Technische Universität Wien
Wirtschaftsinformatik
Institut: Softwaretechnik und Interaktive Systeme

Diplomarbeit

The AudioSquare

A collaborative virtual Music Environment

ausgeführt von

Ronald Genswaider
Mohsgasse 33/36
1030 Wien

unter der Anleitung von:
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
und
Dipl.Ing. Dr.techn. Helmut Berger

Vienna, 2008

Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde eine dreidimensionale kollaborative Umgebung namens *The AudioSquare* entwickelt, die ein virtuelles Erkunden von Musikarchiven ermöglicht. In Form von Avataren können User einander in einer 3D Welt begegnen, über ein einfaches Chatsystem kommunizieren und die angebotene Musik gemeinsam anhören. Die Musik wird anhand von zwei verschiedenen Methoden räumlich angeordnet. Die erste Methode basiert auf der Verwendung von *Self-Organizing Maps*, welche die Musikstücke anhand ihrer Soundeigenschaften auf einer Art Landkarte verteilen. Die zweite Methode bietet die Möglichkeit, Musikdateien in einer bestimmten Ordnerhierarchie auf dem Dateisystem abzulegen, um in der virtuellen Welt entsprechend abgebildet zu werden.

The AudioSquare stellt Zusammenhänge zwischen den repräsentierten Medien in ein räumliches Bezugssystem, in das der User selbst eingebunden ist. Es entsteht eine intuitive Schnittstelle, die realen Handlungs- und Navigationsformen nahe kommt. Die derzeitige Entwicklung von Online-Welten und 3D Spielen zeigt außerdem, dass die Metapher des virtuellen Raumes immer mehr an Bedeutung gewinnt. Konzepte wie der hier vorgestellte Prototyp zur Aufbereitung von medialen Inhalten, also eines Contents, innerhalb dieses Raumes sind jedoch noch spärlich vorhanden.

Die Umsetzung von *The AudioSquare* erfolgte mittels der *Torque Game Engine*, die ein umfangreiches Rahmenwerk für die Gestaltung virtueller Welten bietet. Durch die Client-Server-Architektur wird es Usern möglich gemacht, die virtuelle Welt nach dem Download einer entsprechenden Client-Applikation über das Internet zu betreten. Ein zusätzlicher Streaming-Server ermöglicht es, die eingebundenen Musikstücke, entnommen aus einer freien Musiksammlung von *Magnatune*, gemeinsam anhören zu können.

Abstract

This work presents *The AudioSquare*, a three-dimensional collaborative environment allowing users to explore automatically organized music archives. Users, impersonated as avatars, may interact with each other, communicate through a simple chat-system and enjoy the presented music. Two different methods for music organization are implemented by the system. The first method takes advantage of a *self-organizing map*. It uses the sound characteristics of each audio track to calculate its position on a two-dimensional map. The second method utilizes folder hierarchies on the file system for the organization of media. The music files stored in the folders are transformed into a spatial representation within the virtual environment.

The AudioSquare transforms correlations between the represented media into a system of spatial relations and involves the users directly into the scene. This produces an intuitive interface that allows forms of interaction and navigation similar to real life. Since the number of people participating in online worlds and 3D games increases steadily, the metaphor of virtual space is gaining momentum. The proposed prototype shows a novel approach for organizing and presenting media as content within such environments.

The development of *The AudioSquare* is based on the *Torque Game Engine*. It provides a powerful framework for the development of state-of-the-art virtual 3D environments. Based on a client-server architecture it allows users to enter the world through the Internet by downloading a client-application. The presented music, gathered from *Magnatune's* royalty-free online archive, is distributed over the Internet by a streaming server to allow a collaborative experience when users listen to the music together.

Contents

Contents **vii**

1 Introduction **1**

2 Related Work **5**
2.1 Music Information Retrieval (MIR) 5
2.1.1 Metadata-based MIR 6
2.1.2 MIR based on musical Data 8
2.1.3 Content-based MIR 9
2.2 3D Game Engines 11
2.3 3D Virtual Communities 14

3 Technological Foundations **21**
3.1 Audio Feature Extraction 22
3.1.1 Preprocessing the Audio Files 22
3.1.2 Specific Loudness Sensation - Sone 23
3.1.3 Rhythm Patterns 24
3.2 Self-Organizing Maps 25
3.3 A Game Engine as Development Framework: Torque 28

4 Conceptual Design **33**
4.1 SOM-based Music-Organization 34
4.2 Manual Music-Organization 34

5 Implementation **37**
5.1 System Architecture 37
5.2 Implementations in Torque 39
5.2.1 The User Interface for the Torque Client 39
5.2.2 Customized 3D Objects 40
5.2.3 Setting up the virtual environment in Torque 42
5.2.4 Creating assets for the Repository 43
5.2.5 Importing the Objects-File 44

5.3 Implementation of the the Wrapper 44
5.3.1 Class-Representation of Torque Objects and Input Files . . 46
5.3.2 Processing the Output-Files 47
5.4 The Icecast Streaming Server 49

6 Visit The AudioSquare **51**

7 Conclusions and Future Work **59**

8 Acknowledgements **61**

Bibliography **63**

A Game Engine Overview **71**

B Example of an Objects-File **73**

C Example of Marker-Objects in the Mission-File **79**

List of Figures **81**

List of Tables **83**

Chapter 1

Introduction

In the last few years the Internet experienced a paradigm shift, placing emphasis on the community of people. Today the Internet is no longer an instrument just for representation, rather it becomes a platform for users to extend their real-life by means of a virtual presence. On the one hand this change can be seen on community-based Web sites summarized as "Web 2.0", featuring user-generated content and providing extensive tools to share personal information. On the other hand, the Internet is also a virtual stage for users to encounter each other. The most advanced examples for such environments today are *Massively Multiuser Online Roleplaying Games* (MMORPGs) [65, 8], consisting of huge virtual worlds inhabited by thousands of concurrent users impersonated as avatars. While the predominant motivation for participating in MMORPGs is still "playing", an increasing number of users is spending a significant amount of time in 3D virtual worlds without following a predefined quest. Generating, publishing and experiencing content in 3D virtual spaces is an emerging trend on the Internet. The recent media hype around *Second Life*[1] gave an outlook on the future of such virtual communities. On the one hand virtual environments broaden the possibilities of social interaction that go beyond text-based chat rooms. Especially one's inherent presence in space and the awareness of others facilitate the initiation of social contacts. On the other hand, using 3D virtual worlds has the advantage of communicating via commonly accepted spatial metaphors [18]. Similarity of objects can be expressed by spatial relations, i.e. the more similar two objects are, the closer they are located together. Furthermore, users can interpret each other's interests by how close they are to one another and to the objects in space. Consequently, users are supported in building a mental model of the information space. They begin to understand its characteristics and to recognize which information is present and how the respective objects are related to each other.

[1] http://www.secondlife.com (date of access: 20.04.2007)

In this context we utilized the possibilities of virtual environments for music representation. We developed a prototype, *The AudioSquare*, which allows users to explore music in a collaborative virtual world. It takes advantage of spatial metaphors to describe the relationships of the represented music tracks to one another. The prototype offers the organization of music libraries by a *self-organizing map* (SOM) as introduced in "Island of Music" [40], and later in the PocketSomPlayer and PlaySom [39]. Sound-specific characteristics are extracted from music files by applying methods from digital signal processing and psycho-acoustics. The resulting features are used for the training of the SOM that arranges similar music tracks in spatially adjacent regions. More specifically, a *self-organizing map* is an unsupervised neural network model that provides a topology-preserving mapping from a high-dimensional input space onto a two-dimensional output space [28]. Alternatively, a manual organization method is also provided. First, audio files are organized by setting up a specific directory structure on the local file system. This folder hierarchy is then automatically transformed into an architectural setting. Folders are represented by buildings that are arranged by a selectable layout algorithm. The music tracks are placed inside of the buildings according to the their location in the folders.

The technical foundation for the realization of *The AudioSquare* is the 3D game engine *Torque*[2]. Built as a framework for game development it already comes with many efficient features for creating collaborative virtual environments. A key feature is the built-in client-server architecture that makes it easy to develop an Internet-based virtual community. After installing the client-application a user may start the program and enter the *The AudioSquare* online. To ensure that users are listening to the same music when on the same place at the same time, the music is streamed by a media server over the Internet[3].

The remainder of this thesis is structured as follows. Chapter 2 provides an overview about related work and includes the areas *music information retrieval*, game engines and virtual communities. Chapter 3 explains the technological foundations used to build the prototype. On the one hand, the principles of feature extraction and *self-organizing maps* are explained. On the other hand the *Torque Game Engine* that was used as development framework for the creation of the prototype is discussed. The conceptual design of the prototype is described in Chapter 4. It explains how a virtual world should be utilized in order to represent media and discusses the two different methods for organizing and visualizing the music tracks. Chapter 5 describes the technological details of the prototype. It outlines the general system architecture of the prototype and discusses the components of the system. These are *Torque's* client-server architecture, the media server used for audio streaming over the Internet and the

[2]http://www.garagegames.com/products/torque/tge/ (date of access: 27.07.2007)
[3]http://www.icecast.org/ (date of access: 27.07.2007)

Java-based preprocessing-facility, the *Wrapper*, which sets up the data to start the system. Moreover, the workflow of creating the virtual world is explained. Chapter 6 takes the reader on a tour through *The AudioSquare* and provides screenshots of the virtual world. A summary followed by suggestions for future work are given in Chapter 7.

Chapter 2

Related Work

The theoretical background for this thesis comprises three areas that are discussed in this chapter. *The AudioSquare* is an interface that provides exploration of music in a spatial environment. This approach expands existing methods from the area of *music information retrieval* (MIR). Thus, the first section explains the different approaches in MIR and gives examples for the visualization of sound and music. Since the technical realization of the prototype is based on the *Torque Game Engine*, the next section describes the origin and present use of game engines, especially in non-game scenarios and research projects. As a collaborative environment *The AudioSquare* is a basis for *3D virtual communities* which are discussed in the last section.

2.1 Music Information Retrieval (MIR)

A research project of the University of Berkley, called *How much information? 2003*[1], highlights the fast growth of digitally stored information. As presented in Table 2.1, the researchers and students estimated the amount of produced information in terabytes for the year 2002, regarding different media, namely paper, film, magnetic media and optical media (CD, DVD). The result has been compared with the same estimates for the year 1999. It turned out that the difference of annually produced data is over 70 percent, whereby the greatest change was on digital media.

This trend is similar in digital music. The number of available music files has grown rapidly with the emergence of large archives driven by powerful compression algorithms like *MP3* and their distribution over the Internet. For example *Apple* states that their online music store *iTunes* offers more than 6 million songs and 2 billion tracks have been downloaded already[2]. *eMUSIC*, the largest online

[1] http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/ (date of access: 04.02.2008)

[2] http://www.apple.com/de/itunes/store/ (date of access: 06.02.2008)

Storage Medium	*2002 (Terabytes)*	*1999 (Terabytes)*	*Change (Percent)*
Paper	1,634	1,200	36.0
Film	420,254	431,690	-3.0
Magnetic	5,187,130	2,779,760	87.0
Optical	103	81	28.0
Total	5,609,121	3,212,731	74.5

Table 2.1: How much information? Comparing the estimates for the amount of generated digital media in the years 1999 and 2002 in the United States.

store for independent labels, already hosts 2 million online tracks that were downloaded over a hundred million times[3]. The enormous amount of available music online points out the need for efficient strategies in music retrieval. Considering the different interests of music listeners, strategies are very diverse.

Music information retrieval can be roughly classified into three main paradigms that are discussed in the subsequent sections. The first relies on *metadata* which describes a music piece by text properties such as title or artist. Approaches based on *musical data* focus on melodies and the notes of a song. Finally, *content-based* strategies use the raw sound information for analysis.

2.1.1 Metadata-based MIR

A common way of querying music is to search in metadata such as title, artist, genre or lyrics. In most cases, ID3 tags are used to store metadata in music files[4]. ID3 tags offer a broad range of possible attributes for the description of a song. However, the tags are often assigned by users and, thus, the provided information is sometimes inconsistent or simply false.

On Internet platforms users are often invited to create relations between music tracks. Following the Web 2.0 paradigm, *Last FM*[5] gives users the possibility for social tagging of songs. Listeners enter desired songs and the Web application creates an audio stream containing songs with similar tags. This empowers the users to find related music by stating in their personal preferences.

A strategy that goes beyond textual search is the creation of music maps that visualize the relations between songs according to metadata. A music map as seen on the *Dimvision* Web site[6] is depicted in Figure 2.1. When a user enters an artists name or a song title, a network of interconnected nodes representing a band or artist is displayed. The similarities of the nodes have been calculated by comparing descriptions stored in Amazon's music database.

Torrens proposes three different visual representations for private digital mu-

[3] http://www.emusic.com/about/pr/pr20061213.html (date of access: 06.02.2008)
[4] http://www.id3.org (date of access: 06.02.2008)
[5] http://www.lastfm.de (date of access: 06.02.2008)
[6] http://www.dimvision.com/musicmap (date of access: 06.02.2008)

Figure 2.1: A music map as shown on the *Dimvision* Web site. The nodes represent artists and bands. The edges identify the relations of the nodes to one another.

sic collections [56] (see Figure 2.2). The first representation shows a disc that is divided into sectors according to each of the genres of the library. Each sector is divided into sub-sectors representing the artists of the associated genre. The radius of the disc, from the center to the perimeter, is equivalent to the time axis. The rectangle visualization is quite similar to the disc-view. The time axis goes along the vertical axis and the genres divide the rectangle into columns. The third visualization follows a tree structure. A rectangular area representing the complete library is recursively split three times: First into genres, each genre into its sub-genres and finally each sub-genre is split into its artists. The colors of the divisions provide additional information about quantitative attributes, such as the play-count or user ratings. All three visualizations are intended as alternative user interfaces. Relations are shown in easy-to-understand graphics to provide efficient interaction. Another approach for creating relations in music takes the geospatial location of the music pieces into account. *AROOOGA* [26] is a Web crawler that scans entire Web sites for music tracks. The location of the Web server is determined by applying the "whois" command applied to the servers IP address. The main idea was to create a world map where the locations of the servers hosting the sound files are marked. Unfortunately, the technological restrictions of location detection on the Internet resulted in low accuracy. The *MUSICtable* provides a collaborative interface that invites all participants to select music tracks in a playful manner [50]. A tabletop display standing in the middle of a room shows a colored map that graphically represents a music archive. A floating cursor marks the music track on the map that is played next. Users can press buttons around the table to influence the direction of the cursor and each others decision at the same time.

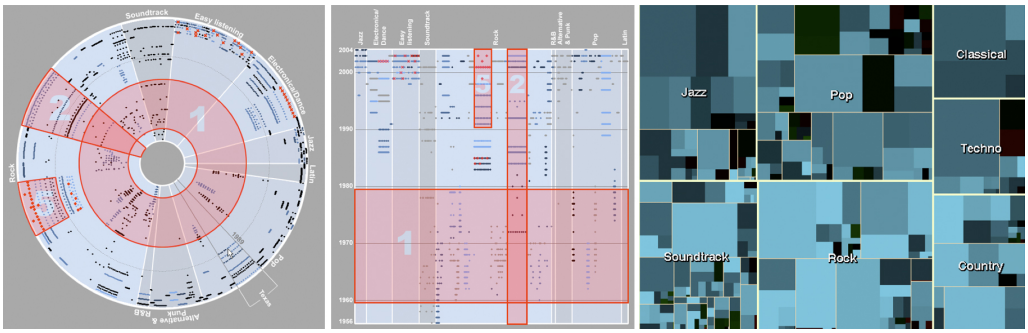

Figure 2.2: Three visual representations as proposed by Torrens: The disc view, divided (left), the rectangular view (middle) and the tree view (right).

2.1.2 MIR based on musical Data

The most popular way to represent musical data is the *Common Music Notation* (CMN) (cf. Figure 2.3). It arose in the Middle Ages, developed to preserve the extensive plain chant repertoire of the Roman Catholic church and refined several times, especially in the Renaissance. As Isaacson pointed out in his comparison of different representations of musical information, CMN is very adaptable for live performance, but it is not very suitable to show information about harmonic or motivic relations and the musical form [20].

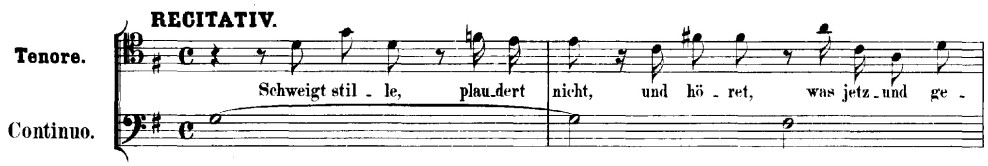

Figure 2.3: Excerpt from Bach's score for his piece *Schweigt stille*.

For computers the first standard to represent music in an abstract form was MIDI, an interface standard for interchanging musical events between electronic instruments. In the early 1990s Stephen Malinowski created the *Music Animation Machine*[7], a computer tool that transformed the score of a music piece into an animated visual representation. It was a very early attempt in creating a computer program that can help unexperienced listeners to understand musical structure.

The focus on music seen as events of notes and intervals leads to queries that are related to musical notation. Dovey [14] proposes an algorithm for the *OMRAS* framework, a Java-based development suite for testing search algorithms. It

[7] http://www.musanim.com (date of access: 06.02.2008)

allows music queries in a regular expression-style for specific notes and melodies. Such techniques are more of a textual kind and hard to perform by an average user. The query-by-humming approach was introduced in the mid 1990s. It allows users to query songs by simply singing or humming melodies [16, 37, 42]. Bainbridge et al. [2] developed a music library that stores the musical notation of music pieces retrieved from scanned pages, song books and MIDI files. Users can combine hummed queries with textual search to narrow down the results. Today, query-by-humming has reached a mature state and can be used in the commercial online archive *midomi.com*[8]. Polyphonic queries that address harmonies and more complex passages of a music piece are addressed by Suyoto et al. [52]. Query-by-example approaches use segments of a song as search criteria and are used in queries for similar songs [19] and cover versions [57].

2.1.3 Content-based MIR

Since electronic processing evolved, many different representations of the sound itself were introduced. The simplest method is to display the amplitude of the sonic wave against a time line, comparable to the analogue audio tracks on film reels. Spectrograms offer even broader possibilities. They display the distributions of frequency bands over a certain time. On a typical spectrogram as seen in Figure 2.4, time is represented by the x-axis and the frequency by the y-axis. The more intense a certain frequency is at a certain time, the brighter it is plotted. In 1984 Cogan analyzed a broad range of music pieces by plotting spectrograms [11]. Such visualizations can help to emphasize the view on changes in orchestration and the dynamics more than with traditional notations. In content based approaches of MIR spectrograms are used for analyzing dynamics and rhythmic patterns to create qualitative patterns describing the underlying music pieces. Logan and Salmon propose a method to compare songs based solely on their audio content [33]. Each music track is described by a signature based on K-means clustering of spectral features. West and Lamere combine this approach with textual information such as the genre, style or emotion of the music to build a similarity metric [64].

Music similarity measurements can also be transformed into an euclidean space. Visualized as a kind of map, the similarity of music tracks is displayed by their distance to one another. Pampalk proposes an approach that takes advantage of a *self-organizing map* (SOM) to organize music on a two-dimensional plane [41]. Spectral features are extracted from each music track and used as input for the SOM. This organization method is also used for the automatic organization approach within *The AudioSquare* and is described in detail in Chapter 3. Pampalk uses this technique to create a map of music which can be used as an alternative view for a jukebox, referred to as *Island of Music*. The *PlaySOM*

[8] http://www.midomi.com (date of access: 06.02.2008)

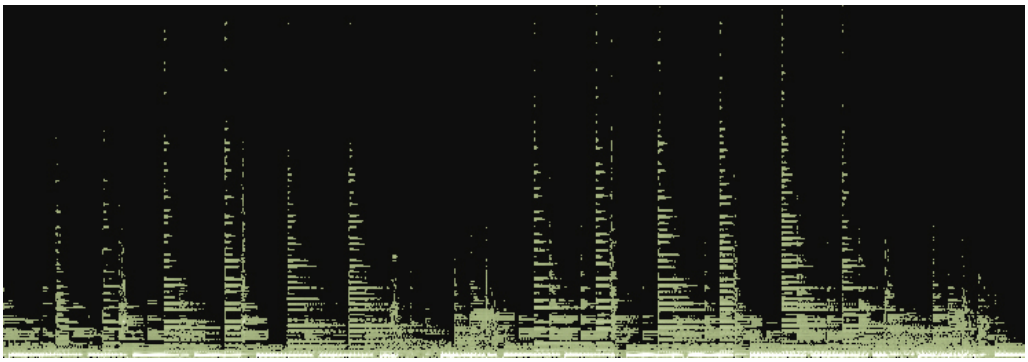

Figure 2.4: Spectral analysis of a guitar playing a melody.

and the *PocketSOMPlayer* take up this approach to create an interface for small devices such as palmtops and mobile phones [39]. As seen on Figure 2.5, the main part of the interface shows a height-map. Different colors indicate the density of music tracks of a certain location. Users can create playlists by simply drawing a path or marking a region on the map (c.f. Figure 2.5).

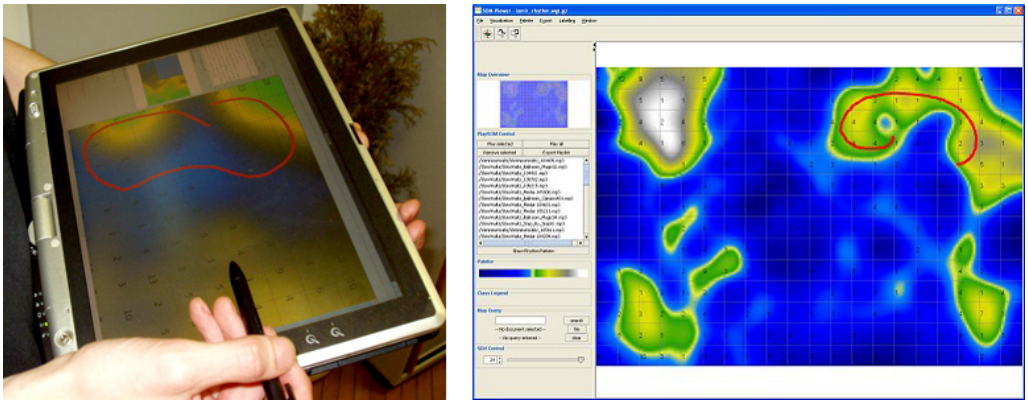

Figure 2.5: Utilizing content-based music organization - the *PocketSOMPlayer* on a tablet PC (left) and the graphical user interface (right).

Knees transforms the SOM-generated height-map into a 3D landscape [25]. The height of the terrain indicates the density of music tracks in a certain location. The music pieces are represented by images which are located in the landscape. Each image shows the most prominent artist on a certain place. The image is retrieved over the Internet by simple keyword-search. A virtual camera can be moved to fly around the map and listen to the music nearby played as spatial sound. Luebbers also addresses SOM-generated music maps [34]. He tries to improve the human interface design by focussing on the sound of the organized music. He created two graphical interfaces using spatial sound metaphors

to improve the users experience when exploring the music map. The *Sonic Radar* is visually comparable to a radar-screen. The center of the screen denotes the actual position of the listener. The volume of the music tracks around the listener increases and decreases according to the actual direction the listener heads towards. The *Sonic SOM* is very similar to the visual design of Pampalk's concept. It shows a map from top-view with the organized music tracks displayed as dots. The listener can choose a standpoint by selecting one of these dots. Similar to the *Sonic Radar*, the actual direction influences the loudness of the surrounding tracks. The user can move through the map by clicking on the dots around the current location which represent the currently playing songs. Tzanetakis and Cook introduce *Marsayas3D*, an audio browser and editor for collaborative work on a large-scale multiuser-screen [58]. The framework offers different audio analysis methods such as classification, segmentation, similarity-retrieval, principal component analysis (PCA), beat-detection and clustering. Each method can be applied to different browsers, real-time monitors and an editor. For example sound files can be mapped into a browser called "TimbreSpace3D" by using PCA and clustering can be used to color them. A multi-speaker display consisting of 16 speakers supports the users operating the various 2D and 3D interfaces by playing spatial sound.

2.2 3D Game Engines

Tennis for Two created in the year 1958 is known as the first video game in history[9]. Its "game logic" was hard-wired on an analogue computer system with a small oscillator display. Commercial game titles followed in the 1970s with simple video consoles and arcade cabinets [24]. Most of the early titles could be developed by a single person in a very short time. With the advancement of computer technology video games became more and more complex. The production of a current high-class title demands long-term plans comparable to those of film-making and a development team consists of many people with different professions. The complexity of today's games states the need for modular frameworks, which abstract fundamental programming routines from game design. These frameworks are referred to as game engines and used in nearly every modern computer game. One of the first development frameworks which follows the principle of game engines is *SCUMM*, written to improve the development workflow of the point-and-click adventure *Maniac Mansion*. It was a scripting language that supported instructions such as "walk to" and "pick up". This helped game designers to concentrate on artwork, the gameplay and the plot of the adventure. Today *SCUMM* is still maintained by an online community[10].

[9] http://www.bnl.gov/bnlweb/history/higinbotham.asp (date of access: 04.02.2008)
[10] http://scumm.mixnmojo.com/?page=scumm (date of access: 06.02.2008)

Game engines are used for all kinds of game genres but they are generally known as foundations for so-called *first-person shooter* games. The plot of such games is usually to run through a realistic 3D environment and to kill as many opponents as possible. Despite the moral issues, the underlying engines are technically very sophisticated. Photorealistic 3D graphics, surround sound, multiuser support and a very fast gameplay create a highly immersive experience for the player. The era of first person shooters started with *Wolfenstein 3D* and *Doom*, both developed by *id Software* in the early 1990s. A very interesting aspect was that the players started to change level designs of the games by themselves. For example, there are still many "custom levels" for *Doom* made by fans available for download on the Internet[11]. The creation of self-made levels called *modding* has become that popular that many game developers distribute their titles together with dedicated level editors, scripting interfaces and other development tools. A very popular editor comes from *Epic Games* and is called *Unreal Editor*. Version 2 is a very comfortable development suite featuring terrain editing, different mesh types, creation of indoor sceneries, texturing, audio effects, various special effects and a scripting language called *UnrealScript*[12]. *id Software* goes other ways by releasing the source code of discontinued game engines under the GNU/GPL license, which makes them attractive to the open-source community. Another popular feature of game engines is the possibility for users to create cinematografic content. *Machinimas* are movies created entirely within a 3D game engine and are usually distributed over the Internet as a package of code, objects and textures [61]. Other users who downloaded a *machinima* need the same game engine as the creator for playback. Today there are annual contests such as the *Machinima Festival* which awards such movies in different categories every year[13].

The flexibility of 3D game engines for building virtual environments makes them applicable for non-game usage. In research game engines are used for projects addressing realtime visualizations and simulations. Usually, power workstations have been used for this task which require specially qualified people for maintenance and programming. The high costs of such machines made research in many cases impossible. Game engines are an alternative to these heavyweight systems. The costs to establish a workstation for developing applications with a game engine are approaching zero. A common personal computer with a relatively new graphics card is often already available or can be upgraded very easily. The learning curve is also much faster than on a professional 3D workstation since these engines are dedicated to non-professionals. Lewis et al. point out that game engines are the tools of choice for research projects which require reliable maintenance of player states, objects and terrain locations [32]. However,

[11] http://www.doomwadstation.com (date of access: 08.02.2008)
[12] http://udn.epicgames.com/Two/UnrealEd.html (date of access: 08.02.2008)
[13] http://festival.machinima.org (date of access: 06.02.2008)

if the task is simulation where high accuracy is crucial, photorealistic solutions still are the better choice.

In visualization projects for architecture the use of game engines instead of supercomputers have shown several advantages. Students of the Technical University of Denmark tried to visualize their campus on an *SGI* graphics workstation [49]. The programming was very time-consuming and the project was about to stop. Eventually, the use of a game engine let them succeed. The possibility of establishing multiuser worlds over networked applications driven by a game engine lead to a proposal for a collaborative environment for experimental architectural design [47]. Moloney et al. propose a tool for architectural design critique realized in *Torque* [38]. The goal was to allow students to publish their architectural models in a virtual environment where others could examine the buildings. Notes could be placed at particular locations for discussing details of the presented objects. With the *Quake III Engine* a source code comprehension tool for distributed work has been created to collaboratively examine programming projects [29]. Fritsch and Beck suggest a 3D system for indoor-visualization of buildings [4, 15]. Their approach addresses a cost-effective virtual environment for *Computer Aided Facility Management-Systems* (CAFM). A prototype created with the *Quake 3 Arena Engine* features a virtual building that users can walk through. By "shooting" objects textual data could be queried showing contextual information. *VR Kon-Tiki* goes new ways for presenting a museum and shows the virtual representation of the *Kon-Tiki Museum* in Norway [55]. The application based on the *Torque Game Engine* can be downloaded on the homepage of the institution[14]. A similar project but from a more artistic view is called *Museum Meltdown*[15]. The artists used the level editors of different game engines to recreate the architecture of contemporary museums and to use them as virtual exhibition spaces. *ARQuake* is an augmented reality game where game levels are written to match actual physical locations [54]. Players wearing head-mounted displays encounter monsters superimposed over a real scenery. *CaveUT* takes advantage of *Unreal's* network ability to use multiple player's viewpoints to construct a Cave-like display [21].

Schools and other educational institutions can benefit from game engines by teaching the basic principles of modding [48]. Scholars learn different concepts in game development such as computer science, mathematics, physics and aesthetic principles. Artificial intelligence routines in computer games are used for simulating the behavior of virtual opponents. *PSDoom* utilizes these routines for managing system processes in *Doom* [9, 10]. As a usual behavior of monsters in the game the artificial entities start to fight against each other when the place becomes too crowded. In this case "important" processes represented by

[14] http://www.kon-tiki.no/VR/ (date of access: 08.02.2008)
[15] http://www.bernstrup.com/meltdown/main.html (date of access: 08.02.2008)

Figure 2.6: *PSDoom* is a process management tool realized with *Doom 1*.

stronger monsters kill weaker ones resulting in an automatic process management (c.f. Figure 2.6). The idea of managing system resources inside of a first person shooter environment is also realized in *Brutal File Manager*[16]. Folders represented by rooms are inhabited by files represented by monsters which can be killed to delete them on the file system. The recent version also supports moving and copying of files. *GameBots* is a multi-agent system infrastructure [22] based on *Unreal Tournament*. It was designed to study human team behavior and allows the construction of artificial agents that collaboratively interact with users of other agents. Laird and Young use the *Unreal Tournament Engine* to create artificial intelligence characters that are able to interact with each other within a virtual scenery [30, 66].

2.3 3D Virtual Communities

The Internet introduced many new forms of communication and interaction. The apparent impact of the Internet on human community led to a controversy in opinions about its influence on the quality of our everyday life [3, 43, 62, 63]. Indeed, there are more and more people spending a noteable amount of time on the Internet. With a recent paradigm change the Internet itself transformed into a place for people to socialize and to extend their own subsistence on arbitrary virtual stages. This change has been summarized under the notion "Web 2.0" which should express the advancement of the Internet away from an information base towards a social platform for its users[17]. In addition, efforts are made to create virtual places dubbed *Cyberspace* in the 1990s. These immersive 3D virtual worlds address the satisfaction of the user's social needs and are complemented by immersive characteristics. Virtual worlds support the way humans act and communicate in real life to a certain extent and offer an environment to meet

[16] http://www.forchheimer.se/bfm/ (date of access: 11.02.2008)
[17] http://www.oreilly.de/artikel/web20.html (date of access: 12.02.2008)

other people. Such interfaces go beyond the text-based approaches dominating the Internet and graphically represent the user in terms of an avatar [12]. The virtual embodiment gives users the experience to be literally *in* the Internet rather than *on* it. 3D virtual worlds address the issue of social interactions much more since location awareness, presence, as well as direct communication are intrinsic elements. A notable number of users participate in *Massively Multiuser Online Roleplaying Games* (MMORPGs), the most successful virtual worlds today. Some examples for the increase of active subscribers over the years until 2006 can be seen on Table 2.3, taken from the independent Web site *mmogchart.com*[18]. The most successful MMORPG today, *World of Warcraft* (cf. Figure 2.7) has more than 10 million active subscribers worldwide[19]. From an economic view such online worlds are gaining momentum. The borderline between the virtual and the real world tends to blur in current titles. In particular the buying and selling of items for the game provide some gamers with a notable source of real income [59]. For *EverQuest* an economic study has revealed that the virtual *Norrath* has become the 77$^{\text{th}}$ richest country in the world, roughly comparable to Russia [7]. Other research revealed that some people start to spend more time in the virtual realms of an MMORPG than they spend in their job [65].

Figure 2.7: Masses of avatars in *World of Warcraft*.

Until now, three-dimensional collaborative virtual worlds are mostly different kinds of online games, especially MMORPGs. However, since the 1990's literature such as the books *Neuromancer* by William Gibson [17] and *Snowcrash* by Neal Stephanson [51] have been indicatory for the idea of virtual spaces without any game context. The latter author introduced the term *Metaverse* as a concept of a virtual habitat for users as well as the term *Avatar* for the virtual

[18] http://www.mmogchart.com/ (date of access: 05.02.2008)
[19] http://blizzard.co.uk/press/080122.shtml (date of access: 05.02.2008)

Game Title	*Developer*	*July 2004*	*July 2006*	*Delta*
City of Heroes	Cryptic Studios	180,000	160,000	-11 %
Dark Age of Camelot	Mythic Entertainment	245,000	125,000	-49 %
Everquest	Verant Interactive	420,000	200,000	-52 %
Everquest II	Verant Interactive	150,000	175,000	+17 %
Final Fantasy XI	Squaresoft	500,000	650,000	+30 %
Lineage	NCsoft	2,659,502	1,497,287	-44 %
Lineage II	NCsoft	1,474,280	1,302,340	-12 %
Star Wars Galaxies	Verant Interactive	300,000	170,000	-43 %
Toontown Online	Disney Interactive	85,000	110,000	+29 %
Ultima Online	Origin Systems	170,000	135,000	+21 %
World of Warcraft	Blizzard Entertainment	250,000	6,600,000	+2,450 %
Other MMORPGs		137,031	560,337	+309 %
Total		*6,664,402*	*12,466,740*	+87 %

Table 2.2: Listing of the most relevant *MMPORGs* and the number of active subscribers in the years 2004 and 2006 (According to *mmogchart.com*). The right columns on the right show the title of the game and the name of the developing company. The next two columns show the total number of active subscribers in the years 2004 and 2006. The rightmost column shows the difference between these two years in percent.

representation of a user. As an early example the virtual chat environment called *Onlive Traveler* [13], developed in the year 1999, tried to improve the experience of conversations online. The surroundings had a rather decorative character and were used as places to meet as groups of several users. The avatars were only heads without a body to emphasize on mimics and lip movements, because the chat was supported by a full duplex audio stream. Although users commended the impression of realistic and lively conversations, they felt decontextualized. The researchers concluded that having shared interests is crucial to virtual communities. According to Kaplan et al. virtual communities have to provide some basic properties to be satisfying for users [23]. First, they should provide the possibility to define a user's identity in a broad way, thus avatars representing the users have to be very customizable in shape and style. Communication should be supported by a wide variety of expressions that come close to face-to-face conversations. Typically this is done via gestures which can be triggered via buttons or voice-chat such as in *Traveler*. 3D worlds should also empower users to contribute to the collaborative space by building parts of the virtual

environment. Last but not least the researchers also mention the importance of a shared interest that gives the users a reason to interact. They suggest to encourage the users in steadily creating new contexts by building new facilities and improving the virtual world. 3D environments that mostly fit these criteria are referred to as *Metaverses*. One of the first is the open virtual space *Active Worlds*, online since 1995[20]. Via a client-application users can enter different virtual worlds and contribute in creating the environment. The originators had the vision of an alternative to traditional Web browsing. Instead of building a Web site, users could create buildings such as offices to represent their products or information. *Active Worlds* never became very popular and seems to keep the status of a niche product. Eight years later, a very similar but much more prominent virtual world called *Second Life*[21] was published by *Linden Lab* (cf. Figure 2.8). It is the first virtual online world without a gaming context that generated media attention. As *Linden Lab* CEO Pilip Rosedale says, in the year 1996 he had been inspired by Neal Stephensons *Metaverse*, which people could walk through with their avatars and independent programmers called "hackers" built the world and the virtual objects [35]. In correspondence to this most of the things in *Second Life* are made by the users. If they are familiar with scripting, they can change behavior of things in a very dynamic way. Now it seems that the big media-hype around *Second Life* is over and the number of active users is not growing any more. However, *Second Life* still shows the capability of virtual online environments for social interaction as well as for new business models. A very potential feature is the *Linden Dollar*, the official currency in *Second Life* that can be exchanged against real US Dollars. As an estimate users spend about one Million US Dollars a day [60]. Some companies started to contribute with large projects. IBM started to build virtual meeting venues and presentation areas for employees. Toyota created a virtual representation where people could obtain virtual cars for free as an advertising gimmick. Many other companies followed with interesting ideas, mostly to refresh their image as innovative companies for the press.

Second Life's short-term success led to a renaissance of the cyberspace and sensitized the public for the relevance of virtual communities. Several companies started to follow with their own concepts for online worlds. *Sony* announced a 3D virtual environment referred to as *Home* on the game developers conference in March 2007 [31] (cf. Figure 2.9). *Home* should enable all owners of their latest game console, the *PlayStation 3*, to enter an exclusive online 3D community world. In his keynote, *Sony's* Phil Harrison mentioned the notion *Game 3.0* as an emerging trend of mashing up game concepts with virtual communities. The Chinese *HiPiHi*[22], which has been online as a Beta version since April 2006,

[20] http://www.activeworlds.com/ (date of access: 05.02.2008)
[21] http://secondlife.com (date of access: 05.02.2008)
[22] http://www.hipihi.com (date of access: 05.02.2008)

Figure 2.8: Impressions from the virtual online world *Second Life*

shows conspicuous similarities to *Second Life* while the originator, Xu Hui, sees these similarities as a coincidence. The German company *Metaverse* looks at the development of their product *Twinity*, which was announced to be online by the end of 2007, in a more practical manner [5]. The appearance of the avatar should correspond with the users appearance in real life and the goal is to merge actual trends on the Internet. *vSide*[23] is a 3D online community for teenagers to meet in virtual nightclubs and listen to popular music (cf. Figure 2.10). The young company *Pixel-Orange* combines Web browsing with the strengths of collaborative virtual environments[24]. A Java applet, embedded in an HTML page presents architectural 3D models. Users may explore the objects and start conversations with others nearby.

[23] http://www.vside.com (date of access: 05.02.2008)
[24] http://www.pixel-orange.com (date of access: 15.03.2008)

Figure 2.9: Home for the *PlayStation 3*.

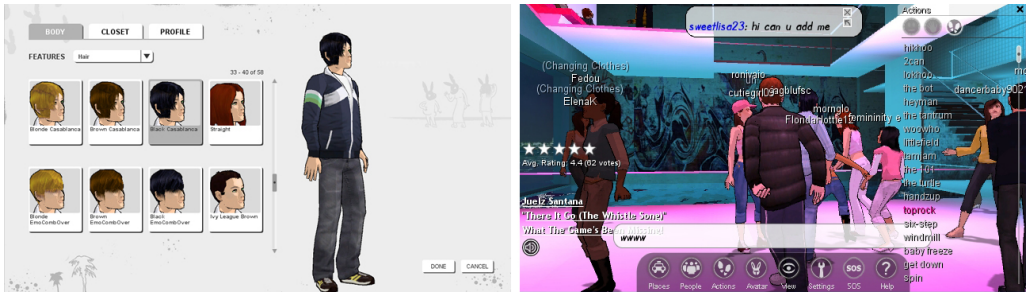

Figure 2.10: *vSide*, a virtual online community for teenagers.

Chapter 3

Technological Foundations

This chapter explains the technological foundations used for developing *The AudioSquare*. As depicted in Figure 3.1 this concerns the organization of music archives and the visual representation of the results in a collaborative virtual environment. The organization is done by means of a combination of *audio feature extraction* and *self-organizing maps* (SOM) as already proposed in prior research projects [39, 41, 44, 45]. Each audio file of a music archive is analyzed by a series of digital signal processing algorithms to create a set of audio-specific parameters, called the *Rhythm Pattern*. A SOM uses this *Rhythm Pattern* as input features to calculate a position on a two-dimensional map. The result is a map of music, in which tracks with similar characteristics are located close to one another. It is represented in a three-dimensional environment based on the *Torque Game Engine*.

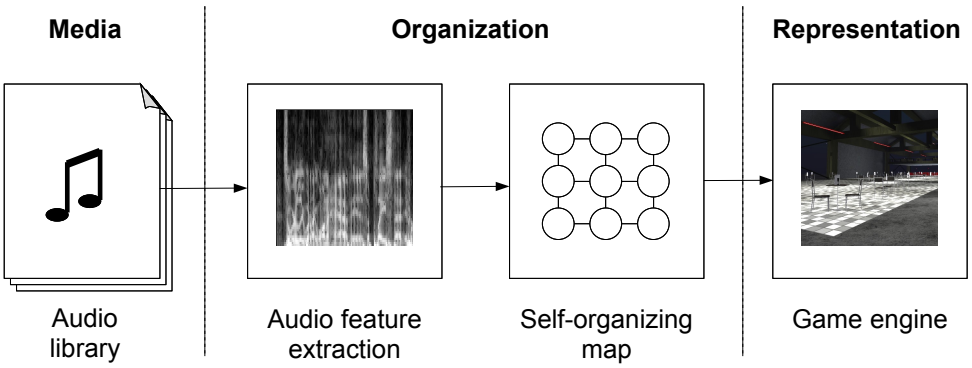

Figure 3.1: Organization and representation of music libraries with a *self-organizing map*.

3.1 Audio Feature Extraction

The process of *audio feature extraction* analyses the music files in a music library to extract meaningful parameters. In this case, each audio file runs through several processes to achieve a time-invariant measure that allows the comparison of the music files. The process involves three main steps. First, sound files are reduced in quality to allow fast analysis. Then the sound data is transformed to a power spectrum representing the loudness sensations per critical band over time. The loudness sensation is calculated to *Sone*, a loudness measure that reflects the perception of the human ear. Finally, the power spectrum is transformed into a time-invariant representation which focusses on the modulation of the loudness over time, called *Rhythm Pattern*.

3.1.1 Preprocessing the Audio Files

High-quality music in digitized format consumes a high amount of storage. For example one minute of sound data in CD quality requires about 10 MB. For detailed analysis of large music libraries the large amount of data needs to be reduced by a certain degree. Figure 3.2 shows the steps taken to transform a specific audio file from the library to audio data ready for analysis. First, the original audio file is decoded to raw *Pulse Code Modulation* (PCM) format. Since the raw audio format in good quality requires a huge amount of storage, the quality is reduced from 44 kHz stereo sound to 11 kHz mono. Within this poor quality it is still easy for humans to identify the music. Finally, the sound data is split up into 6-second segments. This length has been chosen because it is enough for humans to recognize the style of a piece of music within this duration. To avoid lead-in and fade-out effects, the first and the last segments are dropped. In addition, only every third segment is kept, which is still sufficient to identify the underlying music easily.

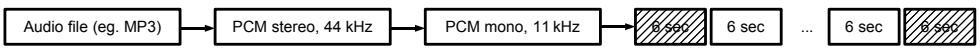

Figure 3.2: Preprocessing a music file in the library, which results in several 6 second sequences. The first and the last sequence are dropped to avoid lead-in and fade-out effects.

The result of the preprocessing steps are segments of 6 seconds of music every 18 seconds at 11 kHz for each music file. Without loosing relevant information for analysis, the music is reduced by a factor of over 24.

3.1.2 Specific Loudness Sensation - Sone

First, the power spectrum of the audio signal is calculated. This is is done by decomposing the audio data into its frequencies using *Fast Fourier Transformation* (FFT). The frequencies are then bundled into the first 20 out of the 24 critical bands of the so-called Bark scale, that refers to the frequency resolving ability of the human auditory system. The inner ear can be thought of as a complex system with bandpass filters separating the frequencies to concentrate them at certain locations along the basilar membrane. The response to the different frequencies is asymmetric. For example, we can distinguish low frequencies of up to about 500 Hz very well, while our ability decreases significantly below 500 Hz. In the next step spectral masking effects, meaning the occlusion of a quiet sound by a louder sound, present at the same time with similar frequencies, are calculated. The resulting so-called spreading matrix identifies the influence of the critical bands on each other and is used for their adjustment.

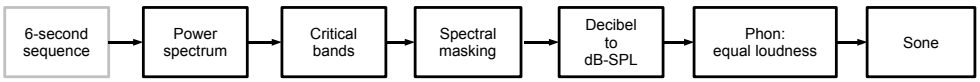

Figure 3.3: Transformation of an audio signal to a representation regarding the specific loudness sensation in Sone.

PCM data corresponds to the sound pressure which is measured in *Pascal* (Pa). Before the calculation of *Sone* values it is necessary to transform the data into decibel, which is calculated as the value between the sound pressure and the pressure of the hearing threshold. This is also known as dB-SPL, where SPL is the abbreviation for sound pressure level. Since the perceived loudness depends on the frequency of the tone, the relationship between the sound pressure in decibel and the hearing sensation measured in *Sone* is not linear. The dB-SPL values are thus transformed to the equal loudness levels with their unit *Phon*. By definition, one *Phon* is equal to 1 dB-SPL at a frequency of 1 kHz. For example, a value of 40 *Phon* corresponds to 40 dB-SPL at 1 kHz. Finally the perceived loudness sensation varies for different loudness levels. When transformed to *Sone*, this is taken into respect. The loudness of an audio signal at 1 kHz and 40 dB-SPL is to be defined as 1 *Sone*. A tone perceived twice as loud is defined to be two *Sone* and so on. Within this stage of processing the raw audio data has been finally transformed into a spectrum that measures the specific loudness sensation for 20 frequency bands along the time axis.

3.1.3 Rhythm Patterns

Since the current data representation is not time-invariant, it may not be used to compare two pieces of music point-wise. Thus, this stage of processing is used to calculate a time-invariant representation of music, namely the *Rhythm Pattern*. These *Rhythm Patterns* indicate how strong and fast rhythms are played within specific frequency bands. Figure 3.4 outlines the steps needed to achieve the *Rhythm Pattern*.

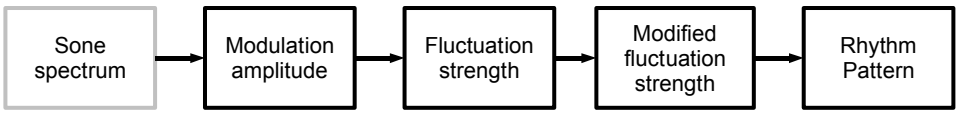

Figure 3.4: Transformation of the power spectrum in Sone to the time-invariant *Rhythm Pattern*

For every critical band the loudness usually rises and falls more or less periodically over time. This results in a more or less regular pattern also known as rhythm. These changes can also be seen as a signal that has been sampled at discrete points over time. Thus, the periodical patterns can be processed by a Fourier transformation for each critical band. The resulting amplitude modulation concerns *Rhythm Patterns* from 0 Hz up to 43 Hz. The sensation of *fluctuation strength* is most intense at frequencies from 4 Hz and gradually decreases up to 15 Hz. Focussing on the frequencies relevant for rhythm sensation, 20 values between 0 Hz and 10 Hz are obtained for each of the 20 critical bands. A gradient and a Gaussian filter are then applied to better distinguish the *Rhythm Patterns* and to remove irrelevant information. While the gradient filter applied to the modulation frequency should emphasize distinctive beats, the Gaussian applied across both, the modulation frequency and the critical bands should increase the similarity between two *Rhythm Pattern* characteristics. By simply using the median of the 6-second segments, the final result can now be taken for further cluster analysis by the *Self-organizing map* which is described in the next section.

The final result of the feature extraction process is summarized in Figure 3.5. The graphics show the *Rhythm Patterns* for each analyzed segment as well as the median. The left-hand example shows the representation of Beethoven's classical piano piece *Für Elise*. The modulation amplitude shows that there is no beat present. On the other hand, the right-hand *Freak on a Leash* has a strong beat at around 7 Hz in all frequency bands.

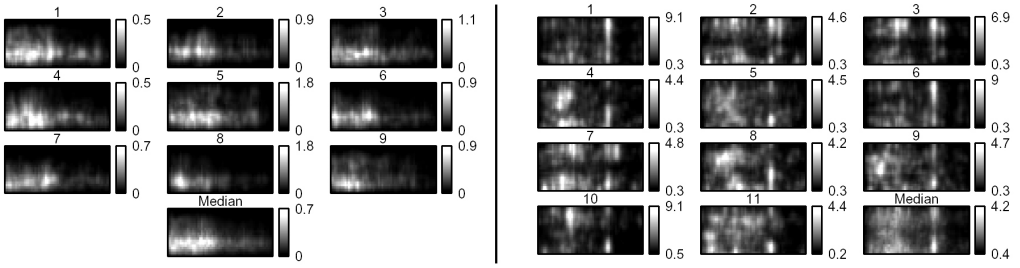

Figure 3.5: Two results of the feature extraction process (left: *Für Elise* by Beethoven, right: *Freak on a Leash* by Korn)

3.2 Self-Organizing Maps

The representation of the music tracks in the three-dimensional virtual world follows the schema of a topographic map, which consists of several units, whereby each unit can contain multiple music tracks. For this kind of organization a *self-organizing map* (SOM) is used, introduced in the year 1982 by Theuvo Kohonen, also known as *Kohonen map* [27, 28]. The principle of a SOM follows observed mechanisms in the human brain. Studies in the field of neurobiology revealed that many structures in the cortex have a linear or planar topology, whereas sensory perception is often multidimensional. An example is the sensation of color, which consists of the three primary colors red, green and blue, by the visual cortex. Additionally the eye delivers information about texture, position and structure of an object. This raises the question how the visual cortex achieves the representation of this multidimensional data through it's planar structure. The studies showed that relationships in sensorial patterns are represented as spatial relationships on the cortex. Of course, all details of how the cortex processes sensory signals have not yet been discovered. However, it seems a safe presumption that the first representation of the world built by the brain is a topological one. Kohonen's model of self-organizing networks goes to the heart of this issue. The model works with elements not very different from the ones used by other researches. The main difference is the definition of the neighborhood of its elements because of their lateral connections to their neighbors. If the state of an element changes, this also effects others to a certain degree.

A *self-organizing map* consists of so-called *units* that usually constitute a one-, two- or three-dimensional grid. The input for a SOM is an n-dimensional *feature space*. Each unit has a *weight vector* with the same dimensionality as the feature space. Like many other artificial neural networks, a SOM has two modes of operation, training and mapping. When trained, the n-dimensional input x is passed to every unit to compute the corresponding excitation by using the weight vectors. The main task of the SOM is to move its units as close as

possible towards the represented features. Figure 3.6 shows a one-dimensional SOM that attunes to a two-dimensional feature space with triangular shape [46].

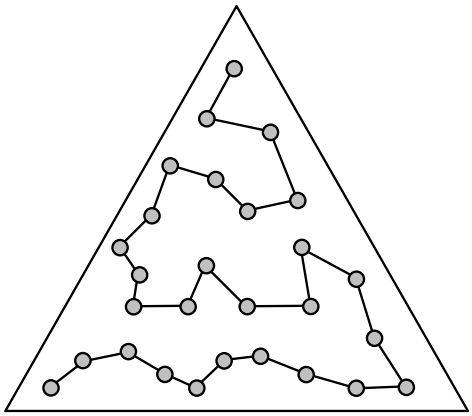

Figure 3.6: A one-dimensional SOM maps a triangular region.

The goal of the learning algorithm is to associate different parts of the SOM matrix to respond similarly to certain input patterns. Each training iteration t starts with the random selection of one input pattern $x(t)$. This input pattern is delivered to the *self-organizing map* and each unit determines its activation. The calculation of the activation is usually done with the Euclidean distance between the weight vector and the input vector. In this case, the unit with the lowest activation value is referred to as the *winner*, c, of the current training iteration (cf. Expression 3.1).

$$c : m_c(t) = \min_i ||x(t) - m_i(t)|| \tag{3.1}$$

Finally the weight vector of the *winner* c as well as the weight vectors of its adjacent units are adapted. The adaptation is implemented as a gradual reduction of the difference between the corresponding components of the input pattern and the weight vector, as shown in Expression 3.2.

$$m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t) \cdot [x(t) - m_i(t)] \tag{3.2}$$

In terms of geometry the weight vectors are moved towards the input pattern by a small value. The amount of this movement depends on the so-called learning rate, α, and its decrease over time. The number of units that are affected by an adaption process is described by the so-called *neighborhood function* h_{ci}. Typically this function is unimodal and symmetric around the location of the *winner* decreasing monotonic with increasing distance from the *winner*. With the advantage of time the functional form shrinks until it only affects the *winner*.

As shown in Expression 3.3 a typical neighboring function may be a Gaussian curve, with r_i is representing the two-dimensional vector pointing to the location of the unit i within the grid, and $||r_c - r_i||$ denoting the distance between the units c, i.e. the *winner* of the current training iteration, and i in terms of the output space. Since the distance range of the neighboring function reaches many units at the beginning of a training session of a *self-organizing map*, it allows the formation of large clusters. This is followed by a steady refinement towards the end of the training process. The spatial width is described by means of the time-varying parameter σ.

$$h_{ci}(t) = \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right) \tag{3.3}$$

The translation of weight vectors has the consequence, that the Euclidean distance between the weight vectors and the input patterns are minimized. The weight vectors get more similar to the input patterns, thus it is more likely to be a *winner* at future representations of this input pattern. The adaption of the *winner* to the input pattern together with its neighbors leads to clusters which provide representation of similar input patterns. Thus, similarities between input patterns that are represented in the n-dimensional vector space are mirrored in the two-dimensional output-space of the *self-organizing map*. The training process of the *self-organizing map* is a topology preserving mapping of a high-dimensional input space onto a low-dimensional output space where patterns that are similar in the input space are also close to each other in the low-dimensional output space.

Figure 3.7 gives an example for a graphical representation of a *self-organizing map*. In this case the map consists of a square arrangement of 7 x 7 units depicted as a matrix of circles on the left hand side of the figure. The black circle in the center of the matrix represents the *winner* regarding to the input pattern $x(t)$. The weight vector of the *winner* $m_c(t)$ is moved towards the input pattern and thus, the new weight vector $m_c(t+1)$ of the *winner* is closer to $x(t)$ than was $m_c(t)$. The grey shaded units in the grid of the *self-organizing map* are also adapted but not as strong as the *winner*, thus the degree of shading corresponds to the strength of adaption.

Self-organizing maps have proven their usefulness in many applications. For example, they are used for medical purposes such as the classification of cancerous tissue [36] as well as in the financial sector such as the clustering of high-frequency financial data [6] or the prediction of stock prices [1]. They provide a very robust and relatively fast method for clustering data with high dimensionality.

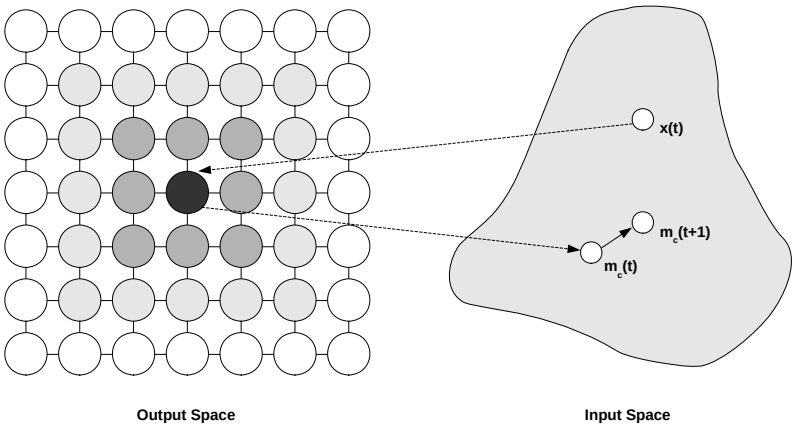

Figure 3.7: Architecture of a 7×7 *self-organizing map*

3.3 A Game Engine as Development Framework: Torque

A three-dimensional environment such as *The AudioSquare* needs a framework with a broad range of features. Besides real-time rendering of 3D graphics, it has to provide multi-user support, a 2D graphical user interface, support for various input devices and the ability to play back spatial sound. Many of today's game engines fulfill these requirements and come with even more modules such as physics engines or artificial intelligence routines to simulate opponents. The choice of the appropriate game engine stood at the beginning of the development process for *The AudioSquare*. The first criteria that narrowed down the number of possible choices was the type of game engine. We decided to use a game engine for first person shooters (FPS). These games have the most immersive characteristics, the fastest gameplay and the most sophisticated graphics routines. When supporting multiplayer-mode, they also feature a very efficient network code that maintains accurate update rates. Obtaining a full software development kit (SDK) for the recent commercial products such as the *Doom 3 Engine*[1] or the latest *Unreal Engine*[2] is very expensive. Since the project demanded a whole SDK that goes beyond the capabilities of a level editor, the commercial engines were skipped. Some older commercial engines such as the *Quake Engine* series are now licensed under the GNU/GPL but technically outdated and sparsely documented. Today, a broad range of low-cost and open-source engines are also available. For a detailed overview of examined low-budget products the reader may be referred to Appendix A. One of these low-cost engines, the *Torque*

[1] http://www.doom3.com/ (date of access: 20.10.2007)

[2] http://www.unrealtechnology.com/features.php?ref=technology-overview (date of access: 21.10.2007)

Game Engine (TGE) from *GarageGames*[3] turned out to be the best choice to develop *The AudioSquare*. It has been the top-rated engine on the game developer community Web site *Devmaster.net*[4] for a long time. It is also the only affordable product that provides everything for the development of a full virtual environment. While most of the investigated engines had poor or no network support, *Torque* comes with an award winning and very efficient network code. The graphics are fairly good and indoor as well as outdoor scenarios are supported. The reason for *Torque's* superiority amongst the other engines is it's origin as a commercial product, the *Tribes 2 Engine*, which was created for the multiplayer egoshooter game *Tribes 2* published in 2001 by *Dynamix*. Shortly after the release of the game, several members of the *Dynamix* team left the company and created their own company, GarageGames. They bought the *Tribes 2 Game Engine* and after extensive modifications the *Torque Game Engine* was created [53]. Since GarageGames provides a unique licensing model for their products, the *Torque Game Engine* is a compromise between commercial and open source software. Non-commercial developers can purchase an *Indie-license* for 150 US Dollars, a commercial license costs 500 US Dollars per developer workstation, a very low price when compared to other professional game engines. With the license of the *Torque Game Engine* comes a development kit that includes the full source code and a few demo games. Besides a Wiki-based online documentation there is a broad online community of license owners who support each other with forum discussions, tutorials, artwork and code enhancements. *Torque* supports the platforms Windows, Linux and Mac OSX natively.

Torque follows a strict client-server architecture. Even when run on a single computer, it acts as both a client and a server on the same machine. Over the Internet, communication between client and server is supported through a very robust and fast networking protocol that allows accurate update rates even for low bandwidth connections. The *Torque* server is responsible for the organization of the virtual world. It creates the environment on startup and coordinates objects and the actions of the users. A *Torque* client is mainly responsible for the audiovisual representation of the environment and the user interface. The logic of a game or 3D application is written in the byte-compiled scripting language *TorqueScript*. Although very flexible and powerful, sometimes it is necessary to enhance the source code of the engine written in C++. In the case of *The AudioSquare* the support for Internet audio streaming had to be added because originally *Torque* only supports the playback of local sound files. The management of assets such as 3D objects, textures and sounds also follows the client-server principle. When a client connects, it synchronizes the missing data with the server via file download.

[3] http://www.garagegame.com (date of access: 26.04.2007)
[4] http://www.devmaster.net/engines/ (date of access: 24.01.2008)

Figure 3.8: The *Torque SDK* comes with a first person shooter example. It can also be edited with the *World Editor* as shown here.

Torque comes with a built-in *World Editor* that provides a broad range of tools to design virtual environments (cf. Figure 3.8). Terrains can be created automatically with a palette of different algorithms or by loading a bitmap-based height map. Manual adjustments are then made with a mouse-controlled brush tool. Pre-built 3D assets can be imported, positioned, rotated and scaled. These include 3D objects such as trees, buildings or interiors as well as additional objects to emphasize the atmosphere such as water areas, a sun, sky textures, volumetric fog, etc. The resulting virtual world is stored in so-called *mission-files*. A *GUi Editor*, as shown in Figure 3.9, is also a standard tool of *Torque*.

The modeling of 3D objects cannot be done in the *World Editor*, rather there are two different file-types of objects that need different applications to create them. Efficient realtime rendering needs highly optimized 3D shapes, therefore the two file-types are suitable for different demands. DIF stands for *Dynamix Interior Format*, it is the file format for large and static, mainly architectural objects and interiors. These objects can be designed and exported in the free program *QuArK*[5] and the shareware product *3D World Editor*[6]. The other format for 3D objects, DTS is mainly used for characters, vehicles and smaller objects such as interiors. The models can be animated via skeletal and morph target animation, moreover animation sequences can be blended together. *Torque*

[5]http://quark.planetquake.gamespy.com (date of access: 09.02.2008)
[6]http://3dworldstudio.thegamecreators.com (date of access: 09.02.2008)

provides exporter plug-ins for different 3D programs, such as *3D Studio Max*, *Maya* and *Blender*.

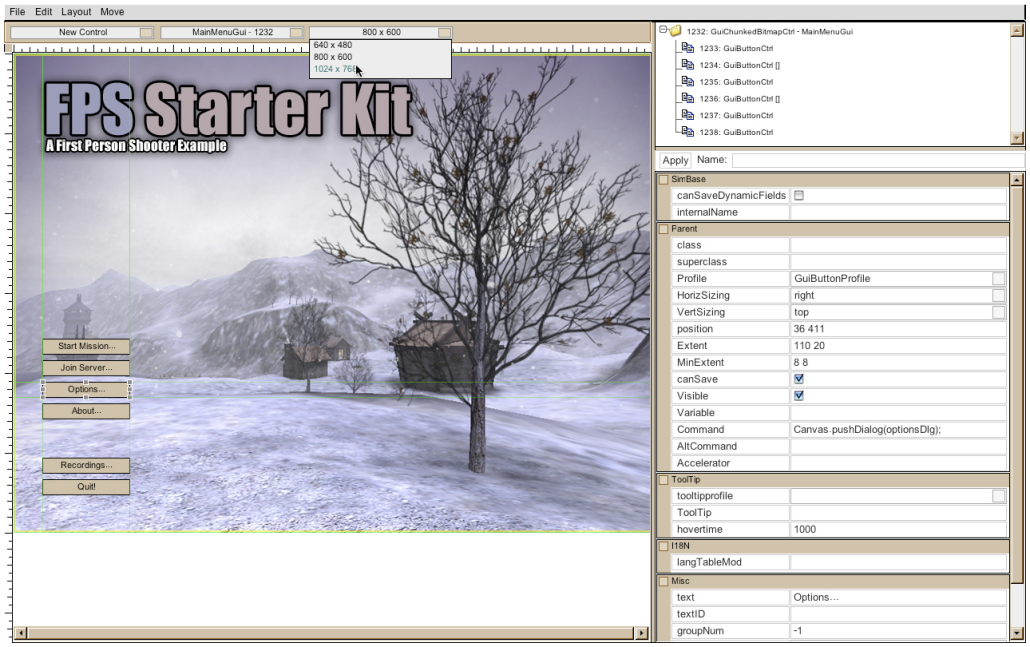

Figure 3.9: The *GUI Editor* is a standard feature of the *Torque SDK*. The image shows the start menu for the first person shooter example.

Chapter 4

Conceptual Design

Virtual three-dimensional environments yield the potential of reproducing interaction scenarios known from real-life situations. Avatars, for example, give users the opportunity for self-identification and encourage them to start social interactions with each other. Walking, running and jumping are navigation forms everyone is familiar with. In contrast to other human interface concepts such as desktop applications or Web sites, navigating through a virtual space prevents users from experiencing visual cuts and, thus, loosing their context. A virtual continuum implicitly creates spatial relations between the objects it contains. When used for representing content it helps users to grasp inherent relations of its entities by creating a mental map of the perceived environment.

The AudioSquare takes advantage of the virtual world paradigm for representing music archives. After downloading and starting a client-application, a user can choose an avatar and enter the virtual world over the Internet. A dedicated server delivers the virtual environment and manages the positions of present avatars. The client-server approach enables a social platform where users are encouraged to start conversations about the presented content through a simple text chat. All objects, avatars and the landscape designed for *The AudioSquare* are reminiscent of real-life scenes. This is based on the assumption that users don't want to learn the principles of every virtual environment from the ground on. Rather, they are supported in quickly orienting themselves in a scenario that looks familiar to them and are able to focus on the main purposes of the virtual world. The music is represented by 3D objects emitting spatial sound. To ensure that users hear the same music when in the same location the sound is streamed over the Internet by a media server.

The workflow for creating the virtual world of *The AudioSquare* comprises three main steps. The first step is the creation of a basic environment with a terrain, buildings, interior and other objects. The second step is the creation of 3D objects for music representation which can be either a single object such as a

radio or a group of objects, for example a table with a speaker on its top. However, these objects are stored as assets in a simple repository, from where they can easily be used for representation. The last step is to place *marker-objects* in the basic environment which specify locations for the music representation. An automatic process places the assets in the virtual world, whereby two different approaches for organizing and representing the underlying music archives are supported. On the one hand, automatic organization by means of a *self-organizing map* is provided. On the other hand, music tracks can be organized manually by generating a specific folder hierarchy on the file system. These two approaches are offered in order to ensure high flexibility for different demands. If the representation of large music archives is desired, the automatic method should be considered. For showing only a small to medium audio collection and for emphasizing on individual tracks, the manual option may be the better choice. However, both modes can be used in the same virtual environment at the same time.

4.1 SOM-based Music-Organization

The SOM-based approach relies on automatic organization according to the sound characteristics of the music tracks. Every audio file within an archive is analyzed for a time-invariant representation of the sound in terms of a set of features (cf. Section 3.1). The features are passed to a *self-organizing map* which positions each music piece on a two-dimensional matrix (cf. 3.2). Such a matrix consists of n x m units, each containing several audio tracks. Since the SOM classifies music according to the sound, similar audio tracks are located close to one another or even on the same unit. *The AudioSquare* transforms the SOM-generated matrix into a three-dimensional representation of objects. A *marker-area* can be defined in the virtual world to specify the boundaries wherein the representation should take place. It also designates a matrix considered for representation, which section thereof and an asset from the repository. As depicted in Figure 4.1, these areas can be used to distribute a matrix over multiple places. Two *marker-areas* located in different rooms in the virtual world address two different sections of a SOM-generated matrix. Additionally, each area refers to another asset of the repository. The result in the virtual space is a matrix of 3D objects separated into two sections with different visual designs.

4.2 Manual Music-Organization

The manual organization approach addresses the three-dimensional representation of a simple folder hierarchy on the file system. Top-level folders stand for an umbrella term wherein further folders contain the audio files. Each folder contains a text file that describes its contents by name, date and an optional short

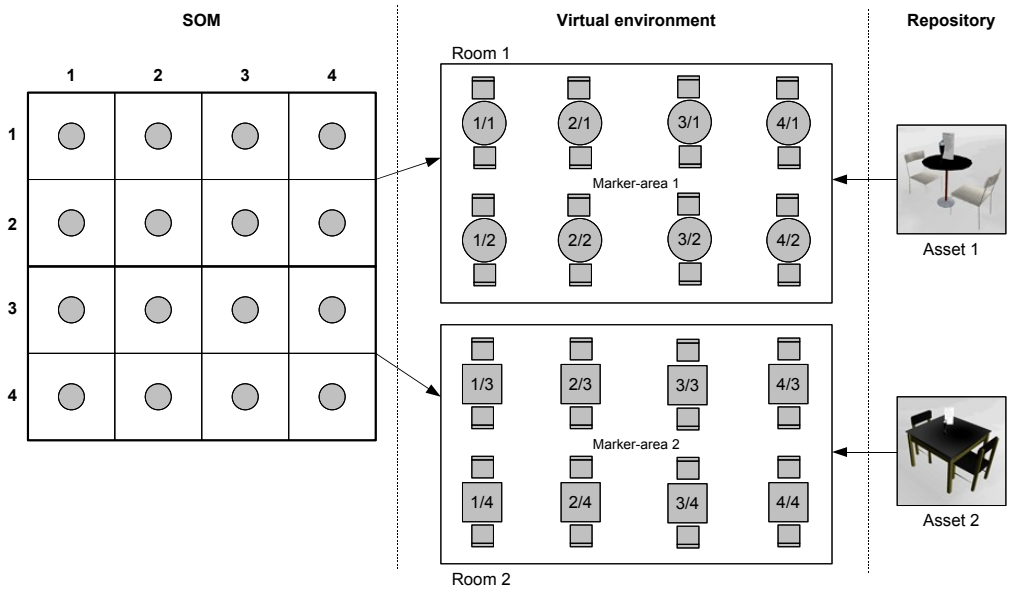

Figure 4.1: Automatic organization based on a SOM.

description. Within the virtual environment, the top-level folders are represented by buildings. The low-level folders are represented by objects that are located inside these buildings. The descriptions in the text files are displayed on virtual signboards attached next to the respective objects. The design of a building as well as its interior objects are stored as an asset in the repository.

Figure 4.2 shows an example for how a simple folder hierarchy is transformed into a visual representation in the virtual environment. The top-level folder is considered as a collection of albums from the rock band "The Beatles". It contains two folders which refer to the albums "White Album" and "Abbey Road", which contain the respective audio files. Transformed into the virtual world, the top-level folder is represented by a house. An attached sign shows a description as stored in the text file. Two hi-fi systems standing on a table are located inside of the building. They represent the two lower-level folders containing the audio files. Two signs above the stereos show the title and the date of publication of the respective albums.

A *marker-object* set in the virtual world designates a point where the music representation should start. It also identifies the location of the folders on the file system and a template from the repository. Additionally, one out of three layout algorithms as depicted in Figure 4.3 is defined with this object. When linear layout is specified, the buildings are placed sequentially next to another. The first building is aligned to the starting point, the subsequent ones are placed one after another with a definable distance d. The generated arrangement is comparable

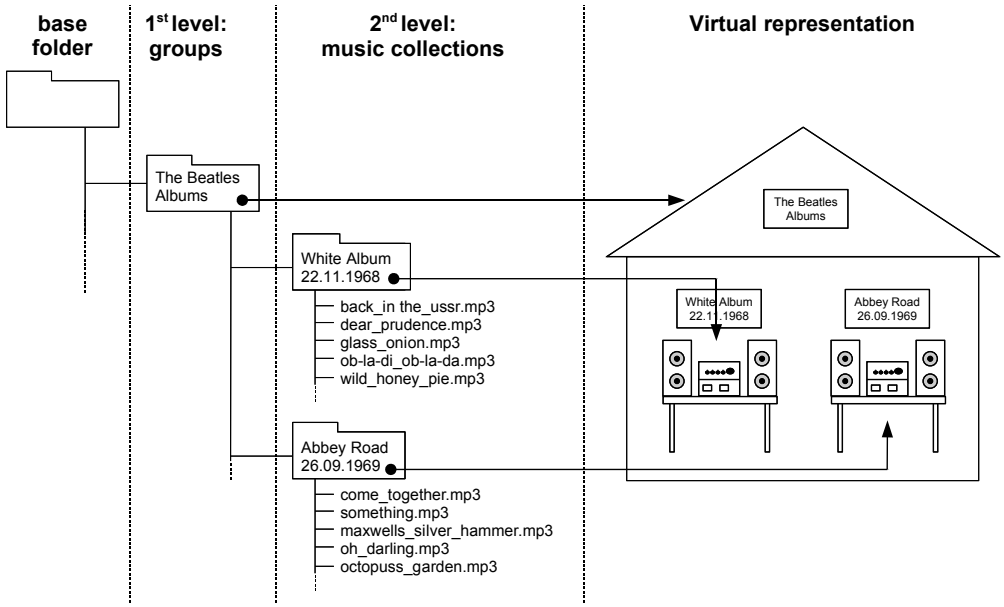

Figure 4.2: The manual organization of music is based on a folder structure. In the virtual environment, the top-level folders are represented as buildings and the low-level folders containing the audio files are represented as interior.

to a residential area with separate houses. In case of a circular layout the starting point acts as the center of a circle with a given radius r. The buidings are placed along the circle with a given distance d to one another. Last but not least, the matrix-style layout arranges buildings similar to a checker board. The number n of objects in a row can be defined as well as the horizontal and vertical distances dx and dy.

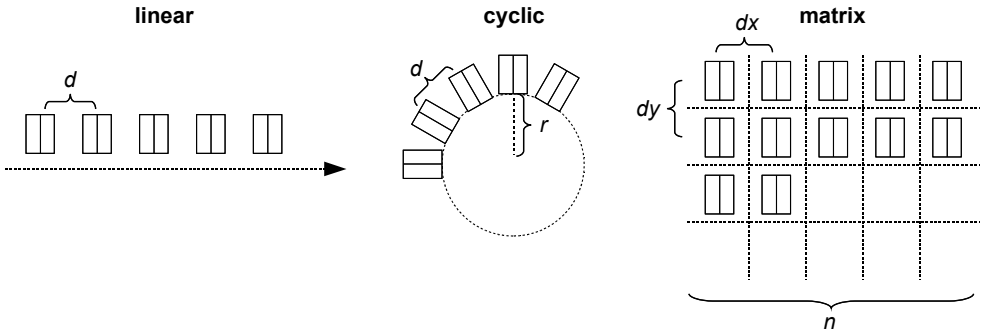

Figure 4.3: Layout algorithms for manual music organization.

Chapter 5

Implementation

This chapter discusses the implementation of *The AudioSquare*. The core of the system is based on the *Torque Game Engine* as introduced in Chapter 3.3. Its integrated client-server architecture facilitates the development of a collaborative virtual world. Other features such as scripting and the availability of the full source-code make it possible to adapt the game engine to own requirements. The broadcasting of the music is done with the open source media server *Icecast*[1]. The playback of the streams on the client-side is realized by integrating the *FMOD*[2] API, which is dedicated to all kinds of sound-processig, into *Torque's* source code. While *Torque* and *Icecast* are used on runtime, a Java-based application, henceforth referred to as the *Wrapper*, preprocesses the data needed to integrate music representation into the virtual environment.

5.1 System Architecture

The system architecture of *The AudioSquare* is depicted in Figure 5.1. Basically, it can be divided into a preprocessing-layer and a runtime-layer. The core of the runtime-layer is a dedicated *Torque* server, responsible for the execution of the virtual environment. This includes the instantiation of the virtual world on startup and the coordination of the connected clients. For broadcasting audio streams over the Internet, *Icecast* is used as a media server. A *Torque* client is mainly responsible for the user interface and the audiovisual representation of the virtual world. It is connected to both, the *Torque* server as well as to the *Icecast* server.

The basic definition of the virtual world is stored in a *mission-file*. It contains information about the terrain, buildings and other objects. The objects used for representing the music are described in a separate XML-file, the *objects-file*. It

[1] http://www.icecast.org/ (date of access: 23.02.2008)
[2] http://www.fmod.org/ (date of access: 22.02.2008)

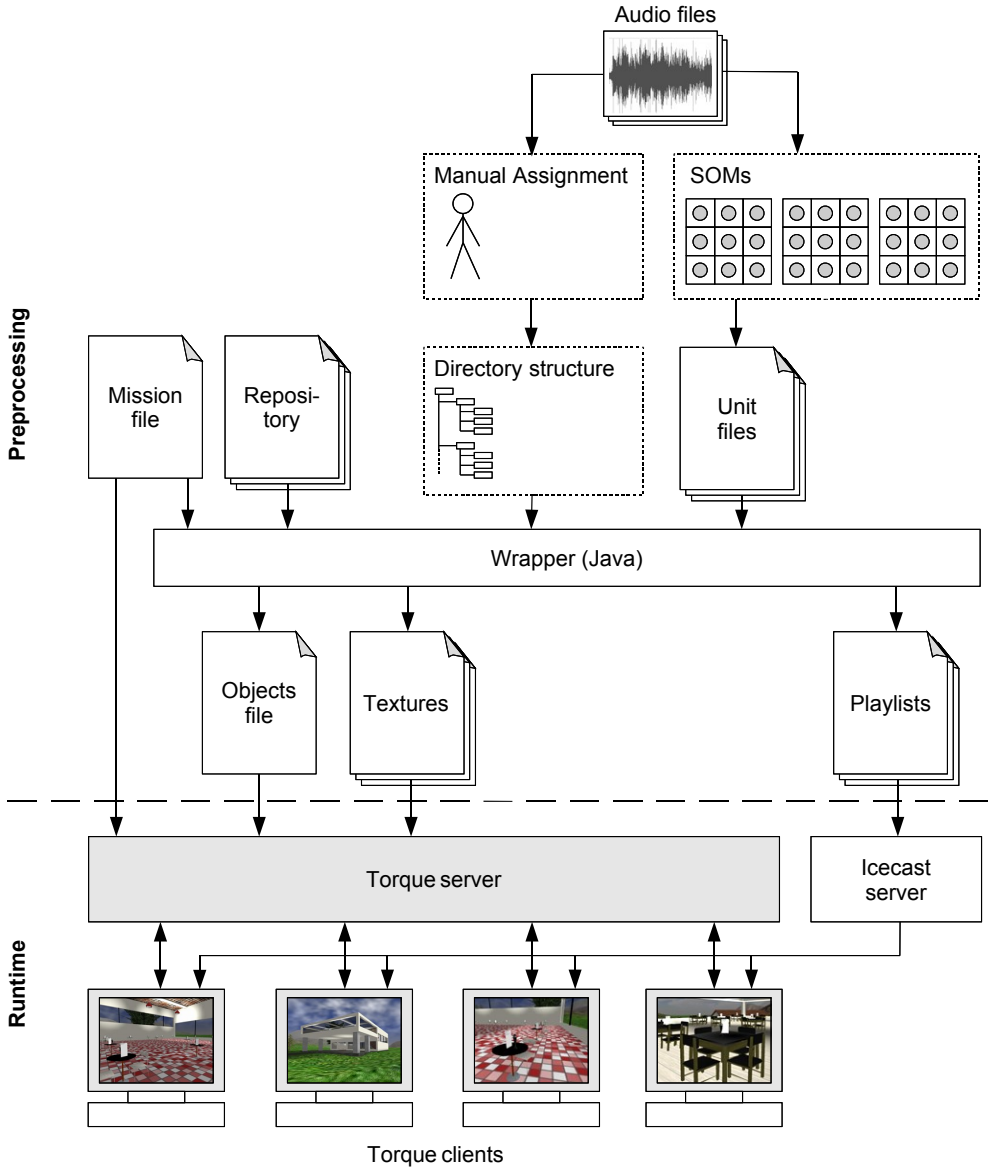

Figure 5.1: System architecture.

is generated by the Java-based *Wrapper* that performs all steps necessary before starting the *Torque* server. When the *Wrapper* starts, it loads a configuration-file and scans the *mission-file* for *marker-objects* that designate locations for music representation. According to the encountered objects it loads the data that describes the organized music files. When a *marker-object* refers to a SOM-based organization it parses the respective *unit-file*, a text file describing the SOM-generated matrix. When a *marker-object* refers to a manually generated

folder hierarchy, it scans the related directories. When the *Wrapper* has processed all input data, it starts to generate the *objects-file* which describes every single music-related object. The information about the appearance and arrangement of these objects is gathered from predefined asset-files in a simple repository. Playlist-files and a start-script are created in order to configure the *Icecast* server. The playlists are also used to generate JPEG-images that show the title and the artist for each music track. The images are used as textures for playlist-objects within the virtual environment. Further images are created for each text file that describes a folder for directory-based organization. In this case, the images are used as textures for signboards-objects. When the *Wrapper* has finished its tasks, the *Torque* server as well as the *Icecast* server can be started. The *Torque* server creates the basic environment by loading the *mission-file* and integrates the music representation according to the definitions in the *objects-file*. The start-script initializes the *Icecast* server and creates an audio stream for each playlist-file. When both servers are successfully started, the system is running and ready for client connections.

5.2 Implementations in Torque

The *Torque* SDK comes with an example project that already provides a lot of functionality. Client-server communication, text-chat, user navigation, a graphical user interface (GUI) and much more can be used and adapted to own needs. This section describes the main changes and additional implementations in *Torque* necessary for the realization of *The AudioSquare*. The visual design of the GUI has been adapted and new elements have been added. Additional routines written in *TorqueScript* were implemented for importing the *objects-file*. New *Datablocks*, *Torque's* concept for describing object-types in *TorqueScript*, have been created. In order to enable playback of broadcasted audio streams over the Internet within the virtual environment, a new audio object has been integrated into *Torque's* source code.

5.2.1 The User Interface for the Torque Client

The GUI of *The AudioSquare* follows a straightforward black-and-white scheme. A window appears in transparent white with black text, while the colors of its title-bar are displayed inversely. The menu on the start-screen contains buttons for entering the virtual world, changing the display settings and closing the application. When the start-button is pressed, a small window shows up where a user may select an avatar and type in a name for identification. On confirming, the client-application connects to the *Torque* server.

The view of the virtual world fills the whole window or the whole screen, depending on the selected display-option. When the avatar encounters an audio

source, a small head-up display (HUD) appears on the lower left of the screen. It describes the currently playing track by title, author, album and genre. Aligned to the right border of the screen, an additional HUD appears showing the whole playlist of the audio stream. This HUD can be disabled by pressing "F6". A small chat-HUD can be found on the top of the screen. The key "C" may be pressed in order to start chatting. The chat message is sent to all other users within the same area as the sender. More information about navigation and interaction within the virtual environment may be found in Table 5.1.

Key / Mouse Move or Button	*Description*
Horizontal mouse movement	Turn left / right
Vertical mouse movement	Look up / down
"w"	Move forward
"s"	Move backwards
"a"	Move left
"d"	Move right
"e"	Zoom view (useful for viewing details such as playlists)
space	Jump
Left mouse button	Skip actual music track (only when close to an audio source)
"c"	Start chat (when chat-HUD is open)
"F4"	Toggles the visibility of the chat-HUD
"F5"	Toggles the visibility of the debug-HUD
"F6"	Toggles the visibility of playlist-HUD

Table 5.1: Keyboard and mouse settings for navigation

5.2.2 Customized 3D Objects

Torque provides many 3D objects that can be used in the virtual world. Some of these objects had to be extended or rewritten to fit the needs of *The AudioSquare*. Usually this is done by creating new *Datablocks* in *TorqueScript*, a concept of the game engine for creating new object types described by a shape, properties and a behavior.

Objects representing Playlists and Signboards

Two similar 3D objects show metadata about the presented content within the virtual world. In both cases images mapped as textures onto the objects show descriptive text. The corresponding image files are automatically generated by the *Wrapper*. Figure 5.2 shows an example for such objects. As seen in the left picture, a playlist-object is reminiscent of a menu card, showing a list of music tracks designated by title and artist. The signboard-object on the right is used for directory-based mapping and shows the metadata that describes the contents

of a folder in a text file. It indicates that the corresponding audio stream plays music from the genre “Rock & Pop” whereas the featured band is called “Very large Array”.

Figure 5.2: 3D objects representing metadata. The left picture shows a playlist-object, the right picture shows a signboard-object.

Marker-Objects

As described in the conceptual design in Chapter 4, *marker-objects* integrated in the basic virtual environment specify locations for music representation. For SOM-based organization a *Datablock* called “SomMapping” has been created. The corresponding object is a simple cube that is only visible in the *World Editor*. Its position, rotation and scale describes the area wherein a SOM-generated matrix should be represented. Further properties specify the name of the *unit-file* describing the matrix, the section of the matrix and the name of the asset-file from the repository. The corresponding *Datablock* for directory-based organization is called “DirectoryMapping”. Similar to the other *marker-object*, it is a cube invisible in runtime. Its location indicates a starting point from where on the representation starts. Other properties specify the location of the folder structure, the name of the asset-file from the repository and the information needed for the layout of the representation (cf. Chapter 4.2). Code-examples for both types of *marker-objects* are demonstrated in Appendix C. Figure 5.3 demonstrates how *marker-objects* are displayed within the *World Editor*. The left example shows a “SomMapping” object placed inside of a building. Its boundaries specify the area wherein the SOM-generated matrix is to be represented. The right example shows a “DirectoryMapping” object, displayed as an outlined cube.

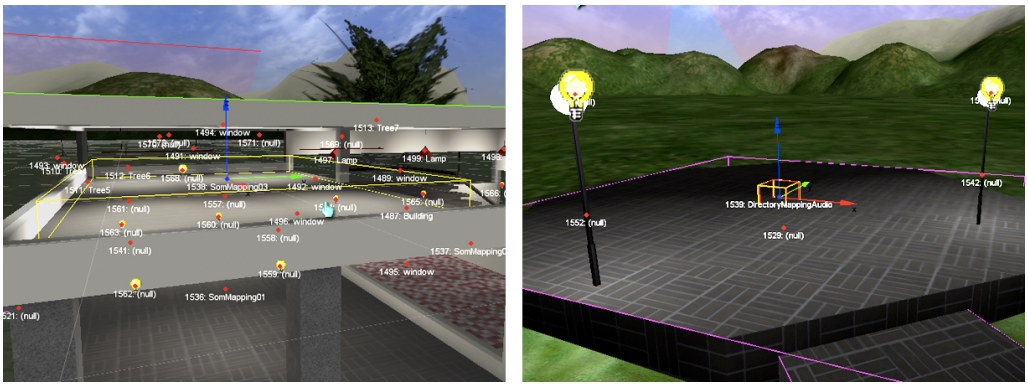

Figure 5.3: *Marker-objects* indicate locations for music representation. The objects are signified by the yellow outlines. The left picture defines an area for SOM-based representation. The right picture defines a starting-point for directory-based mapping.

Audio Triggers

A *Trigger* in *Torque* is an invisible cube which initiates an event as soon as a user enters or leaves it. A customized child-*Datablock* of *Trigger* handles the head-up displays which show information about a specific audio source. When a user enters such a *Trigger*, the embedded meta-data of the respective audio stream is used to show information about the currently playing track and the playlist on the HUDs.

AudioStreamEmitter

Torque's original *AudioEmitter*-object plays audio spatially when placed in a virtual environment. Since it only supports playback from the local file system, a new object called *AudioStreamEmitter* has been added into the source code. This object takes advantage of FMOD, a proprietary cross-platform programming interface for audio playback. It is free of charge for non-commercial use, easy to implement and supports the playback of MP3-based audio streams over the Internet. Moreover, the streams can be played as 3D audio sources.

5.2.3 Setting up the virtual environment in Torque

At the beginning of a new virtual world which is supposed to act as a venue for the exploration of music archives stands the basic design. In *Torque* this is made with its built-in *World Editor*. Creating environmental objects such as a terrain, the sun and water areas are supported as well as importing and placing 3D objects. When the design of the basic environment is completed, the locations for music representation are defined by placing the *marker-objects* for

SOM-based and directory-based representation. The whole world definition is stored in a single *mission-file*, a text-based file that describes the 3D objects via *TorqueScript*.

5.2.4 Creating assets for the Repository

In the virtual environment the music is represented in terms of spatially positioned audio sources as well as visually in the form of 3D objects. A set of these objects is stored as an asset in a simple repository. Such an asset is created in the *World Editor* and stored in a separate *mission-file*. The repository is thus a directory containing *mission-files* which are used for music representation. An asset representing a unit of a SOM-generated matrix contains the following objects:

- At least one 3D object for visual representation.
- One playlist-object.
- One *AudioStreamEmitter* for playing a broadcasted audio stream.
- One *Trigger* object for identifying avatars close to the audio source.

In case of directory-based representation an asset covers even more objects. It contains a building wherein several objects for music representation are located. Accordingly, the following objects are needed:

- A building which acts as a container for music representation.
- Other 3D objects related to the building (eg. windows).
- Multiple sets of objects as described for SOM-based mapping for each potential audio stream within the building. Thus, one such set consists of 3D objects, one playlist-object, one *AudioStreamEmitter* and one *Trigger*.
- Several signboard-objects, one attached to the building and multiple objects placed next to the audio sources.

The objects used for representation are collected in a *Torque*-specific *SimGroup* which represents a hierarchical element for collecting objects. An asset can contain multiple collections in order to facilitate variations of the same visual design.

Figure 5.4 depicts four different examples for assets as seen in the *World Editor*. The rightmost picture shows an example for directory-based representation. It is a building containing several stereos on a table. A small signboard next to the entrance of the building displays the description of the music presented inside. The other pictures show designs of assets for SOM-based representation.

All three examples show a table surrounded by chairs, with a small speaker on the top. The third picture from the left shows, how variations of the same visual design are realized within an asset.

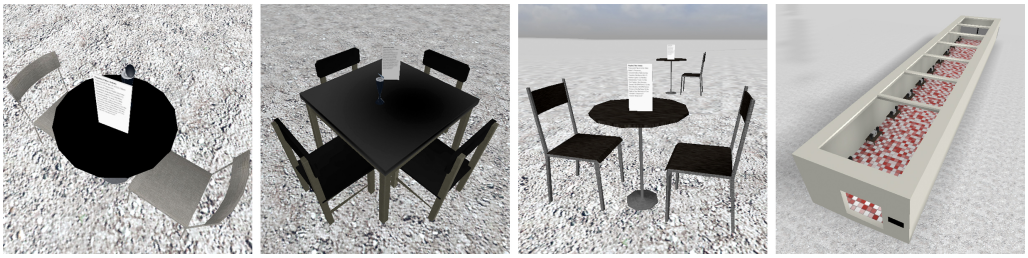

Figure 5.4: Assets for the repository. The rightmost picture depicts an asset for directory-based mapping, the others address SOM-based representation.

5.2.5 Importing the Objects-File

When the *Torque* server starts, it loads the *mission-file* that describes the basic virtual environment. A custom script parses the XML-structure provided by the *objects-file* and creates an internal tree-structure of *Torque's ScriptObjects*, very flexible objects that can contain randomly assigned attributes. This structure is then used to integrate and configure 3D objects for music representation in the virtual world. When this process is finished, the server is ready for client-connections.

5.3 Implementation of the the Wrapper

The *Wrapper* is a Java-based command-line application that processes the data specified for music organization in order to create a definition of the visual representation within the virtual environment. Several internal Java-classes represent the data provided by the input-files, locations and objects in the virtual world, the audio streams for the *Icecast* server and the XML-output. When started, the *Wrapper* performs the following tasks consecutively:

- Read the configuration-file for the *Wrapper* itself. It contains file paths, settings for the graphical output and settings for the *Icecast* server.
- Scan the *mission-file* that describes the basic virtual environment for *marker-objects* defining locations for SOM-based as well as directory-based representation.
- Load asset-files from the repository as specified by the *marker-objects*.

- Load a text file that contains metadata extracted from all audio files in the music archive.
- Load the *unit-files* referenced by *marker-objects* in the *mission-file*.
- Load the folder structures as specified by *marker-objects* in the *mission-file*.
- Create images for playlist-objects and signboard-objects as used for representing metadata within the virtual environment.
- Create a startup-script and playlist-files for the *Icecast* server.
- Create the XML-based *objects-file* which specifies all 3D objects used for music representation.

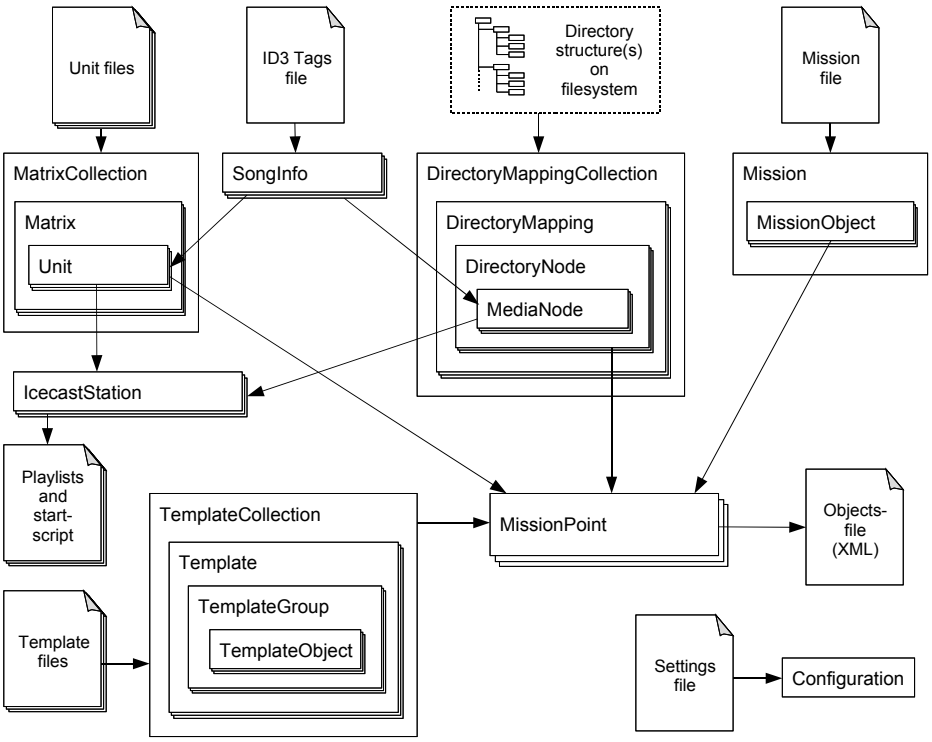

Figure 5.5: The Java-classes of the *Wrapper* support the representation and processing of the *mission-file*, one or more *unit-files*, one or more manually created directory structures, song-metadata, locations in the virtual environment and the *Icecast*-streams.

Parameter	*Description*
$PLATFORM	Operating system for runtime (“Windows”, “Mac”, “Linux”)
$UNIT_PATH	Path where the unit-files reside
$MISSION_FILE	Path for the mission-file which describes the virtual environment
$DIRECTORY_ROOT	Path for the directories that represent the manual mapping
$TORQUE_OUTPUT	Path for the final XML-file that is used by the game engine
$ID3_INFO_FILE	Path for the file that contains the extracted ID3 tags
$TEMPLATES	Directory of the repository
$IC_SOURCEPATH	Directory where the templates for the Icecast files reside
$IC_TARGETPATH	Directory where the compiled Icecast files should be written
$IC_SONGPATH	Absolute path on the server where the audio files reside
$IC_PORT	TCP/IP-port used by Icecast
$IC_ICES_ENTRY	Bash-command template for starting an Ices instance
$IC_MAIN	Name of the Icecast start script
$IC_PLAYLISTFILENAME	File name template of a playlist file
$IC_CONFIG_XML	File name of the configuration for Icecast
$IC_URL	URL of the Icecast server
$PLAYLIST_WIDTH	The width in pixels of the image that represents a playlist
$PLAYLIST_HEIGHT	The height in pixels of the image that represents a playlist
$PLAYLIST_FONT_TYPE	The font used for a playlist image (e.g. Arial)
$PLAYLIST_FONT_SIZE	The font size for a playlist image
$PLAYLIST_FONT_LINEHEIGHT	Line spacing between two entries on the playlist
$PLAYLIST_MAX_CHARS	Maximum of characters in a playlist entry
$PLAYLIST_IMAGE_FOLDER	Folder where the playlist image are stored
$LABEL_WIDTH	The width in pixels of the image that represents an object label
$LABEL_HEIGHT	The height in pixels of the image that represents an object label
$LABEL_FONT_TYPE	The font used for an object label image
$LABEL_FONT_SIZE	The font size for an object label image
$LABEL_FONT_LINEHEIGHT	The line spacing used for an object label image
$LABEL_FONT_MAX_CHARS	Maximum characters in a line of an object label

Table 5.2: Parameters in the configuration-file for the *Wrapper*.

5.3.1 Class-Representation of Torque Objects and Input Files

The Java-classes of the *Wrapper* and their major dependencies are depicted in Figure 5.5. The first file loaded is the configuration-file which is represented by the static class *Configuration* to provide general settings across all other Java-classes (cf. Table 5.2). The *Mission* class scans the *mission-file* for *marker-objects* which specify the locations for music representation. According to the *Datablock*-types of the *marker-objects* the *Wrapper* considers different representation scenarios. When the object is of the type “SomMapping”, its position, size and rotation around the vertical axis define the area for visualizing a SOM-based matrix. Additional properties, “xfrom”, “yfrom”, “xto” and “yto”, specify the section of the matrix that should be imported. When the *marker-object* is of the type “DirectoryMapping”, its position and rotation around the vertical axis define a starting point for the representation of a directory structure. By setting one of the three properties “linear”, “circular” and “matrix” to true the layout algorithm for placing the buildings is selected. Depending on the layout, further properties define the distances between the objects, the radius of the circle or the size of the matrix. The property “directory” refers to the location of the directory structure. Finally, the attribute “templatename” specifies the filename of the asset in the repository.

A 3D object within an asset is represented as an instance of the *TemplateObject* class. Several of these objects are collected in instances of the *TemplateGroup*

class. Accordingly, *TemplateGroup* stands for a set within an asset. In order to allow variations of the same design, multiple sets are aggregated in the *Template* class which finally represents a whole asset-file. Different assets are collected in *TemplateColletion*.

The *Matrix* class represents the data loaded from a single *unit-file*. Since multiple *unit-files* are supported, the representations are collected in the *MatrixCollection* class. Instances of the *Unit* class describe the units of a SOM-generated matrix and contain attributes for the position in the matrix, a list of filenames of the audio tracks and a unique number to identify the audio stream for the *Icecast* server.

The *DirectoryMapping* class represents a folder hierarchy for manual organization. Several instances of *DirectoryNode* refer to a low-level folder containing the music files. The audio files are represented by instances of *MediaNode* which describe an audio track by its filename, title, artist, genre and a stream number for the *Icecast* server. Since it is possible to represent more than one directory hierarchy in the same virtual environment, the class *DirectoryMappingCollection* aggregates multiple *DirectoryMapping* instances.

The *Wrapper* retrieves the metadata describing the music files from a single text file. The file is generated by a Unix-based shell-script that scans all audio files in a given folder for meta-tags. This file is loaded by the class *SongInfo* that creates instances of itself for every title. *IcecastStation* is responsible for the creation of the audio streams broadcasted by the *Icecast* server. An *IcecastStation* instance represents an audio stream and is created for each instance of *Unit* and *DirectoryNode*. It stores a list of filenames within a playlist and a unique stream number.

MissionPoint represents a specific location in the virtual world. Its instances specify where representation of music should take place. On the one hand this is the case for SOM units. Every instance of *Unit* creates a *MissionPoint* according to the location and size of the respective *marker-area* and the number of columns and rows in the matrix. On the other hand, every *DirectoryNode* instance creates a *MissionPoint* according to the starting point and the layout algorithm as given in the corresponding *MissionObject* instance.

5.3.2 Processing the Output-Files

The main output of the Wrapper, the *objects-file*, contains XML-data that describes all objects used for the representation of the music. It is generated by iterating through every instance of *MissionPoint* and calling the method for building an XML-node on the *Template* instance. If a *Template* instance contains more than two *TemplateGroup* objects, one of them is selected randomly. An example of an *objects-file* is given in Appendix B. The following nodes and sub-nodes are generated in this XML file:

- *unit*: This node contains all objects that represent a single unit of the respective SOM.
- *dirmapping*: This node contains all objects that represent top-level folders in directory-based mapping.
- *songinfo*: Represents a collection of music tracks that belong to an audio stream.
- *song*: Sub-nodes of *songinfo* representing a music track by title, artist, album and genre.
- *static*: Describes a 3D object that represents the media visually. It refers to a 3D file in *Torque's* DTS-format, its position, rotation, scale and name.
- *interior*: Describes a 3D object used for representing buildings or similar objects. The XML-node refers to a 3D file in *Torque's* DIF-format, its position, rotation, scale and name.
- *audioemitter*: Stands for an object that plays the audio stream and contains its name, position, rotation, scale and the URL of the stream including its unique number. Further properties describe audio-specific parameters. These are the distance wherein the sound is played at full volume (reference distance) and the maximum distance wherein the sound is audible.
- *playlist*: Represents a 3D object that shows the playlist for the audio stream as an image-texture. It refers to a 3D file in *Torque's* DTS-format and contains properties for its position, rotation, scale, name and the filename of the image to use as texture.
- *label*: Represents the objects that act as signboards. It has the same properties as the *playlist* object listed above.
- *objecttrigger*: Describes a *Trigger* that initiates an event for showing the HUDs with information about the audio tracks when the user crosses its boundaries. The node specifies the *Trigger* by position, rotation, scale and name. Additional information is provided for debugging purposes.

Besides the *objects-file*, the *Wrapper* creates JPEG-images which are used as textures in the virtual environment. One the one hand, this is the case for playlist-objects. The corresponding image contains the artist and title of each track. On the other hand the images are used for signboards to describe the buildings and audio streams when directory-based organization is used.

For the *Icecast* server, every instance of *IcecastStation* creates a playlist-file with a unique name. Such files simply contain the file path of each audio file, separated by a line break. A start-script is also generated providing the code for starting the *Icecast* server and configuring the audio streams.

5.4 The Icecast Streaming Server

The *Torque* server does not support broadcasting of audio over the Internet. This is done by *Icecast*, an open source media server, developed by the non-profit organization *Xiph.org*[3]. *Icecast* provides a large number of audio streams at the same time. It supports the MP3-format as used for *The AudioSquare* as well as the license-free OGG-format. The decision for MP3 was made due to the technical requirements of FMOD which receives the streams on the client-side. In order to run a streaming environment, a single *Icecast* process and multiple instances of *Ices*, one for each audio stream, are needed. An *Ices* instance acts as a transcoder which continuously converts audio files on-the-fly according to a defined playlist. The converted audio stream is passed to the *Icecast* server that delivers the data over HTTP, where each stream is identified by a different TCP-port.

Figure 5.6 depicts the *Icecast* setup used for *The AudioSquare*. The streaming environment is started by a shell-script called "icecast.sh". Just like the playlist-files, this script is generated automatically by the *Wrapper*. When the script executes, it starts the *Icecast* server and one *Ices* instance for each playlist. When all processes are started, the streaming environment is ready for broadcasting audio to the *Torque* clients.

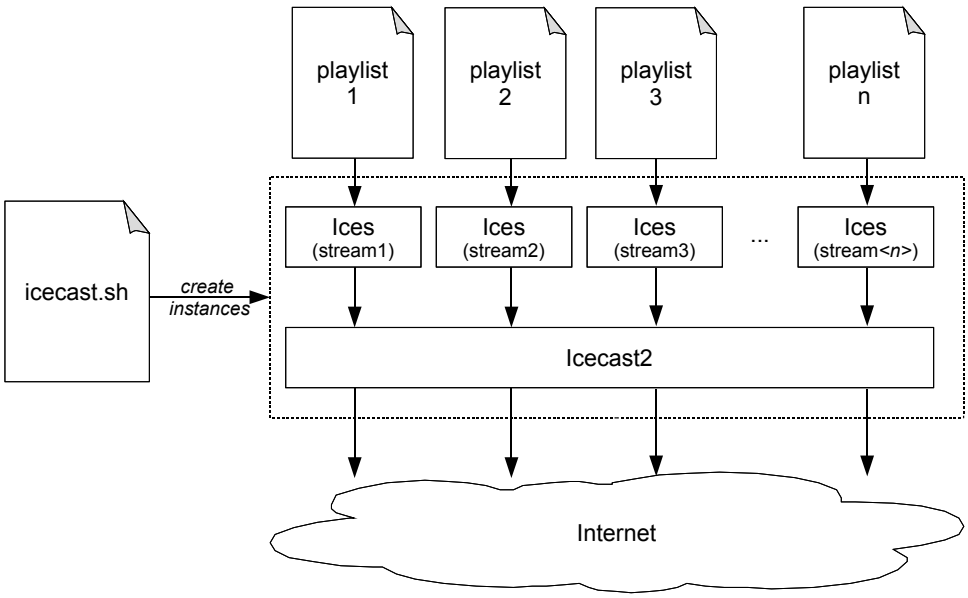

Figure 5.6: Icecast architecture

[3]http://www.xiph.org/ (date of access: 23.02.2008)

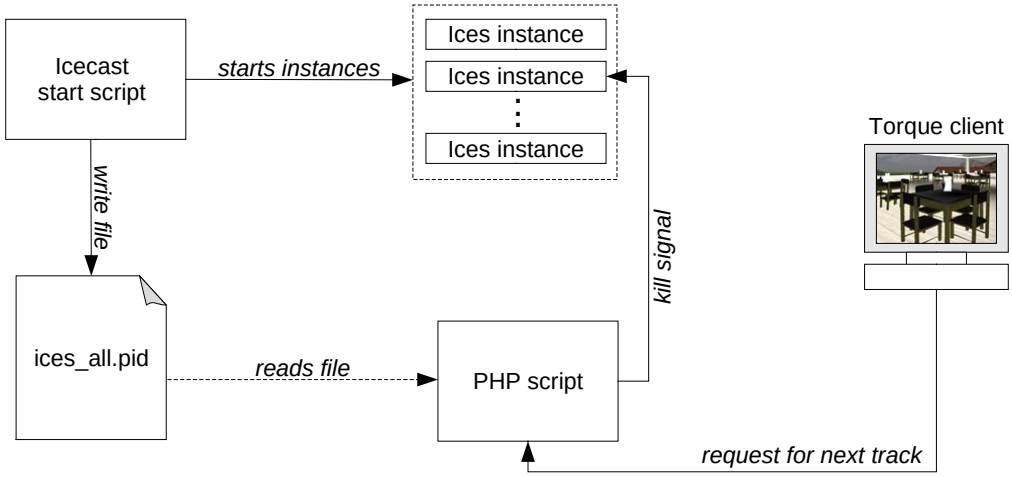

Figure 5.7: Setup for skipping tracks

Icecast also integrates metadata for the currently playing music track by using the ID3-tags[4] embedded in the audio files. Since it only features information about title and artist in the original version, the source code has been modified for *The AudioSquare*. The customized version provides additional tags which are album title, genre and the position of the actual track in the playlist. The *Torque* clients use the metadata for displaying information about the currently playing track in the HUD and for determining its position in the playlist.

Users can skip to the next music track when they are close to an audio source by pressing the left mouse button. Since the only way for skipping in *Icecast* is to send a Unix-based "kill"-signal to the respective *Ices* process, a special setup has been implemented for this feature. As depicted in Figure 5.7, the startup-script for *Icecast* writes the process numbers and the stream names into a text file called "ices_all.pid". When a user wants to skip an audio track by pressing the mouse button, a PHP-script is called via HTTP-request containing the name of the stream. The script reads the text file in order to determine the process-id of the respective *Ices* instance. By sending the kill-signal "SIGUSR1" to this process, the current track is skipped and the next one starts playing.

[4]http://www.id3.org/ (date of access: 22.03.2008)

Chapter 6

Visit The AudioSquare

The final version of *The AudioSquare* can be downloaded from the Web site *http://www.theaudiosquare.net* and runs on Windows XP, Linux and Mac OS X. The latter operating system only covers the visual representation of the virtual world, since FMOD's support for OS X relies on a different version which is not used in *Torque's* source code. In order to run *The AudioSquare*, just extract the downloaded package to any location on the local hard drive. A broadband Internet connection is required in order to connect to the *Torque* server and to listen to the audio streams. There is also an option for starting the application offline if no connection is present. *The AudioSquare* features music from *Magnatune*, an online-disributor of loyalty-free music. As depicted in Figure 6.1, the virtual world has four areas. Users start their exploration in the welcome-area (1) at the center. Basic information about the virtual world can be found on the screen in the information-area (2). The *SOM showroom* (3) is a building, where the music tracks are organized by a *self-organizing map*. The *manual showrooms* (4) represent music organized according to a predefined directory structure. Each building covers a different genre, according to *Magnatune's* library. Hence, the genres are "Classical", "Electronic", "Jazz & Blues", "Metal & Punk" and "Rock & Pop". The subsequent text takes the reader on a tour through *The AudioSquare* by telling a short fictitious story.

The virtual tour starts at the imaginary home of Bob. He likes to listen to all kinds of music and has recently heard about a new virtual place on the Internet where people can explore music and talk about their favorite songs online. Fortunately he remembers the name of the place, *The AudioSquare*, and starts to search for it on the Web. He quickly finds the project page and downloads a package containing the client-application to his computer. After unpacking the contents, he immediately starts the program. The start-screen of *The AudioSquare* pops up, presenting a background image that gives him a clue of how the virtual world looks like (cf. Figure 6.2). A simple menu in the form of buttons with small icons

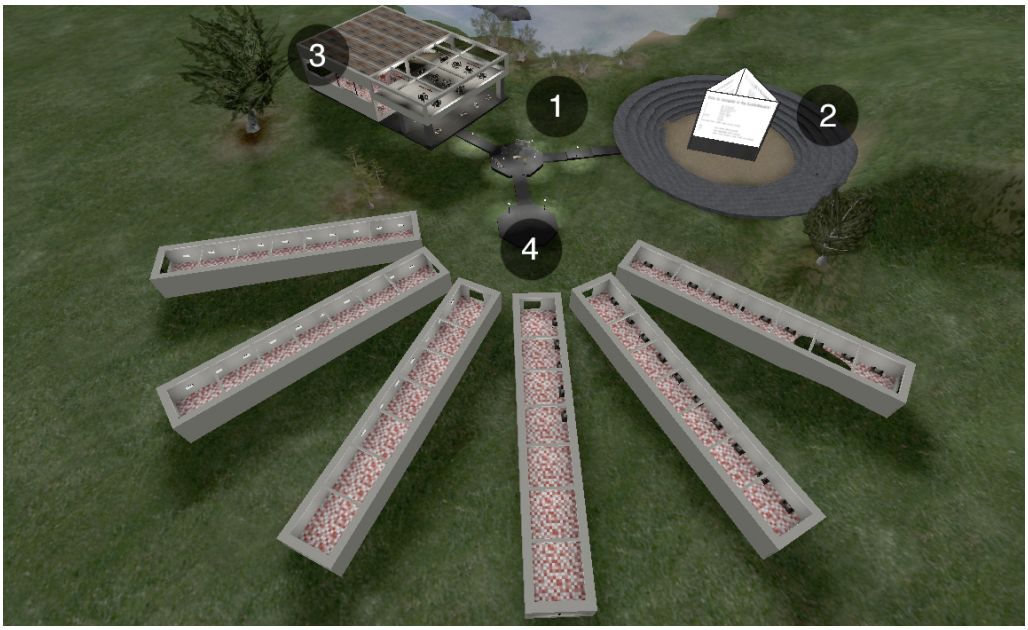

Figure 6.1: Bird's-eye view on *The AudioSquare*

tell him what he can do next. He decides to enter the virtual world online and clicks on the first button in the menu.

A small window appears where Bob is encouraged to select an avatar and to enter a name (cf. Figure 6.3). He chooses the avatar that looks like a man with a yellow-striped pullover. Since he has nothing to hide, Bob simply enters his real name. Then he clicks on the start-button located on the bottom left of the window. Immediately he is informed that *The AudioSquare* is being started.

When the loading from the server has been completed, Bob finds himself in the middle of the virtual world (cf. Figure 6.4). As he looks around, he recognizes a big house, an architectural setting of long buildings arranged along an imaginary circle and a big screen showing some text. At the center, where Bob is standing, he can see guideposts telling him the names of the different locations.

Since he is new to this world, he decides to head towards the screen, expecting to be informed about the virtual world (cf. Figure 6.5). By reading the text, he learns more about *The AudioSquare* and how he can interact and communicate with other users. Next he decides to visit the big house.

Inside the building which has been referred to as the *SOM-showroom* by the guidepost, he sees an arrangement of tables (cf. Figure 6.6). He sees speakers standing on these tables. This suggests him that he can find music there. As he encounters one of the tables, he immediately starts hearing music. The playlist on the table next to him shows him what kind of music is being played here. Since he likes the music, he starts to walks around a little bit.

Figure 6.2: The start-screen shows a preview of the virtual world and a menu.

After a while Bob wants to hear more music and decides to go upstairs (cf. Figure 6.7). The playlist on one of the tables shows a song he might know. Wondering if he is right, he wants to skip the currently playing track so he can immediately listen to the song in question. Since the featured music archive comes from *Magnatune*, a rather unknown music-label, Bob must admit that he doesn't recognize the track.

Bob wonders what the other people think about this place. He decides to ask someone close to him. He walks over to an avatar called Sally, opens the chat-HUD by pressing "F4" and starts his message by pressing "c". After saying "hello", the two start a small conversation about *The AudioSquare* (cf. Figure 6.8).

During their conversation, Sally tells him about the other area, where music is organized in a different way. She tells him that she found a very beautiful classical piece there. She offers to show him the area. Bob accepts and they start walking towards the other showrooms (cf. Figure 6.9).

Sally leads him to the place where her favorite song is being played and switches to the track she wanted to show him. Bob listens for a while, then they start to talk about the music piece (cf. Figure 6.10).

The two virtual strangers stick together for a while, chatting about music and other topics while listening to the music played at their location. Finally, Bob

Figure 6.3: The login-window lets a user select an avatar and type in a name.

and Sally say good bye and leave *The AudioSquare*. This is the end of our short story. What happened to Bob and Sally is kept secret.

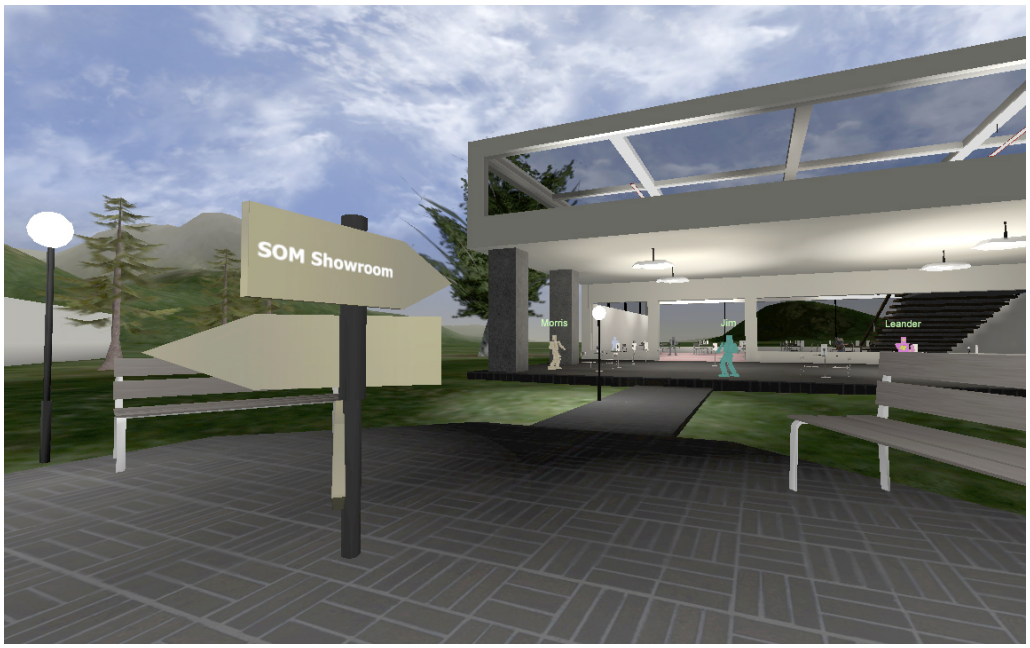

Figure 6.4: The entrance-area of the virtual world is located in the middle of the whole scenario.

Figure 6.5: The information-screen informs users about the purpose of *The AudioSquare* and how to navigate and interact within the virtual world.

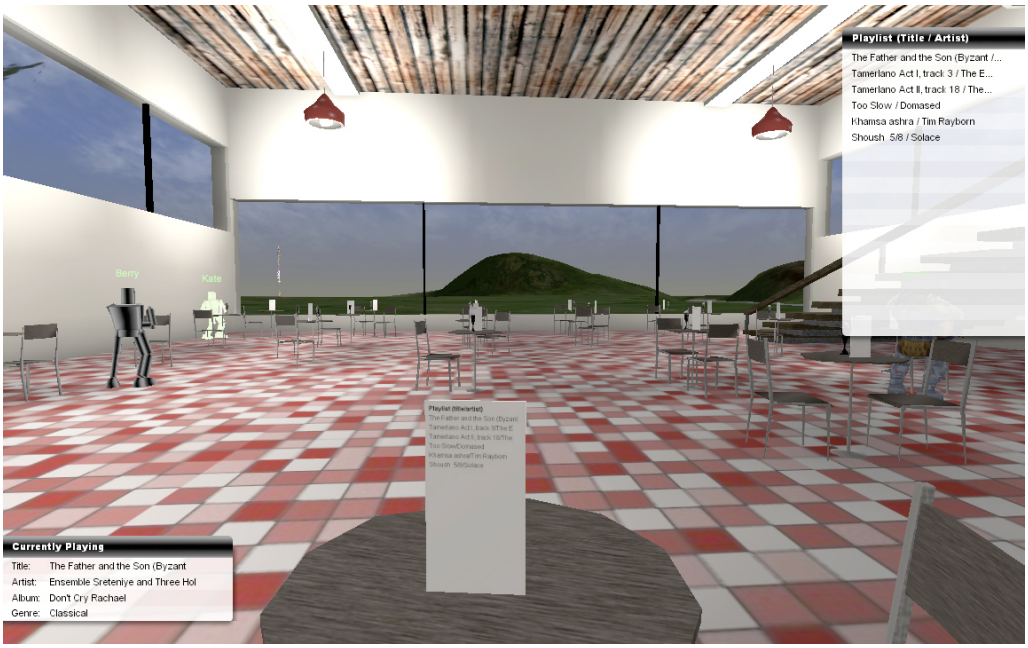

Figure 6.6: Inside the "SOM showroom". Each table represents a unit of a SOM and plays an audio stream.

Figure 6.7: Another area in the same building. The appearance of the interior is different here.

Figure 6.8: The small chat-HUD on the top of the screen is used for conversations with other users.

Figure 6.9: In front of the showrooms for directory-based organization.

Figure 6.10: This showroom is dedicated to classical music.

Chapter 7

Conclusions and Future Work

This chapter summarizes the work presented in this thesis. Moreover, some suggestions are made for future developments concerning media representation within three-dimensional virtual environments.

A prototype, referred to as *The AudioSquare*, was developed in order to propose a new approach for representing music archives within a virtual environment. The *Torque Game Engine* was used to set up a system architecture that allows multiple users to connect to a dedicated server over the Internet. The users, impersonated as avatars, can walk through a collaborative virtual world and explore music represented by 3D objects playing spatial sound. A simple chat-system encourages people to exchange their experiences. A media server was set up to take over the broadcasting of the audio streams, whereas the *Torque* clients were modified in order to receive the streams over the Internet.

The creation of the virtual world was done in three main steps. First, the basic environment was designed, containing the terrain, buildings and environmental objects. In the second phase, areas were defined within the environment which specify where the music is to be represented. Finally, an automatic process, performed by a Java-based application, mapped all objects necessary for the representation into the virtual space. In order to meet different demands, two methods for organization and representation were developed. On the one hand, the music can be organized by means of a *self-organizing map*. The resulting representation in the virtual world are objects arranged according to a matrix. The distance between the objects to one another indicates the sound-similarity of the respective music. The other organization-method relies on manually setting up a folder-structure on the local file system. The hierarchy is preserved as an architectural setting within the virtual world, where buildings represent collections and objects inside represent the music.

The AudioSquare is an attempt of showing the relevance of virtual worlds for media representation. On the one side, they are multimedia "by nature",

meaning they usually come with the ability of representing images, sound or even videos. On the other side, the virtual space can be used for showing relations between the entities of a data set by transformation of similarities into spatial relations. Moreover, an increasing number of people are getting used to 3D environments. Unfortunately, online worlds such as *Second Life* are simply not advanced enough to fulfill the requirements for creating a complex system capable of visualizing media content in a dynamic way. For this reason, *The AudioSquare* was developed on the basis of a highly customizable technical framework, the *Torque Game Engine*. However, a transformation of the present concept into the realms of a popular virtual world would be interesting for future work.

A further approach for refining the principal concept is to make the system more dynamic. Currently, the visual representation is created on startup of the server. Enabling the system to add and remove content on runtime would create a persistent and more vibrant virtual world. Another improvement could be achieved by adding semantic layers to the objects that represent the music in order to enhance the user's experience. For example, artworks of albums or pictures of artists could be retrieved from a Web service such as Amazon's online-database and used as images within the virtual world. Additional objects could change their appearance according to variable parameters. For instance, the size of a 3D object could indicate how many people already listened to a specific audio stream.

The strategies used for organizing and representing music in *The AudioSquare* can also be applied to other types of media. This has already been done with the development of an extended version, called *The MediaSquare*[1]. Created in cooperation with the EC3 in Vienna, it is able to present music, text and images within one virtual world by applying the same organization methods as used in its predecessor.

[1] http://mediasquare.ec3.at (date of access: 20.03.2008)

Chapter 8

Acknowledgements

I would like to thank all the people who supported me during the long time of work on this thesis. In particular, I would like to express my deepest gratefulness to my advisors Helmut Berger and Andreas Rauber. It were the lots of conversations with Helmut in combination with his patience that helped me to develop my ideas and come through the hard times of my thesis. Andreas was the one who made it possible to find an interesting and challenging topic which inspired me through the whole time of writing. I would also like to thank the members of the EC3 in Vienna, where I could work for a long time in a friendly atmosphere. Thanks to Florian Boschitsch for proof-reading and for not complaining too much about my English. Further thanks to Bernhard Faiss for his professional suggestions concerning the poster.

I would like to express my deepest esteem and biggest thanks to my dearly-loved Agnes. During the two and a half years we know each other, she had to stand the vexations of two diplomas. What I definitely can tell is that I will never do another thesis and, even more essential, that she is the most important person in my life. I would like to sincerely thank my mother and my father for always being loving parents. In memories of the early times of my study, I would like to thank Rainer Müller for being the closest fellow student I ever had.

Bibliography

[1] M. O. Afolabi and O. Olude. Predicting stock prices using a hybrid Kohonen self organizing map (SOM). In *HICSS-40: Proceedings of the 40th International Conference on System Sciences*, page 48, Washington, DC, USA, January 2007. IEEE Computer Society. [cited at p. 27]

[2] D. Bainbridge, C. Nevill-Manning, I. Witten, Smith L., and R. McNab. Towards a digital library of popular music. In *DL '99: Proceedings of the fourth ACM conference on Digital libraries*, pages 161–169, New York, NY, USA, 1999. ACM Press. [cited at p. 9]

[3] J. Barlow, S. Birkerts, M. Slouka, and S. Kelly. What are we doing on-line? *Harper's Magazine*, pages 35–44, August 1995. [cited at p. 14]

[4] M. Beck. Realisierung eines Geoinformationssystems - Visualisierung und Analysefunktionalität mit einer 3D Engine. Master's thesis, University Stuttgart, Stuttgart, Germany, June 2002. [cited at p. 13]

[5] D. Bergert. Twinity: Second Life-Konkurrent "Made in Berlin", `www.pcwelt.de/start/gaming_fun/sonstiges/news/86634/` (accessed: January 17, 2008). *PC Welt*, August 2007. [cited at p. 18]

[6] A. Blazejewski and R. Coggins. Application of self-organizing maps to clustering of high-frequency financial data. In *ACSW Frontiers '04: Proceedings of the second workshop on Australasian Information security, Data Mining and Web Software Internationalization*, pages 85–90, Darlinghurst, Australia, 2004. Australian Computer Society, Inc. [cited at p. 27]

[7] E. Castronova. Virtual worlds: A first-hand account of market and society on the cyberian frontier. *CESifo Working Paper Series No. 618.*, December 2001. [cited at p. 15]

[8] E. Castronova. *Synthetic Worlds: The business and culture of online games.* University of Chicago Press, Chicago, Illinois, USA, 2005. [cited at p. 1]

[9] D.L. Chao. Doom as an interface for process management. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 152–157, New York, NY, USA, 2001. ACM Press. [cited at p. 13]

[10] D.L. Chao. Computer games as interfaces. *Interactions*, 11(5):71–72, September 2004. [cited at p. 13]

[11] R. Cogan. *New Images of Musical Sound*. Harvard University Press, Cambridge, Massachusetts, USA, December 1984. [cited at p. 9]

[12] B. Damer. *Avatars: Exploring and Building Virtual Worlds on the Internet*. Peachpit Press, Berkeley, California, USA, October 1997. [cited at p. 15]

[13] S. DiPaola and D. Collins. A social metaphor-based 3D virtual environment. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Educators Program*, pages 1–2, New York, NY, USA, July 2003. ACM Press. [cited at p. 16]

[14] M.J. Dovey. A technique for "regular expression" style searching in polyphonic music. In *ISMIR 2001: Proceedings of the 2nd Annual International Symposium on Music Information Retrieval 2001*, pages 179–185, Bloomington, Indiana, USA, October 2001. Indiana University. [cited at p. 8]

[15] D. Fritsch and M. Kada. Visualisation using game engines. In *Proceedings of the XX ISPRS Congress, Commission V*, pages 621–625, Istanbul, Turkey, July 2004. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences. [cited at p. 13]

[16] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming: musical information retrieval in an audio database. In *Proceedings of the third ACM international conference on Multimedia*, pages 231–236, New York, NY, USA, 1995. ACM Press. [cited at p. 9]

[17] W. Gibson. *Neuromancer*. Heyne, Munich, Germany, 1984. [cited at p. 15]

[18] S. Greenberg and M. Roseman. Using a room metaphor to ease transitions in groupware. In *Beyond Knowledge Management: Sharing Expertise*, pages 203–256, Cambridge, Massachusetts, USA, 2002. MIT Press. [cited at p. 1]

[19] H. Harb and L. Chen. A query by example music retrieval algorithm. In E. Izquierdo, editor, *Digital Media Processing for Multimedia Interactive Services*, pages 122–128. Queen Mary, University of London, UK, 2003. [cited at p. 9]

[20] E. Isaacson. What you see is what you get: On visualizing music. In *ISMIR 2005: Proceedings of the 6th International Conference on Music Information Retrieval*, pages 389–395, London, UK, September 2005. University of London. [cited at p. 8]

[21] J. Jacobson. Common office equipment and caveut make a cheap portable cave. In *VR '03: Proceedings of the IEEE Virtual Reality 2003*, page 311, Washington, DC, USA, March 2003. IEEE Computer Society. [cited at p. 13]

[22] G.A. Kaminka, M.M. Veloso, S. Schaffer, C. Sollito, R. Adobbati, A.N. Marshall, A. Scholer, and S. Tejada. GameBots: A flexible test bed for multiagent team research. *Communications of the ACM*, 45(1):43–45, 2002. [cited at p. 14]

[23] F. Kaplan, A. McIntyre, C. Numaoka, and S. Tajan. Growing virtual communities in 3d meeting spaces. *Lecture Notes In Computer Science*, 1434:286–297, 1998. [cited at p. 16]

[24] S. L. Kent. *The ultimate History of Video Games*. Three Rivers Press, New York, NY, USA, 2001. [cited at p. 11]

[25] P. Knees, M. Schedl, T. Pohle, and G. Widmer. An innovative three-dimensional user interface for exploring music collections enriched. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 17–24, New York, NY, USA, October 2006. ACM Press. [cited at p. 10]

[26] I. Knopke. Geospatial location of music and sound files for music information retrieval. In *ISMIR 2005: Proceedings of the 6th International Conference on Music Information Retrieval*, pages 29–33, London, UK, September 2005. University of London. [cited at p. 7]

[27] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. [cited at p. 25]

[28] T. Kohonen. *Self-organizing maps*. Springer Verlag, Berlin, Germany, 1995. [cited at p. 2, 25]

[29] B. Kot, B. Wuensche, J. Grundy, and J. Hosking. Information visualisation utilising 3D computer game engines - case study: A source code comprehension tool. In *CHINZ '05: Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction*, pages 53–60, New York, NY, USA, July 2005. ACM Press. [cited at p. 13]

[30] J. E. Laird. Research in human-level AI using computer games. *Communications of the ACM*, 45(1):32–35, 2002. [cited at p. 14]

[31] J. Langer. GDC: PlayStation 3 - mit Sony Home gegen Xbox Live, `http://www.golem.de/0703/50934.html` (accessed: January 16, 2008). *golem.de*, March 2007. [cited at p. 17]

[32] M. Lewis and J. Jacobson. Game engines in scientific research. *Communications of the ACM*, 46(1):27–31, 2002. [cited at p. 12]

[33] B. Logan and A. Salomon. A music similarity function based on signal analysis. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 745–748, Washington, DC, USA, July 2001. IEEE Computer Society. [cited at p. 9]

[34] D. Luebbers. Sonixplorer: Combining visualization and auralization for content-based exploration of music collections. In *ISMIR 2005: Proceedings of the 6th International Conference on Music Information Retrieval*, pages 590–593, London, UK, September 2005. University of London. [cited at p. 10]

[35] K. Maney. The king of alter egos is surprisingly humble guy, `http://www.usatoday.com/printedition/money/20070205/secondlife_cover.art.htm` (accessed: January 11, 2008). *USA Today*, page 1B, February 5, 2007. [cited at p. 17]

[36] M. K. Markey, J. Y. Lo, G. D. Tourassi, and C. E. Floyd Jr. Self-organizing map for cluster analysis of a breast cancer database. In *Artificial Intelligence in Medicine*, pages 113–127, Oxford, UK, 2003. Elsevier Science Ltd. [cited at p. 27]

[37] R. J. McNab, L. A. Smith, I. H. Witten, C. L. Henderson, and S. J. Cunningham. Towards the digital music library: tune retrieval from acoustic input. In *Proceedings of the first ACM International Conference on Digital Libraries (DL '96)*, pages 11–18, New York, NY, USA, 1996. ACM Press. [cited at p. 9]

[38] J. Moloney, R. Amor, J. Roberts, J. Furness, and B. Moores. Design critique inside a multi-player game engine. In *Proceedings of the CIB W78's 20th International Conference on Construction IT, Construction IT Bridging the Distance*, pages 255–263, Waiheke Island, New Zealand, 2003. CIB Publication. [cited at p. 13]

[39] R. Neumayer, M. Dittenbach, and A. Rauber. PlaySOM and PocketSOMPlayer, alternative interfaces to large music collections. In *ISMIR 2005: 6th International Conference on Music Information Retrieval*, pages 618–623, London, UK, September 2005. University of London. [cited at p. 2, 10, 21]

[40] E. Pampalk. Islands of music: Analysis, organization, and visualization of music archives. Master's thesis, Vienna University of Technology, Vienna, Austria, December 2001. [cited at p. 2]

[41] E. Pampalk, A. Rauber, and D. Merkl. Content-based organization and visualization of music archives. In *Proceedings of the 10th ACM international conference on Multimedia*, pages 570–579, New York, NY, USA, December 2002. ACM Press. [cited at p. 9, 21]

[42] C. Parker. Applications of binary classification and adaptive boosting to the query-by-humming problem. In *ISMIR 2005: Proceedings of the 6th International Conference on Music Information Retrieval*, pages 245–251, London, UK, September 2005. University of London. [cited at p. 9]

[43] A. Quan-Haase, B. Wellman, J. Witte, and K. Hampton. Capitalizing on the Internet: Social contact, civic engagement, and sense of community. In B. Wellman and C. Haythornthwaite, editors, *The Internet and Everyday Life*, pages 291–324, Oxford, UK, 2002. Blackwell. [cited at p. 14]

[44] A. Rauber and M. Frühwirth. Automatically analyzing and organizing music archives. In *ECDL '01: Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries*, Springer Lecture Notes in Computer Science, pages 402–414, Darmstadt, Germany, September 2001. Springer. [cited at p. 21]

[45] A. Rauber, E. Pampalk, and D. Merkl. Using psycho-acoustic models and self-organizing maps to create a hierarchical structuring of music by musical styles. In *ISMIR 2002: Proceedings of the 3rd International Symposium on Music Information Retrieval*, pages 71–80, Paris, France, October 2002. Ircam, Centre Pompidou. [cited at p. 21]

[46] R. Rojas. *Neural Networks: A Systematic Introduction.* Springer Verlag, Berlin, Germany, 1996. [cited at p. 26]

[47] T. Scheiblauer. Anwendung von Game Engines für kollaborative virtuelle Umgebungen in der Architektur. Master's thesis, Vienna University of Technology, Vienna, Austria, 2004. [cited at p. 13]

[48] M. Seif El-Nasr and B. K. Smith. Learning through game modding. *Computers in Entertainment (CIE)*, 4(1):7, January 2006. [cited at p. 13]

[49] B. Stang. Game engines - features and possibilities. Project Report, Technical University of Denmark, Lyngby, Denmark, 2003. [cited at p. 13]

[50] I. Stavness, J. Gluck, L. Vilhan, and S. Fels. The MUSICtable: A map-based ubiquitous system for social interaction with a digital music collection. In F. Kishino, Y. Kitamura, H. Kato, and N. Nagata, editors, *ICEC 2005: Proceedings of the 4th International Conference on Entertainment Computing*, pages 291–302, Berlin, Germany, September 2005. Springer Verlag. [cited at p. 7]

[51] N. Stephenson. *Snow Crash*. Bantam Books, New York, NY, USA, 1995. [cited at p. 15]

[52] I. Suyoto, A. Uitdenbogerd, and F. Scholer. Effective retrieval of polyphonic audio with polyphonic symbolic queries. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 105–114, New York, NY, USA, September 2007. ACM Press. [cited at p. 9]

[53] C. Svensson. Roar of the indy, `http://www.businessweek.com/innovate/content/nov2005/id20051114_582047.htm` (accessed: January 19, 2008). *Business Week Online*, November 2005. [cited at p. 29]

[54] B. Thomas, B. Close, J. Donoghue, J. Squires, P. De Bondi, and W. Piekarski. First person indoor/outdoor augmented reality application: Arquake. *Personal and Ubiquitous Computing*, 6(1):75–86, February 2002. [cited at p. 13]

[55] I. Tjostheim and J. Lous. Attracting visitors - using computer games technology to build a VR-museum. In M. Hitz, M. Sigala, and J. Murphy, editors, *Information and Communication Technologies in Tourism 2006*, pages 44–54, Berlin, Germany, 2006. Springer Verlag. [cited at p. 13]

[56] M. Torrens, P. Hertzog, and J. Arcoss. Visualizing and exploring personal music libraries. In *ISMIR 2004: Proceedings of the 5th International Conference on Music Information Retrieval*, pages 421–424, Barcelona, Spain, October 2004. Universitat Pompeu Fabra. [cited at p. 7]

[57] W. Tsai. A query-by-example technique for retrieving cover versions of popular songs with similar melodies. In *ISMIR 2005: Proceedings of the 6th International Conference on Music Information Retrieval*, pages 183–190, London, UK, September 2005. University of London. [cited at p. 9]

[58] G. Tzanetakis and P. Cook. MARSYAS3D: A prototype audio browser-editor using a large scale immersive visual and audio display. In *Proceedings of the 7th International Conference on Auditory Display*, pages 250–254, Espoo, Finland, July 2001. Helsinki University of Technology. [cited at p. 11]

[59] M. Wallace. The game is virtual. The profit is real. `http://www.nytimes.com/2005/05/29/business/yourmoney/29game.html` (accessed: January 15, 2008). *New York Times*, May 29 2005. [cited at p. 15]

[60] K. Weber. The "new" new economy in second life? `http://www.heise.de/tp/r4/artikel/24/24584/1.html` (accessed: February 17, 2008). *Telepolis*, February 6, 2007. [cited at p. 17]

[61] K. Wehn. Machinima - Was Ego-Shooter und Puppentheater gemeinsam haben, `http://www.heise.de/tp/r4/artikel/17/17818/1.html` (accessed: February 22, 2008). *Telepolis*, July 13, 2004. [cited at p. 12]

[62] B. Wellman. *Networks in the Global Village: Life in Contemporary Communities.* Westview Press, Boulder, Collorado, USA, July 1999. [cited at p. 14]

[63] B. Wellman, J. Boase, and W. Chen. The networked nature of community online and offline. *IT & Society*, 1(1):151–165, 2002. [cited at p. 14]

[64] K. West and P. Lamere. A model-based approach to constructing music similarity functions. *EURASIP J. Applied Signal Processing*, 2007(1):149, 2007. [cited at p. 9]

[65] N. Yee. The psychology of massively multi-user online role-playing games: Motivations, emotional investment, relationships and problematic usage. In R. Schroeder and A. Axelsson, editors, *Avatars at Work and Play: Collaboration andd Interaction in Shared Virtual Environments*, pages 187–207, Secaucus, New Jersey, USA, 2006. Springer Verlag New York. [cited at p. 1, 15]

[66] R. M. Young. An overview of the mimesis architecture: Integrating intelligent narrative control into an existing gaming environment. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Menlo Park, California, USA, March 2001. AAAI Press. [cited at p. 14]

Appendix

Appendix A

Game Engine Overview

Table A shows a list of low-budget 3D game engines that were considered for developing *The AudioSquare*.

Table A.1: Game Engines for less than $300.

	3Impact	*3D Game Studio*	*C4*	*Crystal Space*	*Dark Basic Pro*	*Irrlicht*	*Ogre 3D*	*Torque*
OS	Windows	Windows	Windows, MacOS	Windows, Xbox, PS 3	Windows	Windows, Linux, MacOS	Windows, Linux, MacOS	Windows, Linux, MacOS
Current release	3.8	6.5	134	1.0.1	-	1.3	1.4.0	1.5
Programming language	C/C++, Delphi, Basic	C/C++, Delphi	C/C++	C/C++	Basic	C++, C#, VB.NET	C/C++	C/C++
Source code available	No	No	Yes	Yes	No	Yes	Yes	Yes
Graphics API	OpenGL, DirectX	DirectX	OpenGL	OpenGL, Software	DirectX	OpenGL, DirectX	OpenGL, DirectX	OpenGL, DirectX
Shaders	-	Shader model 3.0	Vertex, pixel	CG, ARB	Vertex, pixel	Vertex, pixel, GLSL	Vertex, pixel, HLSL, GLSL	-
Bump maps	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Light maps	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Environment maps	Yes	Yes	Yes	-	Yes	Yes	Yes	Yes
Normal maps	-	-	Yes	-	-	Yes	-	-
Scene management	BSP	BSP, LOD	BSP	Portals, kd-trees	BSP	BSP	BSP, LOD, Octrees	BSP, LOD
Dynamic shadows	Vol.	Stencil	Various techniques	Stencil	Vol.	Stencil	Stencil, texture-based	Mapping, vol., projected planar
Terrain	Yes	Yes	Yes	Yes	Yes	-	Yes	Yes
GUI module	-	Yes	Yes	Yes	-	-	Yes	Yes
3D file formats	.x	.3ds, .x, .map, .mdl, .md2	.3ds, .obj, .xsi, .blend	.3ds, .mdl, .md2, .obj, .pov, .ase	.x	.3ds, .obj, .dae, .dmf, .ms3d	.3ds, .blend	.3ds, .max, .lwo, .blend, .dts, .scn, .ma, .ms3d
Keyframe	Yes	Yes	-	-	-	-	-	Yes
Skeletal	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Blending	Yes	Yes	Yes	-	-	-	Yes	Yes
Sound								
3D sound	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes
File formats	.wav, .ogg	.wav, .ogg, .mp3, .mid	QuickTime compatible formats	.wav, .ogg, .au, .aiff, .iff, .mod	.wav, .wma, .aiff, .au, .snd, .mp3, .midi	.wav, .ogg, .mp3, .mod, .xm, .it, .s3m	-	.wav, .ogg
Networking	-	Client-server	Yes	-	Yes	-	-	Client-server
Physics	Yes	Yes	Yes	Yes	-	-	-	Yes
Scripting	-	C-Script	Yes	Python	-	LUA	-	TorqueScript
Level editor	-	Yes	Yes	-	-	Yes	-	Yes
GUI editor	-	-	Yes	Yes	-	-	-	Yes
Model editor	-	-	-	Yes	-	-	-	-
Price	$99	Free	$200	$49 - $899	$100	Free	Free	$150

Appendix B

Example of an Objects-File

The subsequent XML-code has been extracted from an *objects-file* as used in *The AudioSquare* for the import of media-related 3D objects. The example contains the description of one unit of a SOM as well as of one building containing one audio stream.

```
<?xml version="1.0" encoding="UTF-8"?>
<objects>
  <unit unitfile="ismir_genre" x="5" y="0">
    <unitfile>ismir_genre</unitfile>
    <x>5</x>
    <y>0</y>
    <mediatype>audio</mediatype>
    <songinfo>
      <streamnr>5</streamnr>
      <song>
        <title>Yedi tekrar</title>
        <artist>Tim Rayborn</artist>
        <album>Veils of Light</album>
        <genre>Ethnic</genre>
      </song>
      <song>
        <title>Tas</title>
        <artist>Tim Rayborn</artist>
        <album>The Path Beyond</album>
        <genre>Ethnic</genre>
      </song>
      <song>
        <title>Laz  7/8</title>
        <artist>Solace</artist>
        <album>Rhythm Of The Dance</album>
        <genre>Unknown</genre>
      </song>
```

```
  <song>
    <title>Raghba</title>
    <artist>Solace</artist>
    <album>Iman</album>
    <genre>Unknown</genre>
</song>
</songinfo>
<static>
  <shapename> /data/interiors/mediasquare/table11.dts</shapename>
  <name>Table_5_0</name>
  <position>37.93565871993286 -264.3509500400416 50.5</position>
  <rotation>0 0 1 106.07939516125099</rotation>
  <scale>1.0 1.0 1.0</scale>
</static>
<static>
  <shapename> /data/interiors/mediasquare/stool06.dts</shapename>
  <name>Stool01_5_0</name>
  <position>37.6873459227665 -265.2160591525028 50.525</position>
  <rotation>0 0 1 106.07939516125099</rotation>
  <scale>1.0 1.0 1.0</scale>
</static>
<static>
  <shapename> /data/interiors/mediasquare/stool06.dts</shapename>
  <name>Stool02_5_0</name>
  <position>38.18589033478738 -263.486477275504 50.525</position>
  <rotation>0 0 1 286.079395161251</rotation>
  <scale>1.0 1.0 1.0</scale>
</static>
<audioemitter>
  <sound>http://admin.ec3.at:7800/stream5</sound>
  <streamnr>5</streamnr>
  <refdistance>1.0</refdistance>
  <maxdistance>10.0</maxdistance>
  <infodistance>1.9</infodistance>
  <matrixpos>5 / 0</matrixpos>
  <unitfile>ismir_genre</unitfile>
  <name>AudioEmitter01_5_0</name>
  <position>37.96120482570963 -264.4103909197427 51.475</position>
  <rotation>0 0 1 106.07939516125099</rotation>
  <scale>1.0 1.0 1.0</scale>
</audioemitter>
<static>
  <shapename> /data/interiors/speaker_usb.dts</shapename>
  <name>Speaker_5_0</name>
  <position>37.74307690691523 -264.30061152303085 51.945</position>
  <rotation>0 0 1 106.07939516125099</rotation>
  <scale>1.0 1.0 1.0</scale>
```

```
  </static>
  <objecttrigger>
    <organization>unit</organization>
    <type>audio</type>
    <matrixpos>5/0</matrixpos>
    <streamnr>5</streamnr>
    <name>ObjectTrigger01_5_0</name>
    <position>39.59485461056448 -263.197116742632 50.6</position>
    <rotation>0 0 1 106.07939516125099</rotation>
    <scale>3.0 3.0 1.0</scale>
  </objecttrigger>
  <playlist>
    <skin>pics/playlist5</skin>
    <shapename> /data/shapes/playlist/playlist.dts</shapename>
    <name>TSPlaylist_5_0</name>
    <position>38.04655446672187 -264.3309211847546 51.925</position>
    <rotation>0 0 1 186.079395161251</rotation>
    <scale>1.0 1.0 1.0</scale>
  </playlist>
</unit>
<dirmapping directory="../../SDK 1.5/example/torquesom/data/dirmapping/audio01/01/">
  <directory>../../SDK 1.5/example/torquesom/data/dirmapping/audio01/01/</directory>
  <songinfo>
    <streamnr>251</streamnr>
    <song>
      <title>Gigue</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
    </song>
    <song>
      <title>Aria de Fiorenza</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
    </song>
    <song>
      <title>Fandango</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
    </song>
    <song>
      <title>Rujero</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
```

```
    </song>
    <song>
      <title>Pavanas</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
    </song>
    <song>
      <title>Marionas</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
    </song>
    <song>
      <title>Caprice de chaconne</title>
      <artist>James Edwards</artist>
      <album>Canarios</album>
      <genre>Classical</genre>
    </song>
  </songinfo>
  <interior>
    <interiorfile>~/data/buildings/projection_building.dif</interiorfile>
    <name>Building</name>
    <position>-24.218820261851334 -226.90305250468305 50.32425</position>
    <rotation>0 0 1 -45.2637</rotation>
    <scale>1.0 1.0 1.0</scale>
  </interior>
  <label>
    <skin>../../dirmapping/audio01/01/title</skin>
    <shapename>~/data/shapes/projection/projection_label.dts</shapename>
    <name>TSProjectionLabel00</name>
    <position>-4.4065931962842475 -200.8050742952335 52.15505</position>
    <rotation>0 0 1 135.0003</rotation>
    <scale>1.0 1.0 1.0</scale>
  </projectionlabel>
  <static>
    <shapename>~/data/interiors/window10x10.dts</shapename>
    <name>window</name>
    <position>-45.62821140279159 -248.35698393664023 51.32505</position>
    <rotation>0 0 1 -45.2637</rotation>
    <scale>1.0 1.1 0.3</scale>
  </static>
  <label>
    <skin>../../dirmapping/audio01/01/01/title</skin>
    <shapename>~/data/shapes/projection/projection_label.dts</shapename>
    <name>TSProjectionLabel01</name>
    <position>-8.247764074169838 -203.6660596336936 52.15505</position>
```

```
  <rotation>0 0 1 45.836600000000004</rotation>
  <scale>1.0 1.0 1.0</scale>
</label>
<objecttrigger>
  <directorypos>1</directorypos>
  <organization>directory</organization>
  <nr>1</nr>
  <streamnr>251</streamnr>
  <name>TSObjectTrigger01</name>
  <position>-9.446593482194398 -205.5676616940773 50.22425</position>
  <rotation>0 0 1 -45.2637</rotation>
  <scale>5.0 9.0 2.0</scale>
</objecttrigger>
<static>
  <shapename>~/data/interiors/table06.dts</shapename>
  <name>Table01</name>
  <position>-1.5278482049767406 -209.95838767556864 50.25505</position>
  <rotation>0 0 1 -45.2637</rotation>
  <scale>1.0 1.0 1.0</scale>
</static>
<static>
  <shapename>~/data/interiors/stereo01.dts</shapename>
  <name>Hifi01</name>
  <position>-1.5508634609465628 -210.012446421324 50.90505</position>
  <rotation>0 0 1 44.7363</rotation>
  <scale>0.75 0.75 0.75</scale>
</static>
<static>
  <shapename>~/data/interiors/speaker01.dts</shapename>
  <name>SpeakerRight01</name>
  <position>-2.17611902250455 -210.56008497250608 50.90505</position>
  <rotation>0 0 1 -45.2637</rotation>
  <scale>0.5 0.5 0.5</scale>
</static>
<static>
  <shapename>~/data/interiors/speaker01.dts</shapename>
  <name>SpeakerLeft01</name>
  <position>-1.029485309223455 -209.4028477722886 50.90505</position>
  <rotation>0 0 1 -45.2637</rotation>
  <scale>0.5 0.5 0.5</scale>
</static>
<audioemitter>
  <sound>http://admin.ec3.at:7800/stream251</sound>
  <streamnr>251</streamnr>
  <refdistance>6.0</refdistance>
  <maxdistance>10.0</maxdistance>
  <infodistance>1.0</infodistance>
```

```
      <name>AudioStreamEmitter01</name>
      <position>-1.2789893005001423 -209.6719924143252 51.32425</position>
      <rotation>0 0 1 -45.2637</rotation>
      <scale>1.0 1.0 1.0</scale>
    </audioemitter>
  </dirmapping>
</objects>
```

Appendix C

Example of Marker-Objects in the Mission-File

The following code samples describe different *marker-objects* as used in the *mission-file*. The first example with the name "SomMapping01" is an object with the *Datablock* "SomMapping". This means, that is defines an area wherein a part of a SOM-generated matrix is represented. The properties "xfrom", "xto", "yfrom" and "yto" define which section of the matrix is used within the area. The property "mappingfile" defines the name of the *unit-file* used for the representation. The second example with the name "DirectoryMapping01" has the *Datablock* "DirectoryMapping" which means that is is used as a starting point for representing directory-based organization. The property "Directory" identifies the name of the base-folder where the directory structure is located. The circular layout algorithm is selected used since the property "circular" is set to one. "Radius" and "spacing" define the size of the underlying circle and the distance between the buildings aligned along the circle.

```
new StaticShape(SomMapping01) {
  canSaveDynamicFields = "1";
  position = "101.482 -221.926 3.43";
  rotation = "1 0 0 0";
  scale = "32 21 2";
  dataBlock = "SomMapping";
  receiveSunLight = "1";
  receiveLMLighting = "1";
  useAdaptiveSelfIllumination = "0";
  useCustomAmbientLighting = "0";
  customAmbientSelfIllumination = "0";
  customAmbientLighting = "0 0 0 1";
  useLightingOcclusion = "1";
  templatename = "template01";
  xfrom = "0";
```

```
  xto = "4";
  yfrom = "0";
  yto = "3";
  autorotate = "1";
  mappingfile = "ismir_genre";
};

new StaticShape(DirectoryMapping01) {
  canSaveDynamicFields = "1";
  position = "72.7833 -278.761 3.3";
  rotation = "0 0 -1 93.4384";
  scale = "1 1 1";
  dataBlock = "DirectoryMapping";
  receiveSunLight = "1";
  receiveLMLighting = "1";
  useAdaptiveSelfIllumination = "0";
  useCustomAmbientLighting = "0";
  customAmbientSelfIllumination = "0";
  customAmbientLighting = "0 0 0 1";
  useLightingOcclusion = "1";
  Directory = "audio01";
  templatename = "dir_audio_template01";
  Radius = "28";
  spacing = "13";
  circular = "1";
};
```

List of Figures

2.1 A music map as shown on the *Dimvision* Web site. The nodes represent artists and bands. The edges identify the relations of the nodes to one another. 7
2.2 Three visual representations as proposed by Torrens: The disc view, divided (left), the rectangular view (middle) and the tree view (right). 8
2.3 Excerpt from Bach's score for his piece *Schweigt stille*. 8
2.4 Spectral analysis of a guitar playing a melody. 10
2.5 Utilizing content-based music organization - the *PocketSOMPlayer* on a tablet PC (left) and the graphical user interface (right). 10
2.6 *PSDoom* is a process management tool realized with *Doom 1*. 14
2.7 Masses of avatars in *World of Warcraft*. 15
2.8 Impressions from the virtual online world *Second Life* 18
2.9 Home for the *PlayStation 3*. 19
2.10 *vSide*, a virtual online community for teenagers. 19

3.1 Organization and representation of music libraries with a *self-organizing map*. 21
3.2 Preprocessing a music file in the library, which results in several 6 second sequences. The first and the last sequence are dropped to avoid lead-in and fade-out effects. 22
3.3 Transformation of an audio signal to a representation regarding the specific loudness sensation in Sone. 23
3.4 Transformation of the power spectrum in Sone to the time-invariant *Rhythm Pattern* 24
3.5 Two results of the feature extraction process (left: *Für Elise* by Beethoven, right: *Freak on a Leash* by Korn) 25
3.6 A one-dimensional SOM maps a triangular region. 26
3.7 Architecture of a 7×7 *self-organizing map* 28
3.8 The *Torque SDK* comes with a first person shooter example. It can also be edited with the *World Editor* as shown here. 30

3.9 The *GUI Editor* is a standard feature of the *Torque SDK*. The image shows the start menu for the first person shooter example. 31

4.1 Automatic organization based on a SOM. 35
4.2 The manual organization of music is based on a folder structure. In the virtual environment, the top-level folders are represented as buildings and the low-level folders containing the audio files are represented as interior. 36
4.3 Layout algorithms for manual music organization. 36

5.1 System architecture. 38
5.2 3D objects representing metadata. The left picture shows a playlist-object, the right picture shows a signboard-object. 41
5.3 *Marker-objects* indicate locations for music representation. The objects are signified by the yellow outlines. The left picture defines an area for SOM-based representation. The right picture defines a starting-point for directory-based mapping. 42
5.4 Assets for the repository. The rightmost picture depicts an asset for directory-based mapping, the others address SOM-based representation. 44
5.5 The Java-classes of the *Wrapper* support the representation and processing of the *mission-file*, one or more *unit-files*, one or more manually created directory structures, song-metadata, locations in the virtual environment and the *Icecast*-streams. 45
5.6 Icecast architecture 49
5.7 Setup for skipping tracks 50

6.1 Bird's-eye view on *The AudioSquare* 52
6.2 The start-screen shows a preview of the virtual world and a menu. . . 53
6.3 The login-window lets a user select an avatar and type in a name. . . 54
6.4 The entrance-area of the virtual world is located in the middle of the whole scenario. 55
6.5 The information-screen informs users about the purpose of *The AudioSquare* and how to navigate and interact within the virtual world. . 55
6.6 Inside the "SOM showroom". Each table represents a unit of a SOM and plays an audio stream. 56
6.7 Another area in the same building. The appearance of the interior is different here. 56
6.8 The small chat-HUD on the top of the screen is used for conversations with other users. 57
6.9 In front of the showrooms for directory-based organization. 57
6.10 This showroom is dedicated to classical music. 58

List of Tables

2.1 How much information? Comparing the estimates for the amount of generated digital media in the years 1999 and 2002 in the United States. 6
2.2 Listing of the most relevant *MMPORGs* and the number of active subscribers in the years 2004 and 2006 (According to *mmogchart.com*). The right columns on the right show the title of the game and the name of the developing company. The next two columns show the total number of active subscribers in the years 2004 and 2006. The rightmost column shows the difference between these two years in percent. 16

5.1 Keyboard and mouse settings for navigation 40
5.2 Parameters in the configuration-file for the *Wrapper*. 46

A.1 Game Engines for less than $300. 72