Stockholm, Dec 2002

# AAR
## An Audio Augmented Reality System

**NILS MONTAN, 711017 – 0193, MONTAN@KTH.SE**

Final project for the degree of Master of Science performed at the Fraunhofer IGD.

KTH, Royal Institute of Technology, Stockholm
Department of Microelectronics and Information Technology

# Abstract

As we hear sound in three dimensions the reproduction of spatial aspects of audio is essential to digitally create, recreate or enhance an environment. Only recently has computer processing power reached levels enabling performance of synthesis and reproduction of 3D soundscapes in real-time, also on inexpensive hardware. An implementation of a real-time spatial audio rendering system for augmentation of real life situations, and an exa mination of some of the possibilities such a system provides, is the goal of this thesis.

Providing intuitive access to an increasing amount of information in everyday environments is a great challenge. Augmented reality systems address this issue but have so far mainly focused on visual enhancements, which usually require rather immoderate means in terms of perceptual effort. There exist surprisingly few attempts of utilising audio user interfaces in real life environments. The major aim of this thesis has been to implement a low cost audio augmented reality prototype system and to implement and explore some applications. The result is the AAR system, providing a full framework to design and create spatial audio applications. Using the AAR system, three test applications have been implemented were a museum tour guide turned out especially well.

The three major components of the system are the listener, the emitters and their environment. In an application a listener moves around free in space where different locations, such as physical rooms or parts of rooms, provide boundaries between different acoustic landscapes. Audio objects, the emitters, are placed within these environments and can be made to interact with the listener based on his location.

The system is implemented using a client-server architecture and the rendering of the 3D audio and of the environmental acoustics is done through the DirectSound3D and the EAX API's. A head-tracking device makes it possible to use head related transfer functions. An interface to a graphical soundscape design tool is also included. In evaluating the system a brief comparison with a sophisticated audio rendering system, as well as the implemented test applications, showed a satisfying quality of the produced spatial sound.

## Sammanfattning

Att reproducera ljud på ett adekvat sätt kräver att man tar hänsyn till de tre dimensionella och akustiska aspekterna som den mänskliga hjärnan utnyttjar för att analysera ljud. Dagens tekniska utvecklingsnivå tillåter att man, i realtid, simulerar spatiala egenskaper hos ljud på ordinära persondatorer. Syftet med föreliggande arbete är implementering och utvärdering av ett realtidssystem för spatial ljudrendering.

Det implementerade systemet kallas AAR och möjliggör gestaltning och realtidsproduktion av interaktiva ljudmiljöer. Systemet bygger på att en användare kan röra sig fritt i ett rum där olika ljudobjekt är utplacerade. Användarens placering styr den akustiska ljudbilden och interaktionen med de olika objekten.

Systemet är tillämpat med en client-server-arkitektur i vilken en server renderar de tredimensionella och akustiska egenskaperna med hjälp av DirectSound3D- och EAX-biblioteken. En positionsgivare, som registrerar en användarens huvudposition, möjliggör att huvudrelaterade överföringsfunktioner används i ljudrenderingsprocessen. Ett gränssnitt till ett grafiskt verktyg för ljudmiljödesign är också inkluderat i systemet.

Tre testapplikationer är implementerade och särskilt en virtuell guide för museum kan visa på de stora möjligheter som ljudsystemet har. En utvärdering av systemet visar att den spatiala ljud kvaliteten håller hög standard.

## Acknowledgements

I would like to express my gratitude to the following people.

Gregor Heinrich, my advisor at Fraunhofer IGD for proposing the project and for the guidance.

Professor Gunnar Karlsson, my examiner at KTH, for motivation and advice.

Dave Goodwin and Tobias Stjernfeldt, without whom this thesis would have been impoverished.

Feh Reichl, for her excellent photos.

Hannes Guddat, Stefan Noll and the rest of Fraunhofer IGD for making my stay in Darmstadt possible.

Amelie and my family for all the support.

# Table of Contents

# List of Figures



# List of Tables

# Acronyms

AAR - Audio Augmented Reality

API - Application Programming Interface

AR - Augmented Reality

AUI - Audio User Interface

COM - Component Object Model

DLL - Dynamical Link Library

DSP - Digital Signal Processor

EAX - Environmental Audio Extensions

EAGLE - Environmental Audio Graphical Librarian Editor

.eal - Environmental Audio Library (data file)

GUI - Graphical User Interface

HRTF - Head-Related Transfer Function

I3DL - Interactive 3D Audio Rendering Guidelines

IASIG - Interactive Audio Special Interest Group

IGD - Institut Graphische Datenverarbeitung

ILD - Interaural Level Difference

IP - Internet Protocol

ITD - Interaural Time Delay

MFC - Microsoft Foundation Classes

MMA - Midi Manufacturer's Association

OpenAL - Open Audio Library

SAS - Spatial Sound Server

TCP - Transport Control Protocol

UDP - User Datagram Protocol

VR - Virtual Reality

.WAV - Waveform (data file)

# 1.0 Introduction

Providing intuitive access to an increasing amount of information in everyday environments is a great challenge. Augmented reality systems address this issue but have so far mainly focused on visual enhancements, which usually require rather immoderate means in terms of perceptual effort. Even though sound generally carries less information than light, there are a lot of situations where the obtrusiveness of visual cues could make audio an interesting option, either in the complimentarily between the visual and auditory senses, or as a stand alone interface.

Thanks to new advances in auditory rendering techniques and to the decreasing cost of computational power, spatial audio augmentation may become an approach accessible to implementation in inexpensive information systems. An implementation of such a system and an examination of some of the possibilities it provides is the goal of this thesis. The requirement on the system is a flexible implementation without demands of specific audio hardware.

## Why spatial audio as an interface

The term AR is in this thesis given a rather wide definition: a technical system providing information in natural situations. Most information systems today utilise interfaces using different concepts of visualisation. If electronic environment augmentation tools are going to become a more natural part of everyday life, I think the interface must be re-examined.

Exploring spatial audio as an interface to information could probably be fruitful as most of our frequently used electronic devices rapidly are shrinking in size and their traditional man-machine interfaces, keyboard and display, are unable to follow beyond a certain limit.

Yet a motivation for exploring the world of spatial audio interfaces is a concrete scenario, also implemented as a part of this thesis, using spatialised audio in a typical exhibition situation. This is illustrated in figure 1.


**Figure 1** *Illustration of a personalised 3D-audio exhibition guide.*

Given the means of personalised 3D audio and environmental acoustic treatment of sounds, I believe, information provided by museum audio guides could be revolutionised.

## Aim of the study

One aim of this study is to provide a comprehensive guide of spatial audio as well as to take a look at previous attempts in augmenting physical environments using audio. The main objective though, is to implement a prototype audio-only augmented reality system utilising different off-the-shelf products. The system should provide cues of information on the surrounding auditory scene such as sound source direction, its velocity and a sense of the general acoustic environment. Also more sophisticated hints of objects obstructing a sound path and occlusion should be included. The challenge lies in creating a low cost, easily manipulated and accurate spatial audio system. An evaluation of the system and implementation of some test applications also lies within the scope of this project.

## Scope of this study

The first part of this project is a study of different aspects of the theory behind spatial audio. It is followed by a review of existing audio augmented environment implementations.

The major part of the thesis concerns my implementation and evaluation of a spatial audio system for augmenting environments. I also describe a number of implemented applications that are used to further evaluate the system and to explore the possibilities of an audio-only augmented reality.

This study is organised in the following manner. Chapter 2 provides an overview of human spatial hearing and environmental acoustics and explains briefly how these can be modelled. Chapter 2 also describes some playback techniques of immersive sound and some API's for synthesising spatial audio. In Chapter 3 work related to audio augmented realities is reviewed. Chapter 4 describes the implementation of my system and its evaluation. In Chapter 5 describes and briefly evaluates a number of applications implemented in the developed system and Chapter 6 conclude this thesis.

# 2.0 Background

The common technique to synthesise and reproduce sound is to use stereo. A stereo sound captures differences in intensity and phase between points in a sound field. From these variances, the listener is able to imagine a position of the sound source. The experienced position of the sound source is, however, usually along a line between two speakers or, in the case of using headphones, along an axis through the middle of the head. This limitation of stereophonic reproduction is due to the fact the playback audio in stereo is a poor model of how real life sound waves arrive at the ears. In order to create more realistic soundscapes, 3D treatment of the sound is required. This is done using better models of the human auditory perception systems and allows a sound, together with acoustic environmental modelling, to emanate from any direction, carrying cues of distance, motion and ambience. Ultimately the sound waves that arrive at the eardrums during playback should be an approximation as close as possible to what would have actually arrived at a listener's eardrum in a real life situation. A simulation like this, with a complete acoustic audio scene, is sometimes called *spatialisation* and provides the full framework for creating realistic sound environments.

## Spatial audio and its modelling

To understand how to model, implement and use artificially spatialised audio the human perception of spatialisation has to be investigated. There has been substantial research in this area and eight particularly important cues [1] for giving a sound a direction have been identified. Interaural time delay, interaural level difference, pinna[1] response and shoulder echo, all of which are modelled in head related transfer functions[2], are considered particularly essential when it comes to the localisation of a sound. Further, there are the cues of head motion, vision, early reflections and reverberation.

To achieve realistic spatial audio, the objects emitting a sound must interact with the surrounding environment. This is referred to as simulating the acoustic environment and it involves a number of different acoustic behaviours. The previously mention reverberation and early reflections are the most obvious ones but also phenomena such as obstruction and occlusion play significant roles in forming the natural acoustics of an environment.

In the following the above mentioned spatial cues and acoustic phenomena and how they may be modelled will be described. In order to be able to portray some of these cues clearly, as well as for further discussions, a spherical coordinate system about the head needs to be established.

### Coordinate system

The centre of the coordinate system is defined as the point halfway between the ears, see Figure 2. The *azimuth* is defined as the deflection from front centre (0°) in the horizontal plane, with positive angles defined to the right. 90° is directly to the right and –90° is directly to the left. Positions directly behind the head maybe described as either 180° or –180°.

1 Outer ear.
2 The HRTF is determined differently depending on the criteria set for a particular application [2]. Some measurements incorporate only the outer ear and the head, others also make account of features of the body.

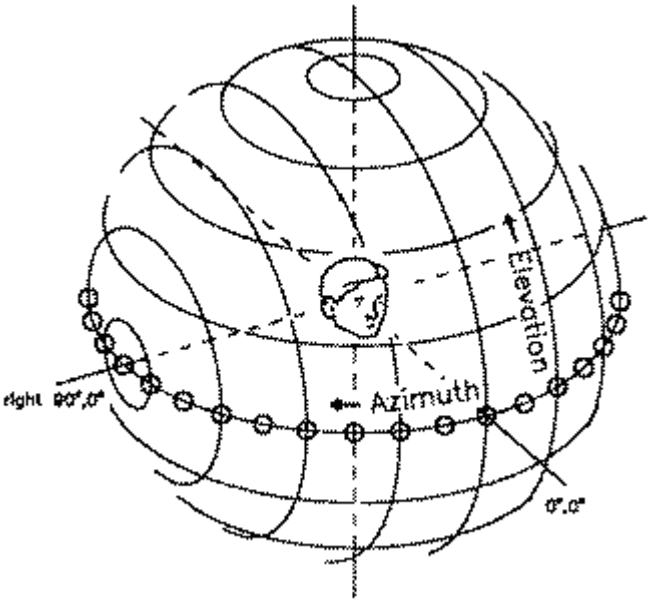


**Figure 2** *Azimuth and elevation.*

The *elevation* is the horizontal deflection, with positive values defined above and negative below. 90° is directly over the head and –90° is directly under the head. The *distance* is simply defined as meters from the centre of the head.

### Pinna, ITD, head shadow and shoulder echo

Reflection and diffraction mainly caused by the pinna and the head, and to somewhat lesser extent by the shoulders, give raise to variations in a perceived sound and play a key role in sound localisation. The interpretation of these cues has been thoroughly examined by researchers for the last five decades.

#### Pinna

The different folds in the pinna modify a sound frequency in a manner that depends on the azimuth, elevation and the spectrum of the sound. Reinforcing some frequencies and attenuating others, the pinna acts as a filter and has responses that allow the brain to estimate the arriving sounds direction. Since everyone's pinna is different, so is the acoustic stamp placed on a sound entering the brain [3].

#### Interaural time delay

The ITD is the delay between a sound reaching the ear closer to the sound source and then the farther ear. It provides a primary cue for the azimuthal information except in the case of a sound source position with an azimuth of either 0° or 180°. A sound source coming directly from the left or the right has an ITD of around 0,63 ms [3]. The frequency of, as well as the linear distance to, a sound source also affect the ITD value.

#### Interaural level difference

The fact that a sound has to go through or around the head to reach an ear account for a significant attenuation of sound intensity. The head also has a filtering effect on the sound, which together with the attenuation of the head gives indication of both direction and distance to a sound source.

### Shoulder echo

Frequencies in the range of 1-3 kHz are reflected from the upper torso and produce echoes that the brain perceive as a time delay relative to the direct sound. Even though this cue is not considered as a primary one, it holds some spatial information [4].

## HRTF

The above cues can be modelled or measured and form a set of head-related transfer functions. Usually the HRTF's are measurements of a sound source made through inserting miniature microphones into the ear canales of a human subject or a mannequin. The measurement procedure is repeated for many locations of the sound source relative to the head, resulting in a database of hundreds of values, describing the sound variation characteristics produced by a particular head [3] [4]. To reproduce the recorded effect of the position with an arbitrary sound, the sound has to be transformed into its frequency components where the HRTF then can be applied and then inversely transformed back into the time domain.

A drawback of this technique is that HRTF's vary considerably from person to person, resulting in poor performance in the synthesised directional cues from a non-personalised measured HRTF. This use of non-individualised HRTF's can result in front/back and elevation errors when reproducing 3D audio [5]. The types of distortions imparted by the pinna and the head fortunately follow some general patterns. Thus meaningful estimations of a median human may be made using average-shaped human models as measuring subjects. There is also a variant where people with proven good sound localisation skills are used as models that achieve good HRTF's.

## Head movement and vision

Since the human anatomical constitution does not allow the ears to move individually we have to move our head to get a better sense of a sound's direction. This fact is well documented and a recent study preformed by Miner et al. [6] states that sound localization generally improves significantly when head movements are allowed.

Our primary tool for localisation is our vision and we rely on it so heavily that we ignore auditory directional cues of a sound source if they disagree with the visual ones.

To satisfy the head movement cue, a head-tracking device can be used with the audio rendering system. As for vision, it is essential to calibrate whatever system one is using in order to make sure that the visual cues match the auditory ones.

## Reverberation

Reverberation comprises all the different reflections produced by a sound in an environment, typically a room. Assuming a direct path exists between the listener and a sound source, this direct sound, or *direct signal*, will be heard first. This will be followed by reflections off nearby surfaces, called *early reflections*, within the first 80 ms after the sound starts [2]. These early reflections are a set of well defined, and directional, reflections that are directly related to the shape and size of the room, as well as the position of the source and listener in the room.

After a few tenths of a second, the number of reflected waves becomes very large and the resulting reverberation is characterized by a dense collection of sound waves travelling in all directions, called the *late reverberation*, see Figure 3. Simulating reverberation is essential for establishing the spatial context of a soundscape. Reverberation gives information about the size and character, such as shape and surface materials, of a space and if modelled correctly it adds greatly to the realism of the simulation.

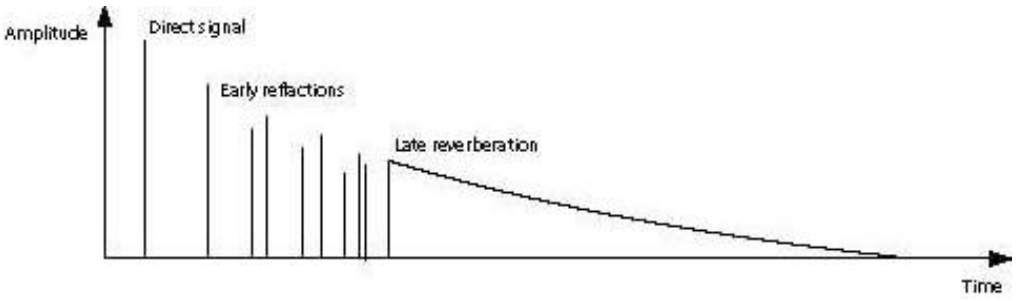


**Figure *3*** *Distinction between the direct signal, early reflections and late reverberation over time.*

A measure that is used to characterize the reverberation in a room is the *reverberation time*. Technically speaking, the reverberation time is the amount of time it takes for the sound pressure level, or intensity, to decay to one millionth (60 dB) of its original value or thousandth of its original amplitude. Longer reverberation times mean that the sound energy stays in the room longer before being absorbed. Typical values of reverberation times run from about 0,3 seconds for a living room to up to 10 seconds for large churches. Most large rooms have reverberation times between 0,7 and 2 seconds [7]. The reverberation time is controlled primarily by two factors, the surfaces in the room and the size of the room. The surfaces of the room determine how much energy is lost in each reflection. Highly reflective materials, such as a concrete or tile floor, brick walls and windows, will increase the reverberation time. Absorptive materials such as curtains, a heavy carpet and people reduce the reverberation time. Further, the absorption of most materials usually varies with frequency.

The loudness of reverberation, in relation to the direct sound, also plays an important role in determining distances [8]. The direct sound decreases in amplitude as the distance to the listener increases. For every doubling of the distance, the amplitude of the direct sound decreases by about a factor of one half, or 6 dB. The amplitude of the reverberation though, does not decrease considerably with increasing distance. The ratio of the direct sound amplitude to the reverberation amplitude is greater with nearby sounds than with it is with more distant sounds, producing an important distance cue.

### Early reflections

The early reflections, also called early echo, does on their own hold many different sound cues such as source direction, source distance, environment dimensions and environment characteristics. The full brain-ear interaction on early reflections, as for a lot of other psycho-acoustic matters, is not fully understood today, but it is an active field of research. An amazing example of information contained in the early reflections is the phenomena of *echolocation* [2]. Experiments have shown that both blind and sighted blindfolded subjects could make use of clicking or hissing sounds from the mouth to estimate distance, width and, in some cases, material composition of objects placed in front of them.

### Late reverberation

The late reverberation is the primary factor establishing a sense of a room's size. In a room, the late reverberation, is often considered nearly diffuse and its impulse response as a exponentially decaying random noise [9].

### Modelling reverberation

In acoustic environment modelling, some parts of the reverberation is often approximated using geometrical models of the simulated space [9]. Geometrical modelling methods use specular reflection to model the sound waves and certain behaviours, such as diffraction and interference, are generally ignored. In other words, a modelled sound reflects off a surface with the same angle as it hit the material. Object dimensions and surfaces are assumed large and its curvatures and imperfections small compared to the sound wavelength. The most commonly used geometrical methods are ray-tracing and image-source.

#### Ray tracing

The ray-tracing method sends a number of non-diverging rays out from a source, usually modelled as a point source, which then are reflected from the surfaces they strike. The listener is penetrated by a number of rays, simulating the sound reflections. In the standard algorithm specular reflection is used. The listener is normally modelled as a sphere since it provides the most pure response patterns and is easy to implement.

A shortcoming of this method is the large number of rays necessary to ensure that all paths from the source to the receiver are covered. A problem that arises as the number of rays has to be approximated with a finite number, and as the rays radiate from a point source, is that the ray-tracing representation gradually becomes less exact with increasing ray lengths, see Figure 4.

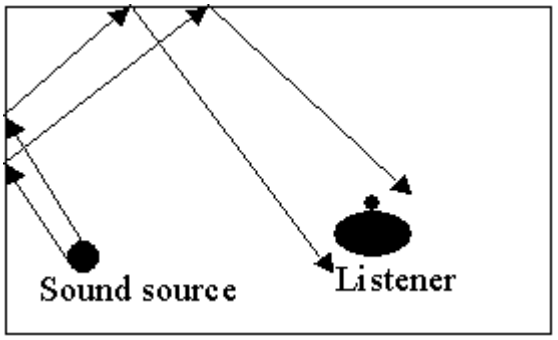


**Figure 4** *Increasing ray length increases the distance between the rays and may leave regions erroneously unaffected.*

More sophisticated algorithmic extensions to the ray-tracing method exist, trying to overcome the problem of unfair detection due to ray length. One such approach is beam tracing where the rays are represented as cones or pyramids, emanating from the sound source [10]. However, these methods face problems with double coverage and may still leave regions erroneously unaffected.

### Image source method

The basic idea of the image source method is to compute specular reflection paths by considering virtual sources, generated by mirroring the location of the sound source over each surface. The locations of the image sources are independent of the receiver's position and when positions in a room change, recalculation of which image sources the listener "sees" has to be done, see Figure 5. This, so called visibility check, is in a brute force implementation done through analysing the direction of the normal vector of each surface. In general, O(*nr*) image sources have to be calculated for *r* reflections in a room with *n* surface planes. This expected computational complexity allows, in practice, only a set of early reflections to be calculated in even a simple environment [10]. There have been several refinements of the image source method and, in using it together with other algorithms, one may reduce the computational load [9].

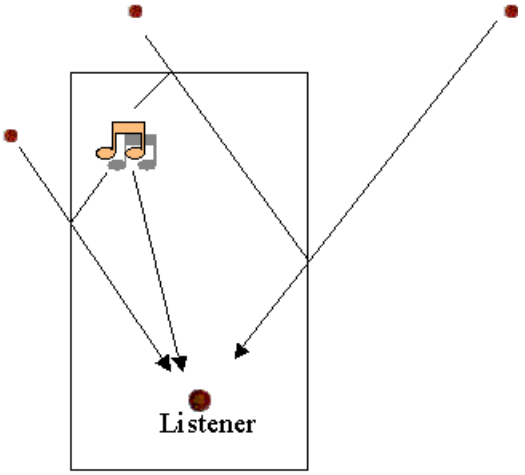


**Figure 5** *The direct sound path and the image sources that the listener "sees" in this particular position.*

## Occlusion and obstruction effects

Occlusion and obstruction are two physical phenomena that also have to be taken into account when considering simulating an acoustic environment.

### Occlusion

Occlusion occurs when a material separates two environments and comes between the sound source and the listener. Since no open-air sound path exists, all sound reaching the listener has to travel through a, more or less, muffling material.

### Obstruction

The sound from a sound source behind an obstructing object diffracts around the object to reach the listener. Wavelengths larger than the occluding object are not affected much, but in the case of wavelengths smaller than the object a considerable attenuation will be the result.

Sounds that are transmitted through material structures undergo a frequency dependent attenuation, depending on material and thickness, and will usually have a character that may be simulated using low-pass filters. In the case of obstruction the reflected sound remains unaffected and in occlusion all sound paths are affected.

The air in a space will of course also attenuate the sound waves, resulting in lower loudness as well as a reduction of the reverberation time. This attenuation varies with the humidity and temperature.

## Systems for synthetic spatialisation

There exists a great number of systems for artificially creating spatialised sound. Many of these systems are heavily relying on hardware whilst others are software based.

Sound synthesis languages, such as Csound[3] or Music-V[4], have been used by the audio synthesis research community for over 30 years. These techniques are useful for limited applications in music synthesis and sound effects processing, but they do not generalize to the task of creating sounds for use in more demanding applications, such as a real time augmented audio reality system.

The purpose of this project is to create a lightweight and flexible spatial sound system on an of-the-shelf platform. This has narrowed the investigation of existing systems for synthetic sound spatialisation to only include three rendering systems, namely Microsoft's DirectSound3D, its open standard counter part OpenAL and the EAX technology. DirectSound3D and OpenAL mainly provide 3D audio capabilities and EAX provides an interface to render the more CPU intensive effects, such as reverberation, reflections and occlusion, in the audio hardware.

The Lake DSP audio rendering product, Huron, is also briefly described as it will be used in the coming spatial sound quality evaluation.

### Standardisation

As the consumer PC market is flooded by products labelled 3D audio[5] the Midi Manufacturer's Association (MMA) has formed an Interactive Audio Special Interest Group (IASIG). IASIG has a 3D sound Working Group who has defined a specification of Interactive 3D Audio Rendering Guidelines (I3DL) [11]. In 1999 the second version of these guide lines were formulated, I3DL2, that has set a standard for what producers should call 3D audio in terms of positional audio and environmental acoustic modelling. The IASIG standard is supported by both DirectSound3D and OpenAL.

3 www.csound.com
4 The MUSIC series software went through an evolution following the development of the IBM computers which ended with Music-V written in FORTRAN running on the IBM 360 machines.
5 3D audio is from the commercial point of view pretty much the same as spatial sound. A clear distinction between surround, 3D and environmental effects in a sound is though usually lacking.

### OpenAL

The Open Audio Library, OpenAL[6], is an effort to create an open and vendor-neutral API for spatialised audio. Like the two next rendering systems to be described in this thesis the OpenAL is a software interface to audio hardware. The interface constitutes a number of functions that allow the programmer to produce audio output of 3D and environmental arrangements of sound sources around a listener.

The OpenAL is currently defined through *The Final Draft of the OpenAL 1.0 Specification* that was released in October 2000 and it supports the I3DL2 standard. The status of this API[7] is that there are still some crucial functionality suffering from not having full support, such as certain environmental effects, such as occlusion and obstruction.

### DirectSound3D

The *DirectSound3D* API is a part of Microsoft DirectX[8] software suite that was released in 1996 and is a set of different multi-media API's. In August 2000, version 8 of DirectX was released with an updated audio interface, DirectX Audio, which the *DirectSound3D* library. The *DirectSound3D* interface supports the I3DL2 guidelines as well as it is including HRTF[9] simulation. *DirectSound3D* is object orientated and the two most central objects are the *sound buffer,* which provide the mechanism for creating sound sources and listeners, and the *interface* that ties certain characteristics to a buffer.

#### Sound buffers

The sound buffers are used in DirectSound3D to contain a set of values in a waveform table. Converted into an analogue representation these values will produce sound on an audio system.

There is always a primary buffer from which to feed the waveform data directly to the audio system through the digital-to-audio converter. The primary buffer can support multi-channel playback by interleaving samples for each channel within a single table. Since the primary buffer is the direct waveform to be output and played it represents the *listener*, and is often referred to as such. The primary buffer is therefore assigned listener settings such as position, orientation and velocity.

When running an application, the primary buffer receives a mix of waveform data from other sound buffers, called secondary buffers. The number of secondary buffers supported is limited to system RAM capacity. The secondary buffers are usually seen as the sound sources and each holds a waveform table created by the application that may be assigned a position, minimum and maximum distances from the listener and so on. DirectSound3D keeps track of the location of the secondary sound buffers in relation to the primary buffer and alters their output to simulate three dimensional audio.

#### Interfaces

To provide control of the buffers the DirectSound3D uses interface objects. An application controls each sound buffer through the buffer's interface by calling member functions on the interface. The standard interface includes functions such as volume and frequency control. In order to render 3D characteristics it is also necessary to tie a 3D interface to the buffer.

It is also possible for third party vendors to define interfaces through so called property sets. One example of a property set, forming an interface object, is the EAX which provides an environmental acoustics interface to apply on top of the standard and the 3D buffer interfaces.

---

[6] www.openal.org
[7] October 2001
[8] www.microsoft.com
[9] HRTF's in DirectSound3D are vMax technology licensed from Harmon-Kardon.

## EAX

The open standard EAX stands for *Environmental Audio Extensions* and is created by Creative Labs Ltd. It is a layer on top of DirectSound3D or OpenAL providing optimised hardware rendering of environmental effects.

EAX includes two different property sets, one for the primary buffer and one for secondary buffers. The properties of the primary buffer property set control the overall aural environment and affect the way all sound sources are perceived in the environment. The secondary buffer property set control the environmental effects applied to each individual sound source. It controls the amount of attenuation and tonal filtering applied to the source's direct and reflected sounds, which determines the amount of reverberation, obstruction and occlusion the listener hears for the source. See Appendix B - EAX Buffer Properties, for the property sets description of the EAX interface. Tweaking all these parameters separately when designing a sound scene can be very tedious, so Creative Labs has developed a DLL called EAXManager that at run-time can provide the system with predefined settings previously retrieved from an Environmental Audio Library (.eal) file. These files are created through a graphical editor called EAGLE where applied sound effects may be rendered and listened to directly. Doing this outside the programming environment gives an opportunity to design soundscapes more intuitively.

### EAXManager

The EAXManager is a COM interface that resides in a DLL. It provides the handling and the processing of data settings provided through an .eal file. The application has to assign the .eal file to the EAXManager and may then query the EAXManager for the appropriate render settings of the sound objects. The application does for example query for occlusion values of a particular sound source given its current position. The EaxManager do not take care of actually setting the values, but returns the proper data for the application to set.

### EAGLE

EAGLE (Environmental Audio Graphical Librarian Editor) is a graphical editor for creating and designing sound environments. EAGLE supports the EAX standard. The editor produces .eal files that may be interpreted by an application including the eaxman.dll.

The EAGLE reads a number of standard geometry files such as 3D Studio Max and Lightwave 3D Object files. Once the geometry is imported the designer can start assign different areas with acoustic effects. Sound sources may be placed and assigned whatever properties necessary. Partitions between different acoustical environments, such as a door opening, can be given occlusion parameters and objects may be assigned obstruction values. Figure 4 is an example of what the EAGLE design environment looks like.

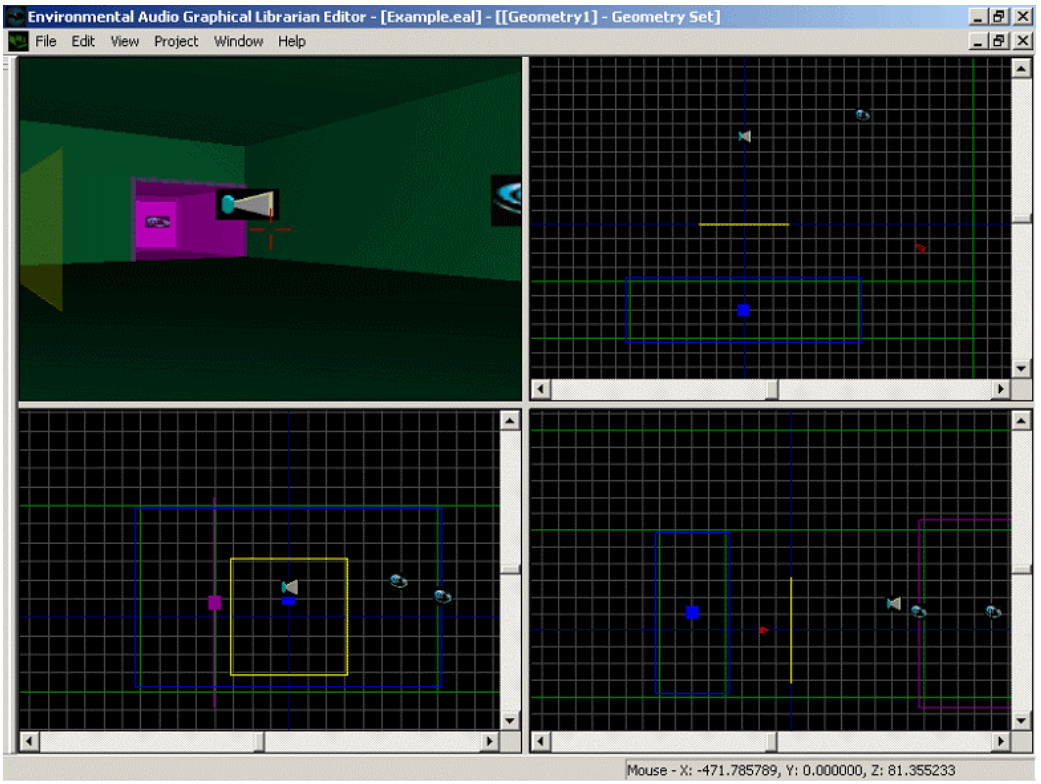


**Figure 6** *Example of an EAGLE design environment.*

### Huron

The HURON workstation, manufactured by the Australian company Lake DSP[10], is a high quality, and very expensive spatial audio rendering machine. By using twelve DSP's the system is capable of mapping up to twenty audio streams in real time using up to ten loudspeakers at 48 kHz sampling rate and 18 bits resolution. Headsets can also be used as the workstation provides filters for calculating HRTF.

Reverberation and other environmental effects for the acoustical behaviour of rooms can be applied to the audio. Software support is provided via an application framework, providing a Windows NT-based user interface to the DSP hardware and the firmware system.

[10] www.lakedsp.com

## Spatial audio reproduction schemes

Spatial audio reproduction schemes can be divided into three different categories [12], namely biaural (headphones), crosstalk-cancelled binaural (two loudspeakers) and multi-channel reproduction (several loudspeakers).

### Binaural

It is straightforward to deliver the appropriate sound fields to each ear with the use of headphones. No consideration of the environment of the playback is needed and head tracking can be integrated with the headphones allowing effective HRTF rendering. Headphone reproduction also allows great mobility for the user, since they could be wireless.

A major drawback of this technique is that headphones generally do not allow real-world sounds to enter the ears, which may be an important feature in an augmented reality. A lot of people also experience fatigue from long time listening while wearing headphones. Attempts with shoulder mounted speakers, enabling spatial reproduction in the same way as with headphones but without suffering from the previously mentioned caveats, have been tested in an audio augmented reality system called Nomadic Radio [13], see Chapter 3. At an initial phase of this project some attempts of using shoulder-mounted speakers was made. Figure 5 shows testing of a system manufactured by Sennheiser.

**Figure 7** *Shoulder mounted speakers.*

The poor sound quality of this model, as well as the lack of other models, led me to abandon further investigation of shoulder mounted speakers.

### Crosstalk cancelled binaural

Today there is a large number of desktop computers with two loudspeakers mounted on either side of the monitor. This has motivated the development of binaural sound reproduction using two loudspeakers placed in front of a listener. In doing this, it is necessary to eliminate the crosstalk [14] that arises due to the sound each loudspeaker sends out to the ipsilateral ear. In Figure 6, the crosstalk paths are labelled $A_{LR}$ and $A_{RL}$. The crosstalk severely degrades localization performance.

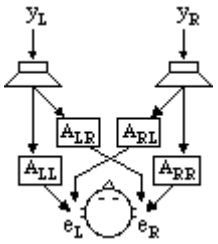


**Figure 8** *Crosstalk terms $A_{LR}$ and $A_{RL}$ needs to be cancelled out in order to achieve spatial sound.*

Considering this cross-talk factor and cancelling it with appropriate filters is called cross-talk cancellation. To be efficient, this reproduction system also must incorporate head-tracking or the best listening area, called the sweet spot, will be very small. It should further more take account of reflections from nearby surfaces in the playback environment, such as the desk the computer is placed on.

### Multi-channel

The most common multi-channel spatial sound reproduction techniques applied today are typically based on amplitude panning [15]. This means that the same sound signal is applied to a number of loudspeakers equidistant from the listener, each speaker reproducing the sound with appropriate amplitude to spatialise the sound. The most common systems, based on amplitude panning, used [9] are Ambisonics and Vector Base Amplitude Panning (VBAP).

#### Ambisonics

Ambisonics is an amplitude panning method in which a sound signal is applied to all loudspeakers placed evenly around a listener. The feature that a sound emanates through all speakers is, simply speaking, what spatialises the sound. As a 3D loudspeaker set up, Ambisonics is typically applied as eight speakers in a cubical arrangement or as twelve loudspeakers as two hexagons on top of each other.

#### VBAP

The vector base amplitude panning (VBAP) is an amplitude panning method developed by Pullki [15]. The VBAP can be used with any number of loudspeakers in any position to simulate 3D virtual sound source positions.

# 3.0 Related work

A common concept of audio augmented information systems is the audio guide services currently offered by almost all big museums around the world. These systems offer a random access library of stored information that you retrieve by either entering a specific code on a portable device, such as a CD-player, or through positioning yourself in zones were infrared-based techniques play loops of the information around the exhibit. Drawbacks with this kind of systems is either the inconvenience of carrying around a playback system or having a system were the timing of the audio clips not are individually controllable.

Providing more sophisticated auditory cues based on a person's position and actions have been explored in several previous projects. An early prototype of an audio augmented reality tour guide, with the capability to provide individualized information and triggered by the users position was implemented by Benderson [16]. In this system you still had to carry around the audio source. A processor was added to the playback device and controlled the playing of the audio clips depending on the user location, which was delivered through an infrared tracing system.

This concept was developed further by the Guided by Voices [17] system who also used a simple wearable computer and a radio frequency based location technique and then play different digital sounds, narrations, sound effects or ambient sounds, corresponding to the user's location. They also added a state to each user that could be manipulated through actions. One notable conclusion from this system, which was implemented as a medieval fantasy world role-play, was the importance of utilizing different *layers of sound*, such as narrations, sound effects and ambient sounds in creating a non-trivial immersive audio augmented reality.

A third approach of a augmented reality audio system, called Hear & There [18], was developed by the Social Media Group at the MIT Media Lab. In this system, users create the content of the augmented audio space by recording their own sounds and then embed them at a particular location. All users traversing the designated location can then hear these *audio imprints*. The hardware in this system, headphones with a digital compass, a laptop, a palm pilot, a GPS receiver, a battery and a microphone is rather bulky and has to be placed on a luggage cart in order to allow mobility. The digital compass together with the GPS allows the head position of the user to be known and thereby the reproduced sound to be spatialised. One of the main explorations of this ongoing project is the feasibility of navigation in an augmented audio environment.

Two more extensive as well as better documented projects involving audio augmented environments are Audio Aura [19] and Nomadic Radio [13].

## Audio Aura

The Audio Aura project explores augmented audio tied to people's physical actions in office environments. The primary goal of the system is to provide useful serendipitous information, that is information not actively asked for, via different ambient soundscapes. The audio, primarily non-speech, creates a non-distracting peripheral display with a low perception cost. Since information needs and interface preferences do vary a lot between users, emphasis was put on creating an easily configurable system for the end users.

The Audio Aura system is implemented using active badges, a server and wireless headphones. An active badge is a small electronic badge, to be worn by a person, that emits a unique infrared signal. Sensors distributed in a building pick up the signal and this position information, combined with other sources of information such as emails in the users inbox and personal agendas, triggers the system server to provide the auditory cues to be sent to the user via the wireless headphones.

Three different sample scenarios, in which to provide and test the serendipitous information ideas, guided the design and development of Audio Aura.

- Email notification through audio cues. When for example entering the bistro in between meetings you will hear a cue conveying approximately how many new email messages you have and indicating messages from particular persons and groups.
- When people drop by other people's offices finding no one there, the Audio Aura provides cues on whether the person has been in that day, been gone for some time or if the person was just missed. The system does not deliver information like "Mr. K has been gone for 45 minutes" but tries to, via auditory cues, provide an augmentation of the empty rooms status: is the light on, is there a briefcase by the table, audio footprints and so on.
- Since many people are not co-located with their collaborators the last scenario envisioned tries to create a "group pulse". Whether people are working that day, on what they are working and if some are working on the same thing, maybe even in a face to face situation, are things that trigger changes in the systems audio cues.

The Audio Aura system explores three different types of sound; speech, musical and sound effects. Within these different sound domains *sonic ecologies* were created. For example, one sound effect design mapped particular sets of functionalities to various beach sounds. The amount of email was mapped to seagull cries, email from particular persons or groups were mapped to various beach birds and seal cries, group activity was represented as surf, the wave volume and the wave activity, and audio footprints are mapped to the number of buoy bells.

No complete evaluation from this project exists at the moment. One conclusion from initial user reactions of the Audio Aura system, when trying out the sound effects of the above described sonic ecology, is that some users found the meaning of the sounds hard to remember.

## Nomadic Radio

Nomadic Radio is a message application utilising spatialised audio, speech synthesis, speech recognition and location awareness, developed by the Speech Interface Group at MIT Media Lab. The user can choose one or more message categories, such as email, news or personal calendar, and the messages are then, in order to enable the listener to better segregate the multiple information sources, presented simultaneously as spatialised audio streams. A speech recognition module provides means for navigation of the system.

In Nomadic Radio the clients run on a wearable computer that provides the real-time spatialisation of the sound as well as the speech recognition interface. A remote server deals with the filtering and the prioritisation of the incoming messages and includes an *audio classifier* that detects whether the user is speaking to the system or is engaged in another conversation. The system then dynamically adjusts the level of notification for incoming messages. To provide as unobtrusive interface as possible the audio reproduction platform used is a system called *SoundBeam Neckset*, developed by Nortel[11]. It is worn around the neck and consist of two directional speakers placed on the user's shoulders and a microphone over the chest. A button on the *SoundBeam Neckset* activates or deactivates the speech recognition.

The system has primarily been used to explore and evaluate different schemes for Audio User Interfaces (AUI) in nomadic situations. Topics such as contextual recognition, peripheral awareness and spatial listening have been examined thoroughly.

[11] www.nortelnetworks.com

Some of the experiences learnt in the Nomadic Radio project were:

- Acquiring a particular service from the application is not easily done since quite many different commands must be recalled by the user for an efficient utilisation of the system. Hence the interaction with the system must be designed in a truly intuitive way allowing the user to gradually become familiar with the syntax.
- Hearing synthetic speech can be tedious due to its sequential and transient nature.
- A particular problem of the Nomadic Radio system, since it produces the output audio on loud speakers, is that other persons can take part of the messages. The contextual awareness of the system is therefore of utter importance.
- One major conclusion drawn from user evaluation was that ambient audio provided the most benefit while requiring least cognitive effort. The users in these particular test also wished to hear ambient audio at all times in order to remain reassured that the system was operational and on.

## Summary of related work

From my investigation on previous work on audio augmented realities I conclude that surprisingly few attempts has been done in using sound interfaces in AR environments. Trying to bring user friendly interfaces into situations where the visual perception should be undisturbed seems rather unexplored.

One conclusion to be drawn from the above implementations is that the sound design, what sounds to play and when to play them, in an audio environment is crucial. This might seem obvious but the importance of not underestimating the complexity in creating natural and intuitive soundscapes must not be neglected. None of the systems covered use sophisticated spatial rendering of the audio. I believe that with more accurate rendering of the sound more information may be put into it, which could enable more refined interfaces to be created.

# 4.0 AAR system

The system design *overview* of the Audio Augmented Reality (AAR) system gives a conceptual presentation of the different parts, and how they interact. The *implementation* part of this chapter describes some more technical aspects of the system and the evaluation gives a hint on its strengths and weaknesses.

The design of the system is built on the DirectSound3D and the EAX API's, to render the 3D audio and the environmental acoustics. The three corner stones of the system are the listener, the emitter and their environment. The ability to easily create and manipulate these entities is provided through an interface to the EAGLE software.

## Overview

As a listener moves around freely in a space, different locations, such as physical rooms or parts of rooms, provide boundaries between different acoustic landscapes. In changing location the listener experiences a morphing between acoustic sceneries. Different audio objects, emitters, are placed within these environments and can be made to interact with the listener based on his location.

The listener hears the audio played through headphones. As he moves, a tracker, mounted on the headphones, registers head position and the system renders the audio appropriately. In other words, as the user moves around and turn the head the audio objects will always appear as coming from wherever the designer of the scene has chosen. As he moves around, the acoustical environment may change as he passes predefined borders, or according to actions taken.

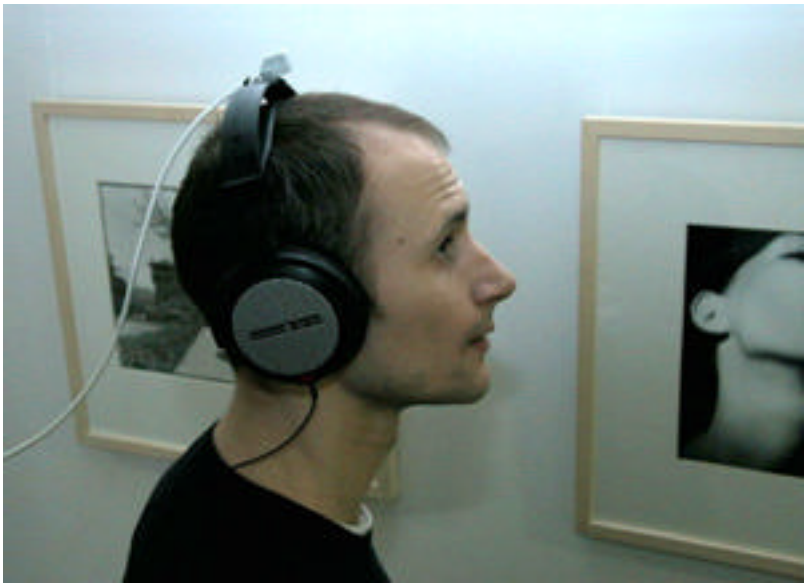

**Figure 9** *The author poses, wearing the AAR system head set.*

Figure 7 shows the AAR system head set. The head set is made up of a pair of head phones and a tracking device.

### Client/server

The networking is based on a straightforward client-server model. One or more clients connect to a server. The server makes sure the appropriate audio is rendered and synchronised.

The client creates emitters, one listener and the acoustical environment. These objects are registered with the server. The client continuously provides positional data of the listener's whereabouts to the server. The emitter might as well move depending on what the programmer has chosen. Finally the acoustics connected with the listeners location may be altered in runtime or predefined in a .eal file.

The server, using its different libraries, renders the audio according to the wishes of the clients. The rendering is performed in real time according to the positional data of the listener and the emitters.

### Listener

There is one listener per client. A position tracking system delivers real time values of the listener position in X, Y and Z co-ordinates and, optionally, of his head orientation. The orientation is defined by the relationship between two vectors, both with origin at the centre of the listener's head. The first vector points forward through the listener's face and the second point's straight up through the top of the head at right angle to the forward vector.

### Emitter

The emitter is a sound source. The system resources limit the number of emitters in a session. The DirectSound3D provides an emitter model with minimum and maximum distance values in relation to the listener.

- Emitter minimum distance. As a listener gets closer to a sound source the sound gets louder. Past a certain point, however, it is not reasonable for the volume to increase. This is the emitter's minimum distance.
- Emitter maximum distance is the distance beyond which the sound does not get any quieter. This can also be used to prevent a sound from becoming inaudible as a listener moves away from it.

By default, distance values are expressed in meters. The emitters may, through unique group ID's be synchronised and manipulated as a group.

Emitters also have an orientation. The model that is supported in the AAR system, provided through the DirectSound3D library, is called sound cones. It describes the loudness of the orientated sound and is made up of an inner cone and an outer cone with differences in attenuation.

Within the inner cone the volume of the sound is just what the designer has set it to in accordance with the above described distance model. At any angle outside the outer cone the volume is attenuated by a factor set by the AAR application. Between the inner and outer cones is a zone of transition, from the inside volume to the outside volume, where the volume increases as the angle decreases, see Figure 8.

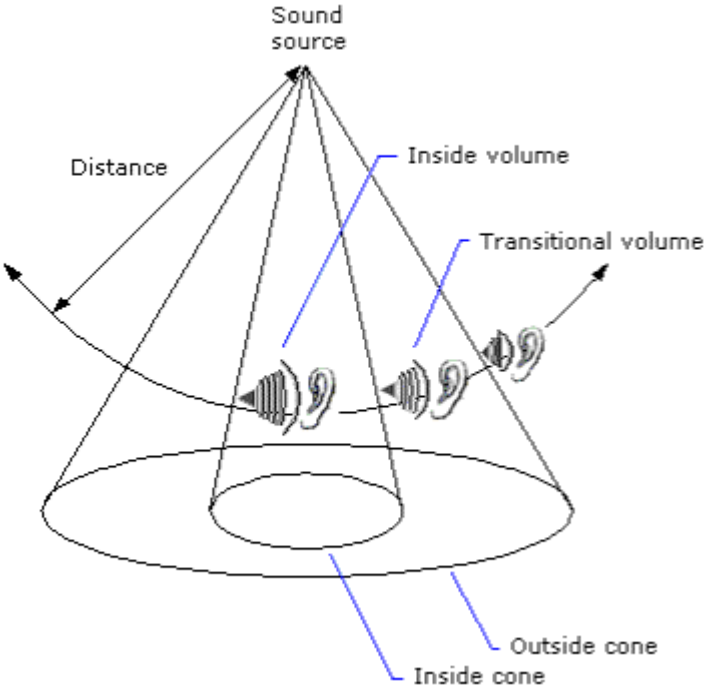


**Figure 10** *Sound cones defining a sounds orientation.*

The default value is set to 360° for both the inner and the outer cones, creating omni directionality.

### Acoustical environment

In addition to the 3D positional information regarding the listener and the emitters, the system also add environmental audio effects to the rendered sounds. Different areas in space may be assigned different environmental acoustics. The included obstruction effect allows emitters to be put behind objects, physical or virtual, in an application. The occlusion effect provides the possibility to put emitters in an adjacent room or outside a window.

Figure 9 illustrates a schematic overview of the AAR system.

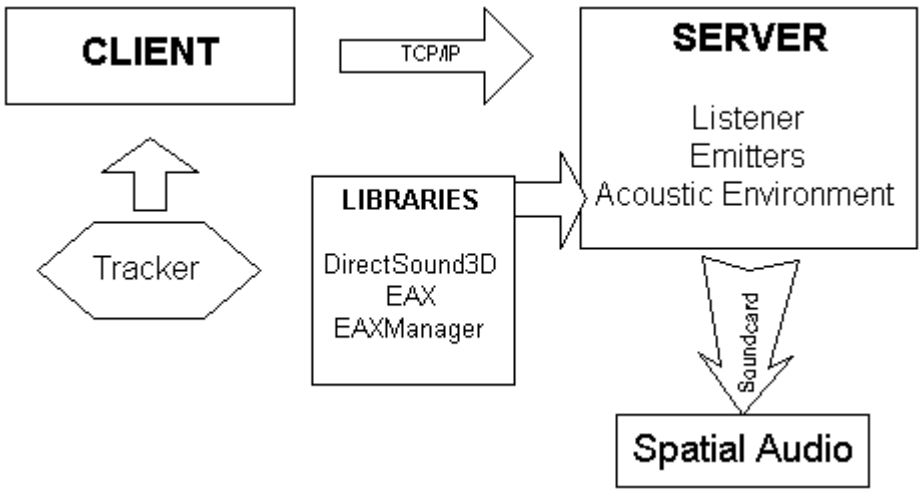


**Figure 11** *Schematic overview if the AAR system.*

## Implementation

The intention with this description of the AAR system implementation is that it should be fairly straightforward to re-implement a similar system with the same structure and the same functionality. The code is written in C++ using Microsoft Visual Studio version 6.0. The system is implemented based on an existing framework for spatial audio transmission over TCP/IP and UDP/IP, called the Spatial Audio Server (SAS), developed at Fraunhofer IGD in 1997.

### System hardware

The AAR system is implemented on a Microsoft Windows based standard PC equipped with a Pentium III processor. For the audio playback a pair of Sennheiser HD565 headphones was used.

A sound card that supports the EAX 2.0 standard is required in order to use the environmental capabilities of the implementation. In my set up an ordinary SoundBlaster Audigy was used.

A Polhemus FasTrack[12] collects the positional data, where the sensor is mounted on the headphones. The FasTrack positioning device allow the simultaneous tracking of both head position and orientation and promises a less than 1 mm range accuracy over the X, Y and Z axis, and 0.15° angular resolution of the orientation. The trackable space is a hemisphere of up to 3 meters. A DSP provides an update rate at 120 Hz with a 4 ms latency. The data is transmitted via a serial interface.

The tracker uses electromagnetic induction and is very sensitive to metal objects in the tracked environment. If there are lots of metal cabinets, desks and computers around the active area, the tracker will not function properly.

### Hardware limitations

The implementation suffers from some hardware and system constrains. In order to be able to render 3D position and environmental effects on the audio in real time I rely on the DirectSound3D and the soundcard to output this. In the DirectSound3D system there is only one primary buffer containing the waveform data to be fed directly to the sound card. This means that I can only render the audio for one client at a particular given point in time.

The FasTrack might not be considered an inexpensive “of the shelf product”. I use this device nevertheless since including the calculation of HRTF’s, where head position is crucial, is essential when using a biaural reproduction of the audio. Providing positional data could probably be done in other, less expensive ways, in future implementations, such as using a digital compass.

[12] www.polhemus.com

### Client

The client is implemented as an API and resides in a DLL. This library provides all different network and audio related commands the client can use, see Appendix A - Client.dll.

To start a session the `Clientconnect` command registers with a particular host submitting port number and a user name. `initialize_audio` initialises the Directsound3D and the EAX sound libraries as well as the listener object. The command `create_Cached_emitter` registers sound sources, typically a .WAV file, which later may be controlled through a variety of commands such as play, pause, synchronise, position etc. If using an .eal file, through the `AARloadEAL_file` command, the previous command is redundant since all that logic is included together with all the environmental acoustics variables via the eaxman.dll (see Chapter 2, EAX). If not running a previously designed .eal file, the client API also allows setting all emitter and environmental parameters separately through commands such as `reverbationSetAllParameters and setEAXEmitterProperties.`

The data for the positions is acquired via a serial interface to the Polhemus FasTrack. A thread is running in order to catch positional updates from the tracking device and execute the `AARset_listener_position` and `AARset_listener_orientation` commands as the user moves around in the created sound environment

### Network protocol

The network protocol runs over TCP/IP. Support for UDP/IP exists, but the AAR system is currently not implemented in a manner to use it.

The two different data packages used in the AAR system are an *initialisation package* and a *data package*. The *initialisation package* is shown in Table 1. The two possible *types* of the *initialisation package* are shown in Table 2. The second package, the *data package*, is shown in Table 3.

| Field name | Field length | Description |
|---|---|---|
| Type | 4 bytes | Value identifying the package type. |
| Port | char[PORTSIZE_MAX] | The server uses this field to return the port number used for the data connection to the client. |
| Username | char[STRINGLENGTH_MAX] | The name of the client's user. |
| Hostname | char[STRINGLENGTH_MAX] | The client's hostname. |
| IP address | char[STRINGLENGTH_MAX] | The client's IP address. |

**Table 1** *Initialisation package.*

| Type | Value | Description |
|---|---|---|
| INI_CONTACT | 0 | Used by the client to request a data connection with the server. |
| INI_ANSWER | 1 | Used by the server to tell a client that the requested data connection has been established. |

**Table 2** *The different types of the initialisation package.*

| Field name | Field length | Description |
|---|---|---|
| Length | 4 bytes | Length of the whole package. |
| Command | 4 bytes | Command |
| Data | Char[DATASIZE_MAX] | Data belonging to the command. |

**Table 3** *The data package.*

The command field definitions are corresponding to the different client functions calls and are presented in Appendix A - Network Commands.

### Server

There are four main classes; the *server* class, the *sound interface* class, the *listener* class and the *emitter* class. Apart from these main classes are there a number of classes and functions, within the server implementation or in different libraries, providing the server with error handling, networking and audio buffer synchronisation among other things.

The tool used for implementation was the Microsoft Visual Studio C++. The only Microsoft specific features used in the implementation are as listed below.

- The Microsoft Foundation Classes (MFC) - for a small Graphical User Interface (GUI).
- A CWinApp object - as an entry point when running the server.
- The DirectSound3D API - for rendering parts of the audio.

In other words, if deciding to implement the server on a Linux platform the code is very much reusable if OpenAL is used instead of DirectSound3D.

### Server class

The server class is initialised through a standard CWinApp object with a pointer to a *sound interface,* created by the CWinApp object, and a port number as construction parameters. The constructor of the server then sets up and initialises the socket communication.

The server class implements all client administration and control. When a client connects to the server the `handleEvent` method creates a new client object and calls `addClient`, which then registers the client to the server. Further, a number of client administrative commands exist such as `clientCount and deleteClient`. The server receives and unwraps the data packets coming from the clients in the `reciveTcpData` method and parses the first two fields (see Table 2). The *command field* value decides which of the methods the `reciveTcpData` should call.

The server implements all methods necessary to respond to the client commands, examples of these are `audioInit`, `createEmitter` and `listenerPosition`, see Appendix A - server.h. These methods parse the *data field* of the data packet, extracting the particular information needed to call the *sound interface* objects corresponding method. In the `createEmitter` method the server also adds on a session unique *emitter id* tag.

### Sound interface class

The sound interface class implements all initialisation and control of the environmental effects, the emitters and the listener. Its methods are called from the server class and it handles most of the API calls to DirectSound3D and the EAX API. The AAR system sound interface must not be mistaken for the DirectSound3D interface objects that are tied to the different buffers by the *listener* and *emitter* classes, see below.

In `initializeAudio` the DirectSound3D interface object, the primary buffer and the listener object are created and initialised. The EaxManager interface is also set up. As seen in Appendix A - soundInterface.h, all the audio related methods that are represented in the *server* class also are available in the sound interface class.

### Listener class

The listener class constructor takes, among others, a primary buffer as in parameter. It initialises the primary buffer and assigns it the necessary Directsound3D and EAX interface objects. In `setEAXManListenerEnvironment` it is possible to register the listener with the EAXManager. `setEAXManListener` should then be called for positional updates, not the `setPosition` method. For the full definition of the listener class see Appendix A - listener.h.

### Emitter class

When creating an emitter object the emitter class dynamically allocates a new secondary sound buffer through the DirectSound3D API. It fills the buffer with the assigned sound data as well as creates the 3D-buffer and the EAX interface objects. Cone angles, cone orientation, max/min distances are set to default values. `setOrientation` and `setEAXProperties`, see Appendix A - emitter.h, are some of the available methods applicable to the emitter. If running an .eal file, the `sourceEAXman` should be used instead of `setPosition`.

## AAR system evaluation

In evaluating the implementation of the AAR system I will consider some general application level performance issues and spatial sound quality.

My evaluation is limited to running one client at a time as the system setup needs one audio hardware unit per client. The audio rendering of the AAR system depends largely on the underlying implementation of the DirectSound3D and since Microsoft does not use an open source principle I am not allowed to analyse it on a level more close to the driver. My system evaluation therefore only concerns some general aspects on the application level.

The spatial quality of reproduced sound is evaluated through listener tests. A structured and formalised methodology of the evaluation of spatialised sound is yet to be formulated but attempts has been made [20]. Pulkki [15] suggests attributes such as envelopment, naturalness, sense of space, directional quality and timbre to be included in a spatial sound quality evaluation.

Since conducting listener tests are rather time-consuming several attempts have also been made to create objective tests of spatial sound quality [21]. No such evaluation was made in this project due to the lack of high quality measurement instruments, which are required in these tests.

### General system performance evaluation

In terms of CPU effort the system at run time consumes moderate amounts of resources, up to playing about eight sound sources at the same time. Above eight emitters my application do affect performance of the system with mainly clicks and other missounds as result. With a more powerful CPU or maybe a more optimised audio driver the number of emitters playing simultaneously would increase. The environmental effects put on a sound do not affect the performance in any notable way. The run time mapping of the environmental affects from the .eal file runs with no apparent effect on system resources no matter what size of the designed environments.

The tracking device turned out to cause some problems. The FasTrack's positional data was at times totally out of range for up to half a second. This caused the server to produce sounds positioned in the wrong place, with a confusing effect for the listener as a result. This problem could be caused of malfunctioning hardware or because I didn't manage to provide a good enough surrounding, free from interfering metal object. The symptom continued even after my effort to clean the lab of potential disturbances, which together with the fact that the fault did not occur too often led me to give up further action in order to fix this bug. One possible solution would otherwise have been to interpolate the positional data series to smoothen sudden, and incorrect, changes out.

### Spatial sound evaluation

This evaluation is conducted through listener tests set up to investigate the AAR system spatial sound quality. This is done through a comparison of the spatial sound of the AAR system and a Huron system. The task the test subjects underwent is to listen to different recordings and then scale certain aspects of its spatial qualities. The Huron machine will here be considered as a reference system providing good quality spatialised audio.

Experience from similar tests [22] has shown that the number of test variables per task should be kept at a minimum. I decided to limit my test variables to naturalness, externalisation and directional quality, focusing on the first two. The directional capacity of the system is to be further explored in the implemented applications. The aim was to keep the tests variables as easily distinguishable from each other and as straightforward as possible in order to avoid any potential misunderstandings by the test subjects.

## Method

Twelve unpaid test subjects were asked to listen to seven differently generated sounds. They where asked to determine whether the sound localisation appeared to be inside or outside the head, which is called externalisation. The authenticity of the acoustic environment, also called its *naturalness*, was investigated through a question if the sounds seemed *very natural*, *natural*, *synthetic* or *very synthetic*. Together with both of these questions came the option of being able *not to decide*. The questions were presented on a paper in a multiple-choice fashion, see Figure 10. The subjects had the opportunity to listen to every sound as many times as they needed. Further the perceived sound direction was also asked for as the subjects had to mark the sound direction in a X-Y coordinate plane.

*The sound localization appears to be:*

| Inside Head | Outside Head | Can't Tell |
|---|---|---|

*The acoustic environment seems:*

| Very Natural | Natural | Can't Decide | Synthetic | Very Synthetic |
|---|---|---|---|---|

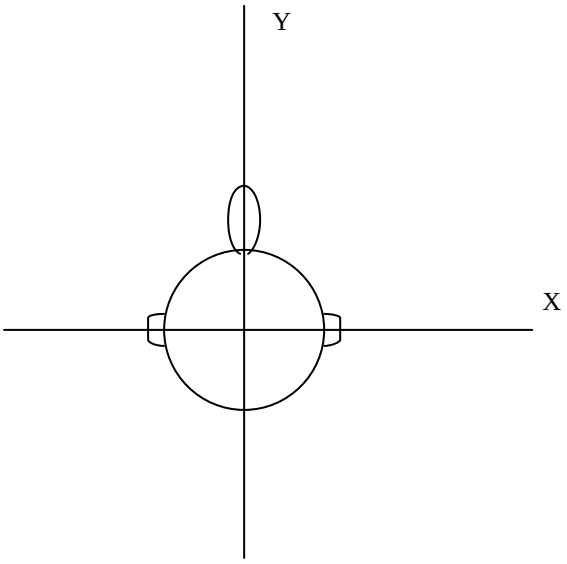


**Figure 12** *Multiple-choice questions and the 2D plane in which to mark sound direction.*

Four different recorded anechoic sounds where rendered either with the AAR system, or by the Huron machine, in three different acoustical environments and with the sound emerging from three different locations. The three different acoustical sets were a very small room (small cellar), large sized room (large living room) and a huge room (larger hall).

The positional test I wanted to implement was at first also going to test for sound sources in three dimensions, but from performing the test myself a couple of times I decided to exclude these. There were already many parameters to ask for in the test and since head movement is essential in more precise localisation of a sound I would have to use the tracking device for the tests. This I wanted to avoid since the test was designed to be performed sitting in front of a computer and the closeness to the monitor could interfere with the electromagnetic components of the FasTrack tracking device. The 3D directional capabilities of the system were evaluated later in the implemented applications.

The three acoustical environments were rendered using the two different systems but with different recordings. In the different environments the sound sources where placed according to the following:

*Small Room – 2 meters*

*Large Room – 20 meters*

*Huge Room – 40 meters*

The recordings where a snare drum, a man reading a text, a flute playing and a violin playing. Not having too similar sounds was the only thought behind choosing these particular recordings. I randomly rendered the sounds in the different milieus. The positions of the sound sources were chosen differently in order to be able to perform the directivity test on the same sound samples.

The mapping of the different recordings to a numbered test is described below, with the angle being the azimuth of the sound:

*Test 1 = AAR Small Room (SR), 0°*

*Test 2 = Huron Small Room (SR), 0°*

*Test 3 = Huron Large Room (LR), 45°*

*Test 4 = AAR Hall (H), -135°*

*Test 5 = Test 1*

*Test 6 = Huron Hall (H), -135°*

*Test 7 = AAR Large Room (LR), 45°*

The order in which the tests were presented was given no significance. As seen, *Test 1* and *Test 5* are used as a pseudo pair to give a hint of the test subject's reliability and of possible bias. Before each subject started performing the tests I carefully confirmed that the task was properly understood.

## Result

Table 4shows the results from the test regarding the naturalness of the acoustic environment.

| | Small Room | | Large Room | | Hall | | Pseudo Pair | |
|---|---|---|---|---|---|---|---|---|
| | AAR SR | Huron SR | AAR LR | Huron LR | AAR HR | Huron HR | AAR SR 1 | AAR SR 2 |
| Subject 1 | 1 | 0 | -1 | 2 | 0 | -2 | 1 | 1 |
| Subject 2 | -1 | 1 | 1 | -1 | -1 | 1 | -1 | -2 |
| Subject 3 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | -1 |
| Subject 4 | 1 | 1 | -1 | -1 | -1 | -2 | 1 | 1 |
| Subject 5 | 1 | 0 | 1 | -2 | 0 | -2 | 1 | 0 |
| Subject 6 | -2 | -2 | 1 | 1 | -2 | 1 | -2 | 2 |
| Subject 7 | -1 | -1 | -2 | -1 | 1 | -1 | -1 | 1 |
| Subject 8 | 1 | 2 | 2 | 2 | -1 | 1 | 1 | 2 |
| Subject 9 | 2 | 1 | 2 | 1 | -1 | 1 | 2 | 1 |
| Subject 10 | 1 | 1 | 1 | -1 | -1 | -1 | 1 | -1 |
| Subject 11 | 1 | 1 | -1 | -2 | 1 | 1 | 1 | 2 |
| Subject 12 | -1 | 1 | 2 | -1 | -1 | -2 | -1 | -1 |
| **SUM** | **4** | **7** | **7** | **-2** | **-5** | **-4** | **4** | **5** |

**Table 4** *Naturalness of the acoustic environment.*

In the test of the naturalness of the sound I asked the subjects to judge whether they thought the acoustic environment seemed *Very Natural, Natural, Synthetic* or *Very Synthetic*. The option *Can't Decide* was also avalible. In Table 4 the responses are mapped according to the following scheme:

*Very Natural = 2*

*Natural = 1*

*Can't Decide = 0*

*Synthetic = -1*

*Very Synthetic = -2*

As seen in table 4 the small room comparison adds up to +7 for the Huron machine and +4 for the AAR system. The response range is going over the full spectra, from –2 (*Very Synthetic*) to +2 (*Very Natural*). It seems the Huron machine produced more natural sounding acoustics in the small room simulation, but since the answers given are so spread out the result seem rather questionable. The large room simulation produced some perculiar results. The Huron sum is –2 while the AAR system scores a +7. Again the answers range from –2 to +2. The hall simulation shows some not very flattering values for both of the systems. –5 for the AAR and –4 for the Huron. Not a single +2 was given to either system that indicates all test subjects perceived some quite non-natural acoustics in this test. The final comparison is the pseudo pair which turned out with rather similar sums, namely +4 and +5. Looking at the response ranges we see that they also go from –2 to +2, even though the majority of the answers lies on +1.

The externalisation test results are shown in Table 5.

| | Small Room | | Large Room | | Hall | | Pseudo Pair | |
|---|---|---|---|---|---|---|---|---|
| | AAR SR | Huron SR | AAR LR | Huron LR | AAR LR | Huron LR | AAR SR 1 | AAR SR 2 |
| Subject 1 | 1 | -1 | 1 | -1 | -1 | -1 | 1 | 1 |
| Subject 2 | 1 | 1 | 1 | 1 | 1 | -1 | 1 | -1 |
| Subject 3 | -1 | -1 | 1 | 1 | -1 | 1 | -1 | 0 |
| Subject 4 | 1 | 1 | -1 | 1 | 1 | -1 | 1 | -1 |
| Subject 5 | 0 | 1 | 1 | -1 | 1 | 0 | 0 | 1 |
| Subject 6 | 1 | 0 | -1 | -1 | -1 | -1 | 1 | -1 |
| Subject 7 | 1 | -1 | 1 | -1 | 1 | 1 | 1 | 1 |
| Subject 8 | -1 | 0 | 1 | -1 | 0 | 1 | -1 | 1 |
| Subject 9 | 0 | -1 | 0 | 1 | -1 | 1 | 0 | -1 |
| Subject 10 | -1 | 1 | -1 | -1 | 1 | -1 | -1 | -1 |
| Subject 11 | 1 | 1 | -1 | 1 | -1 | -1 | 1 | -1 |
| Subject 12 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| **SUM** | **3** | **2** | **2** | **0** | **1** | **-1** | **3** | **-1** |

**Table 5** Externalisation test results.

The externalisation test question asked the subject to decide whether a sound localisation appeared to be *Inside Head* or *Outside Head*. Again the option *Can't Tell* was given. All the sounds were as, previously told, simulated to be outside the head at different distances for the different environmental simulations. The responses map to the table values according to the following:

*Inside Head = 1*

*Outside Head = -1*

*Can't Tell = 0*

Looking at table 5, the small room simulation adds up to +3 for the AAR system and to +2 for the Huron system. We can see that there is just a small favour for the *Outside Head* option for both the systems. In the large room case we also see a small tendency against the outside head option for the AAR system while the Huron produced an answer sum of 0. The hall simulation, which in the naturalness test was perceived as rather poor acoustics in both systems, proved to be perceived as *Inside Head* by, almost, an average of 50% of the test subjects in both tests. Finally the pseudo test shows that this test reliability might be questioned. Only 2 subjects responded with the same answer at this dummy test.

Testing the directivity was not the primary goal of this test. Adding reverb and other effect do affect the ability to determine a sound source direction. In a large room the sound source can be quite tricky to narrow down to an exact direction and this shows in the test results. Again, concerning the directivity and 3D qualities of the system the implemented application (Chapter 5), including the head tracking capabilities of the system, should be a better indicator. The results of the directivity test is shown in Table 6.

| | Small Room | | Large Room | | Hall | | Pseudo Pair | |
|---|---|---|---|---|---|---|---|---|
| | AAR SR | Huron SR | AAR LR | Huron LR | AAR HR | Huron HR | AAR SR | AAR SR |
| Subject 1 | 25 | -25 | 125 | 25 | 180 | 0 | 25 | -35 |
| Subject 2 | 0 | 0 | 55 | 45 | -90 | 180 | 0 | 0 |
| Subject 3 | 0 | 45 | 45 | 0 | -45 | -90 | 0 | 45 |
| Subject 4 | 135 | -90 | 35 | 0 | -135 | -90 | 135 | 55 |
| Subject 5 | 0 | 0 | 0 | 90 | 0 | No Answer | 0 | 0 |
| Subject 6 | 45 | 0 | 0 | 0 | 0 | -90 | 45 | 70 |
| Subject 7 | 25 | -45 | 35 | No Answer | 50 | 0 | 125 | 0 |
| Subject 8 | 0 | 0 | 90 | 0 | No Answer | 45 | 0 | -45 |
| Subject 9 | 0 | 160 | 0 | 90 | 180 | 0 | 0 | 0 |
| Subject 10 | 35 | 0 | 45 | 20 | 0 | 180 | 35 | -35 |
| Subject 11 | 0 | 0 | 0 | 90 | -70 | No Answer | 0 | 0 |
| Subject 12 | -45 | 0 | 180 | 0 | 0 | No Answer | -45 | 0 |
| **Correct value** | **0** | **25** | **35** | **65** | **-135** | **135** | **0** | **0** |

**Table 6** *Directivity test.*

From the test results in Table 6 we see that the range of answers are sometimes widely spread. My estimations of the graphical responses from the test subjects are quite rough. For me the interest lies in what quadrant they perceive that the sound is coming from.

The small room simulation shows that only test subject 4 deviated from the correct quadrant. It was obviously quite easy for most to tell the right direction of the sound in a small room simulation on both systems. For the large room version the response values still are rather good, nearly all test subjects are placing the sound direction in the right quadrant even though the spread is a bit larger than for the small room simulation. The large room simulation produced responses ranging over the whole azimuth. This was, as previously mentioned, anticipated. The pseudo pair test shows that almost every test subjects are consistent in determining the right quadrant of the sound source direction.

### Conclusion of the evaluation

The spatial sound quality test that I have performed leaves a lot to refine and evaluate closer. One might question whether any significant conclusions may be drawn since almost all tests show such a high variance. My objective was to get a hint on whether my AAR system, based on relatively inexpensive hardware, could stand a chance against a more sophisticated system such as the Huron. The conclusion is that it does. When it comes to real time rendering of complex acoustical environments the Huron is of course outstanding, but for less complicated environments my system clearly stand a chance of producing quality acoustical effects and positional sounds.

For the test I would like to point out some of the major weaknesses that I have recognised. First of all I experienced some problems in communicating the questions to some of the test subjects. The *Inside/Outside Head* question for example raised discussions on precisely what I was asking for. Maybe a small education on different sounds, such as just pointing out the sensational difference between a stereo and mono recording, would have helped.

Another problem might have been the fact that different sounds where played on the AAR and the Huron in the same simulations. For example, the violins higher frequency range compared to a man reading a text might affect the rendered simulation in an undesirable way through emphasising the early reverb parts of the acoustical effect.

A third possible flaw is the existence of “bad” listeners. Subjects should maybe first have been evaluated by a preliminary subjective test for checking their capability to give valid spatial sound responses to very evident modifications of the sound field. According a study performed by Farina and Ugolotti [23] only about 1/3 of their test subjects passed such a test, and was found suitable in a spatial audio quality test. When conducting such a preliminary selection test it was also revealed that there were a certain number of subjects who gave inconsistent responses and if included in any test, their “random” responses could degrade the overall confidence of the data.

# 5.0 Applications

After implementing and briefly evaluating the AAR system, see Chapter 4, my aim was to set up a couple of augmented audio reality applications in order to further test my system as well as try to explore some of the potential of audio only AR. The first two applications were developed to further evaluate the 3D audio capacity of the system. The third application is an attempt to construct a typical museum situation. A user is approaching a piece of art and will be presented with cues and stories that interact with the beholders movements and head orientation. The audio Pac Man application was unfortunately not implemented due to the limited range of the tracking device, but is worth mentioning since the concept is inspiring. In evaluating these applications I did not conduct any formal tests. This was due to limited time. I did however test them on a number of person, giving me a pretty good idea of how well they worked and what could be improved.

## Clock

The first application is rather trivial. I created an application that tells the time through a 3D audio interface. The user actively asks for the time, in this case by typing `time` in a command shell. Upon this command a 3 second "Tick, tack" sound moves clockwise in the azimuth, from approximately 2 hours back, to the position of the current time. Another second "coo-cuu" sound tells the time by being played at the appropriate position. 12 o'clock is right in front of the listener.

With some training I could tell the time with the accuracy on the hour.

## Game- "Draw the sound"

The idea behind this application is simply to follow the sound you hear on a sheet of paper or on a white board in front. This application was designed to test the 3D properties of the AAR system. The sound the user should follow is either a short "beep" played with a frequency of once every other second or the same "beep" played continuously.

Evaluating this application I soon realised that trying to draw a sound was rather tricky. The first try usually was very confusing for the test subject. Either to slow or to fast speed of the sound was a common complaint. I tried tree different drawings; a circle, a horisontal line and a diagonal line. After trying a couple of times with some subjects I could conclude that this application did not turn out very well. My system did not perform, for some of the subjects, much better than a stereo encoding would have. One subject got a chance to train with me drawing the different paths and then try to tell which was which. This turned out to be more successful. I leave any deeper interpretations of this test open since I believe the lack of data together with the lack of a formulated strategy in how to perform the test would make it rather speculative.

## Virtual museum narrator

This scenario implements a narrator or guide. Imaging a museum where you walk around and stop in front of a piece of art. A virtual guide not just tells you facts about the piece, but can actually also interact with it. It might ruin your experience or it might just be the difference to make a mediocre painting worth seeing. The narrator can move around within the picture, he can move around the picture or just tell the story of it. Different locations in the exhibition hall may be attached with different ambiances. In Figure 11 the beholder is exposed to different soundscapes depending on his whereabouts.

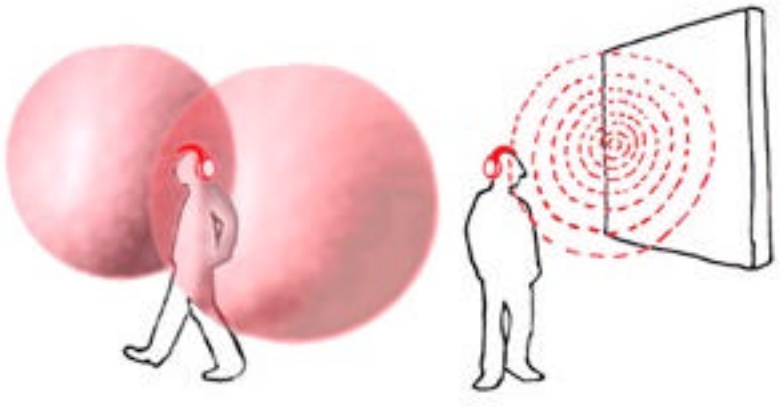

**Figure 13** *The user of the AAR system is exposed to different soundscapes depending on his location.*

This application is implementing a virtual guide for a 3x5 meters print of a famous photograph. The photograph is portraying construction workers having a lunch break on a steel bar on the not yet completed Empire State Building.

The soundscape is including men chatting, the traffic below and the wind blowing. Depending on the whereabouts of the user a narrator tells stories about the picture.

The sound design was created using the EAGLE editor. The range of the FasTrack limited the physical space in which to move around to a 2x2 meters square. Different acoustical environments were created which interacted with the movements of the user. Different user positions triggered different audio events to take place.

This application turned out rather well, providing some hints on the potential of spatialised audio to augment an environment. The different acoustical zones made it possible to hint to the user what to expect if moving in a certain direction. Playing the same sounds in different sound zones gave a sense of moving in to the picture. Standing 2 meters away from the picture moving up to 1 meter and changing the acoustics from a small room to being outside made an impression of entering the picture. Getting the attention of the beholder to look at a particular object by first hinting in what direction to look before describing the object to closely enhanced the sense of participation with the picture.

## Audio Pac Man

Pac Man is a famous arcade game where ghosts hunt you while you are supposed to collect cookies. A Pac Man game in audio build on the same principal; monsters moving around in defined tracks where the player must try to collect items. Different acoustical spaces represents “safe zones” or give hints on where items can be found. As previously mentioned was the implementation of the Audio Pac Man left out due to hardware constraints.

# 6.0 Conclusions

## Conclusions

As seen in the related work chapter the spatial aspect of audio is not very well explored in augmented realities. There exist surprisingly few attempts of utilising AUI's in real life environments. This fact together with new cheap off-the-shelf technologies make audio augmented reality a very exciting area to delve into. The aim of this thesis has been to implement a low cost audio augmented reality prototype system and to implement and explore some applications. The result is the AAR system, providing a full framework to design and create spatial audio applications. Using the AAR system, three test applications have been implemented were a museum tour guide turned out especially well.

The three corner stones of the system are the listener, the emitters and their environment. The rendering of the 3D audio and the environmental acoustics is based on the DirectSound3D and the EAX API's. A tracking device provides the positional data of the listener and all audio is play-backed in real-time. The ability to easily create and manipulate these entities is provided through an included interface to the EAGLE software. The porting of the AAR system on to a Linux platform is doable if an API such as the OpenAL is used instead of DirectSound3D. In an application a listener moves around freely in a space where different locations, such as physical rooms or parts of rooms, provide boundaries between different acoustic landscapes. When changing location the listener experiences a morphing between acoustic sceneries. Different audio objects, emitters, are placed within these environments and can be made to interact with the listener based on his location.

In evaluating the implementation of the AAR system I considered some general application level system performance issues and spatial sound quality. Running the AAR system on a Pentium III allows up to eight sound sources with environmental effects and 3D audio playing simultaneously. The spatial sound quality test that I performed leaves a lot to refine and evaluate closer. My objective was to get a hint on whether my AAR system, based on relatively inexpensive hardware, could stand a chance against a more sophisticated system such as the Huron. The conclusion is that it does.

The implemented applications turned out rather differently in terms of success. My testing, trimming and evaluation of the test applications did suffer from a limiting time schedule. Unfortunately not enough time could be spent on developing and evaluating my implemented applications. Yet did the virtual museum narrator application turn out well, providing some hints on the potential of spatialised audio in augmented environments.

## Future work

As a summary of the entire project I may conclude that the fascinating topic of spatial audio reproduction, rather brutally implemented in the AAR system, spans over a vast spectrum of disciplines. The most direct future work would be to more closely design and evaluate the different test applications. Further, parts of the AAR system are missing proper documentation, as well as an overall users manual. A next step in developing the AAR system would also involve documenting it thoroughly. Listed below are some other areas of future work related to my thesis:

- Development of hardware independent acoustical rendering software.
- Definition and set up of standards for evaluating spatial audio quality.
- Integration of a wireless tracking device, preferably based on a non-electromagnetic solution.
- Implementation of the AAR system in a VR cube.
- Further investigation and to some extent implementation of individual HRTF modelling.

# 7.0 References

[1] Burgess, D.A. *Techniques for low cost spatial audio*. Proceedings of the Fifth Annual Symposium on User Interface Software and Technology (UIST '92), ACM, New York, 1992, pp. 53-59.

[2] Begault, D. R., *3-D Sound for Virtual Reality and Multimedia*, Academic Press Ltd., Cambridge, Massachusetts, 1994

[3] Blauert, J. *Spatial Hearing: The Psychophysics of Human Sound Localization*, MIT Press. Cambridge, MA. 1983.

[4] Searle, C. L., Braida, L. D., Davis, M. F. and Colburn, H. S. *Model for auditory localization*, J. Acoust. Soc. Am., 60, 1164 – 1175, 1976.

[5] Kyriakakis, C. *Fundamental and Technological Limitations of Immersive Audio Systems.* Proceedings of the IEEE. Vol. 86. No. 5 May 1998.

[6] Minnaa P., Krarup Olesen S., Christensen F. and Møller H. *The Importance of Head movements for Binaural Room Synthesis*. Proceedings of the 2001 International Conference on Auditory Display, Espoo, Finland, July 29-August 1, 2001

[7] Kuttruff, H. *Room Acoustics*, Third Edition, Elsevier Science Publishing, England, 1991.

[8] Shinn-Cunningham,B.G.(2000a*). Distance cues for virtual auditory space*. Proceedings of the IEEE-PCM 2000, Sydney,Australia.

[9] Savioja, L. *Modeling Techniques for Virtual Acoustics*. Doctoral thesis, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, Report TML-A3, 1999.

[10] Funkhouser, T. A., Carlbom, I., Elko, G., Pingali, G., Sondhi, M. and West, J. *A Beam Tracing Approch to Acoustic Modeling for Interactive Environments*. Computer Graphics (SIGGRAPH '98), Orlando, FL, July, 1998, 21-32.

[11] IASIG, *Interactive 3D Audio Rendering Guidelines Level 2*, MIDI manufactures Assosiation, L.A, CA, 1999

[12] Kleiner, M., Dalenbäck B.I, and Svensson, P., *Auralization - an Overview*, Journal of the Audio Engineering Society, vol. 41(11), pp. 861-875, 1993.

[13] Sawheny, N. and Schmandt, C. *Normadic Radio: Speech and Audio Interaction for Contextual Messaging in Normadic Environments*. 2000. ACM Transactions on Human Computer Interface (ToCHI'2000). Journal Paper, vol. 7 pages 353 – 385.

[14] Kyriakakis, C., Tsakalides, P., Holman, T. *Surrounded by Sound.* IEEE Signal Processing Magazine Published: Jan. 1999 Volume: 16 1 , Page(s): 55 –66

[15] Pulkki, V. *Spatial Sound Generation and Perception by Amplitude Panning Techniques*. PhD thesis, Helsinki University of Technology, Espoo, Finland, 2001.

[16] Benderson, B. *Audio Augmented Reality: A Prototype Augmented Tour Guide*. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95). May 1995, Pages 210 - 211.

[17] Lyons K., Gandy M., and Starner T. *Guided by voices: An audio augmented reality system.* Proceedings of the International Conference on Auditory Display (ICAD'00), Atlanta, GA, April 2000

[18] Rozier, J., Karahalios K. and Donath J. *Hear&There: An Augmented Reality System of Linked Audio*. Proceedings of the International Conference on Auditory Display (ICAD'00) April, 2000

[19] Mynatt, E., Back, M., Want, R Baer, R., and Ellis, J. *Designing Audio Aura*. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'98). 1998. ACM, New York, Pages 566 – 573.

[20] S.Bech, "Methods for subjective evaluation of spatial characteristics of sound" in Proc. AES 16th int conf on Spatila Sound Reproduction, (Rovaniemi, Finland) AES, April 1999

[21]V. Pulkki, M. Karjalainen, J Houpaniemi "Analysing Virtual Sound Source Attributes Using a Binaural Auditory Model", J. Audio Eng Soc. Vol. 47, No. 4 1999 April

[22] Lokki, T. and Järveläinen, H., "Subjective Evaluation of Auralization of Physics-based Room Acoustics Modeling" In Proc. 7th International Conference on Auditory Display (ICAD'2001), pp. 26-31, Espoo, Finland, July 29 - August 1, 2001.

[24] A. Farina, E. Ugolotti – "Subjective evaluation of the sound quality in cars by the auralisation technique" – Proc. of 4th International Conference and Exhibition Comfort in the automotive industry - Bologna (Italy) October 2-3, 1997.

# Appendix A – Header files

## client.dll

NOTE: The" `extern "C" __declspec(dllexport)`" declarations are not included in order to improve the readability.

```
#include "NetworkCommands.h"

/***  Defines to use in combination with emitter creation   ***/

typedef struct
{
long nType;                          //why the packet was send

char zPortNumber[PORTSIZE_MAX];    //used in connection with INI_ANSWER
                                   //contains the port number for the
                                   //established data connection

char zUserName[STRINGLENGTH_MAX];   //used by the client to tell the server
                                    //the user's name

char zHostName[STRINGLENGTH_MAX];   //used by the client to tell the server
                                   //the client computer's hostname

char zIpNumber[STRINGLENGTH_MAX];   //used by the client to tell the server
                                   //the client computer's IP address
} TCP_INIT_DATA;

// TCP/IP data packet
//
#define TCP_HEADSIZE ( 2 * sizeof ( long ) )

typedef struct
{
  float x;
  float y;
  float z;
} Vector3d;

typedef struct
{
    long _length;
    long _type;
   char  _data[DATASIZE_MAX];

} _TCP_PACK;

typedef _TCP_PACK;

// Types used by TCP_INIT_DATA
#define INI_CONTACT              0  //send by the client to the
                                    //server to request a data connection

#define INI_ANSWER               1  //send by the server as response
                                    //to a clients request for a data
                                    //connection
#define SERVER_ERROR          9999

//
// Polhemus Fasttrack
//
#define MAX_TRACKER     4   /* number of supported tracking devices */
#define LISTENER      255   /* targetID for listener                */
                            /* 255 is the error code for            */
                            /* creating emitters                    */
enum kind {Orientation, Position, PositionOrientation };

/*** orthoVectors are used to select the hemispheres of operation ***/
enum orthovector {Xpos, Xneg, Ypos, Yneg, Zpos, Zneg};
```

```
//Preset of aucostict enviroments
enum enviro {
GENERIC,
PADDEDCELL,
ROOM,
BATHROOM,
LIVINGROOM,
STONEROOM,
AUDITORIUM,
CONCERTHALL,
CAVE,
ARENA,
HANGAR,
CARPETEDHALLWAY,
HALLWAY,
STONECORRIDOR,
ALLEY,
FOREST,
CITY,
MOUNTAINS,
QUARRY,
PLAIN,
PARKINGLOT,
SEWERPIPE,
UNDERWATER,
DRUGGED,
DIZZY,
PSYCHOTIC
};

typedef struct {
  double     Xoffset;
  double     Yoffset;
  double     Zoffset;
  double     Xrecent;
  double     Yrecent;
  double     Zrecent;

  double     Azimut_offset;
  double     Elevation_offset;
  double     Roll_offset;
  double     Azimut_recent;
  double     Elevation_recent;
  double     Roll_recent;

  double     delta1;
  double     delta2;
  short      targetID;
  enum kind  tracking;
  double     scaling;
} tracker_data;

  int     Clientconnect ( char* hostname, int port, char* username );
  void    Clientdisconnect ( void );

  int     initialize_audio ( void );
  int     AARreverbation_on ( void );
  int     AARreverbation_off ( void );
  int     AARreverbation_env ( const enum enviro envId );
   int     AARreverbationSetParameters ( const float decay,
                                         const float intensity );
  int     AARreverbationSetAllParameters( long lRoom ,long lRoomHF,
                                          float flRoomRolloffFactor,
                                          float flDecayTime,
                                          float flDecayHFRatio,
                                          long lReflections,
                                          float flReflectionsDelay,
                                          long lReverb,
                                          float flReverbDelay,
                                          unsigned long dwEnvironment,
                                          float flEnvironmentSize,
                                          float flEnvironmentDiffusion,
                                          float flAirAbsorptionHF,
                                          unsigned long dwFlags);
```

```
  int    setEAXEmitterProperties(unsigned char nEmitterId, long lDirect,
                                 long lDirectHF, long lRoom, long lRoomHF
                                 float flRoomRolloffFactor, long lObstruction,
                                 float flObstructionLFRatio, long lOcclusion,
                                 float flOcclusionLFRatio,
                                 float flOcclusionRoomRatio,
                                 long lOutsideVolumeHF,
                                 float flAirAbsorptionFactor,
                                 unsigned long dwFlags );

  int     AARiniteaxman_source (int emitterID  );
  int     AARsourceeaxman_material ( int emitterID, char* preset);
  int     AARseteaxman_env ( char* filename );
  int     AARsource_eax (int emitterID, Vector3d* position  );
  int     AARloadEAL_file ( char* filename );
  int     AARlistener_eaxman  ( Vector3d* position );
  int     AARconnect_listener ( void );
  int     AARdisconnect_listener ( void );
  int     AARset_listener_position ( Vector3d* position );
  int     AARset_listener_orientation ( Vector3d* position, Vector3d* upward );
  int     AARcreate_emitter ( char* name,
                              Vector3d* position, Vector3d* orientation,
                              float minFront, float minBack,
                              float maxFront, float maxBack,
                              float intensity, char* audiofile,
                              unsigned int groupID );

  int     release_emitter ( int emitterID );
  int     release_multiple_emitters ( int* emitterList, int length );
  int     start_emitter ( int emitterID );
  int     stop_emitter ( int emitterID );
  int     mute_emitter ( int emitterID );
  int     unmute_emitter ( int emitterID );
  int     pause_emitter ( int emitterID );
  int     resume_emitter ( int emitterID );
  int     set_emitter_intensity ( int emitterID, float intensity );
  int     set_emitter_pitch ( int emitterID, float pitch );
  int     set_emitter_position ( int emitterID, Vector3d* position );
  int     set_emitter_orientation ( int emitterID, Vector3d* orientation );
  int     set_emitter_model ( int emitterID, float minFront, float minBack,
                              float maxFront, float maxBack, float intensity );
  int     set_emitter_min_front ( int emitterID, float minFront );
  int     set_emitter_min_back ( int emitterID, float minBack );
  int     set_emitter_max_front ( int emitterID, float maxFront );
  int     set_emitter_max_back ( int emitterID, float maxBack );
  int     change_sounddirectory ( const char* directory );
  char**  get_soundfile_list ( void );
  int     read_scenario ( const char* scenario );
  int     set_CPU_Budget ( const enum RSX_CPU_Budget budget );
  int     submitAudioBuffer( short emitterID, char* audio_data,
                             unsigned int audio_length,
                              unsigned int samplingRate );

/*** Polhemus function prototypes ***/
  int  PolhemusInit ( const char* portName, const DWORD baudrate,
                      const BYTE bytesize,
                     const BYTE parity, const BYTE stopbits );
  int  PolhemusConnectToTarget ( const short targetID,
                                 const enum kind tracking, const double scaling
);
  int  PolhemusDisconnectTarget ( const short targetID );
  int  PolhemusDisconnectTracker ( const int tracker_num );
  int  PolhemusStartTracking ( void );
  void PolhemusStopTracking ( void );
  int  PolhemusSetZero ( const int tracker_num );
  int  PolhemusShutdown ( void );
  void PolhemusUpdateTreshold ( const double position, const double angles );
  int  PolhemusSetHemisphere ( const short targetID,
                               const int   tracker_num,
                               const enum  orthovector dir);

/*** local declared functions ***/

int       create_connected_socket ( char* hostname, int port );
int       create_connected_UDPsocket ( void );
```

```
void      disconnect_socket ( int* asocket );
char*     receive ( long* tag, int* message_length );
int       checkReceiver ( void );
int       send ( const _TCP_PACK* command );
int       sendData ( const _UDP_PACK* command );
int       initialize ( char* username );
char**    filename_array ( char* list, int length );
char*     tag_def ( long tag );

/*** serial port and polhemus fkt's ***/

void      initDataStructure ( void );
void      UpdatebyPolhemus ( int num );
int       getFreeTrackerID ( void );
void      deleteTrackerfromTarget ( const short emitterID );
int       rs232open ( const char* portName, const DWORD baudrate,
                      const BYTE bytesize, const BYTE parity,
                      const BYTE stopbits );
int       rs232send ( const char* data, const int length );
int       rs232recv ( char* data, const unsigned long length );
void      rs232down ( void );
```

## Network commands

```
#define LOGOUT                            1
#define INITIALIZE_AUDIO                  1000
#define REVERBATION_ON                    1001
#define REVERBATION_OFF                   1002
#define REVERBATION_SET_PARAMETERS        1003
#define REVERBATION_ENV                   1005
#define REVERBATION_SET_ALL_PARAMETERS    1006
#define SET_EAX_BUFFER_PROPERTIES         1007
#define LOAD_EAX_ENV                      1008
#define LISTENER_EAX_MANAGER              1009
#define SET_EAX_MANAGER_ENV               1010
#define INIT_SOURCE_EAX_MANAGER           1011
#define SET_SOURCE_EAX_MANAGER_MATERIAL   1012
#define SET_SOURCE_EAX_MANAGER            1013
#define CONNECT_LISTENER                  1103

#define DISCONNECT_LISTENER              1104
#define SET_LISTENER_POSITION            1105
#define SET_LISTENER_ORIENTATION         1106
#define GET_LISTENER_POSITION            1107
#define GET_LISTENER_ORIENTATION         1108
#define CREATE_EMITTER                   1200
#define RELEASE_EMITTER                  1201
#define RELEASE_MULTIPLE_EMITTERS        1202
#define START_EMITTER                    1203
#define STOP_EMITTER                     1204
#define MUTE_EMITTER                     1205
#define UNMUTE_EMITTER                   1206
#define PAUSE_EMITTER                    1207
#define RESUME_EMITTER                   1208
#define SET_EMITTER_INTENSITY            1209
#define SET_EMITTER_PITCH                1210
#define SET_EMITTER_POSITION             1211
#define SET_EMITTER_ORIENTATION          1212
#define SET_EMITTER_MODEL                1213
#define SET_EMITTER_MIN_FRONT            1214
#define SET_EMITTER_MIN_BACK             1215
#define SET_EMITTER_MAX_FRONT            1216
#define SET_EMITTER_MAX_BACK             1217
#define GET_EMITTER_INTENSITY            1218
#define GET_EMITTER_PITCH                1219
#define GET_EMITTER_POSITION             1220
#define GET_EMITTER_ORIENTATION          1221
#define GET_EMITTER_MODEL                1222
#define GET_EMITTER_MIN_FRONT            1223
#define GET_EMITTER_MIN_BACK             1224
#define GET_EMITTER_MAX_FRONT            1225
#define GET_EMITTER_MAX_BACK             1226
#define GET_EMITTER_SOUND                1227
#define GET_EMITTER_STATUS               1228
#define SUBMIT_AUDIO_DATA                1229
#define CHANGE_SOUNDDIRECTORY            1300
#define GET_SOUNDFILE_LIST               1301
#define GET_EMITTERLIST                  1302
#define READ_SCENARIO                    1303
```

## server.h

```
#include "soundInterface.h"

class 3dAudioServer
{
public:
  fsa3dAudioMfcServer( int port,3dAudioInterface  & );   // standard
constructor

virtual ~fsa3dAudioMfcServer();                          // standard destructor

void handleEvent( Event & rEvent );

void setSuccessor( fsaChainOfResponsibilityMember *  pSuccessor );

int lowestClientId( void )
throw();

int clientCount( void )
         throw();

void lostClient( 3dAudioServerClient *    pLostClient )
         throw();

void deleteClient( 3dAudioServerClient *  pClientToDelete )
         throw();

void deleteAllClients( void )
         throw();

void addClient( 3dAudioServerClient * )
         throw();

void receivedTcpData( 3dAudioServerClient *, TCP_DATA * )
         throw();

protected:

//
// Attribute control methods
//

bool _clientExists( 3dAudioServerClient *    pClient )
         throw();

3dAudioServerClient * _client( 3dAudioServerClient *    pClient )
         throw();

//
// Event handling
//
void _delegateEvent( Event &    rEvent )
         throw();

//
// Message methods
//

void _displayStatusMessage( const char *    zFormatString )
         throw();

void _displayExceptionMessage( const fException &  rException )
         throw();
```

```
private:
//
// Audio interface methods
//

inline void initializeAudio()
        throw();
//
// Listener control methods
//

void connectListener()
        throw();

void disconnectListener()
        throw();

void setListenerPosition( TCP_DATA * );

void listenerPosition( 3dAudioServerClient * );

void setListenerOrientation( TCP_DATA * )
        throw();

void listenerOrientation( 3dAudioServerClient * )
        throw();

   //
   // Emitter control methods
   //

  void createEmitter( 3dAudioServerClient *,
                      TCP_DATA * )
        throw();

  int createEmitter( string      zSoundDirectory,
                     TCP_DATA *  pTcpPackage )
        throw();

  int createStreamingEmitter( TCP_DATA *  pTcpDataPack )
        throw();

  void releaseEmitter( fsa3dAudioMfcServerClient *  pClient,
                       unsigned char                nEmitterId )
        throw();

  void releaseMultipleEmitters( fsa3dAudioMfcServerClient *  pClient,
                          TCP_DATA *                       pData )
        throw();

  void startEmitter( unsigned char )
        throw();

  void stopEmitter( unsigned char )
        throw();

  void pauseEmitter( unsigned char )
        throw();

  void resumeEmitter( unsigned char )
        throw();

  void muteEmitter( unsigned char )
        throw();

  void unmuteEmitter( unsigned char )
        throw();

  void setEmitterPosition( TCP_DATA* )
        throw();

  void emitterPosition( fsa3dAudioMfcServerClient*,
                        unsigned char )
        throw();
```

```
  void setEmitterOrientation( TCP_DATA* )
        throw();

  void emitterOrientation( 3dAudioServerClient*,
                           unsigned char )
        throw();

  void setEmitterIntensity( TCP_DATA* )
        throw();

  void emitterIntensity( 3dAudioServerClient*,
                         unsigned char )
        throw();

  void setMaxFront( TCP_DATA* )
        throw();

  void maxFront( 3dAudioServerClient*,
                 unsigned char )
        throw();

  void setMinFront( TCP_DATA* )
        throw();

  void minFront( 3dAudioServerClient*,
                 unsigned char )
        throw();

       void
   setMaxBack( TCP_DATA* )
        throw();

       void
   maxBack( 3dAudioServerClient*,
            unsigned char )
        throw();

       void
   setMinBack( TCP_DATA* )
        throw();

       void
   minBack( 3dAudioServerClient*,
            unsigned char )
        throw();

       void
   setEmitterModel( TCP_DATA* )
        throw();

       void
   emitterModel( 3dAudioServerClient*,
                 unsigned char )
        throw();

       void
   setEmitterPitch( TCP_DATA* )
        throw();

       void
   emitterPitch( 3dAudioServerClient*,
                 unsigned char )
        throw();
 //
 // Environment control methods
 //

       void
   reverbationOn()
        throw();

       void
   reverbationOff()
        throw();
```

```
     void
   reverbation_Env(TCP_DATA *   pTcpPackage)
         throw();

     void
   set_reverbation_parameters( TCP_DATA *   pTcpPackage )
    throw();

      void
  set_all_reverbation_parameters( TCP_DATA *   pTcpPackage )
      throw();

    void
  set_EAX_buffer_properties( TCP_DATA * pTcpPackage)
   throw();

   //
   // Sound file control methods
   //
        void
   changeSoundDirectory( 3dAudioServerClient *  pClient,
                         TCP_DATA *                  pData )
         throw();

        void
   soundfiles(3dAudioServerClient *   pClient )
         throw();

  //AudioData
        void
   receivedAudioData( UDP_DATA* )
         throw();

  //Scenario
        int
   readScenario( 3dAudioServerClient *    pClient,
                 TCP_DATA *                   pData )
         throw();

// EAX Manager
   void
  load_EAX_environment(TCP_DATA *  pTcpData )
   throw();

   void
  listener_EAX_manager(TCP_DATA * pTcpData )
   throw();

   void
  setEAXManagerEnv( TCP_DATA *   pTcpPackage )
      throw();

   void
  initSourceEAXManager( TCP_DATA *    pTcpPackage )
      throw();

   void
  setSourceEAXManagerMaterial( TCP_DATA *    pTcpPackage )
      throw();

   void
  setSourceEAXManager( TCP_DATA *    pTcpPackage )
      throw();

private:

  fsaAbstract3dAudioInterface &        _rLocalAudioServer;

  CServerSocket *                     _pServerSocket;    // Mainsocket
  list<3dAudioServerClient*>          _oClientList;     // List of clients
  bool                                _bWaitForClient;  // Temp-client which
                                                        // will be prepared

};
```

## listener.h

```
#include "dsound.h"
#include "eax.h"
#include "EaxMan.h"

//----------------------------------------------------------------------------
// saEAXListener class
//

class saEAXListener
{
public:
   //
   // Construction and Destruction
   //

   saEAXListener(           const 3dVector &      rPosition
                          , const 3dVector &      rOrientation
                          , const 3dVector &      rUpOrientation
                          , LPDIRECTSOUNDBUFFER   pPrimaryBuffer
                          ,  LPDIRECTSOUND8       pDirectSound
                          ,  bool                 bConnect = true )
         throw();

  virtual ~saEAXListener()
     throw(fDoesNotExistException);

//
 // Interface
 //

         virtual
         void
   connect()
         throw();

         virtual
         void
   disconnect()
         throw();

         virtual
         void
   setPosition( const 3dVector &  rPosition )
         throw();

         virtual
         void
   setOrientation( const 3dVector &  rOrientation
                 , const 3dVector &  rUpOrientation )
         throw();

// Sets EAX listener Property Set
  void EAXListenerProps()
      throw();

// Sets EAXManager and keeps track and sets changes.
   Void
setEAXManListener(LPEAXMANAGER eaxManager, const 3dVector &  rPosition,
                  LPDIRECTSOUND3DBUFFER8 _lp3dBuffer)
   throw();

   void
saEAXListener::setEAXManListenerEnvironment(LPEAXMANAGER eaxManager,
                  char* envName, LPDIRECTSOUND3DBUFFER8 _lp3dBuffer)
   throw();
private:

LPDIRECTSOUND3DLISTENER8 _audioListener()
         throw();
```

```
   saEAXListener( const saEAXListener & );   // Not implemented


   //
   // Operators
   //
   //  Environment ID
  long lenvId;

public:

  LPDIRECTSOUND3DLISTENER8      _pListener;
  LPDIRECTSOUNDBUFFER8          _pListenerBuffer;
  LPKSPROPERTYSET               _pEAXListener;
  LPDIRECTSOUND3DBUFFER8        _pListener3dBuffer;
};
```

## emitter.h

```
#include "dsound.h"
#include "soundInterface.h"
#include "eax.h"
#include "eaxman.h"

//
// Forward Declarations
//
//--------------------------------------------------------------------------
// Emitter class
//

class Emitter {
public:

  Emitter()
   throw();

  Emitter( unsigned char               nId,
           string                      zName,
           const 3dVector &            rPosition,
           const 3dVector &            rOrientation,
           const AudioEmitterModel &   rEmitterModel,
           string                      zSoundFile,
           LPDIRECTSOUND8              lpEAXSound,
           unsigned int                groupID
           ,DirectXSync*               _pSyncronize
           );

    Emitter( const Emitter &   rAudioEmitter )
      throw();

         virtual
   ~Emitter()
         throw();

    //
    // Audio Emitter methods
    //

         void
    start()
         throw();

         void
    stop()
         throw();

         void
    pause()
         throw();

         void
    resume()
         throw();

         virtual
        void
    mute()
         throw();

         virtual
       void
    unmute()
         throw();

         void
    setPosition( const 3dVector &    rPosition )
         throw();
```

```
      void
   setOrientation( const 3dVector &    rOrientation )
      throw();

      void
   setIntensity( float  fIntensity )
      throw();

      void
   setPitch( float   fPitch )
      throw();

      void
   setEAXProperties( long lDirect,
    long lDirectHF,
    long lRoom,
    long lRoomHF,
    float flRoomRolloffFactor,
    long lObstruction,
    float flObstructionLFRatio,
    long lOcclusion,
    float flOcclusionLFRatio,
    float flOcclusionRoomRatio,
    long lOutsideVolumeHF,
    float flAirAbsorptionFactor,
    unsigned long dwFlags )
      throw(  );

   //
   // initEAXman
   // Sets the source preset through EAX Manager
   // Called from saEAXinterface::initSourceEAXman(...)
   //

   void
initEAXman(LPEAXMANAGER eaxManager, string preset)
   throw( fDirectXException );

   //
   // setEAXmanMaterial
   // Sets the source material preset through EAX Manager
   // Called from saEAXinterface::setSourceEAXmanMaterial(...)
   //

   void
setEAXmanMaterial(LPEAXMANAGER eaxManager, char* presetName)
   throw( fDirectXException );

   //
   // sourceEAXman
   // Gets update info from EAX Manager and applies this on source
   //
    void
sourceEAXman(LPEAXMANAGER eaxManager, const fsa3dVector &  rPosition)
   throw( fDirectXException );

      LPDIRECTSOUNDBUFFER8
   secondaryBuffer()
      const
      throw();

      LPDIRECTSOUND3DBUFFER8
   spacialBuffer()
      const
      throw();

public:
     void
   _cleanUp()
      throw();
```

```
public:

   BYTE *                _           _pSoundData;
   LPDIRECTSOUNDBUFFER8              _pSecondaryBuffer;
   LPDIRECTSOUND3DBUFFER8            _p3dBuffer;
   DirectXSync&                      _pSync();
   LPKSPROPERTYSET                   pEAXBuffer;
```

## soundInterface.h

```
#include "eax.h"
#include "eaxman.h"
#include “dsound.h"
#include "3dAudioLibrary/DirectXSync.h"
#include "listener.h"
#include "emitter.h"

class  SoundInterface
{

public:
  SoundInterface()
         throw();
        virtual
   ~SoundInterface()
         throw();

        void
   initializeAudio()
         throw();

        unsigned char
   createEmitter( string                          zEmitterName,
                  const 3dVector &                rPosition,
                  const 3dVector &                rOrientation,
                  string                          zSoundFile,
                  unsigned int                    groupID)
         throw();

   void
   reverbationOn()
         throw();

    void
  reverbationEnv(unsigned long envId)
   throw();

    void
  reverbationOff()
         throw();

   void
  reverbationSetParameters(const float & Decay,const float & intensity )
    throw();

  void
  loadEAXFile(string ealFile)
   throw();

  void
  listenerEAXman(const 3dVector &  rPosition)
   throw();

  void
initEAX(Emitter  *Emitter)
   throw();

  //
  // setEAXManEnv
  // Set, in EAX Manager, the environment you have created in EAGLE.
  // Done once for each env. preset.
  //
  void
  setEAXmanEnv(string envName)
  throw();

   //
   // initSourceEAXman
   // Sets the source preset through EAX MAnager
```

```
   //

  void
  initSourceEAXman(int emitterId)
  throw();

  void
  setSourceEAXmanMaterial(int emitterId, string presetName)
  throw();

  void
  sourceEAXman(int emitterId, const fsa3dVector &  rPosition)
  throw();

  // reverbationSetAllParameters
  // EAX feature
    void
  reverbationSetAllParameters(long lRoom ,long lRoomHF,float
flRoomRolloffFactor,
                              float flDecayTime,float flDecayHFRatio,
long lReflections,float flReflectionsDelay,
                              long lReverb,float flReverbDelay,
unsigned long dwEnvironment,float flEnvironmentSize,
                              float flEnvironmentDiffusion,
float flAirAbsorptionHF,unsigned long dwFlags)
   throw( fAudioNotInitializedException );

LPDIRECTSOUND8             _pEAXDirectSound;
  LPDIRECTSOUNDBUFFER       _pEAXPrimaryBuffer;
  DirectXSync*              _pSyncronize;
  saEAXListener*            _pAudioListener;
  EAXLISTENERPROPERTIES     EAXprops;
  LPEAXMANAGER              eaxManager;
  saEAXEmitter*             oEmitter;
  bool                      EAXinit;
};
```

# Appendix B – EAX buffer properties

## Table of primary buffer properties (listener)

| Property Name | Type | Range | Default | Description |
|---|---|---|---|---|
| Environment | DWORD | [0, 25] | 0 | |
| Environment size | FLOAT | [1.0, 100.0] | (*) meters | Apparent size of environment |
| Environment Diffusion | FLOAT | [0.0, 1.0] | (*) | Echo density in reverb. Decay |
| Room | LONG | [-10000, 0] | (*) mB | Master vol. for reflected snd. |
| Room HF | LONG | [-10000, 0] | (*) | HF atten. for reflected sound |
| Decay Time | FLOAT | [0.1, 20.0] | (*) seconds | Reverberation decay time |
| Decay HF Ratio | FLOAT | [0.1, 2.0] | (*) | Spectral quality of the above |
| Reflections | LONG | [-10000, 1000] | (*) mB | Controls initial reflections |
| Reflections Delay | FLOAT | [0.0, 0.3] | (*) seconds | |
| Reverb | LONG | [-10000, 2000] | (*) mB | |
| Reverb Delay | FLOAT | [0.0, 0.1] | (*) seconds | |
| Room Rolloff Factor | FLOAT | [0.0, 10.0] | 0.0 | Distance attenuation |
| Air Absorption HF | FLOAT | [-100.0, 0.0] | -5.0 mB/m | HF distance attenuation |
| Flags | DWORD | [0x0, 0x2F] | (*) | |
| Decay Time Scale | Flag bit | | TRUE | Scales DT with Env. Size |
| Reflections Scale | Flag bit | | TRUE | Scales DR with Env. Size |
| Reflections Delay Scale | Flag bit | | TRUE | Scales RefD with Env. Size |
| Reverb Scale | Flag bit | | TRUE | Scales reverb with Env. Size |
| Reverb Delay Scale | Flag bit | | TRUE | Scales RevD with Env. Size |
| Decay HF Limit | Flag bit | | (*) | Limits D.HF according to Airabs.HF |

## Table of secondary buffer properties (sound source)

| Property Name | Description | Type | Range | Default |
|---|---|---|---|---|
| Direct | Direct sound manual volume | LONG | [-10000, 1000] | 0 mB |
| Direct_HF | Direct sound manual HF volume | LONG | [-10000, 0] | 0 mB |
| Room | Reflected sound manual volume | LONG | [-10000, 1000] | 0 mB |
| Room_HF | Reflected sound manual HF vol. | LONG | [-10000, 0] | 0 mB |
| Obstruction | Obstruction muffling of direct snd. | LONG | [-10000, 0] | 0 mB |
| Obstruction_LF_ratio | Spectral quality of the above | FLOAT | [0.0, 1.0] | 0.0 |
| Occlusion | Occlusion muffling of direct sound | LONG | [-10000, 0] | 0 mB |
| Occlusion_LF_ratio | Spectral quality of the above | FLOAT | [0.0, 1.0] | 0.25 |
| Occlusion_Room_raio | Occlusion atten. for reflected snd. | FLOAT | [0.0, 10.0] | 0.5 |
| Room_rolloff_factor | Attenuates reflected sound | FLOAT | [0.0, 10.0] | 0.0 |
| Air_absorption_factor | High-frequency distance atten. | FLOAT | [0.0, 10.0] | 1.0 |
| Outside_volume_HF | Controls HF directivity attenuation | LONG | [-10000, 0] | 0 mB |
| Flags | | DWORD | [0x0, 0x7] | 0x7 |
| Direct HF Auto | Auto filtering for direct HF sound | Flag bit | | TRUE |
| Room Auto | Auto distance and directivity atten. | Flag bit | | TRUE |
| Room HF Auto | Auto filtering for directivity | Flag bit | | TRUE |