published by ASCII in 1987. "The goal of the game is that players collect a certain amount of "notes" to finish a game while avoiding and destroying enemies. Players launch a missile called a Music Ball to shoot enemies and collect "notes". When players fire a Music Ball, it emits a sound. The most interesting things in the game are that players are allowed to launch a missile in eight different directions which are directly correspond[sic] to the tonal score. "[Takeshita, 2003]

In 1988 Creative Music Systems (now called Creative Labs) releases the *Game Blaster*/*Creative Music System*. The sound card itself offers 12 channels able to output a sine wave or to generate noise, and represents no competition in quality for AdLib, but it marks the entry point of Creative Labs in the business of producing sound cards. Roland also releases its own sound card, the *LAPC-I* based on the *MT-32* chipset by the same company. The card offers 32 MIDI-channels (with reverb effect, 32 simultaneous instruments are not guaranteed[3]) and 128 predefined instruments. Further, the card allows the upload of new instruments and puts an emphasis on high-quality playback. [Goehler, 2003]

At the end of the eighties, in 1989, several new gaming machines are introduced to the market. [Pidkameny, 2002] The portable Nintendo *GameBoy*, one of the best selling consoles in the world, is released with a sound-chip allowing stereo output. The *GameBoy* hardware can create sound on four channels (white noise, quadrangular wave patterns with sweep and envelope functions, quadrangular wave patterns with envelope functions, voluntary wave pattern) that can be controlled independently, and are able to output sound to both or either one of the two stereo channels. [Fayzullin, et al., 1998]

In addition to the *GameBoy*, two other systems are released. The NEC *TurboGrafx-16* (called *PC Engine* in Japan), a collaboration of NEC and Hudson Soft, offers six channels in stereo. [Hernandez, 2007] NEC also releases an attachment to support CD playback. Sega releases the *Genesis* (called *Mega Drive* in Europe) with the Z80 CPU and the Yamaha YM2413 from the later Japanese Master System model as sound processors, offering six-channel stereo sound. [McDonald, 2004]

In 1989 Sega also brings *Moonwalker* to the *Genesis*. In the game the player controls Michael Jackson fighting against criminals while trying to save children. The player as Michael Jackson can beat them by using a variety of dance moves, up to a fully choreographed dance number including the enemies on the screen. Of course, while playing the game the player can hear different songs from Michael Jackson's repertoire, including "Smooth Criminal", "Billie Jean", "Bad", "Another Part of Me",

[3] Although 32 channels are featured, the card uses up to 4 of these channels to recreate an instrument. Theoretically, the card can only play 8 instruments at once. [Goehler, 2003]

"Thriller" and "Beat It". [DeRienzo, 2007] This is the first time a pop artist has such a major involvement in a game's soundtrack. [Pidkameny, 2002]

The same year, Creative Labs releases the first card in a series that defines the market for sound cards in the nineties – the *Sound Blaster*. The card represents an evolution of the previous effort, the *Game Blaster*, improved in the way the game industry requested. One channel allows playback of 8-bit mono sound at 22.05 kHz and recording at 13 kHz is added. [Goehler, 2003] Furthermore, the card features a game port for joysticks and gamepads and can offer compatibility to the AdLib product with the use of the same OPL 2 synthesizer chip. Creative Labs also offers extensive support for its product, shipping out new drivers to developers as fast as possible and providing cheap developer's kits. The company can also offer the card at a cheap price: Unlike *AdLib* sound cards manufactured in Canada, *Sound Blaster* cards are produced in high quantities in Asia. [Weske, 2000] Later versions of the *Sound Blaster* make the compatibility to the *Game Blaster* optional (version 1.5), offer a new redesigned layout that also removes crackling in the sound rendering (version 2.0) and allows sound playback at 44.1 kHz (version 2.1). [Goehler, 2003] The card also inspires other manufactures to produce "Sound Blaster–compatible" hardware. Due to the revisions of the *Sound Blaster* (and subsequent alterations to the digital signal processor hardware) the compatibility is not always guaranteed. Also, many of the products are of inferior quality compared to Creative Labs' hardware. [Weske, 2000]

1990 - 1994

SNK releases the *NeoGeo* in 1990[4], offering a true arcade experience for the home user by basically repackaging the arcade hardware in console format. The system uses a Yamaha YM2610 sound-chip that is able to output 15 channels in stereo (seven sample channels, four FM channels, three programmable sound generators (PSG) and one noise channel) simultaneously. [PC Vs Console, 2007]

The nineties see the rise of sampling as the preferred method of sound generation. The first console[5] to completely rely on sample playback is Nintendo's *Super Nintendo Entertainment System*, is released in Japan in 1990, followed by North America in 1991 and Europe in 1992. The machine uses a Sony SPC700 sound-chip, allowing the playback of samples over eight separate channels. As well as simple playback, every channel also offers "looping, sample interpolation, lowpass filtering and delay effects as well as an ADSR-amplitude envelope." [Tätilä, 2007]

LucasArts releases *Loom* in 1990. The games music is based on Tchaikovsky's "Swan Lake", featuring different sections of the piece in MIDI with additional original music by the game's composer George Sander. *Loom* does not have a continuous soundtrack, "since almost all of the game-play involves the player making his own music; assuming the role of Bobbin Threadbare, a member of the Weavers' Guild, the player weaves the very fabric of reality by casting spells or "drafts" consisting of four musical notes, solving puzzles by learning new sequences of notes from the world around him and applying them in the appropriate situations." [Pidkameny, 2002]

Joe Montana Sportstalk Football II is released on the Sega *Genesis* in 1991, offering the user continuous play-by-play commentary of the action happening on the screen. Since then this has

[4] The console is introduced at a price point of $599 (including two arcade controllers and a game, with games priced at $200 to $300).

[5] The Commodore Amiga (1000) provided sample playback earlier but should be considered more of a general purpose computer than a dedicated gaming machine.

become a staple of the sports genre, with known commentators often being employed to do the commentary. The most famous of these is arguably John Madden, giving play-by-play commentary in the football game series by EA Games named after him, *Madden*. [McDonald, 2004]

In the same year Creative Labs releases the next card of the *Sound Blaster* series, the *Sound Blaster Pro*. The first version offers the same capabilities as the last revision of the original Sound Blaster, but allows playback in stereo as well as stereo recording at 22,05 kHz. Furthermore a second OPL 2 is added onto the card to guarantee stereo music playback. Later revisions of the card replace the two OPL 2 chips with one fully backwards compatible OPL 3 chip. [Goehler, 2003] The Yamaha YMF262, also known as OPL3, offers more complex waveforms, 18 instead of nine voices and stereo capabilities, but no panning, forcing sounds to be one the left, right or on both outputs. The hardware apparently has a bug in some revisions, switching stereo channels around. [Microsoft, 2003]

Creative Labs also has to face strong competition that year. Advanced Gravis Technologies releases the *Gravis UltraSound* card. The card is able to play back 16-bit stereo samples at 44 kHz and can sample 8-bit stereo sound at the same frequency. [Indiana University, 2004] One of the strong points of the card is its ability to output 32 channels simultaneously, albeit at the cost of a quality drop[6]. The *Gravis UltraSound* has one drawback – due to it missing an OPL chip the card has to emulate the AdLib soundcard and *Sound Blaster* compatibility has to be provided through software. Gravis therefore creates its own platform with no initial support, but by publishing extensive programming tools and source code the company can ensure that game developers start supporting the product. [Goehler, 2003]

Almost at the same time as Creative Labs releases the *Sound Blaster Pro*, AdLib updates its product by releasing the *AdLib Gold*. The card maintains full backwards compatibility with its own predecessor while providing 12-bit quality sound at 44.1 kHz. On its release the card is the only one to offer the OPL 3 chipset – the *Sound Blaster Pro* is still in its first revision. As a specialty, the card can also be extended with an additional board that allows virtual surround sound with echo. The problem is that the card is not *Sound Blaster* compatible - a feature demanded by most games of that time to allow sample playback. Although games supporting the card follow, it does not have success and marks the end for AdLib. [Goehler, 2003]

In September 1991 the MIDI Manufacturers Association (MMA) and the Japan MIDI Standards Committee (JMSC) adopt the "General MIDI System Level 1" specification. [MIDI Manufacturers Association, 2008] The standard defines a set of 128 sounds arranged in a specific order and requires the compliant devices to be able to play 16 instruments at once and have a minimum playback of 24 voices. Additionally, one standard drum set or kit is required. [Tyler, 2006] The standard is a big help for composers, who can assure that their music will sound nearly the same on all systems supporting *General MIDI*.

The *Miles Sound System* is released in its first version in 1991. The API becomes one of the most popular libraries for sound, having appeared in more than 4200 games at the time of writing. The middleware is available on nearly all major platforms. [RAD Game Tools, 2007]

[6] „With 14 active voices, the card can play back at 44100 Hz, while at 28 active voices, the playback rate drops to 22050 Hz. The maximum 32 voices yields [sic] a playback rate of 19293 Hz. " [Indiana University, 2004]

QSound brings 3D sound to arcade games in 1991. [QSound Labs, 2007] The QSound chip is used first as an extension of the *Capcom Play System 1* (*CPS-1*), an arcade system allowing interchangeable games similar to the NeoGeo arcade system. Only a few games are released with the QSound hardware as Capcom introduces the *Capcom Play System 2* (*CPS-2*) in 1993, containing QSound by default. [Richards, 2003] *Super Street Fighter 2*, a huge arcade hit, is one of the games to use the technology.

In 1991 (1992 in North America, 1993 in Europe) Sega releases the *Sega CD* as a hardware add-on for the Sega *Genesis* in Japan. [McDonald, 2004] Although this is not the first time that a system uses CD-based media instead of cartridges, the Sega *Genesis* with the *Sega CD* is the first mainstream console to feature usage of CDs, especially outside of Japan.[7] The console also supports QSound technology. [QSound Labs, 2007]

Creative Labs reacts on the advances its competitors make with their products in 1992 and introduces its own 16-bit sound card, the *Sound Blaster 16*. By upping the card's capabilities to 16-bit sound at 44.1 kHz the sound card is able to output CD quality sound. The hardware also features an interface to connect a secondary wavetable board to it, extending its capabilities to match the ones of the *Gravis UltraSound* or the Roland *LAPC-I*. Creative Labs also licenses QSound technology to be used on the card. [QSound Labs, 2007] On the downside the card is not fully compatible to the rest of the *Sound Blaster* family as it is not able to play stereo in the *Sound Blaster Pro* compatibility mode. Nevertheless, the card is adopted fast by users and game developers and represents the standard for sound playback devices for all DOS games to come. [Goehler, 2003]

1993 sees the release of two new consoles which are both unable to win over the consumer. Panasonic releases the *3DO* console, a CD-based system[8]. [McDonald, 2004] The console uses a custom 16-bit chip "specifically designed for mixing, manipulating, and synthesizing CD quality sound." [Terlecki, 1998] Atari releases the *Jaguar*, a system advertised as a 64-bit console actually featuring two 32-bit co-processors. One of these co-processors, nicknamed "Jerry", handles audio and is able to output CD-quality sound in stereo with the number of sound channels limited by software. [Jung, 2003]

The same year *Sonic CD* is released on the *Sega CD* by Sega. "Breaking new ground in home gaming sound fidelity, Sonic CD for the Sega CD system boasts what is perhaps the first truly CD-quality soundtrack. " [McDonald, 2004]

Creative Labs soon realizes that the wavetable add-on card for the *Sound Blaster 16*, called *Wave Blaster* does not provide the quality wanted by consumers. The follow up, the *Wave Blaster II*, includes the EMU 8000 chip by E-mu Systems Inc, a company acquired by Creative Labs, providing effects like reverb and chorus. It also makes a MPU-401 MIDI interface available. The combination is also released as a single board in 1993, the *Sound Blaster AWE32*. The abbreviation stands for "Advance Wave Effects", the number refers to the maximum number of channels that can be

[7] The first console to ever have a CD drive is the FM Towns *Marty*, produced by Fujitsu in 1991. The console, based on a 386 processor, is never released outside of Japan. [Console Database, 2005] Also, the CD addon to the *TurboGrafx 16/PC Engine* is released before the *Sega CD* in the US but sells very poorly.

[8] *3DO* is first and foremost a set of specifications created by the 3DO company. Several companies (Panasonic, Sanyo, Goldstar, Creative Labs and others, some of the projects have been cancelled) build their own versions of the hardware adhering to these specifications. Panasonic is the first company to release *3DO* hardware. [Terlecki, 1998]

processed simultaneously. [Goehler, 2003] The card offers one megabyte of *General MIDI* samples and another 512 kilobytes of space to download additional samples. The card is released in two versions, the standard and the value edition. The cheaper value edition cannot be upgraded with more RAM while the higher priced version can take as much as 28 more megabytes. What's more, only the standard edition features a *Wave Blaster* connector. [Creative Labs Technical Support, 1999]

In 1994 the sixth part of *Final Fantasy*, named *Final Fantasy III* in North America, is released on the *Super Nintendo Entertainment System*. The game is a huge success, providing a game-play based on plot, events, and character. Journalist Tim Rogers even defines the game as a opera in [Rogers, 2004]: "Square, whose *Final Fantasy IV* is one of the masterpieces of the first age of narrative videogames *because* it is bold and dumb, and garish, *gay*, even, yet full of fear, slinked off into their cave in the hills of Tokyo, and set about making their next game neither a comic book nor a novel nor a home drama. Oh no, they were going to make an opera." [Rogers, 2004](emphasis by Rogers) The score of the composer Uematsu reflects the direction the game takes - "every character has his/her distinct theme, and events are scored with an operatic sense of grandeur. " [Schweitzer, 2008]

1995 - 1999

Sega and a newcomer to creating consoles, Sony, present their new systems in 1995 in North America after releasing the Sega *Saturn* and the Sony *PlayStation* in 1994 in Japan. After putting the unsuccessful Sega *32X* on the market in 1994(an add-on for the Sega *Genesis*, extending the *Genesis'* capabilities with a 32-Bit architecture[9]), Sega releases the completely new developed Sega *Saturn*. [Burkart, 2006] The machine uses two chipsets to create sound: A Yamaha FH1 24-bit digital signal processor and a Motorola 68EC000 sound processor. This hardware enables the Sega Saturn to deliver 32 PCM (pulse code modulation) as well as 8 FM channels at 44.1 kHz. [Gerritse, 2006] The console can also play Redbook Audio[10] CDs and supports QSound technology. The first game to make use of QSound is Sega's *Sega Rally Championship*. [QSound Labs, 2007]

"The industry was revolutionized in 1995 with the introduction of the Sony PlayStation."[Belinkie, 1999] The system is capable of playing back 24 sampled voices and uses three methods of sound generation. Next to MIDI[11] and MOD playback the *PlayStation* also offers the option to use Redbook Audio, providing CD sound quality. The machine is able to play "really long samples, even sampled phrases or melodies."[Tätilä, 2007]

The release of *Windows 95* separates the game developers from direct access to the hardware, instead programs communicate through drivers offered by the manufacturers of the hardware. [Weske, 2000] In September of 1995 Microsoft ships the first version of *DirectX* for Windows 95, including *DirectDraw*, *DirectSound*, and *DirectPlay*. This collection of interfaces is meant to help game developers get the most out of the hardware by providing "fast and relatively convenient access (at least in versions 5 and above) to graphics as well as audio hardware." [Weske, 2000] As long as a *DirectX* compliant driver for sound hardware exists, each game can access it through *DirectX*. [Eisler, 2006]

[9] The Sega *32X* extends the available sound channels of the Genesis by two but these as well as the extended graphic abilities are only available to special games made to work with the 32X. [Burkart, 2006]

[10] Redbook Audio define the specifications for CD sound, important for games are the required 16-bit sound quality with a sample frequency of 44.1 kHz.

[11] The built-in synthesizer of the *PlayStation* does not have any stored instruments but offers the ability to load patches. [Tätilä, 2007]

In 1996 (1997 in Europe) Nintendo reacts upon the releases by Sega and Sony and starts selling its own new console, the *Nintendo 64*. The *Nintendo 64* is not based on CD media but relies on cartridges to store game content [Belinkie, 1999], making it at the time of writing the last (non-portable) game console using cartridge-based media with a mass market audience. In an unusual move, the hardware does not contain a specialized chipset for handling audio. Instead, the CPU and the graphics chipset are responsible for audio playback. The hardware is able to output sound at CD quality, but due to the use of cartridge based media no Redbook Audio is possible, making game developers rely on sample based playback. [Tätilä, 2007]

The same year also sees the releases of games renowned for their soundtracks. *Quake* by id Software is released on the PC with a soundtrack composed by Trent Reznor, front man of the band Nine Inch Nails (the logo of the band is even featured in the game). The soundtrack is stored as Redbook Audio on the CD the game is delivered on and thus playback is possible on any CD player.

On Sony's *PlayStation* the second part of the *wipEout* racing series is released. The first part already makes full use of Redbook Audio and features (at least in some versions) tracks by electronic artists like Leftfield and the Chemical Brothers.[12] [MobyGames, 2008b]The successor, *wipEout XL/wipEout 2097*, boasts a soundtrack that "stands out as being made up of licensed songs from popular techno artists such as Underworld, Future Sound of London, The Prodigy, and The Chemical Brothers. The game also allowed players to select which song they wanted to listen to, a feature which now appears in many racing games. " [Pidkameny, 2002] Redbook Audio is used to allow CD quality playback of the songs as it is done in the first part.

On the same system, *PaRappa the Rapper* is released in 1996 in Japan and brought to North America and Europe in 1997. [MobyGames, 2008] The game is based on a toy released by Milton-Bradley, called *SIMON*.[13] "The reason why I call it the "Simon Says" genre is that this kind of game has the same structure as the "Simon Says," in which players have to perform the actions that Simon (a leader) orders to advance to the next level. Each level gets increasingly difficult. In the genre of games with sound, players also have to mimic a master's play by pressing the correct buttons at the correct moments, all in time to the music." [Takeshita, 2003] The game becomes a top-selling hit in Japan. [McDonald, 2004]

With the release of *DirectX 3* in 1996 Microsoft introduces a new part of *DirectX*, *DirectSound3D*. The release of this product creates some controversy between the manufacturers of sound cards and Microsoft. Instead of just defining an API and providing a basic 3D audio algorithm in software that could be accelerated by hardware, Microsoft opts to not allow any third party 3D algorithms by not passing any coordinates, source, or listener parameters down to the device driver. The sound card manufacturers react by creating sets of non-Microsoft API calls that pass the data required down to the device driver. Two of these sets emerge, called *A3D* and *Dev3D*. Later on both are unified into the *3Dxp* standard, created by the IA-SIG 3DWG (Interactive Audio- Special Interest Group: 3D Working Group) which at the same time negotiates with Microsoft to change its stance.[14] [Schmidt, 1997]

[12] The game even spawns a soundtrack, called „Wipeout – the Music", which differs from the in-game soundtrack. The sampler features many well-known electronic artists. [MobyGames, 2008b]

[13] Interestingly the creator of *SIMON* is Ralph H. Baer, himself creator of the first console, the Magnavox Odyssey. The game itself is based on a coin-op machine by Atari, called *Touch-Me*. [Baer, 1998]

[14] 3Dxp is further developed to "The IA-SIG's Interactive 3D Audio Rendering and Evaluation Guide – Level 1" (I3DL1). "The main purpose of *I3DL1* was to help define a consistent behavioral model for interactive 3D sound

Intel (in collaboration with Analog Devices, Creative Labs, National Semiconductor and Yamaha) develops the *Audio Codec '97* (AC'97) as a high-quality, 20-bit audio architecture for use in desktop PCs. The architecture plans the separation of the hardware into the codec and the audio controller, effectively separating analog and digital circuitry to avoid digital noise in analog sound generation. [Barish, 1998] The specification is meant "to provide system developers with a standardized specification for integrated PC audio devices." [Intel, 2008] The specification is widely adopted and used by a lot of manufacturers as an integrated audio solution.

Another important step is taken in PC audio technology in the same year. Diamond releases the *Monster Sound*, the first sound card to support 3D audio acceleration, in the case of this card accelerating Aureal's *A3D* algorithms. [Smith, 1999] *A3D* allows developers to include Doppler effects, volume dependent on the distance between source and listener and sound cones. [Hagén, 2001]

The first part of the *Grand Theft Auto* series comes out in 1997 [Kruczek, 2008]. The game follows the protagonist on his career in crime, which is mostly spent in cars stolen by the player. One of the staples of the series is already included in the first part – the soundtrack consists of a combination of different radio stations playing a variety of music. The audio is once again stored in the Redbook format and can be played on any CD player.

Konami presents the first entry, *Beatmania*, in its soon extended *Bemani* music series in the same year in Japanese arcades. [Bemanistyle, 2008] The game features an unusual controller including a turntable. The goal of the game is to hit notes and scratch the turntable at the right time.

With the shipment of *DirectX 5* in 1997 Microsoft allows third party acceleration of the *DirectSound3D* API. [Eisler, 2006] "With DirectX 5, DirectSound3D has the capability of having sound cards that use third party 3D audio algorithms accelerate DirectSound3D properly, through Microsoft-approved methods (xyz parameters now get sent from the API to the driver). Therefore, *with DirectX 5, the workarounds A3D, Dev3D and 3Dxp become unnecessary*." [Schmidt, 1997](emphasis by Schmidt) The API offered by Microsoft with *DirectX 5* is very similar to Aureal's *A3D*, in fact, many sound card manufacturers achieve *A3D* compatibility by translating calls from *A3D* to *DirectSound3D*. [Hagén, et al., 2002a]

In 1998 Nintendo releases the *Legend of Zelda: Ocarina of Time* on the *Nintendo 64* to rave reviews. "Besides boasting an amazing soundtrack, it's one of the first titles to feature music-making as part of its gameplay. " [McDonald, 2004] The game includes a playable instrument, the ocarina of time, which lends its name to the game. The instrument features a basic five-note scale, "though additional manipulation from other control buttons makes it possible for a skilled player to reproduce a complete scale. Successfully playing a melody fragment unlocks an animation which completes the melody and performs the specified action when appropriate. Not only do these musical themes flavor the experience of play, they are also reproduced in the backgrounds of several of the game's environments." [Whalen, 2004]

Also in 1998, Konami extends its range of music themed arcade games and releases *Dance Dance Revolution* in Japanese arcades. The concept of the game is simple: The player's controller consists of

---

and to help consumers and magazine reviewers differentiate between true 3D sounds and the "pseudo-3D" sound (stereo enhancement) that was popular at the time." [Miller, 1999]

a platform with eight direction arrows, corresponding to arrows shown on the screen. The player's task is to hit the right direction with the feet at the moment the arrow pointing in that direction hits the top of the screen. All of these arrows are synchronized to music – choosing a certain song sets the difficulty for the game. "Other benami [sic] games include Guitar Freaks (play a guitar to music), DrumMania (play a drum kit peripheral), and HipHopMania (scratch turntables to music)." [McDonald, 2004]

"The OpenGL-GameDev mailing list, which hosts more than its average share of developers interested in portable code, spawned in 1998 a list dedicated to discussion of a new, open audio library—tentatively named *OpenAL*. Proposals were written and posted, and several developers spent significant time coming up with a good solution. However, it quickly became apparent that between us, "good" was measured by very different metrics." [Kreimeier, 2001] As a result of these differences, the project does not go ahead and interest in it decreases for some time. [Kreimeier, 2001]

Sega begins the new round of the next generation console releases by starting to sell the Sega *Dreamcast* in Japan in the fourth quarter of 1998 and later on releasing the machine in North America[15] and Europe in 1999. [The History of Computing Project, 2005] The Sega *Dreamcast* uses a Yamaha Super Intelligent sound processor running at 45 MHz. The chipset can use two megabytes of RAM and is capable of producing sound on 64 channels at CD audio quality of 44.1 kHz with additional effects like reverb, delay and surround sound. [Tyson, 2000] As with the other consoles from the *Sega CD* on, the system supports QSound technology for 3D sound generation. [QSound Labs, 2007]

Creative introduces its own 3D sound API in early Fall 1998. The acronym *EAX* stands for Enviromental Audio eXtension. The API is an extension of *DirectSound3D*, "a primitive set of 26 presets and 3 parameters for more accurate adjustment of the Listener Parameters and 1 parameter for for[sic] the Sources." [Menshikov, 2003] These variables allow the developers to specify what type of room the player is located in and consequently set the values for the reverb algorithm. [Hagén, et al., 2002a] Creative Labs own *Sound Blaster Live!* with the new E-mu Systems Inc. EMU10K1 processor is the first card to feature *EAX*. [Hagén, et al., 2002b]

The same year Aureal releases the second version of *A3D*. The *A3D 2.0* API introduces wave tracing, a computationally intensive approach. [Hagén, et al., 2002a] "Aureal's Wave tracing algorithms analyze the geometry describing the 3D space to determine ways of wave propagation in the real-time mode, after they are reflected and passed through passive acoustic objects in the 3D environment." [Menshikov, 2003]

On the Sony *PlayStation* two notable skateboard games are released in 1999. *Trasher: Skate and Destroy* by Rockstar Games and *Tony Hawk's Pro Skater* by Neversoft both include a lineup of licensed songs by popular bands, with "Trasher: Skate and Destroy" using old-school hip-hop from acts like Run DMC, Public Enemy, Sugarhill Gang, Grandmaster Flash, Afrika Bambaataa and Eric B., and Rakim. "The competing title, Tony Hawk's Pro Skater, goes alt-punk instead,[sic] with songs by the Dead Kennedys, Goldfinger and Primus." [McDonald, 2004] The latter game spawns a long

[15] The Sega *Dreamcast* is released under heavy advertisement on September 9, 1999, thus on 9/9/99. [Robinson, 1999]

running series, still continuing today, that offers an extensive soundtrack with licensed music in each entry.

Another game with a unique sound design is also released on *PlayStation* that year. *Silent Hill* by Konami is part of the survival horror genre established by Infogrames's *Alone in the Dark* in 1992 [MobyGames, 2008a] and further developed by Capcom's *Resident Evil* in 1996. [McDonald, 2004] The genre is based on horror movie conventions. Often using dramatic camera angles, slow moving protagonists and a low amount of ammunition, keeping the player in constant state of suspense is a goal of the genre. "By combining conventions both of videogame and horror film, the designers of *Silent Hill* create an experience that is driven musically by the grotesque exaggeration of musical functions familiar from earlier videogames. The safety state/danger state binary of music which drives the motivational function of the music is shifted to correspond to the threatening, intrusive atmosphere of the city. Overall, the music in *Silent Hill* drives home the unique game play aspects that direct home its status as a classic Survival Horror title." [Whalen, 2004]

Microsoft starts shipping *DirectX 6.1* on February 3, 1999, marking the first release of the *DirectMusic* API. [Microsoft, 1999] "DirectMusic starts out by addressing the major problems of Windows' old MidiOut API, such as shaky timing and limited real-time control. It offers consistent playback of custom sound sets using an open standard, Downloadable Sounds Level 1 (DLS1). On top of that, DirectMusic opens more than one door to achieving adaptive musical scores in games." [Hays, 1998] The component of the *DirectX* API, as well as an application called DirectMusic Producer, is intended to allow game developers to create dynamic scores to accompany their games, based on MIDI output. [Hays, 1998]

In September 1999 the IA-SIG 3DWG releases the 1.0a revision of its "Interactive 3D Audio Rendering Guidelines Level 2.0" (*I3DL2*). [Interactive Audio Specialist Interest Group, 1999]. The guideline extends *I3DL1* with an environment reverberation model, an enhanced distance model, taking advantage of the reverberation cues, and occlusion as well as obstruction models for muffling effects. [Miller, 1999]

Creative Labs reacts to the introduction of *I3DL2* (that is largely based on *EAX*) by introducing *EAX 2.0* in 1999 [Miller, 1999], offering the functionality presented in the *I3DL2* paper and adding a few high-end parameters while keeping backwards compatibility to the *EAX 1.0* specification. [Hagén, et al., 2002a]

"Load the game, follow the beat, make the jump, this is neat[sic] Music a bit sad? Change the CD. This game moves to your melody. Once the game is loaded you can put your own music CD into PlayStation.[sic] The beat of the music dictates the game." [Sony Computer Entertainment Inc., 2000] The text from the advertisement page for *Vib Ribbon* is a good description of the game. The player's task is to direct Vibri, a female bunny, along a white line. The lines layout changes according to the soundtrack played in the background. Slow songs produce fewer obstacles, fast songs make the levels themselves frantic and furious. The game itself has a very simple presentation, only showing stick figures. This allows the whole title to fit in the *PlayStation's* internal memory which again allows players to change the CD containing the soundtrack of the game to any audio CD they like, creating an unlimited number of levels. The game is never released in North America, but is distributed in Japan in 1999 and Europe in 2000. [McDonald, 2004] "Interestingly, the mickey mousing sound

effects of Vibri jumping and walking over the obstacles can be turned off, turning the game into a visual performance tool." [Pichlmair, et al., 2007]

"Back in 1999 a Japanese company called Warp released what could be considered to be the very first big commercial audio game. The game's title was Real Sound: Kaze no Riglet (sometimes also Kaze no Regret) which translates into "Regrets in the Wind" or "The Riglet of the Wind". The game, that consists of 4 cd's (!), was only released in Japan for the Sega Saturn and the Dreamcast." [AudioGames.net, 2008b]

2000 - 2004

One of the most popular consoles of all time is released by Sony in the year 2000. The Sony *PlayStation 2* is the first mainstream console to include the ability to play DVDs. The system can output 32-bit stereo sound at a maximum sample rate of 48 kHz, with 48 channels available. [CNET, 2001] As a further improvement to the *PlayStation*, the *PlayStation 2* contains two megabytes of sample memory instead of 512 kilobytes. The console is able to pass through AC-3 and DTS, in 2002 *SSX Tricky* and *NHL 2002* by EA Sports use DTS to generate in-game 5.1 audio. [Shimpi, 2001]

After the release of the *Sound Blaster Live!* Aureal and Creative Labs engage in a court battle which continues until the end of 1999. Although Aureal claims victory, the financial assets of the company are used up and the company has to file for bankruptcy. Aureal is bought by its rival Creative Labs which takes over the *A3D* technology, by then in its third version. The acquisition is performed on September 21, 2000. [ALive!, 2003]

Sega and Nintendo both introduce games featuring voice recognition in 2000. [McDonald, 2004] *Seaman* by Sega is released for the Sega *Dreamcast* and revolves around a creature called Seaman. After the creature hatches, the player has to take care of it and talk to the Seaman. While the Seaman grows to adulthood it begins building up a vocabulary for conversations with the player. The creature will start being more active in conversations, ask questions and even insult the player depending on its mood, using a vocabulary with over 10,000 words. [Cordeira, 2008]

Nintendo releases a similar game, called *Hey You, Pikachu!* for the *Nintendo 64*. The game can be categorized as "friendship simulation". [Provo, 2000] The player takes responsibility for a Pikachu, the best known character from Nintendo's franchise *Pokémon*. Interaction with Pikachu is done through the use of a microphone included with the game. "Pikachu's understanding of the English vocabulary is limited to approximately 200 words and phrases, but the game's voice recognition is fairly solid – it even allows for the sloppy speech patterns of younger children and toddlers." [Provo, 2000]

Sega releases *Samba De Amigo* on the *Dreamcast* in North America and Europe in 2000, after releasing an arcade and a Dreamcast version in Japan in 1999. The game features a special peripheral, consisting of two maracas and a floor sensor to locate the maracas' positions while playing. The game requires the players to shake the maracas the right way in the right place at the right time. The music used in the game is mostly based on Latin music, the non-Japanese version even includes cover versions of Ricky Martin songs. [Gerstmann, 2000]

After *DirectX 7* failed to change much in regards to how music and sound are handled by *DirectX*[16], Microsoft introduces major changes with *DirectX 8* in November 2000. [Microsoft, 2000] DirectSound and DirectMusic are consolidated into one interface and DirectShow is enhanced, providing real-time compositing and editing of audio/video timelines by the Editing Services API and supporting Windows Media Audio and Video files. DirectSound3D receives a major improvement, upping its abilities to the ones laid out in *I3DL2*.[Hagén, 2001]

Since 2000 video game music is allowed to compete in the Grammy Awards. The 42nd Annual Grammy Awards ceremony includes three categories accepting video games as nominees: "Best Soundtrack Album for Motion Picture, Television, or Other Visual Media; Best Song for a Motion Picture, Television, or Other Visual Media; and Best Instrumental Composition for Motion Picture, Television, or Other Visual Media." [Marks, 2000] No video game music has yet been nominated for one of the three categories. Chance Thomas explains in an interview: "In 1999 I wrote a proposal that was successful in getting three new categories added to include game music. That marked the first time in history that game music was eligible to compete for a Grammy Award. But the road since then has been sluggish. I think publishers have been indifferent about making submissions, while audio professionals have been intimidated by the competition. The result is, very few game soundtracks have actually even been submitted for Grammy consideration." [Brightman, 2007] Submitting soundtracks is also made harder by requirements asking for the game score to be commercially available as a separate music CD, stored in Redbook Audio, or stored as an "enhanced" CD. [Marks, 2000]

Development of *OpenAL* picks up in 2000 and in June 2000 the Version 1.0 Draft Edition is released. Aureal expresses interest in the API in 1999, but neither the company nor the *OpenAL* mailing list can report significant progress. This situation changes in late 1999 with the involvement of Loki, a developer tasked with porting *Heretic 2* and *Heavy Gear* to Linux at that time. Creative Labs expresses interest in the project and at the Game Developer Conference in March 2000 both companies announce their initiative leading to the June 2000 paper. [Kreimeier, 2001] At the time of writing, version 1.1 has been released and the following platforms are supported by at least version 1.0: OS 8/9, OS X, Linux (OSS, ALSA), BSD, Solaris, IRIX, Windows (MMSYSTEM, DirectSound, DirectSound3D, NVIDIA *nForce*, Creative *Audigy 1/2/4*, Creative *X-Fi*), Microsoft *Xbox*, and Microsoft *Xbox 360*. [OpenAL, 2007]

Rez is created by United Game Artists in 2001 and published by Sega on the *Dreamcast*, with a later version published on the *PlayStation 2*. [MobyGames, 2008] The game itself is a rail-shooter, forcing the player on a certain path and only granting limited control to the user. "In the game, players at first move around a cursor to lock on an enemy as a target. Players are allowed to lock on a few enemies at once. On the screen, when players have, for instance, three enemies locked on, they simultaneously see three cursors on three enemies. Then when players press a button to destroy them all together, they hear a sequence of the sound effects. By constantly shooting them and hearing the sound, players feel like they are putting the sound on the techno background music." [Takeshita, 2003]

[16] The major changes from *DirectX 6.1* to *DirectX 7* are a new voice manager and a new software 3D sound engine. [Hagén, 2001]

A similar looking, but differently functioning game is released by American developer Harmonix Music Systems in the same year. [McDonald, 2004] *FreQuency* offers game play similar to *Tempest* from the eighties. The player flies down an octagonal pipe with each wall representing a part of the song played in the background. Each part is again split in to a series of jewels, located on the left, middle or right side of the wall. These jewels represent notes the player has to hit at the right moment. "If you successfully complete a section of a wall without any mistakes, the wall will clear out and start playing automatically, letting you move on to another wall." [Davis, 2001] The game features a soundtrack including BT, Crystal Method, Orbital, DJ Q-Bert, Powerman 5000 and Paul Oakenfold. The successor, *Amplitude*, replaces the pipe in favor of a planar playing field with multiple lanes and introduces multiplayer. [Davis, 2003]

Gitaroo Man by Koei is released on the *PlayStation 2* in 2001 in Japan and brought to North America in 2002. The game play of the title alternates between two different modes, switching between game play based on timing and a mechanic requiring the player to follow a line on the screen with movements of the analog stick. "Gitaroo-Man is arguably the most original and inventive rhythm game since the advent of Dance Dance Revolution or the original Parappa the Rapper, and it's easily one of the best rhythm games for the PlayStation 2." [Davis, 2002]

With Microsoft a new company enters the console business. The first product, called Microsoft *Xbox*, resembles in large parts a PC. The sound chip of the *Xbox*, called MCP (Media & Communications Processor) offers Dolby Digital encoding support and is designed "to bring 5.1 channel gaming to the mainstream market". [Shimpi, 2001] The processor is *I3DL2* compliant with abilities to output 64 voices and features 64 megabytes of unified memory and 200 MHz bandwidth to the CPU. [McDonald, 2004] Another important feature is custom soundtracks for games (as long as the game developers allow it), stored on the internal hard disk. [Microsoft, 2007]

Nintendo is the last major producer of consoles to switch to disk based media. In 2001 the Nintendo *GameCube* is released [McDonald, 2004], featuring a proprietary 8cm optical disk, allowing storage of up to 1.5 gigabyte. The console includes a custom Macronix digital sound processor (DSP) connected to 16 megabytes of RAM. The chip is able to produce up to 64 simultaneous channels at 48 kHz and supports stereo, Pro Logic and Dolby Pro Logic 2. [Shimpi, 2001]

The same year the handheld line by Nintendo, the *GameBoy*, receives an update too. Nintendo starts offering the *GameBoy Advance* in 2001. [Thorsberg, 2001] Unlike its predecessor, the *GameBoy Color* [Strietelmeier, 1998], the *GameBoy* Advance features sound hardware much advanced in comparison to the original *GameBoy* system. The new machine is able to output samples on two channels next to the four standard channels of the *GameBoy*. The output is played in mono over the speaker or in stereo when headphones are used. [Strickland, 2002]

With the release of the *Audigy* cards Creative Labs introduces *EAX Advanced HD* in 2001. *EAX Advanced HD* allows 64 simultaneous voices. Environments can be positioned and morphing allows smooth transitions between them. Further, sound reflections can be calculated and sound can be filtered to simulate wide open spaces better. Creative Labs has since released two updates to *EAX Advanced HD*, upping the version number to 5.0. The newer versions support multiple reverb environments, better recreation of effects close to the player, a more flexible creation of hardware effects, inclusion of the player's voice in the environment, and support for 128 simultaneous 3D voices. [ALive!, 2001] [Hagén, et al., 2002a] [Creative Labs, 2008]

Grand Theft Auto: Vice City is released by Rockstar North, formerly DMA Design, in 2002 on the *Playstation 2* with PC and *Xbox* versions released later on. The game is the successor to the smash hit *Grand Theft Auto III*, taking its game play and transferring it to a city similar to Miami in the eighties. The game keeps the series' presentation of sound through radio stations, using a large collection of eighties music to convey the feeling of that era. The soundtrack even sees a standalone release. The developers also use famous actors like Ray Liotta, Dennis Hopper and Burt Reynolds for voice work. The same concept of licensed tracks to immerse the player in the game world and quality voiceovers by famous actors is used again for *Grand Theft Auto: San Andreas*. [Chan, 2007]

In 2003 Konami and Harmonix Music Systems release *Karaoke Revolution*. The game presents the player with a song that has to be sung along with. *Karaoke Revolution* then rates players on how well they can sing. Although the songs scroll along the screen like they would in a Karaoke machine players can sing whatever they want, as long as they stay within the games limits of pitch and timing for that song. [McDonald, 2004] Numerous versions of the title, including different songs, are released.

At the end of that year Creative Labs announces the acquisition of Scipher Sensaura, one of the competitors left in the field of 3D audio technology. [The Inquirer, 2003] At that point Sensaura provides licenses for about 60% of the PC sound chip manufacturers as well as providing 3D sound technology for the Xbox. [Gamasutra, 2001] The company also offers GameCODA, a cross platform solution. [Creative Technology, 2004] With this purchase Creative Labs further reinforces its position as market leader in 3D sound technology.

In 2004 Sony Computer Entertainment Europe releases a game very similar to *Karaoke Revolution*. *SingStar* becomes a smash hit in Europe, spawning numerous special editions, many localized for specific countries, such as *SingStar Deutsch Rock-Pop* and *SingStar Die Toten Hosen* for Germany or *SingStar Svenska Hits Schlager* for Sweden. [Sony Computer Entertainment Inc., 2007] The game itself uses the same game mechanic as *Karaoke Revolution*, whilst changing the presentation of the songs. Instead of using 3D backgrounds and models to represent singers, *SingStar* plays the music video of the song in the background, also supporting the *EyeToy* accessory to put images of the players on the screen. The lyrics of the song are displayed over the video with bars indicating the location of the player in the song and the pitch the passage has to be sung at. [Bramwell, 2004]

The same year Nintendo, together with Namco, releases *Donkey Konga* for the Nintendo *GameCube*. The game is sold including a peripheral for the console in the shape of bongo drums. The DK Bongo controller supports three actions: hitting the left drum, hitting the right drum, and a detection of sound through a small microphone. The game-play of *Donkey Konga* consists of hitting drums with symbols on them at the right time. These drums run over the screen from right to left and can contain a yellow (left bongo), red (right bongo), pink (both bongos), or blue (clap) symbol. *Donkey Konga* allows up to four simultaneous players (with additional bongos or through the use of standard controllers). The game's soundtrack features a wide selection of songs, all of them cover versions. The game sees two sequels, the latter only being released in Japan. [Davis, 2004]

The *Taiko Master* series, published by Namco, can be seen as template for *Donkey Konga*. The series is very popular in Japan, spawning numerous arcade and console versions. The game named *Taiko Drum Masters* is published for the first time outside of Japan on the *PlayStation 2* in 2004. Included

with the game is a plastic replica of a Taiko drum and two drumsticks. The game itself has very similar game play to *Donkey Konga* and supports up to two players. [Calvert, 2004]

In November 2004 Nintendo releases the Nintendo *DS* in North America,with Japan and Europe following. [Duryee, 2004] The handheld console features two screens in vertical alignment, with the lower one being a touch screen. The audio hardware is again improved from the *GameBoy Advance*, offering 16 sound channels. The *DS* further includes a built-in microphone and is equipped with two speakers, allowing stereo sound without the use of headphones. [GBATEK, 2007]

Late in 2004 the Sony *PlayStation Portable* is available in Japan, with the rest of the world receiving the handheld console in 2005. [Williams, 2004] The portable uses Universal Media Discs (UMD) as storage format, allowing the release of other media (audio and movies) alongside games. Owners can also store movies and songs on small flash cards called Memory Sticks PRO Duo that can be read and written by the console. The console is able to output stereo sound over its speakers or connected headphones. [CNET, 2008]

Also in 2004, Intel releases the "High Definition Audio Specification" as a replacement for the AC'97 codec. The High Definition Audio architecture is not backwards compatible to AC'97. [Intel, 2004] "Intel HD Audio hardware is capable of delivering the support and sound quality for up to eight channels at 192 kHz/32-bit quality, while the AC'97 specification can only support six channels at 48 kHz/20-bit. In addition, Intel HD Audio is architected to prevent the occasional glitches or pops that other audio solutions can have by providing dedicated system bandwidth for critical audio functions." [Intel, 2007] The new architecture allows 7.1 channel output.

2005 - Present

As the first of the three big console manufactures Microsoft releases its new machine to the North American market in November 2005, the Microsoft *Xbox 360*. [Morris, 2005] The console includes 32-bit audio processing with over 256 audio channels and 320 independent decompression channels. Supporting 16-bit audio at 48 kHz the *Xbox 360* can output multiple channels for surround sound. [Microsoft, 2007] The console also keeps the custom soundtracks in games from the original *Xbox*, but again the support of that feature is up to game developers. [Ransom-Wiley, 2005]

Late in 2005 Harmonix Music Systems introduces *Guitar Hero* on the *PlayStation 2*. The developer is approached by peripheral producer Red Octane to create a title for a product produced by the company, a guitar. [Simons, 2007] The result lends from previous titles by the developer, keeping the concept of jewels from *FreQuency* and *Amplitude* and the 3D backgrounds as well as the concept of the "crowd meter" (called "rock meter" in *Guitar Hero*) from *Karaoke Revolution*. The goal for the player is to hit all jewels at the right time by holding the same colored button on the neck of the guitar controller and strumming the flipper located where the strings would be on a real guitar. The player can try to hit special jewels to fill up the "Star Power Meter" which in turn allows them to activate the "Star Power" by holding the guitar peripheral vertically. This way the player can attain an even higher score. The game also includes a career mode to help new players get accustomed as well as a multiplayer mode for two players. [Gerstmann, 2005b] *Guitar Hero* becomes a runaway hit, with Harmonix Music Systems developing two sequels (one also appearing on the *Xbox 360*) before handing over the franchise to Neversoft. [Gamespot.com, 2008]

Released as part of the Bit Generations series, *Sound Voyager* is published by Nintendo for the *GameBoy Advance* in 2006. The title, officially only available in Japan, features seven mini games, all

of which are audio games: *Sound Catcher*, *Sound Drive*, *Sound Chase*, *Sound Cock*, *Sound Slalom*, *Sound Cannon,* and *Sound Picker*. [AudioGames.net, 2008a]

The DK Bongo Controller introduced with the *Donkey Konga* in the previous year gets a new use with Nintendo publishing *Donkey Kong Jungle Beat* in 2005 (released in Japan in 2004). The game itself is a platformer. The player controls the monkey "Donkey Kong" by hitting the right drum to move the character to the right and the left bongo to move the character to the left. Hitting both drums at the same time initiates a jump, clapping emits sound waves that can be used to acquire bananas placed within the levels; it also makes the character interact with the environment, if possible. The game-play is broken up by boss fights and some other challenges within the levels, more or less mini games. [Gerstmann, 2005a]

The same year Nintendo releases the localized version of *Osu! Tatakae! Ouendan*, developed by iNis (the Japanese Version is released in 2005) for the Nintendo *DS*. Originally featuring a squad of male cheerleaders helping people master their problems by cheering them on, the localized version, called *Elite Beat Agents* introduces some changes. The male cheerleaders are replaced by agents of a fictional government organization and the stories, as well as the songs, are adapted to the different audience. "Think of the whole experience as a cross between *Charlie's Angels*, Saturday morning anime, and *Mama Mia!*-esque musical theater." [Navarro, 2006] The game play stays the same between the versions. While the story evolves on the upper screen of the Nintendo *DS*, the player has to tap small circles in the beat of the song playing in the background. Occasionally a ball must be dragged along a predefined path or a wheel must be spun. [Navarro, 2006]

Also in 2006, Nintendo brings *Electroplankton* to North America and Europe (the Japanese Version is released on the Nintendo *DS* in 2005). The title can be better classified as a mixture of a toy and an instrument than a game. "Imagine a set of synthesizers that you can manipulate through the familiar microphone and touch screen interfaces of the Nintendo DS, and you're pretty close. Electroplankton is comprised of 10 such musical toys, and a playful visual style is employed to give the impression that each takes place in some sort of bizarre petri dish--or perhaps a very musical aquarium--filled with different species of plankton that can produce sound and light when you interact with them." [Davis, 2006]

Sony introduces its third console to the market in November 2006. The *PlayStation 3* is released in a range of configurations with different hard drive sizes and different levels of backwards compatibility to the previous *PlayStations*. Sound generation is handled by the Cell architecture within the machine. [Kotaku, 2005] The console supports a variety of output formats, including Dolby 5.1, DTS and LPCM. [Dobson, 2006] It is also able to output 7.1 surround sound.

Nintendo follows up the *GameCube* with the Nintendo *Wii* also in November 2006. As with the *GameCube*, the *Wii* supports Dolby Pro Logic II for surround sound. [McDonough, 2006] "But Nintendo does something different with sound, putting the Wii Remote to work in a new way by providing built-in speakers within the controller." [Dobson, 2006]

In the same year Nintendo releases *Rhythm Tengoku* for the *GameBoy Advance* in Japan. The title consists of a number of very short games that are based on rhythm. If five of these mini games are completed, a remix combining all of them with a new musical twist is unlocked. "Completing remixes unlocks new ladders of games, which offer either new, more difficult twists on earlier mini-games or new rhythm concepts. None of them require anything more than timed button-presses from the

player, and all of them, from the clap-along singing to bunny-marching to the extra-terrestrial baseball, are wonderfully bonkers." [MacDonald, 2006]

In March 2007 EA Games releases *Def Jam: Icon* for the *Xbox 360* and the *PlayStation 3*. The fighting game series *Def Jam* uses Hip Hop as an underlying theme – the first part, *Def Jam: Vendetta*, as well as *Def Jam: Fight for New York*, employ music created by the artists of the Hip Hop label Def Jam and use the rappers themselves as fighting characters. Up to *Def Jam: Icon*, the Hip Hop music scene is only used as a theme – the game play itself is akin to the one featured in wrestling games. The third installment of the series introduces music as a game-play element. While the two rappers are fighting each other, the music playing as score is affecting the environment the brawl takes place in. This can be a cosmetic effect, like the spinning hubcaps moving to the beat, but it can also affect the characters themselves by triggering explosions or similar events able to hurt the players. Both protagonists can try to change the music through a special move. If their own song is playing, characters get stronger and by repeating the special move they rewind their song and trigger explosions in the scenery. [Gerstman, 2007]

The *Guitar Hero* franchise sees its official third part in October 2007. The game is released for every major console as well as for the PC and Mac OS X. *Guitar Hero III* features the basic game play of previous renditions with the introduction of boss battles and online play on all platforms, with all but the Nintendo *Wii* version allowing the download of additional content. [Pfister, 2007]

Ubisoft publishes *Jam Sessions* in the last quarter of 2007, based on the Japanese game *Sing & Play Guitar DS M-06*. [Chester, 2007] *Jam Sessions* is less a game than a guitar simulator, turning the Nintendo *DS* into an acoustic guitar. The game presents a string on the lower screen that can be strummed by moving the stylus up and down. The upper screen presents the player with a choice of eight chords with another eight available by pressing the shoulder button of the *DS*. The chords can be selected out of 120 stored. Further, effects can be applied to the guitar. *Jam Sessions* includes chord progressions for a number of songs, varying between the regions (the European version includes 41 songs while the North American version offers 20 songs). The game makes use of the *DS's* microphone if headphone output is selected, but unlike the guitar sound itself, voices cannot be recorded. [Thomas, 2007]

Harmonix Music Systems releases *Phase* in November of 2007. The game can be seen as the final part of the "modulation trilogy", also containing the games *FreQuency* and *Amplitude*. [Graft, 2007] Unlike the aforementioned games that have been published on the *PlayStation 2*, *Phase* is released for Apple's *IPod*.[17] The game features a similar, but simpler game play than *FreQuency* and *Amplitude*, reducing the action to three static lanes. The jewels still show up in three possible positions, left, right and center, but a new element is introduced: Sweeps have to be followed by rotating the *IPod's* click wheel. The real innovation of the game is the use of all songs available on the players *IPod*, with some restrictions. "When you download it, a new Playlist will appear in your iTunes window on your PC, and you have to individually drag tracks over to it. ITunes will then analyze and generate gameplay[sic] data for each track individually, which takes five to ten seconds per song (you'll see the progress bar)." [Kohler, 2007]

[17] "Phase is playable on the 5th generation iPod®, the new iPod® Nano, and the iPod® Classic!" [Harmonix Music Systems, 2007b]

Rock Band, Harmonix Music Systems' follow-up to *Guitar Hero* is released in late 2007. [Sinclair, 2007] The game extends the idea of *Guitar Hero* by introducing three new instruments: The bass guitar, the drum kit, and the microphone. Game play for the guitar peripherals remains largely unchanged, but with the introduction of new instruments the game also contains new game modes. Using the microphone is very similar to game play in games like *Karaoke Revolution* or *SingStar*, with the same measures of success. The drum set consists of four pads and a kick pedal. As with the guitars, the game play for the drums also requires the player to hit the right pad or press the kick pedal corresponding to the jewels on the screen. "There's really no reference point for the drums portion of the game except for, well, real drums. You hit the pads in time as you would with a realistic drum kit, and on expert, the game practically maps out each song's drum part note for note. Make no mistake: When you are playing on expert, you are playing the drums." [Navarro, 2007] *Rock Band* puts a strong emphasis on multiplayer gaming, with the versions for the *Xbox 360* and the *PlayStation 3* also allowing online play, as well as the download of new songs for the game. [Navarro, 2007] The game also appears on the PlayStation 2 with a reduced feature set. [Navarro, 2007]

Console	**Sound Hardware**
Magnavox *Odyssey*	None
Atari *2600*	Two audio channels
Atari *5200*	POKEY - four semi-independent audio channels
Commodore *C64*	MOS Technology 6581 Sound Interface Device (SID) – three independent channels
Nintendo *Nintendo Entertainment System*	Four synthesizer voices and one sample channel
Sega *Master System*	Three channels for music and one for noise generation (later version additionally feature the Yamaha YM2413 FM sound-chip)
Nintendo *GameBoy*	Four synthesizer voices with the ability to output stereo sound
NEC *TurboGrafx-16*	Six stereo channels
Sega *Genesis/Mega Drive*	Z80 CPU and the Yamaha YM2413 – six stereo channels
SNK *NeoGeo*	Yamaha YM2610 sound-chip - 15 channels in stereo
Nintendo *Super Nintendo Entertainment System*	Sony SPC700 sound-chip – 8 channels for stereo sample playback
Sega *Sega CD*	Adds CD playback capabilities to the Sega *Genesis,* QSound technology
Panasonic *3DO*	custom 16-bit chip for CD quality playback
Atari *Jaguar*	CD-quality sound in stereo with the number of sound channels limited by software
Sega *32X*	Two additional sound channels for the Sega *Genesis*
Sega *Saturn*	Yamaha FH1 24-bit digital signal processor and a Motorola 68EC000 sound processor – 32 sample channels and eight synthesizer channels at 44.1 kHz, QSound technology
Sony *PlayStation*	24 sampled voices and three methods of sound generation
Nintendo *N64*	CPU and the graphics chipset are responsible for audio playback (CD quality)
Sega *DreamCast*	Yamaha Super Intelligent sound processor running at 45 MHz – 64 channels at CD audio quality, QSound technology
Sony *PlayStation 2*	32-bit stereo sound at a maximum sample rate of 48 kHz - 48 channels available, DVD playback, AC-3 and DTS
Microsoft *Xbox*	Media & Communications Processor – 64 channels, I3DL2 compliant
Nintendo *GameCube*	Macronix digital sound processor – 64 channels, 48 kHz, Dolby Pro Logic, Dolby Pro Logic 2
Nintendo *GameBoy Advance*	Four synthesizer voices and two sample channels with the ability to output stereo sound
Nintendo *DS*	16 sound channels
Sony *PlayStation Portable*	stereo sound

Microsoft *Xbox 360*	32-bit audio processing, 256 audio channels and 320 independent decompression channels, surround sound capable
Sony *PlayStation 3*	Cell architecture - Dolby 5.1, DTS and LPCM
Nintendo *Wii*	Similar to the Nintendo *GameCube*

Figure 1: The sound hardware of the consoles presented in the chapter "History of Sound in Video Games"

Creating Electronic Audio

What is Sound?

"Sound is created by vibrations, such as those produced by a guitar string, vocal cords, or a speaker cone. These vibrations move the air molecules near them, forcing molecules together, and as a result raising the air pressure slightly. The air molecules that are under pressure then push on the air molecules surrounding them, which push on the next set of air molecules, and so forth, causing a wave of high pressure to move through the air; as high pressure waves move through the air, they leave low pressure areas behind them. When these pressure lows and highs – or waves – reach us, they vibrate the receptors in our ears, and we hear the vibrations as sound." [Adobe, 2003]

Sound waves can be characterized by three properties: The amplitude, the frequency and the wavelength of a wave. The amplitude is the change in pressure the wave produces – points on the wave with a positive amplitude mark increased pressure, negative amplitude indicates lower pressure. [Janus, 2006] The higher the amplitude of wave, the louder it can be heard. "Loudness is measured in decibels, (dB)." [Collins, 2007b]

Frequency is the number of times a wave repeats, measured in cycles per second. The unit measurement is Hertz (Hz), with one Hertz indicating one cycle per second. [Adobe, 2003] High frequencies result in a high pitched sound, low frequencies in a low pitched sound. The wavelength, a measure of the length of one full wave cycle, and the frequency are inversely proportional to each other – a small wavelength gives a high frequency and vice versa. [Collins, 2007b] An important frequency is 440Hz. This tone, also called A440, is the A above the middle C and marks the standard pitch that all instruments are set to. [Heaton, 2007]

When talking about frequency, harmonics should also be mentioned. For any given frequency f, one can create sequence of frequencies f, 2f, 3f, 4f, … nf …etc, with n being a whole number and f being the fundamental frequency. This series of frequencies is called harmonic series and the single frequencies inside the series are called harmonics. [Wolfe, 2007] These harmonics can be used to create specific waveforms.

Waveforms

Sine waves are the most basic waves. They are also called "pure" as they do not contain any harmonics. [Collins, 2007b] Sine waves can be used to support other waveforms (adding additional deep or high pitched frequencies). Figure 2 shows a simple sine wave.

Figure 2: A sine wave, the most basic sound wave. Retrieved from [Preve, 2007]

The opposite of the sine wave is the sawtooth wave, also called ramp wave. A perfect sawtooth wave consists of the sum of the fundamental frequency with all the harmonics of that frequency. The resulting pattern resembles the teeth of a saw, hence the name of the wave. The wave generates an extremely bright and buzzy sound. [Preve, 2007] Figure 3 illustrates a sawtooth wave.

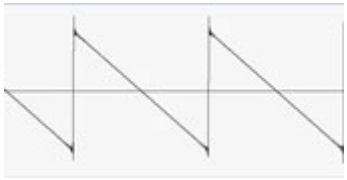

Figure 3: A saw tooth wave, a combination of a sine wave of a certain frequency summed with all harmonics of that frequency. Retrieved from [Preve, 2007]

Square waves consist of all odd numbered harmonics of a fundamental frequency, the volume linearly descending with each added harmonic. [Preve, 2007] "Square waves are often referred to as "hollow" sounding." [Collins, 2007b] A square wave is shown in figure 4.

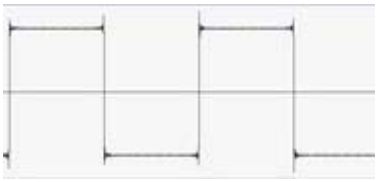

Figure 4: A square wave, the sum of all odd harmonics with the fundamental frequency, with the volume linearly reduced for each successive harmonic. Retrieved from [Preve, 2007]

Triangle waves equal square waves as they also only contain all odd numbered harmonics of a fundamental sine wave. The difference lies in the reduction of the volume – instead of reducing the volume in a linear fashion, the reduction is done exponentially, resulting in a much faster descent of volume. "In plain English, this means that the triangle sounds like a more muted – or duller – square wave." [Preve, 2007] Figure 5 shows the representation of a triangle wave.

Figure 5: Like the square wave, the triangle wave is the sum of all odd harmonics with their fundamental frequency. Instead of reducing the volume in a linear fashion, the volume is reduced exponentially. Retrieved from [Preve, 2007]

Pulse waves, also called rectangle waves, are a more general case of a square wave. [Preve, 2007] The difference lies in a variable duty cycle. The duty cycle represents the ratio between the time the signal strength is at its maximum and the time it's at its minimum (in this case, there are no values between the maximum and the minimum). While the duty cycle of a square wave equals 1 (same time at maximum and minimum), a general pulse wave can have any kind of ratio. [Collins, 2007b]

"White Noise is created by generating every possible frequency at the same volume." [Preve, 2007] Two additional colors of noise exist, pink noise and blue noise. Pink noise filters out higher frequencies slightly while blue noise does the same with lower frequencies. [Preve, 2007] [Collins, 2007b]

Digital Sound

The main difference between analog sound as it was discussed up until now and digital sound is that digital sound is not stored as continuous waveform but as a series of discrete samples. In order to transform the sound wave into this series of samples, the wave needs to be quantized. The bit depth and the sample rate play a big role in how good the quality of the quantization will be. [Collins, 2007a] Bit depth determines the amplitude resolution. An amplitude resolution of 8-bit means that it is possible to store 256 different levels of amplitude, a very low number. Using a low bit rate introduces noise, as the amplitude of a data point sampled from the wave cannot be stored at its exact value. Instead, it has to be matched to the nearest available value.

This problem can be tackled by using a higher resolution. “CD-quality sound is 16-bit, which means that each sample has 65,536 possible amplitude values.” [Adobe, 2003] DVDs offer even higher quality by allowing audio with 20-bit depth(1,048,576 values) or even 24-bit depth(16,777,216 values). [Adobe, 2003]

The second parameter affecting audio quality is the sample rate. It specifies the number of samples taken from an audio wave per second and is measured in Hertz. “The Nyquist-Shannon sampling theorem states that a continuous-time band-limited signal *x(t)* with maximum frequency f_{max} can be recovered or reconstructed exactly from its discrete-time samples *x[n]* if the samples are taken at a sampling rate $f_s > 2f_{max}$.” [Howard, 2007] The human ear can hear sounds between frequencies of 15 - 20Hz and 20,000 Hz.[Errede, 2007] In order to capture the full spectrum of frequencies audible to human beings the sampling rate must thus be above 40,000 Hz.

The CD audio specification calls for a rate of 44.1 kHz and thereby satisfies this condition. The DVD-Audio standard defines even higher rates, allowing 48 kHz or 96 kHz sampling. [Motion Picture Experts Group, 2007]

Creating Digital Sound

Until the beginning of the nineties, sound synthesis is the major, and very often the only, method to create sound and music in games. Starting with simple tone generation, the technology soon goes on to subtractive synthesis, additive synthesis, frequency modulation synthesis and peaks with wavetable synthesis. [Collins, 2007b]

The first kind of synthesis used to create sound is still rooted in analogue technology, using voltage controlled oscillators, filters and amplifiers, a technology invented by Robert Moog in the late sixties. Subtractive synthesis is based on the idea of producing a waveform rich in harmonics and then using lowpass filters to thin out the high frequencies. The voltage controlled amplifier is used to shape the sound wave into its final form with the control of an envelope generator. [Winfield, 2003]

The most common envelope generator is the ADSR envelope generator. The acronym stands for the four phases of the generators output: Attack, decay, sustain and release. The generator takes five parameters. The level parameter sets the output value reached after the attack phase. The next two parameters define the attack and decay durations, the time needed to reach the level value and the time required to fall back down to the value of the fourth parameter, sustain, which is defined as a fraction of the level parameter. The final parameter is the release duration, specifying how long the drop to zero takes. The length of the sustain phase is calculated by subtracting the attack, decay and release values from the total length of the envelope. [Puckette, 2006] This kind of envelope is still in

use today. As an example, Force Feedback technology uses the basic waveforms and ADSR envelopes to define the shape of its effects.

Additive synthesis is the counterpart to subtractive synthesis. Based on Joseph Fourier's discovery of the Fourier series and the fact, that this mathematical expression can represent any periodic signal (provided the signal only contains a finite number of frequencies) the synthesis starts out with a simple wave form like a sine wave. [Puckette, 2006] The final sound is generated by adding more (simple) wave forms to create the complex wave form asked for. Although theoretically able to reproduce any sound, the creation of complex sounds requires a lot of processing. [The Sonic Spot, 2007]

Frequency Modulation (FM) synthesis, published the first time by John Chowning in 1973, is the dominating sound creation method until the beginning of the nineties. [Smith, 2007] It is also the first commercial digital sound synthesis method in the world. [Horner, 1999] "FM synthesis techniques generally use one periodic signal (the modulator) to modulate the frequency of another signal (the carrier). If the modulating signal is in the audible range, then the result will be a significant change in the timbre of the carrier signal. Each FM voice requires a minimum of two signal generators. These generators are commonly referred to as "operators", and different FM synthesis implementations have varying degrees of control over the operator parameters." [Heckroth, 1995]

Wavetable synthesis is the term for two different methods of creating sound. At the end of the seventies, Wolfgang Palmer creates a digital synthesizer that uses "wavetables" with each of these tables storing 64 different waveforms. These waveforms can be accessed through a continuously variable control, implemented in software, allowing other elements of the synthesizer to step through the waveforms. [Cullen, 2006]

The second method called wavetable synthesis, also called PCM synthesis, found widespread use among sound card manufacturers in the nineties. The idea is that instead of recreating the sound of an instrument, for example a grand piano, within the synthesizer, one could also record each note of the instrument, store it, and play it back on demand. This theoretical approach is constrained by the storage size available on a sound card. To circumvent this restriction, not all notes of an instrument are stored, instead, to get the full spectrum of an instrument, certain instrument sounds are stored and then varied in their pitch (by speeding up or slowing down the playback) to achieve the targeted tone. [Winfield, 2003] The problem is finding the right ratio between sounds stored and sounds recreated as shifting the pitch too far from its original state makes the resulting sound tinny or dull, depending on whether the pitch is increased or lowered.

A special case of the wavetable/PCM synthesis is the Linear Arithmetic synthesis. "L/A (Linear Arithmetic) synthesis is Roland's trademarked term for their own wavetable/sample playback principle. A sample of the beginning of a note (called the attack) is spliced onto a simple oscillator waveform. The resulting output goes to a conventional chain of enveloped filters and amplifiers. In acoustic instruments, the attack is usually the most complex part of the sound, and this approach provides an easy way to capture that complexity." [Seum-Lim, 1992]

The main problem with wavetable/PCM synthesis is the varying quality of the sets between the different sound card manufacturers, resulting in a large difference between playbacks of the same song.

From the nineties on sample playback replaces sound synthesizing more and more. The first console to really make use of sample playback (previous consoles only included it as a supplement to other methods of synthesis, like the Nintendo Entertainment System) is the Super Nintendo Entertainment System, offering up to eight simultaneous channels for samples. [Tätilä, 2007]

Storing Digital Audio

Up to the beginning of the eighties, creating sound for games means programming the sound hardware directly. The first big step forward is taken in 1983 with the publication of the official MIDI 1.0 Detailed Specification. The acronym MIDI stands for Musical Instrument Digital Interface. [Akins, 2004] MIDI data does not contain any sound itself. Instead, commands on how to play music are stored which are then executed by the sound devices. As no audio data has to be stored, MIDI files are comparatively small, an important feat at a time where space was expensive and scarce.

Unfortunately, MIDI also has a downside. As already mentioned, no sound data is stored. This in turn means, that the composer does not have any control over how the piece composed will actually sound. The definition on how a certain instrument sounds is up to the manufacturer of the sound device, until the introduction of the *General MIDI* standard in 1991 it is not even guaranteed, which instrument will actually get used. “One of the main problems that had caused headaches for MIDI users, was the inconsistent way that various instrument sounds were numerically arranged between different brands of MIDI synthesizers. On one synthesizer, instrument sound #1 might be piano, while on another it could be some weird sound like "aeroglide." You never knew. It was a salad toss- the instrument and drum sounds were numbered and arranged in completely different orders, from one brand to the next!“ [Tyler, 2006]

The *General MIDI* standard ensures consistency of instrument types, at least for the first 128, next to some other minimum requirements. Nevertheless, this does not mean that a song played as MIDI will sound the same across all platforms – the sound of the instrument is still at the manufacturer’s discretion. Over the years *General MIDI* is unofficially extended by Roland with the GS and by Yamaha with the XG specification. [Tyler, 2006] In 1999 General MIDI is officially extended by the General MIDI 2 specification[18]. [MIDI Manufacturers Association, 2008] Another important step is taken with the introduction of “Downloadable Sounds Level 1.x”[19] (DLS) in 1997. [MIDI Manufacturers Association, 2008] This allows custom sounds to be loaded into the systems memory, basically providing a limitless extension of the sounds controllable by MIDI.

The next big change in music formats is the introduction of the MOD format. The format was used for the first time in 1987 in an application written by Karsten Obarski, called *Ultimate Soundtracker*. [SoundTracker, 2008] As with MIDI files, MOD files contain all control data used to play music, but unlike MIDI the files also contain all sounds used within the music piece. Originally developed for the Amiga hardware, MOD files only support four simultaneous channels and the original *Ultimate Soundtracker* only allows 15 instruments, but after the release of the source code similar programs extend the format’s capabilities. At the beginning of the nineties, applications supporting the format also appear on the PC. [Wright, 1998] Including the samples needed for playback of course results in files larger than the size of a similar MIDI file. On the other hand, composers are guaranteed to always have exactly the same instruments playing exactly the same sounds as the samples are bound

[18] General MIDI 2 is currently in version 1.2, updated in February 2007. [MIDI Manufacturers Association, 2008]
[19] Downloadable Sounds is currently in version 2.2, updated in April 2006. [MIDI Manufacturers Association, 2008]

to the control data – the only difference can lie in the quality of playback, depending on the output capabilities of the sound device.

The final way to store audio is basically storing the whole music in one sample. This method has some benefits to it. The music can be recorded in one piece instead of having to split it up and put the pieces together. This allows for example real orchestral sound instead of the combination of instrument tones. It also allows soundtracks consisting of musical pieces by other artists – a good example for this would be the *wipEout* series; one of its staples is the soundtrack, consisting of electronic music written and performed by famous artists. The same goes for nearly all rhythm-based games like *Dance Dance Revolution*, *Guitar Hero*, *Elite Beat Agents* or *SingStar*. All of these games employ prerecorded music to accompany the game-play.

Unfortunately, there are also downsides to completely prerecorded music. For one, the amount of storage needed is much higher compared to the sizes of MIDI or MOD files. This is one of the reasons for this kind of game music only becoming popular with the introduction of CD technology in gaming platforms – previous methods of storage, such as floppy disks and cartridges just did not have the space to accommodate a few songs in full CD audio quality, not to mention a whole soundtrack. There are, of course, a few methods to reduce the size requirements of prerecorded music but most of the time it is a tradeoff – one has to sacrifice quality in sound or the music needs more processing power to be played.

The main method of quantizing sound to be stored digitally is called Pulse Code Modulation. It is basically the method already described for quantization: Sample a wave with a certain amplitude resolution at a certain sample rate. The problem is the amount of data one gets. Using CD quality as a measure, 44,100 samples are taken at 16-bit size each every second, resulting in 705,600 bits per second. Stereo sound needs double the size, reaching 1,411,200 bits per second. Translated into megabytes this means that one minute of CD quality stereo audio takes slightly more than ten megabytes of storage. Even when the quality is reduced by halving the sample rate and the bit rate, a significant drop in quality, the required space is still approximately 2.5 megabytes. Comparing that to the size of a standard 3.5” inch floppy disk – 1.44 megabytes – one can see that size was a major hindrance in the use of prerecorded audio. Over the years several methods have been devised to cope with this problem, some without losing any information (for example DPCM and ADPCM as different kinds of quantization[20] [Jeffay, 1999] and FLAC[21] as a compression format) as well as some with the loss of information (MP3 is the best known of these). Although this does not pose a problem for most hardware nowadays, decoding compressed audio requires significant processing power not available in earlier machines, especially within the context of a game.

As well as having to deal with a much higher size requirement, prerecorded audio has some other drawbacks as well. Interactive/adaptive music is not possible with prerecorded audio as there is no way to change the music stored. What’s more, the costs of producing complete tracks can be very high. Unlike having a composer arranging instruments together and then letting the sound device play the music, prerecorded audio requires actual musicians playing their instruments. This can start

[20] In Differential-PCM (DPCM), instead of taking the value of the sample, the difference between two samples is measured. This can be represented in much fewer bits. Adaptive DPCM (ADPCM) uses different sized words – small differences are stored in fewer bits than large differences. [Jeffay, 1999]

[21] “FLAC stands for Free Lossless Audio Codec, an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality. “ [FLAC, 2007]

by just taking one or two musicians in to replace samples not up to the required quality standards and stop at employing a whole orchestra to record a complete game's soundtrack[22]. [Wall, 2002] When using songs from other artists, licensing fees can play a big role too.

Dimensions of Sound

The easiest and most forward way to play digital sound, or any kind of sound at all, is to mix all signals into one single channel and push that data to the output. If more than one output is connected, all outputs would receive exactly the same information; in the case of loudspeakers every loudspeaker would play exactly the same sound (with minor delays between them, depending on the distance of the different speakers from the source). This kind of playback is commonly called mono playback, in the context of dimensions one could call this sound one dimensional (this is only concerning positioning, not the sound quality itself).

The next logical step is adding a second dimension to allow basic positioning of sound sources. This is done by introducing a second channel and thus splitting up the sound into a left output and a right output. Called stereo playback, this method represents today's minimum standard in entertainment electronics. The existence of two channels is already enough for the simulation of sound sources positioned in three dimensional space, as well as the movement of these sources.

The most basic way to move a sound source is called panning. Depending on the position towards the listener, the sound is played through both outputs at different levels. If the sound source moves, the level of volume gets adjusted for each speaker. As an example, if one wants to move a sound source from the complete left of the listener to the complete right, the signal strength of that source would be at 100% at the left and at 0% at the right output in the beginning. With the movement of the sound source the left output would continuously lose strength and the right side would gain strength until the ratio is reversed with now 100% of the signal's strength at right output. "Such system has no vertical positioning but it's possible to change the sound a little (for example, by filtering high frequencies) when it comes from behind the listener because in this case he hears it a little muffled." [Menshikov, 2003]

Another easy to implement technique is adjusting the volume based on the distance, also called attenuation. [Microsoft, 2008] The simplest way to implement the attenuation is to set a certain distance from the listener. If the sound source is within that border, the volume stays at a predefined level, past the distance the sound volume is lowered at a rate set by the developer. The sound can be lowered until it reaches zero, but in a game environment the preferred way should be to switch the source off at a predefined distance to free resources. This effect can be extended by the so-called "rolloff", simulating additional atmospheric effects. [Menshikov, 2003]

One final effect that can be implemented with relative ease is the Doppler effect. The Doppler effect is a physical effect that affects the wave length of any wave emitted by a source. If the source moves towards a receiver the wavelength is shortened. The opposite happens when source and receiver move away from each other – the wavelength increases. For sound waves, this means that the pitch of a sound increases if source and listener move closer to each other, and decreases if the distance between source and listener increases. The pitch can be calculated based on the relative speed of the

[22] A 30 minute score, recorded with an orchestra from Seattle, Salt Lake City or San Diego can cost $21,000 to $26,000. [Wall, 2002]

source and the listener to each other. The Doppler effect is mainly used in racing and flight games, but all genres can benefit from it. [Menshikov, 2003]

A more precise method of positioning (or better, the emulating of the position of a sound source) is made possible with the use of the Head Related Transfer Function (HRTF). ”*HRTF* (Head Related Transfer Function) is a transfer function which models sound perception with two ears to determine positions of the sources in space. Our head and body are actually obstacles modifying the sound, and our ears hidden from the sound source perceive sound signals altered; then the signals proceed to our head to be decoded in order to determine the right position of the sound source in space.” [Menshikov, 2003](emphasis by Menshikov) Unlike panning and attenuation, HRTFs also allow adjustments in height of the source. The HRTFs are recorded by inserting microphones into the ears of humans or by using a special head model with built-in microphones. It is important to note, that every human being has different HRTFs – the reason is quite simple. Every human has a different head and ear form. “A complete description of a subject’s head response requires hundreds of HRTF measurements from all directions surrounding the subject.” [Gardner, 2004] Some companies use synthetic HRTFs applying the same laws occurring in real HRTFs while other companies use averaged HRTFs. [Menshikov, 2003] The HRTF is supposed to be used as a filter function for the sound source; a direct convolution can be implemented using a finite impulse response (FIR) filter. [Filipantis Jr., 1994]

The HRTF has some downsides to it. Sound can be distorted and applying the HRTF can be slow. Another problem is immovable sound sources, as humans are used to turning their heads towards unexpected sounds. This allows the brain to get additional samples of the sources position. “If the sound source does not generate a special frequency forming the difference between the front and rear HRTF function, the brain ignores such sound; instead, it uses data from the memory and compares the information about location of known sound sources in the hemisphere.” [Menshikov, 2003] Another problem is the separation of the signals. Only headphones allow an easy method to deliver one signal to each ear. For speaker systems, crosstalk cancellation is needed.

Unlike with headphones, using speakers will always result in a listener hearing both channels, the one for the left, as well as the one meant for the right ear only on both ears. This effect is called “crosstalk”. “Crosstalk cancellation is a technique for sending arbitrary, independent signals to the two ears of a listener from conventional stereo loudspeakers; it involves canceling the crosstalk that transits the head from each speaker to the opposite ear.” [Gardner, 1997] In order for crosstalk cancellation to work, the listener has to be positioned in the middle of the two speakers, with just a small margin to all sides. The area allowing the listeners to hear 3D sound effects as intended with the optimum of crosstalk cancellation is called “sweet spot”. [Menshikov, 2003] More speakers can allow a larger sweet spot.

The introduction of more speakers seems to be the current method of improving the quality of 3D audio. In particular 5.1 systems (one center speaker, two front and two rear speakers, one subwoofer) are supported by many current sound cards and consoles[23]. To simulate 3D sound on such systems, panning is very often used. Some technologies use separate HRTFs for each speaker to simulate the 3D environment, including extended crosstalk cancellation - the resulting sweet spot is

[23] Examples are the Xbox, Xbox360, GameCube, Wii, PlayStation 2, PlayStation 3 and Soundblaster X-Fi as well as Audigy and some Live! models (with the PlayStation 3, Audigy and X-Fi cards supporting up to 7.1 configurations).

larger than the one of stereo speakers. These kinds of systems need a lot of processing power, resulting in hybrid methods, for example using HRTF for front speakers and panning for rear speakers. [Menshikov, 2003]

Next to the positioning of the sound source, some other effects play an important role, such as the already mentioned Doppler effect. Special care is given to the environment the listener is situated in and on how it affects the sound heard by the listener. Two different approaches have been developed to emulate the environment. The first one is called “wave tracing”, the other “reverberation”.

Wave tracing is very similar to ray tracing, a technique used in computer graphics. The concept of both these techniques is to follow the path of emitted elements while they are reflected in the environment. For wave tracing, this means tracking the direct path the sound wave takes, while also taking the 1st order reflections (reflected once) as well as the 2nd order reflections (reflected twice in the environment) of the sound wave in the environment into consideration.

An example for the implementation of such a method is Aureal’s *A3D 2.0*. “Aureal's Wavetracing algorithms analyze the geometry describing the 3D space to determine ways of wave propagation in the real-time mode, after they are reflected and passed through passive acoustic objects in the 3D environment.” [Menshikov, 2003] This approach guarantees a highly realistic representation of the acoustic environment the listener is located in, but wave tracing is a very demanding technique. In addition to requiring a lot of processing power, developers have to represent the 3D environment of the game in a way *A3D* can use. Due to the power requirements, calculations of higher order reflections are also not possible. [Menshikov, 2003]

Another method to simulate a surrounding environment is using reverberation. Instead of calculating the reflections for each sound source, an environment is defined. This environment defines certain parameters used for all sounds occurring, mainly reverb variables. Reverberations are reflections of the original sound source in the environment. They are very similar to echo but arrive in a much shorter timeframe. “'Echo' generally implies a distinct, delayed version of a sound, as you would hear with a delay more than one or two-tenths of a second. With reverb, each delayed sound wave arrives in such a short period of time that we do not perceive each reflection as a copy of the original sound. Even though we can't discern every reflection, we still hear the effect that the entire series of reflections has.” [Harmony Central, 2008] Reverb can be separated in two sets of reflections. Early reflections arrive in a short period after the direct sound and are well defined and directional reflections, relating directly to the environment. Late reflections, also called diffuse reverberation, arrive at a much higher rate after the early reflections and are much more random. “It is believed that the diffuse reverberation is the primary factor establishing a room's 'size', and it decays exponentially in good concert halls.” [Harmony Central, 2008] Of course, the shape, size, and material of the environment as well as the distance between the sound source and the listener affect the reverb.

EAX, developed by Creative Labs, is an extension to the *DirectSound3D* component of *DirectX* that allows developers to use reverb for 3D sound. The basic principle behind the technology is to define a set of reverb variables for the listener and to set the values of these variables so as to reflect the environment the listener is located in. The distance between the listener and the sound source can then be represented by the wet/dry ratio, the ratio between the strength of the reverb and the

strength of original sound source.[24] [Hagén, 1999] Over the years EAX has evolved greatly. Today EAX Advanced HD 5.0 features a large set of defined parameters, including volume level, reverb, reflection and attenuation variables, as well as parameters for delay and decay times, sound tone, granularity, panning and pitch modulation for the listener. Instead of just defining an environment for the listener, sound sources have their own environments too now. The transition of the listener between different environments is greatly improved – instead of just switching between two reverb settings, the transition is done gradually. Sources also have their own set of parameters including volume control, 3D properties as well as occlusions, obstructions and exclusions controls. [Menshikov, 2003]

Occlusion, obstruction and exclusion are important for creating realistic sound. The easiest of these effects is occlusion. Occlusions represent a solid barrier between the sound source and the listener. This means that the direct path, as well as the path of the sound reflections, is blocked. As a result, the listener only receives a muffled sound from both the sound source and its reflections. The effect can be varied by defining various material properties as well as the shape and thickness of the barrier.

Obstructions are obstacles that block the direct path of the sound, but allow the reflections to travel freely. In this case the distortion effect, with material, shape and thickness of the obstacle taken into consideration, has to be applied only for the direct sound reaching the listener.

Exclusions represent the opposite case of obstructions. The direct path between listener and sound source is open, but the reflections are at least partially blocked. To simulate sound travelling through such a barrier the direct sound is left unaffected and the reflections are played with a distortion effect, using again the shape, thickness and material as parameters.

These effects add a great deal of realism to 3D audio, but require a lot of computational power. ”Anyway, no matter how the effects are realized (with Aureal A3D, Creative Labs EAX or manually on your own audio engine), it's necessary to trace geometry (wholly or only the sound part) to find out whether there is a direct contact with the sound source. This is a very strong blow on performance.” [Menshikov, 2003]

Sound in Games

Sound in games has a number of different functions to fulfill and can take different appearances.

One way to differentiate sound is to use a classification from film theory, a separation in diegetic and non-diegetic sound[25]. A sound is called diegetic if the sound source is visible on the screen, or if the sound is assumed to be heard within the action of the film. In games diegetic sound can be defined as sound originating in the game environment. Examples for such sound are voices of characters, sounds made by objects in the game or music created by instruments within the game’s environment. “*Digetic*[sic] sound can be either *on screen* or *off screen* depending on whatever its source is within the frame or outside the frame.” [FilmSound.org, 2007](emphasis by FilmSound.org)

Non-diegetic sound is a term for sounds that do not originate within the story space of a movie or, in the terms of games, sound not originating in the game’s environment. Examples for non-diegetic

[24] 100% dry would mean sole playback of the original sound source, while 100% wet would mean that just the effect is played.

[25] “Diegesis is a Greek word for "recounted story"”. [FilmSound.org, 2007]

sound include commentary by a narrator, mood music or sound effects added for dramatic effect. [FilmSound.org, 2007]

Axel Stockburger gives a more detailed classification of sound in games, creating a typology of sound objects according to their use in [Stockburger, 2003]. He identifies five different objects within the games environment, based on the game *Metal Gear Solid 2: Sons of Liberty*, developed by Konami.

The first group he identifies is the category of speech sound objects. Speech can be used as a diegetic or non-diegetic element in games. "Mostly it is employed as an intrinsical element of the diegetic system, developing the narrative of the game." [Stockburger, 2003] Speech is also an element used to convey emotion and immerse the player in the game world. Up until recently, space requirements of game media restricted the widespread use of speech in games, especially games heavy on dialogues, like role-playing games. Instead of having characters interact which each other by speech, text was displayed to convey information. Voice acting was mostly restricted to cut scenes or relevant lines. Although this does not pose a problem nowadays, some game developers make the conscious choice of not including speech samples for their characters. Examples of such games are the *Zelda* series by Nintendo, which still relies solely on text with a few non-speech emotions like surprised shouts or angry screams added, and the *Half-Life* series by developer Valve. The *Half-Life* games are a special case, as voice acting is used selectively – all characters but the player's character speak within the game.

Stockburger describes effect sound objects as sounds that seem to be originating from visual objects and events within the game world. "The realm of the effect sound objects is generally constituted by all the sounds, which are at the forefront of the user's attention with the exception of intelligible speech." [Stockburger, 2003] Examples of such objects include the opening of doors, the sound of guns fired and cars driving but also sounds the player's own character or other, non player characters, produce. Stockburger further includes sound effects indicating the health status or signals to player, conveying that the player has picked up a special game item or that the player has reached a new high score, for example. This allows effect sound objects to be diegetic or non-diegetic, as some effects, like gunfire, will definitely originate within the game world while others like the signal announcing a new high score will clearly come from outside the game environment. Another example for a non-diegetic effect sound object is "mickey mousing", a technique used to convey the action on the screen through music. A good example would be the "jump" sound from *Super Mario Bros.*, illustrating the action on the screen with "an ascending chromatic glissando or slide – think of a "boing" sound." [Whalen, 2004]

Zone sound objects are defined by Stockburger as sounds that are connected to a specific game environment, in other words, zone sound objects can be described as ambient sound. "Zones are separated by differing causally linked visual, kinaesthetic or auditory qualities. In special cases different zones overlap. Zone sound objects are aurally defining zones within the game environment." [Stockburger, 2003] It is important to note that zone sound objects do not have to have a visual representation within the game environment as long as it is believable for them to originate in that environment. An example for a zone sound object with a visual representation would be the sound of raindrops combined with rain falling within the environment. The sound of birds in a forest without the existence of a visual representation of these birds is a good example of the other type of zone sound object. Although the player cannot see the birds, the sounds still fit the

environment (provided the right bird sounds were chosen) and contribute to the immersive experience. Zone sound objects can be classified as diegetic sounds.

The soundtrack of a game is defined as collection of score sound objects. “The game score or music consists of a number of sound objects that belong to the non-diegetic part of the game environment. In numerous games the player can decide to switch the music on or off independently from the sound effects.” [Stockburger, 2003] Score sound objects have different uses within the game. One of its main functions is to convey emotions and set the mood of a scene. Very often, different pieces of the score are linked to different environments. Score sound objects are also used to mask transitions within the game world and moments of inactivity, like loading times and idle situations. Although Stockburger defines game music as a non-diegetic part of the game environment this classification can prove to be problematic in special situations. For example, many racing games feature a soundtrack consisting of different songs by known artists, most often allowing the user to switch between songs on the fly. Some of these games portray the control of the sound track as controls of the car radio, sometimes even playing radio jingles between the songs. Examples for such games are *Project Gotham Racing 2* by Bizarre Creations and *Burnout 3: Takedown* by EA Games, both released on the *Xbox*. Although being seemingly controlled from within the game environment, the sounds themselves do not originate from the car’s speaker system (as the properties of the sound do not change when switching the perspective from within the car to outside cameras) and can also not be defined as zone sound objects.

The last object listed by Stockburger is the interface sound object. “Interface sound objects share most of the qualities of effect sound objects with the notable exception that they are usually not perceived as belonging to the diegetic part of the game environment.” [Stockburger, 2003] The classification cannot be made distinctively as some games incorporate the interface within the game environment. Stockburger himself lists *Metal Gear Solid 2: Sons of Liberty* as an example - the interface for saving this game is included within the game narrative. To save the game, the player has to contact another character through the player’s own character’s communication device. Generally interface sounds objects are sounds that can be heard traversing and using the menu of a game, for example the settings or the load/save dialogs. [Stockburger, 2003]

Karen Collins uses another method of categorizing game audio. Building upon the already mentioned diegetic/non-diegetic properties of game audio she further specifies sound as being non-dynamic, adaptive, and interactive.

Both adaptive and interactive audio can be grouped together under the term dynamic audio. “I use the term dynamic audio to encompass both interactive and adaptive audio. Dynamic audio, then, is, audio which reacts to changes in the gameplay[sic] environment or in the response to a user.” [Collins, 2007c] Adaptive audio is sound changing without the direct control of the user, responding to the game play. Interactive audio is defined as sound events directly responding to the player’s actions.

Collins defines six groups of game sounds in [Collins, 2007c], making a primary distinction between diegetic and non-diegetic sound. These two broad categories are further split up in three subgroups each, distinguished by being non-dynamic, adaptive, or interactive sounds.

The first group, non-dynamic diegetic audio, is sound that is rooted within the game environment but cannot be affected in any way. The ambient sound of machinery in a factory, provided the player has no control over the machines, is an example for this group of sounds.

Adaptive diegetic audio describes music that is created within the game's world and is affected by game play. Collins uses environmental sound effects that change between the daytime and nighttime presentation of the scene as an example for this group of sounds.

The last group of diegetic sounds, interactive diegetic sounds, can be directly triggered by the player and of course exist within the game environment. Most actions a player is able to perform fall under this category, starting with the sound emitted when firing a gun or the sound of footsteps, and moving on to games like *Electroplankton* giving the player control over animals emitting sounds when touched.

Non-dynamic, non-diegetic sound is mostly used for cut-scenes. "In these cases the player has no control over the possibility of interrupting music (short of resetting or turning off the game)." [Collins, 2007c] According to Collins, this represents the most basic level of game audio.

Super Mario Bros. contains a basic example of non-diegetic adaptive game audio, sound that is affected through game play but not contained within the game environment. The game has background music that is non-dynamic for the most part of the game, but changes when the timer starts to get low. The music gets faster and faster, provoking a sense of urgency in the player. Collins offers *The Legend of Zelda: Ocarina of Time* as an example – the theme of the game environment changes between night and day.

The last group contains the interactive non-diegetic sounds. Again, *Super Mario Bros.* offers a good example for this group, using "mickey mousing" as a technique for many movements of the player character. The already discussed jump sound is an interactive non-diegetic sound.

In addition to categorizing game sounds into these six categories, Collins identifies nine functions of game audio:

- commercial functions: Games can be used as marketing tools. Music released as the soundtrack of a game can become more popular through the game.
- kinetic functions: Some games feature a "direct participatory and performance aspect to listening to the songs." [Collins, 2007c] In this cases music becomes a driving force in motivating the player while remaining in the focus of the player's attention at the same time.
- anticipating action: Game audio can foreshadow certain events by cueing scores informing the player of certain situations lying ahead. Sound effects can play the same role.
- drawing attention: Audio in games can be used to identify goals, environments, obstacles or special events by giving the player cues or making certain aspects of the game world stand out. "Recurring musical themes can situate the player in the game matrix, in the sense that various locales or levels are usually given different themes." [Collins, 2007c]
- structural functions: Game audio can be used to give scenes in a game a frame by using identifiable openings and closing sounds. By fading the music out, players can be animated to move on. Breaks can signal changes in the narrative, continuous music over different scenes can give these scenes a common context.

- reinforcements: Reinforcements can be delivered through dialogue, in a direct way by giving clues and goals, but also indirectly through accents, language used, and the timbre of the delivery.
- illusionary and spatial functions: Game audio can help deepen the immersion by delivering a realistic aural representation of the environment. Through ambient sound, non-diegetic music and sound effects, all orchestrated to create one game world, audio can reinforce the player's feeling of experiencing a "real" world.
- environmental functions: Sound can be used to attract people towards the game, or to mask sounds of the surrounding environment.
- communication of emotional meaning: As in movies, sound is of course also used in games to communicate emotions and induce moods.

Music-Based Games

Music-based games are a special case of audio used in games, as the entire game play revolves around sound. Looking at the list of functions defined by Collins in [Collins, 2007c], the most prominent function taken by sound in music-based games is definitely the kinetic one. "In many music games, the player is placed in the role of the star, the performer, even if these games are meant primarily for home play." [Collins, 2007c]

Collins also mentions kinetic gestural interaction, a very direct method of interacting with audio in games that is prevalent in music-based games. "At its simplest level, a joystick or controller could be argued to be kinetically interactive in the sense that a player can, for instance, play an ocarina by selecting notes through pushing buttons on a controller; but more significantly, here I refer to when a player may physically, gesturally mimic the action of a character, dancer, musician, etc. in order to trigger the sound event." [Collins, 2007c]

Martin Pichlmair and Fares Kayali present in [Pichlmair, et al., 2007] a classification scheme for music-based games. They identify seven qualities of music-based games:

- active scores (the player is allowed to alter the score)
- rhythm action (game play based on timing)
- quantization (actions by the player are automatically synchronized with the score)
- synaesthesia (a combination of visual and aural, sometimes also haptic, sensations)
- play as performance (requirement of a physical performance)
- free-form play (using the game as an instrument without any restrictions imposed through the game)
- sound agents ("Sound Agents are visual elements primarily existing for affecting, emitting, or accompanying sound." [Pichlmair, et al., 2007])

By analyzing the existence of these qualities in different games, Pichlmair and Kayali establish two types of music-based games: Rhythm games, and what they call instrument games. [Pichlmair, et al., 2007]

Presenting an Example: Phase

To illustrate the categorizations described in the last two sections I will employ a game example, *Phase*, developed by Harmonix Music Systems (of *Guitar Hero* fame) and published by MTV Games. The game was released in the beginning of November 2007 for IPods, specifically, the 5th generation

of IPods, the video version of the IPod Nano and the IPod Classic. The game is available through the iTunes music store, costing €4.99 at the time of writing. [Harmonix Music Systems, 2007a]

The game offers two game modes, “quick spin” and “marathon”, with “quick spin” consisting of one song, while “marathon” plays five songs in a row, becoming consistently more challenging. The player has the initial choice of three difficulty settings, “easy”, “medium”, and “hard” with two additional settings that can be unlocked. “Expert” is unlocked by beating a marathon on “hard” and “insane” is available to the player after beating a marathon on “expert”. Both “expert” and “insane” offer the same levels, with “insane” requiring a higher accuracy.

One song equals one level in the game, with no song being allowed to last longer than 30 minutes or shorter than 30 seconds. Phase includes a soundtrack with seven songs, but the game is meant to be played with the player’s own music. These songs are selected in iTunes by dragging them onto a special play list, called “Phase-Musik” in the German version of iTunes, and letting iTunes then analyze these songs to be usable in *Phase*. No matter if a song is chosen from the included playlist or added through iTunes, upon starting a level the user finds herself in one of six different graphic sets, called journeys.

While playing, the game moves steadily through these sets presenting different environments, called “deep sea” or “sonic city” for example. The surroundings always consist of a plane with a straight track on it, presenting three lanes pointing away from the player. Small figures fitting into the context of the surroundings (deep sea fish in the “deep sea” set for example) pass by the player while she moves along towards the goal shown on the horizon. On the three lanes jewels approach the player, colored in green and blue, as well as so called sweeps, dotted blue lines traversing all three lanes.

Figure 6: A typical level of *Phase* by Harmonix Music Systems, in this case using the “music festival” graphics set. Retrieved from [Harmonix Music Systems, 2007]

The player’s role in the game is to hit each of the jewels and to follow the sweeps to gather points. Each level consists of several checkpoints, requiring a certain number of stars to pass. By collecting points the player can get up to five stars. If not enough stars are collected, the number of missing stars is deducted from the number of hearts in the player’s health bar. The player starts out with three hearts but can collect new ones by receiving more stars than required in between checkpoints–

for each of these bonus stars one heart is added up to a maximum of four hearts. If all hearts are lost at a checkpoint, the game ends immediately.

Phase makes use of the unique controls located on an IPod. Jewels have to be hit by pressing the button associated with each lane, the “backwards” button for the left lane, “center” for the center lane, and “forward” for the right lane. The click wheel on the IPod is used for the sweeps – the player has to scroll on the wheel to follow the line presented on the screen. Green jewels don’t give many points but are required to get a high multiplier (by not missing any for an extended amount of time), blue jewels give more points if a whole sequence of them is hit without failing but turn into green jewels if one is missed, and sweeps also give high points but only if followed completely. If the player misses a part of the sweep, the whole line turns green and does not give any more points (but the multiplier is kept). Every missed jewel, blue or green, resets the multiplier.

As already mentioned, iTunes calculates the level a player has to go through by analyzing the song playing for that particular level. This is done once on the PC with the analyzed song then being transferred to the IPod to be playable. Jewels and sweeps are set in the level depending on the song itself, although it is sometimes not clear what parts of a song are used – sometimes the rhythm of the jewels is affected by several different instrument tracks of the song at the same time.

Obviously, Phase can be considered a music-based game, so it is appropriate to apply [Pichlmair, et al., 2007]’s classification scheme. Although the user is allowed to alter the games score by adding new songs and removing others, one cannot speak of an active score, as the changes happen outside of the game. Inside the game the user has to follow a strict pattern, every button press not hitting a jewel is considered a miss. The game is definitely based on timing and rhythm, thus possessing the quality of rhythm action.

If the player hits a jewel at the right time, a small click sound is emitted. A wrong hit is also indicated by a short sound, the same sound is used for failed sweeps. None of these sounds is synced up to the music playing, so no quantization takes place. Synaesthesia is not existent either, as the graphics shown in the game are not affected in any way by the player’s action. Playing the game as a performance, as well using it as an instrument are not applicable. Finally, sound agents also do not exist within in the game – one of the tips presented to the player before the start of a level actually advises users that they can switch off sound effects completely to only hear the music playing.

Out of the seven qualities specified by Pichlmair and Kayali only one, rhythm based game play, is applicable. This puts *Phase* clearly in the category of rhythm games.

The background track is non-diegetic in nature, as it has no representation in the game’s environment. Although the layout of jewels is based on the songs played, the jewels do not generate the music. Based on Collins’ classification in [Collins, 2007c] one can speak of a non-dynamic, non-diegetic sound. The sounds created by the jewels and sweeps can be considered diegetic, as they are created within the game’s world. The sounds are adaptive, as the game decides which sound is made, not the player, by detecting if the button press was a hit or miss.

When trying to categorize sound using Stockburger’s classification of sound objects in [Stockburger, 2003], one can rule out speech sound objects immediately, as the game offers no interaction with other characters, and also no non-diegetic narration. Speech is present in the game, but only as part of the songs playing in the background, making it a score sound object by Stockburger’s definition.

Effect sound objects are also present, if only in a minimal form. The game developers obviously did not want to take attention away from the music itself, and unlike in other games, like Guitar Hero, it is not possible to remove certain parts of the songs to make them "playable" by the user (by basically mixing these back in if the player can hit a certain sequence without miss). Interface sound objects exist in the same minimal way, zone sound objects are not present at all in the game.

As one can see, the game audio is completely based on the song running in the background of the level being played – the music is mainly fulfilling a kinetic function as described by Collins in [Collins, 2007c], all other sound effects are subordinate to the background music. The music defines the whole game: The levels are set to the beats of the song playing, the level ends at the moment the song ends, and the goal for the player is to reach the end to hear the full song, instead of being stopped at a checkpoint in the middle of the track. Thereby the music also satisfies the structural function in the conventional sense (as far as the condensed game play allows it), but also very literally – without the songs loaded into the game the user would not be able to play as there would be no levels to use. *Phase* is, in this way, very similar to *Vib Ribbon*.

Conclusion

The last two chapters are meant to provide a theoretical introduction to sound as it is used in games with an emphasis on music-based video games.

"The History of Sound in Video Games" provides an overview of the advancements in sound technology over the last 30 years. By mentioning several games I also portray the evolution of music-based games in the same timeframe.

"Creating Electronic Audio" establishes the basics of sound generation. In this chapter, I first explain what sound is general and then concentrate on digital sound in particular. I present the methods used in games to generate sound and provide a summary of the possible methods of storing digital audio for playback. Finally, I introduce two different models of game sound categorization by Stockburger and Collins and outline the seven qualities of music-based games, as defined by Pichlmair and Kayali.

All of these classification schemes are then illustrated by employing an example, the game *Phase*. Phase was developed by Harmonix Music Systems and published by MTV Games and is currently available for the 2nd generation of the IPod Nano, the IPod Classic and the 5th generation of the IPod.

The foundations established in this part of the thesis are used to create an own prototype on the basis of the Torque Game Builder engine.

Part 2: Technical Foundations

The Torque Game Builder Engine

The Importance of Engines in Prototyping

In his article on "Common Game Prototyping Pitfalls" [Cook, 2005], the first point Daniel Cook mentions is the need for an engine to create a prototype in. "Of all the pitfalls, the need for infrastructure is the most difficult to overcome. Most games need substantial game engine system[sic] in place before they can be prototyped. The list ranges from graphics engines to networking support. It is nearly impossible to prototype a game like Doom if you don't have a 3D engine. It is also difficult to prototype an online game when your engine had not[sic] networking support." [Cook, 2005]

The solution he offers is very simple – instead of creating the whole system from scratch, most of the time it is much more straightforward to use an already existing engine, be it freeware or a commercial product. This allows designers to go straight to working on the game itself instead of being involved in a process that is "is costly, time consuming and gets in the way of the real task at hand: rapidly exploring a series of game mechanics." [Cook, 2005]

Rapidly exploring a series of game mechanics, in this case the game mechanics of music-based video games, is the goal of the prototype developed by Pichlmair and Kayali. The TGB engine is used as the underlying system for this prototype, called *Radiolaris*.

The TGB engine is described as "the world's most powerful 2D game engine" by its developer, GarageGames [Garage Games, 2008a]. The engine offers an easily accessible editor with drag and drop functionality, as well as its own scripting language, called TorqueScript. "TorqueScript is an easy to use C++ like scripting language that ties all of the various elements of your game together. It supports a large complement of functions including math, object manipulation, fileIO, and more." [Garage Games, 2008b]

The engine is available in two different versions, a standard version and a "pro" version. The only, albeit large, difference between the two versions is the inclusion of the engine's source code in the "pro" version, making it possible to change, extend, or add certain aspects of the engine. The cost of the engine is variable, starting at $100 for the standard version with an "indie" license (only sold to companies making less than $250,000 in a year, with some other restrictions applying in the use of the engine) and continuing up to $1250 for the "pro" license for commercial developers.

As the changes described in the next two chapters, namely the addition of joystick and gamepad support and supplementing the *OpenAL* library with the FMOD library, require access to the source code, it is necessary to buy a "pro" license for "indie" or commercial development – both licenses include the same code. The TGB engine works with Windows, Mac OS X, and Linux, but in this thesis only the Mac OS X version is used. At the time of writing this part of the thesis, version 1.5.1 was the newest version, but recently version 1.6 has been released. [Garage Games, 2008a]

Adding Joystick/Gamepad Support to the Torque Game Builder Engine

Preface

The Torque Game Builder (TGB) engine includes support for gamepads and joysticks in a Windows, as well as a Linux environment. In Windows the Direct Input Library, part of the Direct X API, is used to administer joystick and gamepad functions. The Linux build of the TGB engine uses SDL, the Simple Direct Media Layer, to provide the same utilities.

The Mac OS X version does not recognize any joystick or gamepads at all. This is unfortunate - as already mentioned, Collins describes “kinetic gestural interaction” in [Collins, 2007c] as the most direct audio interaction possible. She argues that this interaction is already present in a joystick or gamepad at the simplest level. Keyboard and mouse, the two control methods present on nearly every PC offer even less of a tangible user interface for games, while depriving the user of analog control sticks and Force Feedback technology. On this account, gamepad and joystick support was the first addition to game engine.

Overview

Basically there are two ways of adding joystick support. The first method involves building the TGB Engine with the Mac OS X version of the SDL library and using the functions offered through SDL (by taking the Linux implementation of the TBG Engine as an example). Unfortunately, this method does not allow for Force Feedback effects in the game, as those are not supported by the current version of SDL at the time of writing, SDL 1.2.

The second method involves using Apple’s own I/O Kit framework to connect to the joystick and to handle the incoming input. Furthermore, Force Feedback effects can be created through the use of the Force Feedback framework. The drawback of this approach is the necessity to create the connection and the device management from scratch. The advantage of Force Feedback still outweighs this disadvantage, thus, this method was chosen.

Implementation

As already mentioned, the basis for the connection to the game controllers is the I/O Kit framework. As described in Apple’s introduction, “the I/O Kit is a collection of system frameworks, libraries, tools, and other resources for creating device drivers in Mac OS X.” [Apple Inc., 2003b] Of main interest is the HID family, a collection of classes that defines the basis for developing drivers for HID devices. The group of HID devices is a subset of the USB devices which “consists primarily of devices that are used by humans to control the operation of computer systems.” [USB Implementers' Forum, 2001] This naturally also includes joysticks and gamepads. [USB Implementers' Forum, 2007]

Using HID instead of straight USB is easily explained: HID devices can be queried for a set of parameters which then can be matched to filter out any unwanted objects. Each element of the device can also be examined for its function, which again allows those properties to be matched against predefined tables and thereby the input scheme to be constructed.

The two steps of first finding a device and then defining it are carried out by two different classes in the implementation. Naturally, more than one joystick or gamepad can be connected to the system at any given time. This requires the implementation of a system which can manage several joystick

devices as well as finding them. The functions necessary are contained in the `JoystickHandler` class. If a matching HID device is found, an instance of the `JoystickDevice` class is created. This class will then further manage the assigned device. Figure 7 illustrates both classes. The source code used is an extended and modified version of the source code presented by Apple in [Apple Inc.,

2001].

Figure 7: UML-Diagram representing the joystick control classes

Finding Devices

The search for the game control devices is handled by the `JoystickHandler` class, more specifically by the "start" method which is called when an instance of the `JoystickHandler` is initiated. Below is an excerpt of the method showing the search for joystick devices, finding gamepad devices differs only marginally.

```
//Look for joysticks
io_object_t hidDevice = IO_OBJECT_NULL;
hidObjectIterator = FindHIDDevices(kIOMasterPortDefault, kHIDPage_GenericDesktop,
kHID_USAGE_GD_Joystick);
counter = 0;
int i = 0;
if(hidObjectIterator != (int)NULL)
{
        while((hidDevice = IOIteratorNext(hidObjectIterator)))
        {
                if(counter < 4)
{
                        joystickDevice[i] = new JoystickDevice(hidDevice, i);
Con::printf("%i. joystick initialized: %s %s\n", i, joystickDevice[i]->getVendor(),
joystickDevice[i]->getProduct());
                        i += 1;
                        counter += 1;
}
        }
}
```

The first step is the creation of an object that will be used to keep a reference to a joystick device. Next, the method `FindHIDDevices` is called. This method returns a collection of device references that match the selected set of usage and usage page - in this case joystick devices that declare themselves as generic desktop devices. The source code of the method shows the two important lines, first the creation of a suitable dictionary to match against with use of the function `MatchHID`, and the retrieval of all matching devices by using `IOServiceGetMatchingServices`. The number of game control devices is capped at four as the game that is intended to be used with this `JoystickHandler` class only needs two inputs and more than four game controllers are very rarely needed.

```
//initialize needed variables
CFMutableDictionaryRef hidMatchDictionary = 0;
IOReturn ioReturnValue = kIOReturnSuccess;
io_iterator_t hidObjectIterator = 0;
//Create a matching dictionary
hidMatchDictionary = MatchHID(usagePage, usage);

if (hidMatchDictionary == NULL)
{
        Con::printf("Failed creating the dictionary.\n");
}

ioReturnValue = IOServiceGetMatchingServices(masterPort, hidMatchDictionary,
&hidObjectIterator);
if (hidObjectIterator == 0)
{
        Con::printf("No usable devices found.\n");
}
else if(ioReturnValue != kIOReturnSuccess)
{
        Con::printf("Could not create iterator.\n");
}
hidMatchDictionary = NULL;
return hidObjectIterator;
```

Unless the returned collection is empty (in that case an error will be printed to console), every element of the collection is passed on to the constructor of the `JoystickDevice` class along with a counter, used as the ID of the device. Afterwards, the ID of the device is written to the console, as well as the vendor name and the device name, if applicable. The same operation is executed with minor adjustments to find all devices reporting themselves as gamepads.

Getting input from the device

Finding devices is only the first step - in order to get input from the device, several more have to be taken. This is the task of the `JoystickDevice` class. This class represents one found device and handles all input and output from that device. The following source code shows the initialization of a device found by the `JoystickHandler` class. The code is located in the `JoystickDevice` constructor.

```
hidDevice = hidDevicePassed;
hidDeviceInterface = NULL;
//Initialize the Force Feedback system
FFinit();
//This is the default value for the dpad
previousValue[0] = 8;
previousValue[1] = deviceNumber;
//Create an interface to the device
hidDeviceInterface = CreateHIDDeviceInterface(hidDevice);
if(hidDeviceInterface != NULL)
{
	//Use the created interface to get the deviceCookies
	gamepadCookies = GetHIDCookies((IOHIDDeviceInterface122**)hidDeviceInterface);
	//Create an eventqueue that sends eventcalls to the QueueCallBackFunction
	GetHIDInterface(hidDeviceInterface, gamepadCookies);
}
```

The constructor of the `JoystickDevice` class includes some more code to allow the display of the vendor and the device name but the main functionality lies in the calls to three methods:

- `CreateHIDDeviceInterface`
- `GetHIDCookies`
- `GetHIDInterface`

These methods are responsible for creating an interface to the device, getting listeners for the services the device offers, and creating an event queue that allows for callbacks. For this reason each will be discussed in detail.

The name of the `CreateHIDDeviceInterface` method is already a good indicator for the function of the method. In order to communicate with the device and query its services, an interface needs to be created. The source code is as follows:

```
//initialize needed variables
HRESULT plugInResult = S_OK;
IOReturn ioReturnValue = kIOReturnSuccess;
IOCFPluginInterface **plugInInterface = NULL;
IOHIDDeviceInterface ** hidDeviceInterface = NULL;
SInt32 score = 0;

//create a core foundation plugin interface for HID devices
ioReturnValue = IOCreatePluginInterfaceForService(hidDevice, kIOHIDDeviceUserClientTypeID,
kIOCFPlugInInterfaceID, &plugInInterface, &score);
if (ioReturnValue == kIOReturnSuccess)
{
	//get the device's interface from the plugin interface
plugInResult = (*plugInInterface)->QueryInterface (plugInInterface, CFUUIDGetUUIDBytes
(kIOHIDDeviceInterfaceID), (void *) &hidDeviceInterface);
```

```
if (plugInResult != S_OK)
        Con::printf("Couldn't query HID class device interface from plugInInterface.\n");
(*plugInInterface)->Release (plugInInterface);
}
else
        Con::printf("Failed to create **plugInInterface via
IOCreatePlugInInterfaceForService.\n");
return hidDeviceInterface;
```

This code is a modified version of the source presented in "Creating a HID Class Device Interface", part of Apple's HID Class Device Interface Guide. [Apple Inc., 2001] After the preliminary initialization of the variables needed an intermediate Core Foundation plug-in interface is created, using `IOCreatePlugInInterfaceForService`. This temporary interface is then used to get the specific type of device interface needed. This is done by calling the `QueryInterface` method of the plug-in interface with several parameters, one of them being the address of the device interface intended to contain the new device interface. [Apple Inc., 2003a]

Cookies are unique identifiers for an element of the HID device. When added to an `IOHIDQueue`, they act as listeners on their respective element and report all events to queue, which then can initiate a callback. The `GetHIDCookies` function takes an `IOHIDDeviceInterface` as parameter, which is queried for all its elements. Next the function iterates over the array containing these elements and tries to match usage and usage page IDs of each element to predefined values representing inputs on the game controller. If a match is reported, the corresponding `IOHIDElementCookie` is stored inside the matching value of the cookie struct.

Instead of just using the struct with four cookies as in the example presented by Apple an adapted version is used. The extended `cookie_struct_t` structure is able to represent every function on the gamepad used for development, a Logitech *Rumblepad 2* controller. Next to all 12 buttons on the gamepad both analog sticks are tracked. Although the descriptions for the HID usage would ask to map the secondary analog stick's axes to the values 0x33 and 0x34 (standing for the Rx and the Ry axes respectively) the axes need to be mapped to 0x32 and 0x35 (being the Z and the RZ axes). According to [Wayper, 2003] and [Stahl, 2003] it seems to be a convention among joystick and gamepad producers to map secondary x and y axes to the z rotation and the slider, a fact that has to be considered in our mapping. The following code shows the matching of the axes in the code.

```
//Check for 1st x axis
if(usage == 0x30 && usagePage == 0x01)
cookies->gX1AxisCookie = cookie;
//Check for 1st y axis
if(usage == 0x31 && usagePage == 0x01)
cookies->gY1AxisCookie = cookie;
//Check for 2nd x axis
if(usage == 0x32 && usagePage == 0x01)
cookies->gX2AxisCookie = cookie;
//Check for 2nd y axis
if(usage == 0x35 && usagePage == 0x01)
cookies->gY2AxisCookie = cookie;
```

Additionally, a directional pad is mapped. HID normally provides a cookie for every direction of the directional pad but in the case of the Logitech *Rumblepad 2* the directional pad is mapped on the coolie hat, a device commonly found on the top of joysticks. Rather than requiring one cookie for every direction of the directional pad it allows every movement to be tracked with one cookie reporting back nine positions. The values for these directions seem to differ a bit between different

game controllers – for the *Rumblepad 2* the following are used (in clockwise rotation, starting north, 45 degree steps) : 0 1 2 3 4 5 6 7. The value 8 represents the Null value reported when no button is pressed.

After creating the HID device interface and mapping the elements of the device to their individual cookies the `GetHIDInterface` function can be called. The function takes the `IOHIDDeviceInterface` and the now filled `cookie_struct_t` structure as parameters. It opens an interface to the device and allocates a queue. All the previously found element cookies are then added to that queue to allow them to report events to it. Again the source follows closely the code presented in [Apple Inc., 2001], combining different functions laid out in the text.

By creating an asynchronous event source, a listener is added to the queue, waiting for new events being reported from the cookies registered with the queue. These events are then handled by calling a callback function that communicates with the game engine. The asynchronous event source is then added to the main loop and the start method of the queue is called, thus making it ready to receive events. The source code shows the creation of the event source, adding the callback, adding the event source to the main loop and starting the queue. One important action in this snippet is the passing of the address of the `previousValue` variable. This allows a specific check in the callback function which in turn permits the implementation of diagonal button presses on the directional pad. The procedure will be explained in detail in the description of the callback function.

```
result = (*queue)->createAsynchEventSource(queue, &eventSource);
result = (*queue)->setEventCallout(queue, QueueCallBackFuntion, queue, &previousValue);
CFRunLoopAddSource(GetCFRunLoopFromEventLoop(GetMainEventLoop()), eventSource,
kCFRunLoopDefaultMode);
result = (*queue)->start(queue);
```

Communicating events to the Torque Game Builder Engine

The callback function `QueueCallBackFunction` represents the connection to the TGB Engine, albeit only a unidirectional connection. This means that events can only be sent to the engine, with no way to retrieve any information. This restriction is eased by including functions in the game engine's console, accessible through TorqueScript, which bypass the engine and affect the game controller directly.

In order to communicate with the engine an `InputEvent` has to be created and passed to the main game loop. The structure of an `InputEvent` is shown in figure 8.

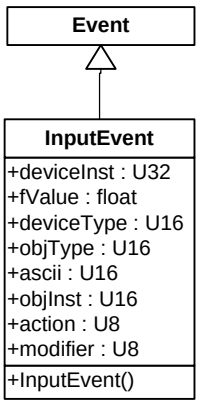

Figure 8: Layout of the InputEvent class

The `deviceInst` data field stores the device instance. Within the engine these IDs are represented as integers, therefore passing a simple '0' for the first game control device and a '1' for the second game control device is sufficient (for each controller added further the ID is increased by one, up until the maximum of four) . The value field stores values between -1.0 and 1.0. Button presses are registered as 1.0, button releases as 0.0. Analog joystick axes pass a value between 0 and 255 that is then mapped to the -1.0/1.0 range.

Mouse and Keyboard control are handled by the TGB Engine by default, so the only option needed for the `deviceType` field is "JoystickDeviceType". The `ascii` (as well as the `modifier`) field is not needed for game control devices and thus is set to zero. The `objType` field contains the type of input – SI_BUTTON, SI_XAXIS, SI_YAXIS and SI_POV (the last value representing the directional pad). The `objInst` variable specifies the event source more accurately – here the number of buttons and the orientation of the directional pad are passed, axes do not need that value. The data field `action` represents the action that is supposed to be executed. For buttons and the directional pad SI_MAKE and SI_BREAK are used to signal pressing or releasing of a button, axes use SI_MOVE. The code snippet shows the creation of an input event for a buttons including code checking if the button was pressed. [Garage Games, 2007a] [Garage Games, 2007b]

```
//Check if the button is pressed or not
S32 action = (event.value == 1) ? SI_MAKE : SI_BREAK;
//Set object instance to buttonnumber + the 0 button
S32 objInst = ((unsigned long)event.elementCookie - 5) + KEY_BUTTON0;
//Set bool according to the button being pressed or not
bool pressed = (event.value == 1) ? true:false;
U8 buttonNum = (unsigned long)event.elementCookie - 5;

InputEvent ievent;

ievent.deviceInst = deviceID;
ievent.deviceType = JoystickDeviceType;
ievent.modifier = 0;
ievent.ascii = 0;
ievent.objType = SI_BUTTON;
ievent.objInst = objInst;
ievent.action = action;
ievent.fValue = (action == SI_MAKE) ? 1.0 : 0.0;
Game->postEvent(ievent);
```

Unfortunately the TGB Engine only supports a four-way directional pad with the directions north, south, east and west. [Garage Games, 2007a] The directional pad on the Logitech *Rumblepad 2* supports 8 directions and being a coolie hat instead of a "real" directional pad it can only report one at a time. To include the diagonals offered by the coolie hat on the gamepad, a workaround had to be implemented. The TGB Engine supports 32 buttons but the gamepad chosen only implements 12. This leaves 20 unused buttons that can be mapped to other functions. To allow other game control devices to be implemented without restricting the number of available buttons too much, the last four supported buttons (KEY_BUTTON28 to KEY_BUTTON31) were selected to represent the diagonals on the gamepad. This still allows a theoretical maximum of 28 supported buttons on a game control device.

After solving the problem of unique identifiers for the diagonals another problem emerged. The directional pad reports button presses and releases with special identifiers: SI_UPOV, SI_DPOV, SI_LPOV and SI_RPOV, one for each of the four directions supported by the TGB engine. These identifiers need to be passed when a direction is pressed but also in the case of a release of a

direction. The directional pad, as implemented on the Logitech *Rumblepad 2*, reports the direction in instances of button presses but returns 8, the specified Null value when a button is released.

The naïve approach to solving this problem is sending an "SI_BREAK" command to all four IDs on the directional pad as well as the four buttons specified as diagonals. This would create an unreasonably large overhead every time the orientation on the directional pad is changed or released. This drawback was circumvented by passing along the last direction pressed on the pad. Every time an orientation is activated, the value passed from the joystick is stored in the first position of the previousValue array (the second being occupied by the ID number of the game control device).

This in turn allows easy distinction between the directions. Every time an event from the directional pad is passed on to the `CallbackFunction`, it gets sorted into one of two different cases. If the value passed through the `previousValue` variable is uneven, the previous orientation of the directional pad was a diagonal. This means that a button press has to be deactivated. The cancellation is then executed with the previous orientation specifying the button ID.

In the other case, the last press was a "real" directional pad orientation, thus a button press on the pad gets cancelled with the appropriate ID. Afterwards the new button press is created, distinguishing between the two cases of "button press" and "directional pad press" by testing if the current value passed is even or uneven. In the case of a value of eight, nothing is passed to the engine (as any previous orientation was already deactivated).

Initiating and Controlling Force Feedback Events

Force Feedback describes the ability of devices to give haptic feedback to actions. In games, these effects are used to augment the sensation of playing the game and can give viable game-play clues to the person playing.

Force Feedback devices for home use on the PC seem to have appeared for the first time by the end of 1997 although haptic support was already introduced earlier to the console market with the release of the Rumble Pak for the Nintendo 64 console in the middle of the same year. [Johnston, 1997]

Apple supports the Force Feedback technology with its own Force Feedback Framework which offers functionality very similar to the one provided by Microsoft's Direct Input API. [Apple Inc., 2006] Please refer to "Apple's Force Feedback Framework" in the Appendix A for an overview of the API.

Basically, Force Feedback devices can be separated into two groups:

- **Devices that support time-based effects.** The effects these devices produce are commonly known as rumble effects as the effect produced makes the game control device vibrate. This is done by rotating one or more unbalanced weights inside the game controller. Most gamepads described as having Force Feedback effects (like the *Rumblepad 2* used in this project) support only time-based effects. The ones supported by Apple's own Force Feedback framework are as follows [Apple Inc., 2007]:
    - Constant force: The game control device emits a Force Feedback effect constant in strength over the whole length of the effect.
    - Ramp force: Starting at one point, the strength of the effect follows a curve towards a second specified point.

  - Square Wave: The strength of the effect follows a square wave.
  - Sine Wave: The strength of the effect follows a sine wave.
  - Triangle Wave: The strength of the effect follows a triangle wave.
  - Sawtooth Up: The strength is determined by a wave looking similar to saw teeth pointing upwards
  - Sawtooth Down: The strength is determined by a wave looking similar to saw teeth pointing downwards. [Microsoft, 2007]
- **Devices supporting additional interactive effects.** These control devices are able to produce time-based effects, as well as interactive effects. These effects "are based on the state of the stick (position, velocity, and/or acceleration)." [Walters, 1997] This means that these effects react to user input. An example for such an effect would be the simulation of a plane controlled by hydraulic steering – the faster the plane is diving towards the ground, the harder it becomes to pilot the plane into a horizontal position by using the ailerons. This effect is emulated by the control device by applying force against users pulling on a joystick. Devices allowing for these effects are mostly joysticks and steering wheels, as gamepads do not include the appropriate controls for such an effect. The following effects are supported by the Force Feedback framework provided by Apple:
  - Spring: The further away the game device is moved from a predefined position, the more force will be applied.
  - Damper: The faster the device is moved, the more force is applied.
  - Inertia: The Inertia effect is similar to the Damper effect but uses acceleration instead of velocity.
  - Friction: Apply force the moment the game device is moved. [Microsoft, 2007]

Apple's Framework additionally supports custom effects and envelopes. "An envelope defines an attack value and a fade value, which modify the beginning and ending magnitude of the effect. Attack and fade also have duration, which determines how long the magnitude takes to reach or fall away from the sustain value, the magnitude in the middle portion of the effect." [Microsoft, 2007]

The actual implementation of Force Feedback effects is straightforward. After verifying that the chosen device does actually support Force Feedback effects, a new device object for handling these effects is created. All further communication to the game device concerning Force Feedback is handled through this device object. The snippet illustrates this process (the error handling for the device creation was removed).

```
HRESULT FFresult
//check if Force Feedback is supported
FFresult = FFIsForceFeedback(hidDevice);
If(FFresult == FFERR_NOINTERFACE)
{
        Con::printf("Device does not support Force Feedback");
        isFFDevice = false;
}
else if(FFresult == FF_OK)
{
        isFFDevice = true;
        //create the FF device object
        FFresult = FFCreateDevice(hidDevice, &FFHandler);
```

The game that is intended to be used with this project only needs minor Force Feedback support. For this reason, the only effect created will be a constant effect. To allow for easy control of the effect,

most variables are predefined – in the end, the only controllable elements of the effect will be the gain and its duration as this has proven sufficient enough for the purposes of the game. Additionally, no envelopes will be used.

In order to get a strong effect, the magnitude of the constant force effect will be set to the maximum allowed value. The Logitech *Rumblepad 2* includes two axes which can both be used at the same time, allowing for a stronger perception of the rumble effect. All of the mentioned variables have to be entered into data fields in order to create an effect which can be passed to the game control device. The source code shows the creation of a constant effect at maximum strength (also using both axes), lasting for 3 seconds.

```
//The axes and direction arrays required by the FFForceEffect struct
DWORD  dwAxes[2] = {FFJOFS_X, FF_JOFS_Y};
LONG           lDirection[2] = {0, 1};

//set the parameters for the constant effect
FFConstantEffect.lMagnitude = FF_FFNOMINALMAX;

//set the general parameters for an effect
FFForceEffect.dwSize = sizeof(&FFForceEffect);
FFForceEffect.dwFlags = FFEFF_CARTESIAN;
FFForceEffect.dwDuration = 3*FF_SECONDS;
FFForceEffect.dwSamplePeriod = 0;
FFForceEffect.dwGain = 10000;
FFForceEffect.dwTriggerButton = FFEB_NOTRIGGER;
FFForceEffect.dwTriggerRepeatInterval = FF_INFINITE;
FFForceEffect.cAxes = 2;
FFForceEffect.rgdwAxes = dwAxes;
FFForceEffect.rglDirection = lDirection;
FFForceEffect.lpEnvelop = NULL;
FFForceEffect.cbTypeSpecificParams = sizeof(&FFConstantEffect);
FFForceEffect.lpvTypeSpecificParams = &FFConstantEffect;
FFForceEffect.dwStartDelay = 0;

//Create the constant effect
FFresult = FFDeviceCreateEffet(FFHandler, kFFEffectType_ConstantForce_ID, &FFForceEffect,
&FFEffectHandler);
```

Once a Force Feedback effect gets requested by the game engine the method `FFEffectSetParameters` is called to edit the effect according to the parameters passed by the game. As only duration and gain are changed, the same struct that was used to create the effect is reused with the duration and gain set to new values. The next snippet shows the usage of the `FFEffectSetParameters` method.

```
//Edit affected parameters
FFForceEffect.dwDuration = length * FF_SECONDS;
FFForceEffect.dwGain = gain;

//Edit the effect to represent passed parameters
FFResult = FFEffectSetParameters( FFEffectHandler, &FFForceEffect, FFEP_DURATION|FFEP_GAIN );
```

After updating its parameters, the effect is ready to be passed on to the game control device. This is accomplished by calling the `FFEffectStart` method with the FFES_SOLO flag as parameter, instructing the game control device to stop any other active effects still playing.

The last methods used are `FFDeviceReleaseEffect` and `FFReleaseDevice` to free the memory in case the `JoystickDevice` object is stopped.

Accessing and Controlling Devices from within the Game

In order to use all of these functions discussed in the chapter, an instance of the JoystickHandler class has to be instantiated at the start of the engine, and has to be accessible by the game itself to communicate with the game control devices connected to the system. This functionality was implemented in two different ways.

The first method used to integrate the `JoystickHandler` within the engine was mimicking the calls made by the Windows and Linux implementations to their respective control devices. To ensure full functionality, activate/suspend functions as well as start/stop functions had to be created and called at the appropriate places within the engine. This worked well, but unfortunately this approach required modification of core engine files instead of keeping all functionality within newly created files, and it also did not allow easy communication with the game device to enable Force Feedback.

The second option proved to be more fruitful: By allowing the `JoystickHandler` class to be controlled by the console module all the functionality could be contained within the macCarbInputManager.cc and macCarbInputManager.h files. Furthermore, console functions provide an easily usable and extendable way of accessing the `JoystickHandler` and by proxy the `JoystickDevice` objects – only a static pointer to a `JoystickHandler` instance is required.

Console functions are easily added by including the console header file from the console folder in the game engine source. As well as making it possible to send messages to the console (by using the Con::print command) this also allows the creation of personal callable console functions with the `ConsoleFunction` method. The snippet presents the usage of the `ConsoleFunction` method, in this case used to specify the `InitJoystickHandler` method.

```
static JoystickHandler *JoystickHandler = NULL;

ConsoleFunction(InitJoystickHandler, void, 1, 1, "InitJoystickHandler()")
{
	if(!joystickHandler)
	{
		Con::printf("JoystickHandler not found");
		Con::printf("*****INITIALIZING JOYSTICKHANDLER*****");
		joystickHandler = new JoystickHandler();
		Con::printf("%i Joystick(s) found", JoystickHandler->getCount());
	}
	else
	{
		Con::printf("JoystickHandler already initiated");
	}
}
```

The parameters of the `ConsoleFunction` method are the name of the new console function, the return type, the number of minimum input parameters, the number of maximum parameters and a small description of the console function displayed if the function is used incorrectly. It is important to note that the minimum amount of parameters is one, as the object calling the function is always passed to the console function. If the minimum and/or the maximum amount of input parameters is set to zero, the function can take an unlimited number of input parameters. Also, the first parameter does not need to be explicitly passed – this means that the first "real" parameter in the parameter array is located at index one, not zero.

The following console functions were defined to handle input from game control devices as well as controlling the devices themselves – this includes suspending the devices, shutting the game controllers down and sending Force Feedback commands:

- `InitJoystickHandler()`: Initiates an instance of the `JoystickHandler` class and reports back the number of game control devices found. If the `JoystickHandler` object is already initiated, the init command is skipped. The function does not have a return value.
- `SwitchJoysticks(OnOff)`: This method needs an integer variable as input. If the integer is zero, all connected game control devices are suspended. If a one is passed all connected game controllers are activated. The function returns false if no `JoystickHandler` object is initiated or if another value than one or zero is passed as parameter.
- `RebindJoysticks()`: This method basically first stops all functionality of the initiated static JoystickHandler object, resetting it back to the status prior to calling the start function of the object. By then calling start afterwards, the object runs through the whole loop of finding game control devices and setting them up. This enables the `JoystickHandler` object to find devices that have been connected since the last call and to remove references to game controllers that have been disconnected from the system.
- `StopJoysticks()`: This function stops all registered game control devices. It returns false if no JoystickHandler object was found. All device objects are released.
- `FFPlayRumble(id, length, gain)`: This method sends a constant rumble effect of specified length and strength to the game control device located at the index ID in the array of control devices. The function returns false if no `JoystickHandler` was found or if playing the Force Feedback effect failed.

Using the FMOD Ex Library with the Torque Game Builder Engine

Preface

The Mac OS X version (like any other version) of the Torque Game Builder supports sound playback. This is accomplished through *OpenAL*, the Open Audio Library.

"OpenAL is a cross-platform 3D audio API appropriate to use with gaming applications and many other types of audio applications." [OpenAL, 2007] *OpenAL* is based on three objects – listeners, sources and buffers. Buffers contain the actual audio data for playback, sources represent the audio emitting source positioned in 3D space and listeners represent the audio receiver, also placed in 3D space. [OpenAL, 2007]

Unfortunately, this approach is less than optimal for use in a 2D context, as the positioning of the sources and listeners still applies and is executed in 3D. *OpenAL* further does not support real panning or pitch shifts and sound effects have to be implemented before use. These drawbacks prompted the search for a substitute which might prove to be more suitable for the task at hand. A good candidate was found in FMOD, specifically the newest version of it, *FMOD Ex*.

FMOD Ex is a proprietary sound library developed and distributed by Firelight Technologies Pty, Ltd. It is the successor of the FMOD 3 sound library and presents a completely new structure to the developer. At the time of writing the library supports 22 input file formats and is available on all major platforms.

FMOD Ex allows easy separation between 2D and 3D sounds. Instead of having to define positions in 3D space for all kind of sources that are eventually used as 2D sounds one can simply ignore all 3D sound controls and just use methods like play and stop to control sounds, at most the pan of the sound, as well as its volume, have to be specified. Another major advantage of the *FMOD Ex* library over its predecessor is the possibility to use a build in DSP effects suite. This allows utilization of several effects supported by default:

- Oscillators
- Low-pass filters
- High-pass filters
- Echo
- Flange
- Distortion
- Normalizer
- Parametric EQ
- Real-time Pitch Shifter
- Chorus
- Reverb [Firelight Technologies, 2007b]

FMOD Ex also supports virtual voices which allow the usage of a great number of sounds on limited hardware, without worrying about handling the logic to switch sounds off and on themselves. [Firelight Technologies, 2007a]

Overview

The current version of *FMOD Ex* is 4.06.23 and was released on the 2nd of August 2007. It can be downloaded from the developer's website and is free of use without any kind of restrictions for non-commercial products.

The download package for the Mac OS X version includes several header files as well as two different versions of *FMOD Ex*, distributed as dynamic libraries – one including all plug-ins (libfmodex.dylib) and one without the plug-ins compiled into the library (libfmodexp.dylib). For ease of use the dynamic library including all plug-ins was used in this project.

The `FMODExPlugin` class, that was implemented to utilize the *FMOD Ex* sound library, allows samples to be loaded as well as playing, pausing and stopping them. Further, each sample can have one or more DSPs connected to it to change its output to match certain effects. As long as a sample is stored it can be played over again, but each sample can only be played once at any given time. If playback is still active on a sample while another play request is issued to that sample, previous playback stops and the sample is played from the beginning.

Further, the volume and pan of each sample can be controlled and each sample can also be muted. Samples can also report their volume and pan and their status on being paused and muted.

Implementation

The following implementation is in a small part based on the FMOD implementation of Zachary McMaster, found in [McMaster, 2002].

FMOD Ex uses eight objects to control sounds:

- `System`: This class represents the underlying system for communication with the soundcard and the file system. Speaker modes, output options, the number of voices in hard- and software as well as recording, geometry and network functions are handled within the class. It also controls the creation and retrieval of instances of the other four classes. System is also the only class allowed to start playback of sounds and DSPs.
- `Sound`: Sounds represents the actual audio data. In addition to the data they also contain information for 3D positioning, synchronization and looping. The usage of a unified sound object is one the improvements from FMOD 3.
- `Channel`: The channel class represents the audio channels that allow playback of sounds. Functionality specific to playback like pausing and stopping, as well as controlling the volume and pan are conducted by this class.
- `Soundgroup`: Soundgroups control a group of sounds. The main function of this class is to control the maximum audibility of the sounds within the group.
- `Channelgroup`: As soundgroups control sounds, `channelgroups` control `channels`. `Channelgroups` can set parameters like volume and pitch relative to the channels connected to the group and also override several parameters (volume, frequency, pan, reverb properties, 3D attributes and the speaker mix). A `channelgroup` can also pause, mute and stop all connected channels `channels`.
- `DSP`: `DSPs` are used to apply effects on channels or `channelgroups` by taking the input from those objects and modifying it. All the effects previously mentioned are enabled by adding a `DSP`. The `DSP` object offers methods to control the `DSP` input in addition to offering functionality to edit the parameters of a `DSP`.

- `Geometry`: The geometry class implements methods for importing geometry into the library for calculation of proper sound reflection and occlusion in a 3D environment.
- `Reverb`: This class is used to store the 3D space attributes necessary for reverb manipulation.

Out of these eight objects, the class `FMODExPlugin` uses five: `system`, `sound`, `channel`, `channelgroup` and `DSP`. Figure 9 shows the layout of the `FMODExPlugin`.

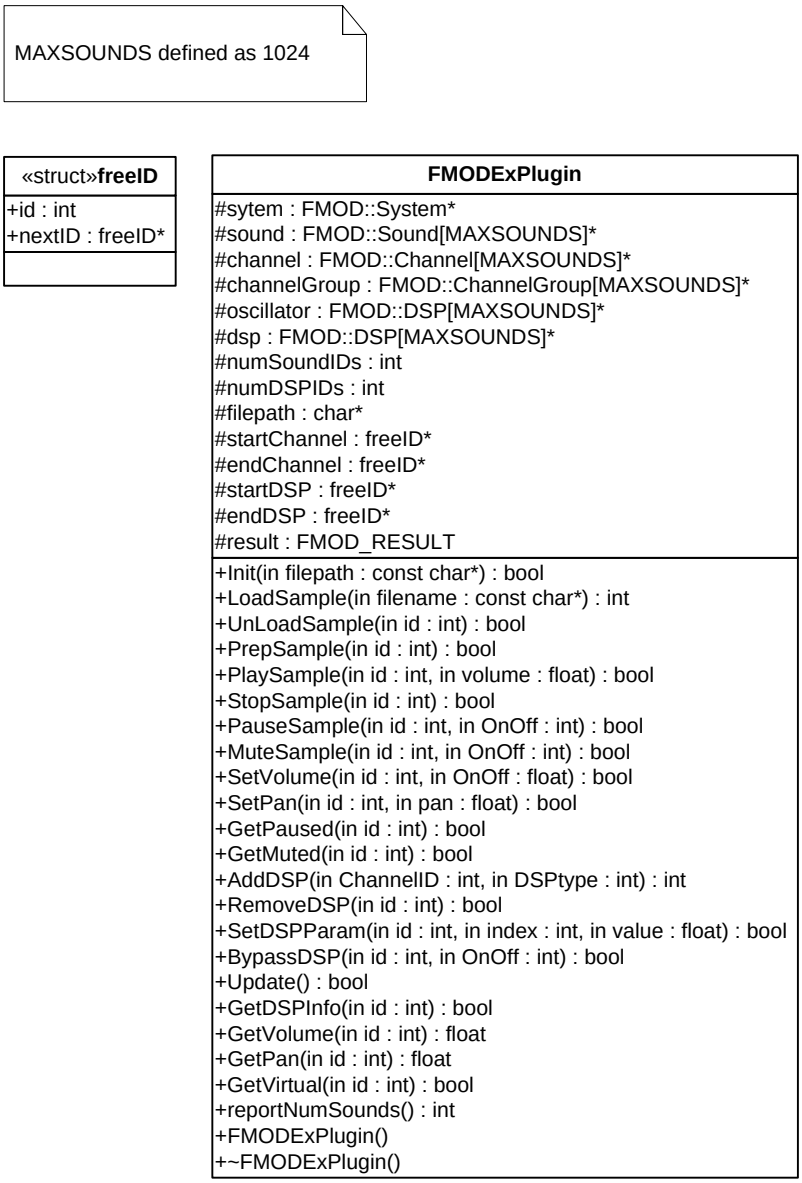

Figure 9: UML-Diagramm representing the FMODExPlugin Class

Creating a Connection to the Fmod Ex Libary

As mentioned before, the `system` class controls all communication with the sound hardware and the file system, as well as keeping track of the other objects. This necessitates the first step being the creation of a `system` object. In order to do this, an instance of the `FMODExPlugin` class has to be initialized. A call to the constructor will create all necessary variables, including two linked lists that will be used to keep track of available sample and `DSP` ids. Please note that samples are not equal to `sound` objects. The `FMODExPlugin` is laid out to play each sound-generating entity (be it a `sound`

object or a DSP generating sound, like an oscillator) in its own channel. For this reason each generator and its corresponding channel are treated as one sample.

Another variable of importance is the file path. By default the FMODExPlugin will search for requested sound files within the root directory of the game it is bound too. By specifying a file path (for example “mp3/”) it is possible to tell the object another search location.

The file path itself is passed on to the class when calling the Init method. After saving it in the filepath variable, the FMOD system object is created. The system object is then initiated normally with the maximum number of sounds audible at the same time passed as variable. The next code snippet illustrates this procedure (error handling has been omitted).

```
//create the FMOD system to communicate to FMOD
result = FMOD::System_Create(&system);
if ( result != FMOD_OK)
{
        //Error Handling
}
//initiate the system, using the defined maximum number of sounds with a normal initiation
//and no extra driver data
result = system->init(MAXSOUNDS, FMOD_INIT_NORMAL, 0);
if ( result != FMOD_OK)
{
        //Error Handling
}
Con::printf("Successfully initialized the FMOD Ex Plugin: %s \n", FMOD_ErrorString(result));
return true;
```

With this call, the *FMOD Ex* system is ready to use.

Loading and Playing Samples

In order to play a sample with the use of the FMODExPlugin class, three steps have to be taken. First, a sample has to be loaded, be it from a sound object source or a generating DSP. Next, the sample has to be prepared which basically means creating a channelgroup for it. Finally, the sample can be played at a specified volume (which can be changed at runtime). These three steps are accomplished by calling three different functions: LoadSoundSample/LoadOscillatorSample, depending on which type of sound generator is required, PrepSample and PlaySample.

LoadSoundSample loads the data from the file system into a *FMOD Ex* sound object, LoadOscillatorSample creates an oscillator DSP of specified type and at the set frequency rate. In both methods the first step lies in allocating a free ID – the methods return with a failure message immediately if the maximum number of available channels is reached. For this purpose, a linked list is created in the constructor with both a pointer for the first and the last element of that list. When the methods call for a free ID, a new temporary pointer is set on the first element in the list and a new sound or DSP object is created with the ID of the element. If the creation was successful, the starting point of the list is moved one element further up the list and the element still referenced by the temporary pointer is deleted (after storing the ID contained in that element to for use as a return value).

The list is composed of freeID structures. Each of these holds a variable of the name id with a unique number between 0 and the maximum number of sounds (set to 1024). Each of these numbers specifies a position in one of the arrays initialized in the class’ constructor. If a number is used, the corresponding struct is removed from the list.

Given one of these numbers, the `createSound` method or the `createDSPByType` method (for the `sound` and the `DSP` object respectively) of the system object are called, allocating the data within the specified position on the appropriate array. The following snippet shows the creation of a `sound` object.

```
//get the first available ID out of the list
freeID * firstFreeChannel = startChannel;
//create the sound
result = system->createSound(path, FMOD_SOFTWARE, 0, &sound[firstFreeChannel->id]);
```

The creation of the oscillator has a few more steps. The `createDSPByType` method creates an oscillator with the default values, requiring two more calls of the `setParameter` method of the `DSP` object to assign the right type of oscillator and the right frequency rate. These steps are shown in the next snippet, error handling was omitted.

```
//get the first available ID out of the list
freeID * firstFreeChannel = startChannel;
//Waveform type.  0 = sine.  1 = square. 2 = sawup. 3 = sawdown. 4 = triangle. 5 = noise.
if(type < 0 || type > 5)
{
	//Error Handling
}
result = system->createDSPByType(FMOD_DSP_TYPE_OSCILLATOR, &oscillator[firstFreeChannel->id]);
if( result != FMOD_OK)
{
	//Error Handling
}
result = oscillator[firstFreeChannel->id]->setParameter(0, type);
if( result != FMOD_OK)
{
	//Error Handling
}
result = oscillator[firstFreeChannel->id]->setParameter(1, rate);
```

Before the sound or oscillator can be played, `PrepSample` has to be called. This method loads a `channelgroup` with the `id` passed as parameter. The `system` object's method for creating a `channelgroup` requires a char variable as title for it. This problem is solved by converting the `id` parameter value into a string.

The `channelgroup` is needed to keep `DSP` objects reusable – if the `DSPs` were connected directly with a `channel`, they would need to be connected again every time the `channel` is replayed. The code illustrates creating the `channelgroup`.

```
char buffer[20];
snprintf (buffer, (size_t)20,"%d", id);
char * title = buffer;
result = system->createChannelGroup(title, &channelGroup[id]);
```

The `PlaySample` method brings the created sound or oscillator and `channelgroup` objects together. After ensuring that a `channelgroup` object exists, a `channel` is created, receiving the `sound` or `DSP` object as input and getting put on pause before starting. The `channel's` index is set to be reused with the FMOD_CHANNEL_REUSE flag. If the `PlaySample` method is called while the `channel` is still in use, the same `channel` will be used again, effectively restarting the playback. Oscillator `DSP` objects will unfortunately report errors when used this way, thus the `DSP` object's remove method is called prior to disconnecting the oscillator from any channels before reusing the channel occupied by the `DSP`.

The `channel` is then connected to the previously created `channelgroup`. As one can see the `sound`/`DSP` object, the `channel` and the `channelgroup` all reside in arrays of the same size. This makes it possible to use the same ID for all three arrays, making it much easier for the user to keep track by just requiring one "sample ID".

The `PlaySample` method also receives a second parameter to set the initial volume of the `channel` playing. Previous versions of FMOD used 256 levels of volume, with *FMOD Ex* the volume control is switched to a floating point value representation. The maximum volume is 1.0, the minimum value is 0.0. Thus, any number higher or lower is adjusted to these boundaries. The volume is then set for the `channel` and finally the `channel` is un-paused.

Due to the `channel` already being paused on creation, the output starts immediately with all connected `DSPs` affecting output and the volume at the specified value – the user cannot hear any adjustments at the beginning of the sample. The next source code snippet shows the operations described above, excluding error handling and controlling the volume's boundaries.

```
if(sound[id] == 0)
{
	//in this case an oscillator will be played->playDSP
	//remove all inputs to avoid cyclic errors
	result = oscillator[id]->remove();
	if (result != FMOD_OK)
	{
			//Error Handling
	}
	//play the dsp with a paused state
	result = system->playDSP(FMOD_CHANNEL_REUSE, oscillator[id], true, &channel[id]);
}
else if(oscillator[id] == 0)
{
	//this is a sound object -> playSound
	result = system->playSound(FMOD_CHANNEL_REUSE, sound[id], true, &channel[id]);
}
else
{
	//Error Handling
}
if (result != FMOD_OK)
{
	//Error Handling
}
//this connects the channel to the dsps meant to be used with it
result = channel[id]->setChannelGroup(channelGroup[id]);
if (result != FMOD_OK)
{
	//Error Handling
}
//control if volume value is within boundaries
//set the volume
result = channel[id]->setVolume(volume);
if (result != FMOD_OK)
{
	//Error Handling
}
//play the channel
result = channel[id]->setPaused(false);
```

Using DSPs to Modify the Output

Digital Signal Processing means "changing or analysing information which is measured as discrete sequences of number". [Bores Signal Processing, 2007] In the context of the *FMOD Ex* library digital DSPs are used to modify the input they receive and to output it again, thus producing various effects. *FMOD Ex* offers by default several effects which can be sorted into effects creating their own output

and effects modifying output from other sources. A sound generating effect, the oscillator, was already described in the prior chapter, this chapter deals now with effects affecting output.

DSPs can be added onto existing channels and channelgroups or, as implemented for the oscillators, be played straight through the system. Unfortunately, stopping a channel or letting a channel finish its playback will result in losing all connected DSPs. For this reason, channelgroups are introduced.

Every sample receives its own channelgroup that acts as a hook for the DSPs. After being created with the PrepSample function, channelgroups remain practically unchanged over the full lifetime of the sample - DSPs are added and removed from them, but a sample finishing, getting stopped, paused or muted does not affect the state of the channelgroup or the DSPs connected to it – the DSPs will still be exactly the same for every time the PlaySample function is called.

This means that although it may look to the user as though DSPs being added to the channel of the target sample, in reality the DSPs are getting all connected to the channelgroup which the channel is then connected to.

Following from the previous paragraphs, DSPs can be added to a sample from the moment its PrepSample function was called. If the DSPs are added before calling PlaySample, they will affect output from the beginning of the sample, but DSPs can be added to and removed from a sample at any time during playback too.

The following effects are available through the FMODExPlugin implementation, sorted by value of the ID that has to be passed to choose them (starting at zero):

- FMOD_DSP_TYPE_LOWPASS: A typical lowpass filter – attenuates high frequencies and passes low frequencies.
- FMOD_DSP_TYPE_HIGHPASS: The lowpass filter reversed – high frequencies are passed, low frequencies are reduced.
- FMOD_DSP_TYPE_ECHO: This effect creates an echo on the input.
- FMOD_DSP_TYPE_FLANGE: “Two identical signals are mixed together, with one of the signals time-delayed by a small and gradually changing amount.” [Cole, 2006]
- FMOD_DSP_TYPE_DISTORTION: This effect distorts the input.
- FMOD_DSP_TYPE_NORMALIZE: Normalize or amplify the input to a certain level.
- FMOD_DSP_TYPE_PARAMEQ: This effect represents a parametric equalizer, attenuating or amplifying selected frequency ranges.
- FMOD_DSP_TYPE_PITCHSHIFT: This effect changes the pitch of the input without slowing the input down or accelerating it.
- FMOD_DSP_TYPE_CHORUS: The input is mixed with identical copies that are slightly shifted in pitch and offset by a few milliseconds.

DSP effects are added by calling the AddDSP method with two parameters, the first specifying the ID of the channelgroup the DSP should be added to and the second defining the type of DSP that should be used. As well as the sample ID specified by the user, DSPs have their own list of IDs available to them, managed in the same way as the list containing the sample IDs. The AddDSP function will return the ID used for that specific DSP. The next source code sample shows the procedure until the

attachment of the DSP to the channelgroup, with error handling and the DSP type cases in the switch statement omitted.

```
if(numDSPIDs == MAXSOUNDS)
{
	//Error Handling
}
if(ChannelID < 0 || DSPtype < 0)
{
	//Error Handling
}
freeID * firstFreeDSP = startDSP;
startDSP = firstFreeDSP->nextID;
if(ChannelID >= MAXSOUNDS)
{
	//Error Handling
}
if(DSPtype > 11)
{
	//Error Handling
}
FMOD_DSP_TYPE type;
switch(DSPtype)
{
	//determine the type of the DSP
}
result = system->createDSPByType(type, &dsp[firstFreeDSP->id]);
if (result != FMOD_OK)
{
	//Error Handling
}
result = channelGroup[ChannelID]->addDSP(dsp[firstFreeDSP->id]);
```

Unlike using channels, channelgroups and sound/DSP (oscillator) objects together transparently under the term “sample”, the user has to track which DSP was added on which channel. This is a result of allowing users to use any number of DSPs on a channel, starting at zero up to the maximum number of DSPs available. Allowing only a certain set of DSPs to be used per channel (one of each type for example) would restrict the user, allowing an unlimited number of DSPs would require an overhead out of the scope of this project. By giving control to the user, a compromise is made – the user has an acceptable number of DSPs available, while the overall number of DSPs and the functionality needed to manage these stays within reasonable boundaries.

All DSPs created with the AddDSP function are initialized with default values that might fit but most of the time at least one parameter has to be adjusted. Each type of DSP has up to eight parameters and none of them share a common set of values. To adjust DSPs after creation, SetDSPParam is used. The method takes an ID, an index and a value as input parameters. It changes the value at a given index of a DSP identified by the id. Of course, this has to be done after the DSP specified by the id was created. It is also important to note that every value of a DSP can be changed at any moment, no matter if the DSP is currently in use or not. Changes are propagated immediately. For an overview of the parameters available for each DSP please refer to figure 13 in Appendix A.

Removing Samples and DSPs

FMOD Ex can by default use an arbitrary number of sounds at the same time without running out of channels to play them in, or having to steal channels from other sounds. Furthermore the value stated when initializing the *FMOD Ex* System object does not represent the maximum number of available channels but the maximum number of channels allowed to be played at the same time.

Even the size of that number does not significantly affect performance, as voices that are played but not audible (as determined by the *FMOD Ex* `system` object) are virtualized. This means that playback of these `channels` is simulated until some part is actually audible. At this moment, the `channel` is switched back to real playback until it becomes inaudible (and by that, set to the virtual state) again.

Nevertheless, another issue remains. The `createSound` method of the System object will try to fully load `sounds` into memory by default, thus making *FMOD Ex* memory-restricted. There are possibilities to access sound files as streams or to keep them in a compressed state within the memory, but these methods require a higher CPU usage which is not advisable when many `sounds` are played at once, a scenario present in the project using the `FMODExPlugin` class.

For this reason, the total number of `sounds` available to users through the `FMODExPlugin` class is limited to the maximum number of `sounds` allowed to play at the same time. Because of the restricted range of IDs available, a way to remove samples must be offered to the users. This also applies for using `DSPs`, as the maximum number of `DSPs` available equals the maximum number of sounds.

In order to free up IDs and memory, two methods are available to the user. `UnLoadSample`, as the name already indicates, allows samples to be released and their ID reused, `RemoveDSP` does the same for `DSPs`.

`UnloadSample` is callable at any stage of the loading, preparing and playing process. The method will always follow the same sequence. First, it will check if a `channelgroup` exists at the ID specified (by checking whether the value at that position is not zero, the default value for not initialized objects). Given an existing `channelgroup`, a call is made to the `stop` function of the `channelgroup` to stop any playback making use of that specific `channelgroup`. If the call is successful the `channelgroup` is instructed to release the memory it is occupying with a call to its `release` function. The last step taken is setting the array position the `channelgroup` was residing at to zero, the default value.

After taking care of the possibly existing `channelgroup`, a similar course of action is taken to remove a potential `channel` object. This object does not contain a `release` method, so it is sufficient to call the `stop` method (once again ensuring that the sample is actually stopped) and set the value in the array position to zero.

The last step in the hierarchy is releasing the `sound` or oscillator `DSP` object. After determining which kind of object is used by the sample in question, the `release` method is called on the object and the value of the position in the appropriate array is set to zero.

By completing the last step, the ID held by the sample can be considered available again for use by the `LoadSoundSample` and `LoadOscillatorSample` methods. To allow these functions to see the ID, it must be appended to the list of free IDs.

One special case exists, however, in which appending is not possible. If all IDs in the list are used up, the pointer usually pointing to the start and end element of the list are no longer available. These hooks must be recreated again by using the now free ID as start and end point. To find out if this case is applicable, the number of used sample is compared to the maximum number of available samples. If both values are the same, the list needs to be recreated. The following source code shows this procedure, error handling was omitted.

```
freeID * newLastChannel = new freeID;
newLastChannel->id = id;
newLastChannel->nextID = NULL;
//if there are no more free ids, create a new starting point for the list.
if(numSampleIDs == MAXSOUNDS)
{
	endChannel = newLastChannel;
	startChannel = newLastChannel;
}
//add id to the back of the list, set pointer from previous end to that node
else
{
	endChannel->nextID = newLastChannel;
	endChannel = newLastChannel;
}
```

Controlling the FMODExPlugin Object through the Console

Even more than the previously explained JoystickHandler class, the `FMODExPlugin` class needs to offer functions that allow the user to access the methods discussed above. Again, console functions are used to allow this access. The process of the implementation of these console functions is the same as the one already explained in the `JoystickHandler` class, so it will not be repeated here.

The following functions are available to the user from within the console:

- `InitFMODExPlugin(filepath)`: This function creates a `FMODExPlugin` object and calls the init method of that object, with the filepath variable (a char array specifying the relative path in the file system from the root of the game to the folder containing the sound files) passed to it as input parameter. If the creation or initialization fails, the function will return false, otherwise true.
- `FMODExPluginUpdate()`: This function should be called at every frame. It contains a call to the update method of the System object. It does not have a return value and also does not need any input parameters.
- `LoadSoundSample(filename)`: Using the filename passed to it as input parameter this console function calls the `LoadSample` method of the `FMODExPlugin`. It returns the ID of the sample created with that sound object and '-1' on failure.
- `LoadSoundSamples(Samples)`: Unlike `LoadSoundSample`, this function allows a batch of samples to be loaded at once. Essentially, instead of passing only one filename at a time an arbitrary number of filenames (configured as char arrays) can be passed, separated with commas. The function will then call the `LoadSample` method for every one of these filenames. Instead of returning the ID of the created element, this function returns Boolean values, thus it cannot really be used programmatically, but it is more intended to be actually used by the user inside the console, as it writes the ID of every file loaded to the console. If one file cannot be loaded, the function quits, but all files loaded up to this point are still available.
- `LoadOscillatorSample(type, rate)`: This function creates an oscillator sample of the specified type at the specified frequency rate. It returns the ID to the user, if the function fails, '-1' will be returned.
- `PrepSample(id)`: `PrepSample` is used to call the `FMODExPlugin` method of the same name. It will return true on success and false otherwise.
- `PlaySample(id, volume)`: This calls the `PlaySample` method of the `FMODExPlugin`, passing its input parameters on to that function. It will return true on success and false otherwise.

- `UnloadSample(id)`: Calls the appropriate method to unload the sample specified by the `id` passed as input parameter. Again, this function will return true on success and false otherwise.
- `StopSample(id)`: Stops the sample located at the specified ID. This function will return true on success and false otherwise.
- `PauseSample(id, OnOff)`: This function can be used to pause a sample. The sample stored at the specified ID will be paused or un-paused, depending on the `OnOff` parameter having the value one (pause) or zero (un-pause). Again, this function returns true if running correctly, false otherwise. If the same state is requested twice in a row (pausing a sample twice, for example), the function will still return true, if no other errors occur.
- `MuteSample(id, OnOff)`: `MuteSample` works the same way as `PauseSample`, except the sample is muted instead paused (or un-muted, if a zero is passed as second parameter). Also, if the same request is repeated, the function will also react the same way as `PauseSample` (return true if no other errors are reported). It is important to note that setting the sample volume to zero and muting the sample are considered two different operations, thus, a sample with a volume of zero can still be muted (and the other way around). A sample with a volume of zero will also not return true when `GetMuted` is called on it and it was not muted before.
- `SetSampleVolume(id, Volume)`: Using the ID passed to identify a specific sample, this function tries to set the volume of that sample to the value stored in the second parameter. This value has a range of zero to one and is passed as a float variable. The function returns true when operating correctly, false otherwise.
- `SetSamplePan(id, Pan)`: This function determines the position of a sample between the left and the right output. Depending on whether the sample is pushed more to the left (by defining the Pan variable closer to minus one) or the right (setting Pan in the vicinity of plus one) the sample will be played louder through the left or right output, with minus one and plus one defining the borders on both sides. By default all sample have a pan of zero and will be played equally loud through both outputs. The function returns true on successful completion and false otherwise.
- `GetPaused(id)`: This function returns the current status on being paused (return value true) or playing (return value false) of the sample specified.
- `GetMuted(id)`: This function returns the current status on being muted of the selected sample. If the sample is muted the function will return true and false otherwise.
- `GetSampleVolume(id)`: `GetSampleVolume` returns the current value of the volume set for the selected sample as a float value.
- `GetSamplePan(id)`: Returns the current pan value of the sample specified by the `id` parameter as a float variable.
- `GetVirtual(id)`: This function reports if a channel was set to virtual playback (return value true) or not (return value false).
- `AddDSP(ChannelID, DSPType)`: As the name already implies, this function adds a DSP of a specific type (as set by the second function parameter) to the channel specified by the `ChannelID` variable. If the function can execute correctly, an integer value containing the ID of the DSP will be returned. If any errors occur, the function return value will be set to minus one.

- `RemoveDSP(id)`: This function represents the counterpart to `AddDSP` and is responsible for removing the DSP specified by the ID. Executed correctly, the function returns true and the return value will be false if any errors occur.
- `EditDSP(id, Params)`: Every DSP type has its own set of variables defining several characteristics of that DSP. This function is dependent on the user to specify the number of input parameters to be used with the specific DSP defined by the `id` variable. That means that the user has to know how many parameters a DSP needs. All DSP parameters are expressed as float variables and the function will always start with setting the DSP parameter at the index zero. The function will return true on successful completion and false otherwise. All DSP parameters up to the one throwing the error will be set to the new values defined by the passed parameters.
- `BypassDSP(id, OnOff)`: Sometimes one or more DSPs added to a channel are not meant to be played for a short time and will be played again shortly after with no values changed. Instead of removing them and adding them back at a later point in time, these DSPs can be bypassed by using this function. The DSP at the position determined by the `id` variable will be bypassed if the second function parameter, `OnOff`, is set to one and will be put back onto the channel if `OnOff` is set to zero. The function itself will return true on success and false otherwise.
- `GetDSPInfo(id)`: This function writes a list of the parameters of the specified DSP into the console window. The parameters values are listed in the same order they are stored in the enumeration belonging to the selected DSP. The function returns false if the DSP does not exist or the retrieval of the parameters fails and true otherwise.
- `OscillatorSetRate(id)`: Unlike samples incorporating a Sound object, the output of oscillators can be edited directly without the use of further DSPs. This function allows the frequency rate of an oscillator sample to be changed at any time. It will return true on success and false otherwise.
- `GetCurrentSampleNumber()`: This function returns the number of samples currently in use as an integer return value.

The Prototype

The goal of all these additions to the TGB engine is to support the development of a prototype of a music-based game. The development of *Radiolaris*, as this prototype is called, has been carried out by Martin Pichlmair and Fares Kayali. At the time of writing, the game was going through many revisions, its look, sound, and game play changing significantly from the early versions.

The current version is pictured in figure 9. The game is meant to be played by two players simultaneously, who try to destroy each other's "mother ship" by shooting at it. In order to create shots, players first have to collect gems emitted by the "mother ships". These gems create a tail on the player's avatar, similar to the well known game *Snake*. These gems are the basis for all weapons and defense mechanisms the players have at their disposal. To create a specific item, the player has to press the appropriate button on the gamepad at the right time – only if the button was pressed in beat with the rhythm of the game's music is the requested item created, otherwise the player's avatar only emits some debris and loses its tail.

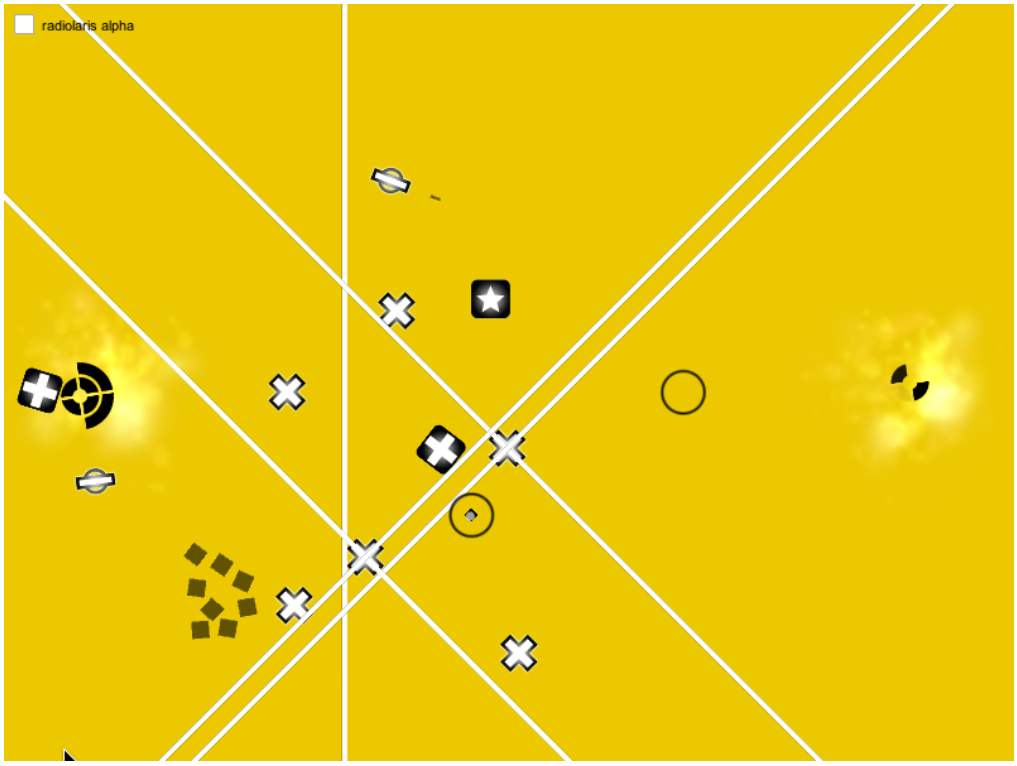

Figure 10: *Radiolaris* in its most current revision (at the time of writing)

The objects created by the player fulfill different offensive and defensive actions in the game, such as being an offensive or defensive tower. More importantly, they also emit sound and become part of the games soundtrack.

The game in its current state incorporates several of the qualities discussed in [Pichlmair, et al., 2007]. For one, *Radiolaris* contains an active score – players can determine the timing of sound effects playing by the position of the object in the game. The white stripe visible in the screenshot in figure 9 moves steadily between the left and the right side of the screen. If it touches an object positioned by the players, that object is "fired" and emits a sound. Another quality possessed by the game is the rhythm based game play – all of the players' actions have to be executed in a certain rhythm to generate the wanted object instead of debris. The game also contains a third quality,

quantization. The playback of every sound in the game is arranged to be consistent with the overall beat of *Radiolaris*.

The prototype makes heavy use of the additional features added to the TGB engine. Currently, gamepads are the only way to control the game, as *Radiolaris* requires at least an analog stick to move the player's avatar. As already mentioned, the game allows two players to play simultaneously, thus supporting two gamepads connected to the system. *Radiolaris* further uses the Force Feedback functions present in the utilized Logitech *RumblePad 2* gamepads to hint the game's rhythm at the player by sending short rumble effects on every beat.

As for the sound in the game, *Radiolaris* uses the *FMOD Ex* library exclusively. The prototype in its current version uses the provided functions (namely two methods to load, play, and stop samples, as well as the volume and pan controls) to create a sequencer written in TorqueScript. By using TorqueScript to circumvent the sound functionality already provided with the engine, the *OpenAL* library could be conveniently ignored.

At the time of writing this thesis, no DSPs are used in the game, including oscillators. *Radiolaris* is still in a prototyping stage though, so further versions of the game could definitely include these functions of the *FMOD Ex* engine as well.

Conclusion

In this part of the thesis I present some technical aspects of creating a music-based game. I start with a short overview of the TGB engine and explain two disadvantages of that system, missing game controller support and an unsuitable audio library.

The next two chapters deal with overcoming these disadvantages. First, I introduce a method to realize game controller support in the TGB engine by using the HID functions provided by the IOKit framework by Apple. I implement the complete functionality of the target hardware, the Logitech *RumblePad 2,* and also include functions for the use of Force Feedback with the help of Apple's Force Feedback Framework.

By including the FMOD Ex audio library in addition to the OpenAL library in the TGB engine I can solve the problem of having an audio library unsuitable for two-dimensional games. I make most the functionality of FMOD Ex available for the use through TorqueScript, the scripting language of the TGB engine, including the ability the load, play, pause, mute, and remove sounds as well setting the volume and the pan of a sound object. Furthermore I include most of the effects provided by the FMOD Ex library.

I conclude this part of my thesis with a short presentation of the prototype developed on the basis of the improved TGB engine. *Radiolaris*, developed by Pichlmair and Kayali, was created to explore different game play aspects of music-based games and is still under development at the time of writing.

Appendix A

Apple's Force Feedback Framework

Figure 11: A small overview over the functions of Apple's Force Feedback Framework

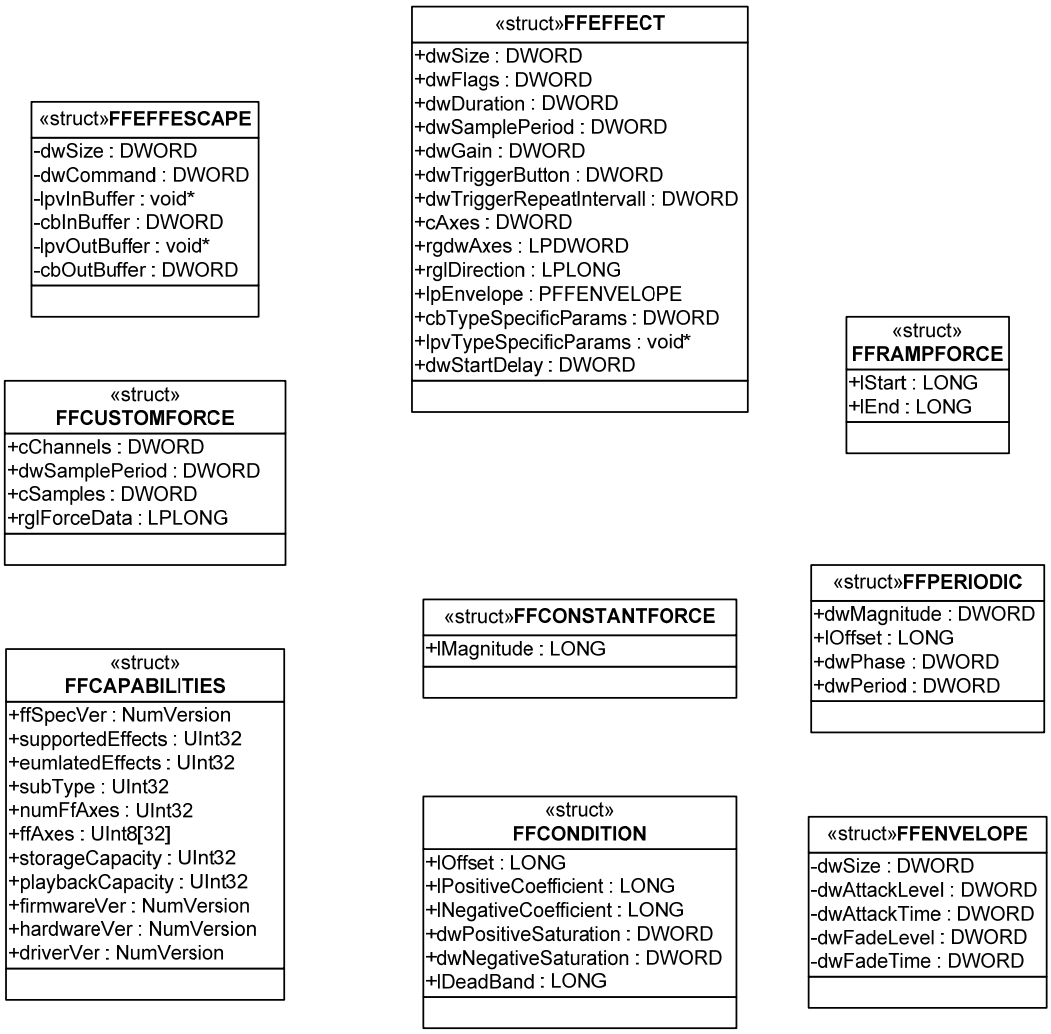

Figure 12: These are the structs used by the Force Feedback Framework. Please note that all listed structs have a type definition of their own name

Variables of the FMOD Ex DSPs used within the FMODExPlugin Class

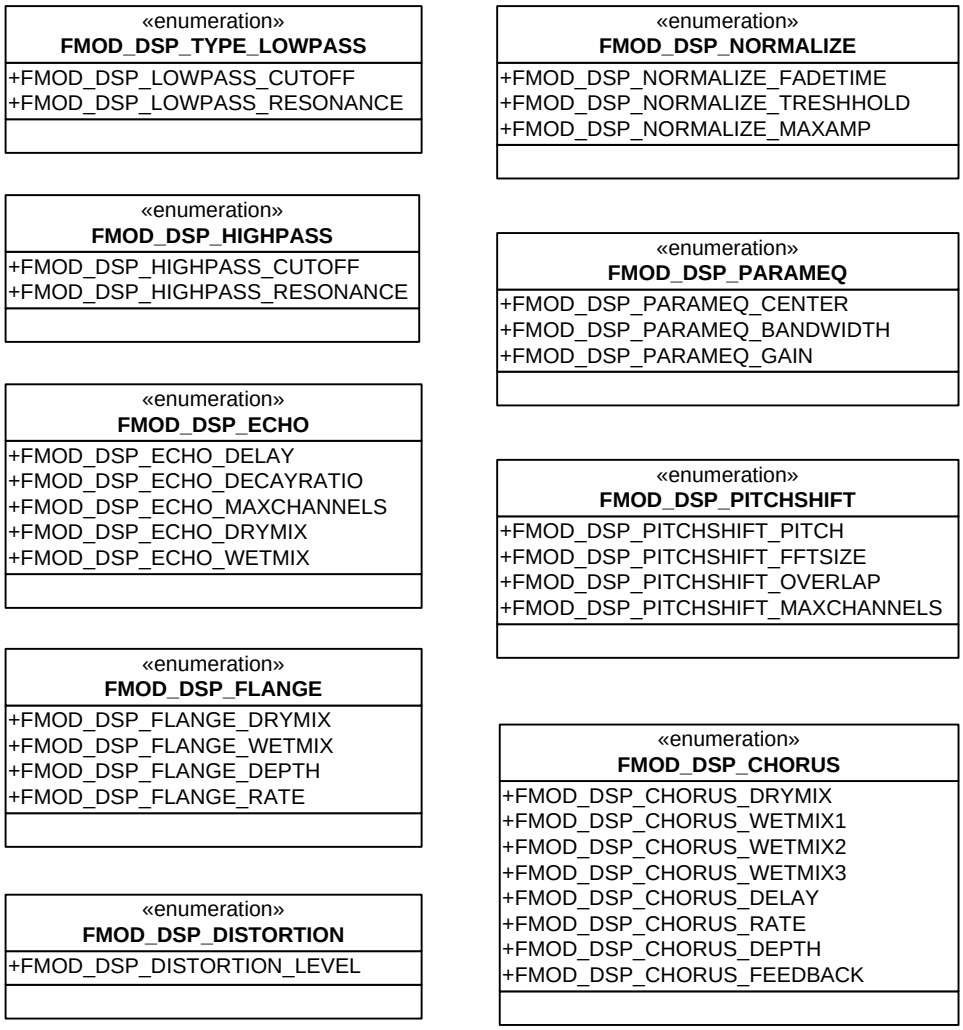

Figure 13: The parameters for the different DSPs are stored in enumerations. Please note that only the DSPs used in FMODExPlugin are listed here

References

Adobe. 2003. A Digital Audio Primer. *adobe.com.* [Online] 2003. [Cited: January 9, 2008.] http://www.adobe.com/products/audition/pdfs/audaudioprimer.pdf.

Akins, Joseph. 2004. A Brief History of MIDI. [Online] April 13, 2004. [Cited: January 10, 2008.] http://www.mtsu.edu/~jakins/4190/4190midihistory.html.

ALive! 2001. Audigy Review - EAX and Gaming. *ALive!* [Online] December 31, 2001. [Cited: January 6, 2008.] http://alive.singnet.com.sg/audigy/review/eax.htm.

—. 2003. Aureal vs Creative. *ALive!* [Online] January 8, 2003. [Cited: January 5, 2008.] http://alive.singnet.com.sg/features/aureal-creative.htm.

Apple Inc. 2003a. Accessing Hardware From Applications: Putting It All Together: Accessing A Device. *Apple Developer Connection.* [Online] 2007-02-08, Apple Inc., May 15, 2003a. [Cited: December 1, 2007.] http://developer.apple.com/documentation/DeviceDrivers/Conceptual/AccessingHardware/AH_Finding_Devices/chapter_4_section_4.html.

—. 2007. Force Feedback Device Access Reference. *Apple Developer Connection.* [Online] Apple Inc., October 31, 2007. [Cited: December 1, 2007.] http://developer.apple.com/documentation/DeviceDrivers/Reference/ForceFeedback/index.html.

—. 2006. Getting Started with Games. *Apple Developer Connection.* [Online] Apple Inc., May 23, 2006. [Cited: December 1, 2007.] http://developer.apple.com/referencelibrary/GettingStarted/GS_games/index.html.

—. 2001. HID Class Device Interface Guide. *Apple Developer Connection.* [Online] 2006-10-13, May 1, 2001. [Cited: December 1, 2007.] http://developer.apple.com/documentation/DeviceDrivers/Conceptual/HID/hid.pdf.

—. 2003b. I/O Kit Fundamentals. *Apple Developer Connection.* [Online] 2007-05-17, September 18, 2003b. [Cited: December 1, 2007.] http://developer.apple.com/documentation/DeviceDrivers/Conceptual/IOKitFundamentals/IOKitFundamentals.pdf.

Atari. 1982. POKEY Datasheet. *The Atari Historical Society.* [Online] March 9, 1982. [Cited: December 28, 2007.] http://www.ionpool.net/arcade/atari_docs/pokey_datasheet.pdf.

AtariAge. 2006. Atari 7800 FAQ. *AtariAge.* [Online] October 11, 2006. [Cited: December 30, 2007.] http://www.atariage.com/7800/faq/index.html.

AudioGames.net. 2008a. Bit Generations Sound Voyager. *AudioGames.* [Online] 2008a. [Cited: January 7, 2008.] http://www.audiogames.net/db.php?action=view&id=SoundVoyager.

—. 2008b. Real Sound - Kaze No Riglet. *AudioGames.* [Online] 2008b. [Cited: January 7, 2008.] http://www.audiogames.net/db.php?action=view&id=realsoundkazenoregret.

Baer, Ralph H. 1998. The SIMON Story. *Classic Consoles Center.* [Online] December 12, 1998. [Cited: January 3, 2008.] http://www.dieterkoenig.at/ccc/english/se_story_simon.htm.

Barish, Jeffrey. 1998. New Sound Technology for PCs. *Gamasutra.* March 6, 1998, Vol. 2, 10.

Belinkie, Matthew. 1999. Video game music: not just kid stuff. *VGMusic.* [Online] Videogame Music Archive, December 15, 1999. [Cited: December 30, 2007.] http://www.vgmusic.com/vgpaper.shtml.

Bemanistyle. 2008. beatmania for JP Arcade Game Information. *Bemanistyle.* [Online] Too Much Dedication, 2008. [Cited: January 4, 2008.] http://www.bemanistyle.com/gameinfo/game.php?game_id=77.

Bores Signal Processing. 2007. Introduction to DSP - basics: what is DSP? *Bores Signal Processing.* [Online] Bores Signal Processing, March 20, 2007. [Cited: December 08, 2007.] http://www.bores.com/courses/intro/basics/1_whatis.htm.

Bramwell, Tom. 2004. Review - SingStar // PS2 /// Eurogamer. *Eurogamer.* [Online] June 1, 2004. [Cited: January 6, 2008.] http://www.eurogamer.net/article.php?article_id=55663.

Bridgett, Rob. 2005. Hollywood Sound: Part Three. *Gamasutra.* [Online] CMP Media LLC., October 12, 2005. [Cited: January 7, 2008.] http://www.gamasutra.com/features/20051012/bridgett_01.shtml.

Brightman, James. 2007. Video Game Music Today & Tomorrow. *GameDaily.* [Online] May 8, 2007. [Cited: January 6, 2008.] http://www.gamedaily.com/articles/features/video-game-music-today-and-tomorrow/70389/?biz=1.

Burkart, Jörg. 2006. Hardware. *sega-32x.de.* [Online] February 8, 2006. [Cited: January 3, 2008.] http://www.sega-32x.de/hardware.htm.

Calvert, Justin. 2004. Taiko Drum Master for PlayStation 2 Review. *Gamespot.com.* [Online] CNET Networks, October 28, 2004. [Cited: January 7, 2008.] http://www.gamespot.com/ps2/puzzle/taikodrummaster/review.html?tag=tabs;reviews.

Chan, Norman. 2007. *A Critical Analysis of Modern Day Video Game Audio.* Nottingham : Department of Music, University of Nottingham, 2007.

Chester, Nick. 2007. Destructiod review: Jam Sessions. *Destructiod.* [Online] October 22, 2007. [Cited: January 7, 2008.] http://www.destructoid.com/destructoid-review-jam-sessions-49986.phtml.

CNET. 2001. Sony PlayStation 2 Specs. Consoles Specifications. *CNET Reviews.* [Online] CNET Networks, November 20, 2001. [Cited: January 4, 2008.] http://reviews.cnet.com/consoles/sony-playstation-2/4507-10109_7-30012264.html.

—. 2008. Sony PSP Specs. Consoles Specifications. *CNET Reviews.* [Online] CNET Networks, 2008. [Cited: January 8, 2008.] http://reviews.cnet.com/consoles/sony-psp/4507-10109_7-30895581.html.

Cole, Jason. 2006. Audio Effects – Noise Gate & Flange. *Buzzle.com.* [Online] Buzzle.com, September 1, 2006. [Cited: December 8, 2007.] http://www.buzzle.com/editorials/8-31-2006-107181.asp.

Collins, Karen. 2007a. An introduction to sampling. *GamesSound.com.* [Online] 2007a. [Cited: January 9, 2008.] http://www.gamessound.com/bitdepth.pdf.

—. 2007b. An introduction to sound waves. *GamesSound.com.* [Online] 2007b. [Cited: January 9, 2008.] http://www.gamessound.com/soundwaves.pdf.

—. 2007c. An Introduction to the Participatory and Non-Linear Aspects of Video Game Audio. [book auth.] Stan Hawkins and John Richardson. *Essays on Sound and Vision.* Helsinki : Helsinki University Press, 2007c.

Console Database. 2005. FM Towns Marty/FM Towns Marty 2. *Console Database.* [Online] April 18, 2005. [Cited: January 2, 2008.] http://www.consoledatabase.com/consoleinfo/fujitsufmtownsmarty/.

Cook, Daniel. 2005. Common Game Prototyping Pitfalls. *Lost Garden.* [Online] August 21, 2005. [Cited: January 22, 2008.] http://lostgarden.com/2005/08/common-game-prototyping-pitfalls.html.

Cordeira, Jim. 2008. Seamn Review for Dreamcast. *Gaming Age.* [Online] 2008. [Cited: January 5, 2008.] http://www.gaming-age.com/cgi-bin/reviews/review.pl?sys=dreamcast&game=seaman.

Creative Labs. 2008. EAX - About EAX. *Sound Blaster - Hear to Believe.* [Online] Creative Technology, 2008. [Cited: January 6, 2008.] http://www.soundblaster.com/eax/abouteax/.

Creative Labs Technical Support. 1999. Frequently Asked Questions for SB AWE32. *Gamedev.* [Online] Gamedev.net, July 7, 1999. [Cited: Januar 1, 2008.] http://www.gamedev.net/reference/articles/article445.asp.

Creative Technology. 2004. Creative Expands Audio Creation Arsenal with New Console Game Development Solution. *PR Newswire.* [Online] United Business Media , March 24, 2004. [Cited: January 8, 2008.] http://www.prnewswire.com/cgi-bin/stories.pl?ACCT=104&STORY=/www/story/03-24-2004/0002133817&EDATE=.

Cullen, Micheal. 2006. Q. Can you explain the origins of wavetable, S&S and vector synthesis? *Sound On Sound.* February, 2006.

Davis, Rayn. 2003. Amplitude for PlayStation 2 Review. *Gamespot.com.* [Online] CNET Networks, March 26, 2003. [Cited: January 8, 2008.] http://www.gamespot.com/ps2/puzzle/amplitude/review.html?tag=tabs;reviews.

Davis, Ryan. 2004. Donkey Konga for GameCube Review. *Gamespot.* [Online] CNET Networks, September 27, 2004. [Cited: January 7, 2008.] http://www.gamespot.com/gamecube/puzzle/donkeykonga/review.html?tag=tabs;reviews.

—. 2006. Electroplankton for DS Review. *Gamespot.com.* [Online] CNET Networks, January 6, 2006. [Cited: January 7, 2008.] http://www.gamespot.com/ds/puzzle/electroplankton/review.html?tag=tabs%3Breviews&page=2.

—. 2001. Frequency for PlayStation 2 Review. *Gamespot.* [Online] CNET Networks, November 27, 2001. [Cited: January 6, 2008.] http://www.gamespot.com/ps2/puzzle/frequency/review.html?tag=tabs;reviews.

—. 2002. Gitaroo Man for PlayStation 2 Review. *Gamespot.com.* [Online] CNET Networks, February 14, 2002. [Cited: January 17, 2008.] http://www.gamespot.com/ps2/puzzle/gitarooman/review.html.

DeRienzo, David. 2007. Hardcore Gaming 101: Michael Jackson's Moonwalker. *GameSpy.com.* [Online] IGN Entertainment, December 30, 2007. [Cited: December 39, 2007.] http://hg101.classicgaming.gamespy.com/moonwalker/moonwalker.htm.

Dobson, Jason. 2006. Dolby Audio Support Clarified For PS3, Wii. *Gamasutra.* [Online] CMP Media LLC., September 22, 2006. [Cited: January 7, 2008.] http://www.gamasutra.com/php-bin/news_index.php?story=10975.

Duryee, Tricia. 2004. Nintendo DS will be on shelves in time for Christmas season. *The Seattle Times.* [Online] The Seattle Times Company, September 22, 2004. [Cited: January 8, 2008.] http://seattletimes.nwsource.com/html/businesstechnology/2002042597_nintendo22.html.

Eisler, Craig. 2006. DirectX Then and Now (Part 1). *Craig's Musings.* [Online] February 20, 2006. [Cited: January 3, 2008.] http://craig.theeislers.com/2006/02/directx_then_and_now_part_1.php.

Entertainment Software Association. 2008. Sales & Genre Data. *Entertainment Software Associations.* [Online] 2008. [Cited: January 16, 2008.] http://www.theesa.com/facts/sales_genre_data.php.

Errede, Steven. 2007. Physics 199 Physics of Music - Lecture Notes Week VII. *University of Illinois.* [Online] 2007. [Cited: January 9, 2008.] http://online.physics.uiuc.edu/courses/phys199pom/Lecture_Notes/P199POM_Lect7_Ch7.pdf.

Fayzullin, Marat, et al. 1998. Everything You Always Wanted To Know About GAMEBOY but were afraid to ask. *Emu-Docs.* [Online] March 12, 1998. [Cited: January 5, 2008.] http://www.emudocs.org/Game%20Boy/Gbspec.txt.

Filipantis Jr., Frank. 1994. 4.2 HRTF Implementation. *Design and Implementation of an Auralization System with a Spectum-Based Temporal Processing Optimization.* [Online] May 1994. [Cited: January 12, 2008.] alumni.caltech.edu/~franko/thesis/Chapter4.html.

FilmSound.org. 2007. Diegetic and non-diegetic sounds. *FilmSound.org.* [Online] July 16, 2007. [Cited: January 14, 2008.] http://filmsound.org/terminology/diegetic.htm.

Firelight Technologies. 2007a. FMOD Ex. [Dokument]. September 14, 2007a.

—. 2007b. FMOD Ex Feature List. *FMOD music & soundeffects system.* [Online] Firelight Technologies, 2007b. [Cited: December 06, 2007.] http://www.fmod.org/index.php/products/fmodex.

FLAC. 2007. introduction. *flac - free lossless audio codec.* [Online] December 12, 2007. [Cited: January 11, 2008.] http://flac.sourceforge.net/features.html.

Gamasutra. 2001. GDC News. *Gamasutra.* [Online] CMP Media LLC., March 24, 2001. [Cited: January 8, 2008.] http://www.gamasutra.com/gdc2001/news.html.

Gamespot.com. 2008. Search results for 'Guitar Hero'. *Gamespot.com.* [Online] CNET Networks, January 7, 2008. [Cited: January 7, 2008.] http://www.gamespot.com/search.html?type=11&stype=all&tag=search%3Bbutton&om_act=convert&om_clk=gssearch&qs=Guitar+Hero&x=0&y=0#game.

Garage Games. 2007a. Input system constants. *Torque Game Engine Documentation.* [Online] 1.3.x, Garage Games, 2007a. [Cited: December 1, 2007.] http://www.garagegames.com/docs/tge/engine/group__input__constants.php.

—. 2007b. InputEvent Struct Reference. *Torque Game Engine Documentation.* [Online] 1.3.x, Garage Games, 2007b. [Cited: December 1, 2007.] http://www.garagegames.com/docs/tge/engine/structInputEvent.php.

—. 2008a. Torque Game Builder. *Garage Games.* [Online] Garage Games, 2008a. [Cited: January 16, 2008.] http://www.garagegames.com/products/torque/tgb/.

—. 2008b. TorqueScript. *Torque Game Builder Features.* [Online] Garage Games, 2008b. [Cited: January 17, 2008.] http://www.garagegames.com/products/torque/tgb/features/torquescript/.

Gardner, William G. 1997. *3-D Audio Using Loudspeakers.* s.l. : Massachusetts Institute of Technology, 1997.

—. 2004. Spatial Audio Reproduction: Toward Individualized Binaural Sound. *The Bridge.* Winter, 2004, Vol. 34, 4.

GBATEK. 2007. DS Technical Data. *GBATEK.* [Online] October 4, 2007. [Cited: January 8, 2008.] http://nocash.emubase.de/gbatek.htm#dstechnicaldata.

Gerritse, Michel. 2006. SaturnSystemInfo. *Eidolon's Inn.* [Online] December 12, 2006. [Cited: January 3, 2008.] http://www.eidolons-inn.net/tiki-index.php?page=SaturnSystemInfo.

Gerstman, Jeff. 2007. Def Jam: Icon for Xbox 360 Review. *Gamespot.com.* [Online] CNET Networks, March 7, 2007. [Cited: January 14, 2008.] http://www.gamespot.com/xbox360/action/defjam3/review.html?tag=tabs;reviews.

Gerstmann, Jeff. 2005a. Donkey Kong Jungle Beat for GameCube Review. *Gamespot.com.* [Online] CNET Networks, March 11, 2005a. [Cited: January 8, 2008.] http://www.gamespot.com/gamecube/action/donkeykongjunglebeat/review.html?tag=tabs;reviews.

—. 2005b. Guitar Hero for PlayStation 2 Review. *Gamespot.com.* [Online] CNET Networks, November 1, 2005b. [Cited: January 7, 2008.] http://www.gamespot.com/ps2/puzzle/guitarhero/review.html?tag=tabs%3Breviews&page=1.

—. 2000. Samba de Amigo for Dreamcast Review. *Gamespot.com.* [Online] CNET Networks, June 16, 2000. [Cited: January 17, 2008.] http://www.gamespot.com/dreamcast/puzzle/sambadeamigo/review.html?tag=tabs;reviews.

Goehler, Stefan. 2003. Phonomenal! *Crossfire Designs.* [Online] Crossfire Designs, September 30, 2003. [Cited: January 1, 2008.] http://crossfire-designs.de/index.php?lang=en&what=articles&name=showarticle.htm&article=soundcards.

Graft, Kris. 2007. Harmonix on Rock Band and Beyond. *Next Generation.* [Online] Future Network USA., October 23, 2007. [Cited: January 7, 2008.] http://www.next-gen.biz/index.php?option=com_content&task=view&id=7570&Itemid=50.

Hagén, Mikael. 2001. A Gamer's Guide to Direct Sound 3D and A3D 1.x. *3D Sound Surge.* [Online] March 15, 2001. [Cited: January 6, 2008.] http://www.3dsoundsurge.com/features/articles/ds3d.html.

—. 1999. A Gamer's Guide to EAX. *3D Sound Surge.* [Online] October 14, 1999. [Cited: January 13, 2008.] http://3dsoundsurge.com/features/articles/EAX.html.

Hagén, Mikael and Muschett, Mark. 2002a. Gamer's Guide to 3D sound and reverb APIs. *3D Sound Surge.* [Online] January 8, 2002a. [Cited: January 4, 2008.] http://www.3dsoundsurge.com/features/articles/APIs/APIs.html.

—. 2002b. Gamer's Guide to 3D sound and reverb engines. *3D Sound Surge.* [Online] January 8, 2002b. [Cited: January 4, 2008.] http://www.3dsoundsurge.com/features/articles/3DSoundEngines/3DSoundEngines.html.

Harmonix Music Systems (2007a): *Phase*, MTV Games, IPod

—. 2007b. Phase | FAQ. *Phase | your music is the game.* [Online] Harmonix Music Systems, Inc., 2007b. [Cited: January 7, 2008.] http://www.yourmusicisthegame.com/faq/.

—. 2007c. Phase | Media. *Phase | your music is the game.* [Online] Harmonix Music Systems, Inc., 2007c. [Cited: January 17, 2008.] http://www.yourmusicisthegame.com/media/.

Harmony Central. 2008. Reverberation. *Harmony Central.* [Online] 2008. [Cited: January 12, 2008.] http://www.harmony-central.com/Effects/Articles/Reverb/.

Hays, Tom. 1998. DirectMusic For The Masses. *Gamasutra.* November 6, 1998, Vol. 2, 44.

Heaton, Barrie. 2007. Standard Pitch of Concert Pitch for Pianos. *UK Piano Page.* [Online] November 30, 2007. [Cited: January 9, 2008.] http://www.uk-piano.org/history/pitch.html.

Heckroth, Jim. 1995. *Tutorial on MIDI and Music Synthesis.* La Habra, CA : The MIDI Manufacturers Association, 1995.

Hernandez, Christopher. 2007. What are the Technical Specs of the Turbo Grafx/PC Engine. *Dark Watcher's Console History.* [Online] September 25, 2007. [Cited: December 30, 2007.] http://darkwatcher.home.att.net/console/details/turbo16.htm.

Horner, Matthew. 1999. Digital Sound Synthesis Using FM. [Online] 1999. [Cited: January 9, 2008.] http://www.wam.umd.edu/~mphoenix/dss/dss.html.

Howard, Sheryl. 2007. EE 348: Lab3, Spring 2007 - Sampling and Interpolation. *Northern Arizone University.* [Online] February 27, 2007. [Cited: January 9, 2008.] http://jan.ucc.nau.edu/sh295/EE348/Labs/Lab3S07.pdf.

Indiana University. 2004. What is a Gravis UltraSound card? *Knowledge Base.* [Online] Indiana University, June 11, 2004. [Cited: January 2, 2008.] http://kb.iu.edu/data/acre.html.

Intel. 2008. Audio Codec. *Intel.com.* [Online] 2008. [Cited: January 3, 2008.] http://www.intel.com/technology/computing/audio/.

—. 2004. High Definition Audio Specification. *Intel.com.* [Online] April 15, 2004. [Cited: January 8, 2008.] ftp://download.intel.com/standards/hdaudio/pdf/HDAudio_03.pdf.

—. 2007. Intel® High Definition Audio. *Intel.com.* [Online] Intel, November 6, 2007. [Cited: January 8, 2008.] http://www.intel.com/design/chipsets/hdaudio.htm.

Interactive Audio Specialist Interest Group. 1999. Interactive 3D Audio Rendering Guideline - Level 2.0. *Interactive Audio Special Interest Group.* [Online] September 20, 1999. [Cited: January 5, 2008.] http://www.iasig.org/pubs/3dl2v1a.pdf.

Janus, Scott. 2006. Audio in the 21st Century. *Audio DesignLine.* [Online] October 17, 2006. [Cited: January 9, 2008.] http://www.audiodesignline.com/howto/193303241.

Jeffay, Kevin. 1999. The Audio Data Type - Coding & Compression Basics. *Department of Computer Science - University of North Carolina at Chapel Hill.* [Online] August 26, 1999. [Cited: January 11, 2008.] http://www.cs.odu.edu/~cs778/jeffay/Lecture2.pdf.

Johnston, Chris. 1997. Rumble Pak Titles On The Rise. *Gamespot.com.* [Online] Gamespot, May 23, 1997. [Cited: December 1, 2007.] http://www.gamespot.com/news/2466717.html?q=Rumble%20Pak.

Jung, Robert. 2003. Atari Jaguar Frequently Asked Questions. *Atari Age.* [Online] August 3, 2003. [Cited: January 2, 2008.] http://www.atariage.com/Jaguar/faq/index.html?SystemID=JAGUAR.

Kohler, Chris. 2007. Hands-On: Phase, Harmonix's iPod Game. *Game | Life from Wired.com.* [Online] CondéNet, Inc., November 6, 2007. [Cited: January 7, 2008.] http://blog.wired.com/games/2007/11/hands-on-phase-.html.

Kotaku. 2005. Playstation 3: Full Specs. *Kotaku.* [Online] Gawker Media, May 16, 2005. [Cited: January 7, 2008.] http://kotaku.com/gaming/playstation-3/playstation-3-full-specs-103733.php.

Kreimeier, Bernd. 2001. The Story of OpenAL. *Linux Journal.* [Online] January 1, 2001. [Cited: January 5, 2008.] http://www.linuxjournal.com/article/4400.

Kruczek, Thilo. 2008. Allgemeine Infos. *gtaplanet.de.* [Online] Gamigo AG, 2008. [Cited: January 4, 2008.] http://gtaplanet.gamigo.de/content/index.php?men_open=gta.

Kubarth, Darius. 2006. SID History. *SID in-depth information site.* [Online] November 23, 2006. [Cited: December 30, 2007.] http://sid.kubarth.com/sid_history.html.

Kuphaldt, Thorsten. 2007. SID 6581 / 8580. *Commodore Computer Online Museum.* [Online] November 25, 2007. [Cited: December 30, 2007.] http://cbmmuseum.kuto.de/zusatz_6581_sid.html.

MacDonald, Keza. 2006. Rhythm Tengoku - Rhythm Heaven // GBA /// Eurogamer. *Eurogamer.* [Online] September 26, 2006. [Cited: January 7, 2008.] http://www.eurogamer.net/article.php?article_id=67953.

Marks, Aaron. 2000. Working the Grammy Angle. *Gamasutra.* [Online] CMP Media LLC., Feburary 25, 2000. [Cited: January 6, 2008.] http://www.gamasutra.com/features/20000225/marks_01.htm.

McDonald, Glenn. 2004. A History of Video Game Music. *Gamespot.* [Online] CNET Networks, March 29, 2004. [Cited: December 28, 2007.] http://www.gamespot.com/features/6092391/index.html?tag=result;title;0.

McDonough, Amy. 2006. Wii Get It Now: Technical Specs from 1UP.com. *1UP.com.* [Online] Ziff Davis Publishing Holdings Inc. , November 6, 2006. [Cited: January 7, 2008.] http://www.1up.com/do/feature?cId=3154939.

McMaster, Zachary. 2002. FMod for torque. *Garage Games.* [Online] Garage Games, March 6, 2002. [Cited: January 18, 2008.] http://www.garagegames.com/index.php?sec=mg&mod=resource&page=view&qid=2305.

Menshikov, Aleksei. 2003. Modern Audio Technologies in Games. *Digit-Life.* [Online] 2003. [Cited: January 4, 2008.] http://www.digit-life.com/articles2/sound-technology/index.html.

Microsoft. 2007a. Basic Concepts of Force Feedback. *DirectX Developer Center.* [Online] Microsoft, 2007a. [Cited: December 1, 2007.] http://msdn2.microsoft.com/en-us/library/bb172346.aspx.

—. 2007b. Conditions. *DirectX Developer Center.* [Online] Microsoft, 2007b. [Cited: December 1, 2007.] http://msdn2.microsoft.com/en-us/library/bb204846.aspx.

—. 2008. How To: Apply Attenuation and Doppler 3D Audio Effects. *MSDN Library.* [Online] Microsoft, 2008. [Cited: January 12, 2008.] http://msdn2.microsoft.com/en-us/library/bb447686.aspx.

—. 2000. Microsoft Announces Release of DirectX 8.0. *PressPass - Information for Journalists.* [Online] Microsoft, November 9, 2000. [Cited: January 6, 2008.] http://www.microsoft.com/presspass/press/2000/Nov00/DirectXLaunchPR.mspx.

—. 1999. Microsoft Ships DirectX 6.1. *PressPass - Information for Journalists.* [Online] Microsoft, February 3, 1999. [Cited: January 5, 2008.] http://www.microsoft.com/presspass/press/1999/feb99/direct61pr.mspx.

—. 2003. Sound System: Description of Yamaha OPL3 and OPL2 Chips. *Help and Support.* [Online] Microsoft, November 17, 2003. [Cited: January 2, 2008.] http://support.microsoft.com/kb/89877/en-us.

—. 2007c. Xbox 360 Technical Specifications. *Xbox.com.* [Online] Microsoft, July 23, 2007c. [Cited: January 7, 2008.] http://www.xbox.com/en-AU/support/xbox360/manuals/xbox360specs.htm.

—. 2007d. Xbox: Description of custom soundtracks. *Help and Support.* [Online] Microsoft, April 25, 2007d. [Cited: January 7, 2008.] http://support.microsoft.com/kb/909942/en-us.

MIDI Manufacturers Association. 2008. About Downloadable Sounds (DLS). *MIDI Manufacturers Association.* [Online] 2008. [Cited: January 10, 2008.] http://www.midi.org/about-midi/dls/abtdls.shtml.

—. 2008. About General MIDI. *MIDI Manufacturers Association.* [Online] 2008. [Cited: January 2, 2008.] http://www.midi.org/about-midi/gm/gminfo.shtml.

Miller, Mark. 1999. 3D Audio. *Gamasutra.* [Online] CMP Media LLC., November 2, 1999. [Cited: January 5, 2008.] http://www.gamasutra.com/features/19991102/gameaudiosupp/3daudio.htm.

MobyGames. 2008a. Alone in the Dark. *MobyGames.* [Online] 2008a. [Cited: January 6, 2008.] http://www.mobygames.com/game/alone-in-the-dark.

—. 2008b. Game Trivia for Wipeout. *MobyGames.* [Online] 2008b. [Cited: January 3, 2008.] http://www.mobygames.com/game/wipeout.

—. 2008c. Release Information for PaRappa the Rapper. *MobyGames.* [Online] 2008c. [Cited: January 3, 2008.] http://www.mobygames.com/game/playstation/parappa-the-rapper/release-info.

—. 2008d. Rez. *MobyGames.* [Online] 2008d. [Cited: January 6, 2008.] http://www.mobygames.com/game/rez.

Morris, Chris. 2005. XBox 360: Good, but not great. *CNNMoney.com.* [Online] CNN, November 21, 2005. [Cited: January 7, 2008.] http://money.cnn.com/2005/11/17/commentary/game_over/column_gaming/index.htm.

Motion Picture Experts Group. 2007. Audio. *mpeg.org.* [Online] September 3, 2007. [Cited: January 9, 2008.] http://www.mpeg.org/MPEG/DVD/Book_B/Audio.html.

Navarro, Alex. 2006. Elite Beat Agents for DS Review. *Gamespot.com.* [Online] CNET Networks, November 6, 2006. [Cited: January 7, 2008.] http://www.gamespot.com/ds/puzzle/elitebeatagents/review.html?tag=tabs;reviews.

—. 2007a. Rock Band for PlayStation 2 Review. *Gamespot.com.* [Online] CNET Networks, December 18, 2007a. [Cited: January 8, 2008.] http://www.gamespot.com/ps2/puzzle/rockband/review.html?mode=gsreview.

—. 2007b. Rock Band for Xbox 360 Review. *Gamespot.com.* [Online] CNET Networks, November 20 20, 2007b. [Cited: January 7, 2008.] http://www.gamespot.com/xbox360/puzzle/rockband/review.html?tag=tabs;reviews.

Nintendo. 2005. The Legend of Zelda. *Das Zelda Universum.* [Online] Nintendo, 2005. [Cited: December 30, 2007.] http://zelda.nintendo-europe.com/deDE/index.php?mId=106.

OpenAL. 2007a. OpenAL. *OpenAL.* [Online] OpenAL, June 1, 2007a. [Cited: December 6, 2007.] http://www.openal.org/.

—. 2007b. Platforms. *OpenAL.* [Online] June 1, 2007b. [Cited: January 6, 2008.] http://www.openal.org/platforms.html.

PC Vs Console. 2007. Console Specs (4th Generation). *PC Vs Console.* [Online] December 30, 2007. [Cited: December 30, 2007.] http://www.pcvsconsole.com/features/consoles/gen4.php.

Pfister, Andrew. 2007. Guitar Hero III: Legends of Rock Xbox 360 Review Index, Guitar Hero III: Legends of Rock Reviews. *1UP.com.* [Online] Ziff Davis Publishing Holdings Inc. , October 30, 2007. [Cited: January 7, 2008.] http://www.1up.com/do/reviewPage?cId=3164068&sec=REVIEWS.

Pichlmair, Martin and Kayali, Fares. 2007. *Playing Music: On the Principles of Interactivity in Music Video Games.* Tokio, Japan : Situated Play, Proceedings of DiGRA 2007 Conference, 2007.

Pidkameny, Eric. 2002. Levels of Sound. *VGMusic.* [Online] Videogame Music Archive, May 15, 2002. [Cited: December 28, 2007.] http://www.vgmusic.com/information/vgpaper2.html.

Polsson, Ken. 2007. Chronology of Personal Computers (1981). *Chronology of Personal Computers.* [Online] November 7, 2007. [Cited: January 1, 2008.] http://www.islandnet.com/~kpolsson/comphist/comp1981.htm.

Preve, Francis. 2007. Oscillators: Essential Waveforms. *beatportal.com.* [Online] October 25, 2007. [Cited: January 9, 2008.] http://www.beatportal.com/blogs/post/oscillators-essential-waveforms/.

Provo, Frank. 2000. Hey You, Pikachu! for Nintendo 64 Review. *Gamespot.* [Online] November 3, 2000. [Cited: January 5, 2008.] http://www.gamespot.com/n64/puzzle/heyyoupikachu/review.html.

Puckette, Miller. 2006. Theory and Techniques of Electronic Music, Draft: December 30, 2006. [Online] December 30, 2006. [Cited: January 8, 2008.] http://crca.ucsd.edu/~msp/techniques/latest/book.pdf.

QSound Labs. 2007. QSound Chronology. *QSound Labs.* [Online] November 23, 2007. [Cited: January 11, 2008.] http://www.qsound.com/corporate/chronology.htm.

RAD Game Tools. 2007. The Miles Sound System. *RAD Game Tools.* [Online] December 21, 2007. [Cited: January 6, 2008.] http://www.radgametools.com/mssds.htm.

Ransom-Wiley, James. 2005. Xbox 360 custom music controlled by developer. *Joystiq.* [Online] Weblogs, Inc. Network, November 9, 2005. [Cited: January 7, 2008.] http://www.joystiq.com/2005/11/09/xbox-360-custom-music-controlled-by-developer/.

Richards, Lee. 2003. Capcom Board List. *Cybercade.* [Online] August 1, 2003. [Cited: January 11, 2008.] http://www.system16.com/cybercade/capcom_list.html.

Robinson, John. 1999. Sega unleashes a 128-bit monster on the gaming world. *CNN.* [Online] CNN, September 9, 1999. [Cited: January 4, 2008.] http://cnnstudentnews.cnn.com/TECH/computing/9909/09/dreamcast/index.html.

Rogers, Tim. 2004. Final Fantasy VI. *insert credit.* [Online] October 16, 2004. [Cited: January 2, 2008.] http://www.insertcredit.com/reviews/ffvi/.

Schmidt, Brian. 1997. What's the deal with 3D sound under DirectX. *GameDev.* [Online] Gamedev.net, June 1997. [Cited: January 5, 2008.] http://www.gamedev.net/reference/articles/article593.asp.

Schweitzer, Ben. 2008. Final Fantasy VI OSV. *RPGFan.* [Online] 2008. [Cited: January 2, 2008.] http://rpgfan.com/soundtracks/ff6ost/index.html.

SegaStuff. 2007. Sega Master System. *SegaStuff.* [Online] September 28, 2007. [Cited: December 30, 2007.] http://www.segastuff.de/index.php?option=com_content&task=view&id=92&Itemid=35.

Seum-Lim, Gan. 1992. Digital Synthesis of Musical Sounds. [Online] 1992. [Cited: January 10, 2008.] http://xenia.media.mit.edu/~gan/Gan/Education/NUS/Physics/MScThesis/.

Shimpi, Anand Lal. 2001a. Hardware Behind the Consoles - Part I: Microsoft's Xbox. *AnandTech.* [Online] November 21, 2001a. [Cited: January 6, 2008.] http://www.anandtech.com/printarticle.aspx?i=1561.

—. 2001b. Hardware Behind the Consoles - Part II: Nintendo's GameCube. *AnandTech.* [Online] December 1, 2001b. [Cited: January 6, 2008.] http://www.anandtech.com/printarticle.aspx?i=1566.

Simons, Iain. 2007. Book Excerpt: INside Game Design: Harmonix Music Systems. *Gamasutra.* [Online] CMP Media LLC., December 5, 2007. [Cited: January 7, 2008.] http://www.gamasutra.com/view/feature/2801/book_excerpt_inside_game_design_.php.

Sinclair, Brendan. 2007. $169 Rock Band on November 23. *Gamespot.com.* [Online] CNET Networks, September 28, 2007. [Cited: January 7, 2008.] http://www.gamespot.com/xbox360/puzzle/rockband/news.html?sid=6180056&om_act=convert&om_clk=newlyadded&tag=newlyadded;title;1.

Smith, Julius O. 2007. Spectral Audio Signal Processing, March 2007 Draft. [Online] March 2007. [Cited: January 9, 2008.] http://ccrma.stanford.edu/~jos/pasp/.

Smith, Will. 1999. Review: Diamond Monster Sound MX300. *Ars Technica.* [Online] Ars Technica, 1999. [Cited: January 4, 2008.] http://arstechnica.com/reviews/1q99/mx300.html.

Sony Computer Entertainment Inc. 2000. music interaction. *vib_ribbon (flash version).* [Online] Sony Computer Entertainment Europe, 2000. [Cited: January 4, 2008.] http://www.vibribbon.com/#.

—. 2007. SingStar. *SingStar.* [Online] Sony Computer Entertainment Europe, 2007. [Cited: January 6, 2008.] http://www.singstargame.com/.

SoundTracker. 2008. What is SoundTracker? *SoundTracker.* [Online] 2008. [Cited: January 10, 2008.] http://www.soundtracker.org.

Stahl, Geoff. 2003. Re: joystick mappings in HID? *Apple Mailing Lists.* [Online] Apple Inc., October 22, 2003. [Cited: December 1, 2007.] http://lists.apple.com/archives/Mac-games-dev/2003/Oct/msg00232.html.

Stockburger, Axel. 2003. The game environment from an auditive perspective. *AudioGames.net.* [Online] 2003. [Cited: January 14, 2008.] http://www.audiogames.net/pics/upload/gameenvironment.htm.

Strickland, Chris. 2002. Audio Programming on the GameBoy Advance Part 1. *Gamedev.* [Online] Gamedev.net, May 21, 2002. [Cited: January 8, 2008.] http://www.gamedev.net/reference/articles/article1823.asp.

Strietelmeier, Julie. 1998. The Gadgeteer - Gameboy Color Review. *The Gadgeteer.* [Online] December 6, 1998. [Cited: January 8, 2008.] http://the-gadgeteer.com/review/gameboy_color_review.

Takeshita, Yohei. 2003. Soundpond. Parsons School of Design : s.n., 2003.

Tätilä, Veli-Pekka. 2007. An Informal History of Game Music. [Online] September 27, 2007. [Cited: December 28, 2007.] http://www.student.oulu.fi/~vtatila/history_of_game_music.html.

Terlecki, Daniel. 1998. Frequently Asked Questions List v5.3. *3DO FAQ - Classic Gaming.* [Online] 1998. [Cited: January 2, 2008.] http://classicgaming.gamespy.com/View.php?view=ConsoleMuseum.Detail&id=39&game=12.

The History of Computing Project. 2005. Videogames - Sega Dreamcast. *History of Computing.* [Online] February 4, 2005. [Cited: January 4, 2008.] http://www.thocp.net/software/games/consoles/sega/sega_dreamcast.htm.

The Inquirer. 2003. Creative snaps up Scipher Sensaura business. *The Inquirer.* [Online] Incisive Media, December 4, 2003. [Cited: January 8, 2008.] http://www.theinquirer.net/en/inquirer/news/2003/12/04/creative-snaps-up-scipher-sensaura-business.

The Purple Owl. 2004. The Commodore Amiga 1985 - 1994. *The Purple Owl.* [Online] July 24, 2004. [Cited: January 1, 2008.] http://home.iprimus.com.au/danmcpharlin/purpleowl/commodore-amiga-history.html.

The Sonic Spot. 2007. Tpyes of Synthesis. *The Sonic Spot.* [Online] April 27, 2007. [Cited: January 9, 2008.] http://www.sonicspot.com/guide/synthesistypes.html.

Thomas, Aaron. 2007. Jam Sessions for DS Review. *Gamespot.com.* [Online] CNET Networks, September 14, 2007. [Cited: January 7, 2008.] http://www.gamespot.com/ds/puzzle/jamsessions/review.html?tag=tabs;reviews.

Thorsberg, Frank. 2001. Nintendo Ships Game Boy Advance. *PCWorld.com.* [Online] PC World Communications, June 11, 2001. [Cited: January 8, 2008.] http://www.pcworld.com/article/id,52406/article.html.

Tyler, Micheal. 2006. General MIDI - Why You Need It. *Computer Music Products.* [Online] November 21, 2006. [Cited: December 2, 2008.] http://musicmall.com/cmp/article3.htm.

Tyson, Jeff. 2000. How Dreamcast Works. *Howstuffworks.* [Online] October 19, 2000. [Cited: January 4, 2008.] http://entertainment.howstuffworks.com/dreamcast.htm.

USB Implementers' Forum. 2001. Device Class Definition for Human Interface Devices (HID). *USB.org.* [Online] 1.11, June 27, 2001. [Cited: December 1, 2007.] http://www.usb.org/developers/devclass_docs/HID1_11.pdf.

—. 2007. USB.org - HID Tools. *USB.org.* [Online] USB Implementer's Forum, 2007. [Cited: December 1, 2007.] http://www.usb.org/developers/hidpage/.

Velden, C.C. van der. 1998. The MIDI FAQ by CC. *MIDI Papa's.* [Online] 1998. [Cited: January 1, 2008.] http://members.aol.com/midipapa/midi_faq.htm.

Wall, Jack. 2002. Using a Live Orchestra in Game Soundtracks. *Gamasutra.* [Online] CMP Media LLC., May 20, 2002. [Cited: January 11, 2008.] http://www.gamasutra.com/resource_guide/20020520/wall_01.htm.

Walters, Chuck. 1997. Cop a Feel....with Haptic Peripherals. *Gamasutra.* [Online] CMP Media LLC., December 19, 1997. [Cited: December 1, 2007.] http://www.gamasutra.com/features/19971219/walters_01.htm.

Wayper, Timothy. 2003. Re: HID Manager shortcomings. *Apple Mailing Lists.* [Online] Apple Inc., January 22, 2003. [Cited: December 1, 2007.] http://lists.apple.com/archives/mac-games-dev/2003/Jan/msg00168.html.

Weske, Jörg. 2000. Sound on the PC. *Digital Sound and Music in Computer Games.* [Online] December 2000. [Cited: January 1, 2007.] http://www.tu-chemnitz.de/phil/hypertexte/gamesound/pcsound-main.html.

Whalen, Zach. 2004. Play Along - An Approach to Videogame Music. *Game Studies.* November, 2004, Vol. 4, 1.

Williams, Martyn. 2004. Sony's PSP Hits the Streets. *PC World.* [Online] PC World Communications, Inc., December 12, 2004. [Cited: January 8, 2008.] http://www.pcworld.com/article/id,118912-c,gameconsoles/article.html.

Winfield, Alan. 2003. The Principles of Electronic Music Synthesis. *University of West of England.* [Online] October 14, 2003. [Cited: January 9, 2008.] http://www.ias.uwe.ac.uk/~a-winfie/teach2003/st_pems1.htm.

Wolfe, Joe. 2007. What is a Sound Spectrum? *School of Physics - The University New South Wales.* [Online] August 10, 2007. [Cited: January 10, 2008.] http://www.phys.unsw.edu.au/jw/sound.spectrum.html.

Wright, Mark. 1998. Retrospective - Karsten Obarski. [Online] March 1, 1998. [Cited: January 10, 2008.] http://www.textfiles.com/artscene/music/information/karstenobarski.html.

Wright, Steve. 1979. 2600 (STELLA) Programmer's Guide. 1979.