Binaural Audio for an Immersive Co-Located Multiplayer Environment

Christian Bartsch

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Oktober 2016

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, October 15, 2016

Christian Bartsch

Contents

Kurzfassung

Binaurales Audio oder wörtlich “Audio beide Ohren betreffend” beschreibt Methoden um realistische, dreidimensionale Sounds über Kopfhörer wiederzugeben. Dreidimensionale Sounds können das Erlebnis von virtuellen Umgebungen wie zum Beispiel Videospielen verbessern: Töne können aus allen Richtungen wahrgenommen werden; Bilder können nur direkt vor den Augen des Benutzers gesehen werden.

Diese Arbeit stellt ein System vor, mit welchem dreidimensionale Sounds mit Hilfe von binauralen Techniken in einem Videospiel wiedergegeben werden können. Es handelt sich um ein Virtual Reality Videospiel, bei dem mehrere Benutzer in dem selben physikalischen Raum in einer virtuellen Welt miteinander interagieren. Binaurales Audio wird verwendet, um die Spielerfahrung zu verbessern, indem ein zusätzlicher Weg, Informationen über die virtuelle Welt zu übermitteln, ermöglicht wird.

Um eine realistische Erfahrung zu erreichen, muss die Richtung der simulierten Töne automatisch angepasst werden, wenn die Benutzer ihre Köpfe rotieren. Um diese Rotationen zu erfassen, können Orientierungs-Sensoren verwendet werden. Verschiedene solcher Sensoren, wie sie zum Beispiel in den meisten Smartphones verbaut sind, wurden verwendet, getestet und werden vom System bereitgestellt.

Abstract

Binaural audio, which literally means “audio regarding both ears”, describes techniques to provide realistic three-dimensional sound experience to a user wearing headphones. Three-dimensional sound can help players immerse themselves in virtual environments, such as video games: Sounds can be perceived from any direction, while visuals can only be seen in front of the user’s eyes.

This thesis presents a system to provide three-dimensional sound using binaural techniques in a co-located multiplayer environment. In this environment, multiple users interact with each other inside a virtual world while being physically located in one room. Binaural audio is used to improve the experience by adding an additional means to convey information about the state of the virtual world.

To achieve a realistic experience, the simulated sound direction must adapt to users rotating their heads. To track such rotations, orientation sensors can be used. Various different orientation sensors, including those present in common smartphones, have been used, tested and implemented and are provided by the system.

Chapter 1

Introduction

1.1 Motivation

With consumer devices such as the *Oculus Rift* and *HTC Vive* being released, virtual reality has become an interesting topic for video games. But it is not only used for games played in front of a computer, but also for co-located multiplayer games, in which players physically move around in the same location to interact with each other. Typically, co-located games use what is called global visuals and audio. This means output which is the same for each player. Global visuals can be produced with video projectors, either on the floor where players move around, or on a wall next to it. Global audio can accompany it by using loudspeakers in the room. Multiplayer games in general, however, normally use customized visuals and audio, which means players see and hear those parts of the game world that are relevant to them. It would be interesting to be able to deliver these in co-located environments.

In the real world, when sounds are heard, the direction they originate from relative to the listener's position can also be perceived. In simulations, and more specific in video games, this feature of sound is still rarely used. Audio output, even if customized for each user in a multiplayer game, mostly consists of background music and sound effects which cannot be localized by listening alone. When simulating virtual worlds in co-located environments, having sounds whose origin direction can be determined just by listening could greatly improve the sense of immersion.

1.2 Problem and goal

The goal of this thesis is to add customized, localizable audio output to an existing co-located environment, namely the *Ars Electronica Deep Space* system. This system already possesses all necessary features for co-located multiplayer games with global audio and visuals. A project is presented that

uses smartphones and headphones in combination with the *Deep Space* system to deliver directional audio to each player in the co-located environment.

Using headphones and so-called binaural audio techniques, directional information can be added to any mono-channel sound. Listeners will be able to tell which direction the sound is coming from as if it would originate from an actual sound source at any defined position in the room.

To achieve realistic directional audio that responds to players moving around in the room, the orientation each player is facing needs to be determined. Only then can the relative direction of the virtual sounds in respect to the player be calculated. Thus, an orientation sensor needs to be carried by each player. Two different solutions to this problem are implemented and presented.

1.3 Outline of this work

Including this chapter, this work is divided into seven chapters, each presenting a part of the research and the project.

Chapter 2 describes the current state of the art in co-located environments. The requirements for using binaural audio in a co-located system are listed. Some existing co-located systems are examined, if and how they are capable of delivering customized directional audio to its users.

In chapter 3, the theoretical basics of binaural directional sounds are described. It shows how humans can determine the direction of sounds, which techniques exist to make any sound directional, and the implications of this for designing virtual environments.

Chapter 4 presents the concept for the implementation of the project. It discusses which problems are being solved and by which methods. A game framework is introduced that can be used to create multiplayer games for the co-located environment with customized binaural audio. Also, an example game using the introduced game framework is created to showcase its features.

The details about the technical implementation of the concept are described in chapter 5. The different parts of the implemented system are explained, how they have been developed and how they communicate with each other.

Using the developed example game, data about the effectiveness of the system has been collected by letting multiple users play a pre-defined scenario and recording their performance. Chapter 6 performs an objective evaluation of this data and determines if the proposed goals could be achieved by the implementation.

Chapter 7 concludes this work by presenting a short summary, the results and contributions. Also, steps to be done regarding this topic in the future are discussed.

Chapter 2

State of the Art: Co-Located Environments

2.1 Overview

The term co-located literally means placing multiple entities inside the same physical location. In terms of virtual environments, it is used to describe environments in which multiple users interact with each other by being located in the same physical location, which usually is an enclosed space or room.

In co-located multiplayer games, users interact with each other to reach an either common or adversarial goal inside a virtual world by sharing the same location. This contrasts regular multiplayer games, in which each player usually uses their own device, such as a computer or smartphone, at arbitrary physical locations and interacts with the game using an input device like a keyboard and a mouse or a joystick.

The interactions between users in a co-located environment arise from their actual physical interactions, like movement. The virtual environment can be changed by various data derived from these interactions, e.g. position and rotation of the users inside an enclosed space or movement of hand and feet.

2.2 Requirements

This thesis creates a concept for using directional audio in a co-located multiplayer environment. In this environment, users should receive realistic, customized three-dimensional audio from virtual sound sources by physically walking inside a pre-defined, rectangular location. To achieve this, the system is required to feature the following ways to obtain data about the physical state of the co-located users, and to report the state of the virtual world back to them.

2.2.1 Position

For a customized sound experience, the system must be able to have information about each user's position inside the co-located environment. Using these positions, the relative directions to each virtual sound source can be calculated and used for delivery of realistic directional sounds.

For the implementation of the thesis project, the *Pharus* [25] system supplies these positions. It uses laser tracking devices to calculate the positions of multiple users inside a pre-defined area and reports them to the application. Technical details about how these positions are obtained are given in section 5.1.

2.2.2 Orientation

Users should be able to locate the incoming direction of virtual sounds just by listening. For this, it is also important that the system is able to obtain the direction in which the users are facing. In the best case, the absolute orientation of each user's head can be obtained, so the virtual sounds change immediately to users rotating their heads.

It might also suffice if only the rotation of the users' bodies could be determined; however, the immersiveness of the experience might be reduced by this restriction.

As part of the thesis project, two ways to obtain orientation data have been implemented and tested: the sensors included in smartphones, and a custom solution using an Arduino microcontroller with an orientation sensor attached to a pair of headphones. Details about this topic are covered in section 2.3.

2.2.3 Audio output

To deliver realistic audio, each user needs to be supplied their own customized 2-channel audio output. This can be achieved using a pair of headphones per user and creating one stereo audio signal per user. Chapter 3 describes how three-dimensional audio can be created using headphones.

The different audio signals can either be created centralized on a server and wirelessly sent to each user's headphones, e.g. using bluetooth. Alternatively, one client device can be used per user to supply the audio signal, for example by connecting each pair of headphones to a smartphone running a client application which is carried by each user.

2.2.4 Visual output

Visual output can improve the quality of the experience, by being able to deliver additional information about the state of the virtual world. While users

will see the position of the other players physically, other virtual entities, such as obstacles or enemies, can be displayed visually.

Visuals can be delivered globally, e.g. by projecting an image on a wall of on the floor, or per user by using individual displays, such as smartphones or head-mounted displays.

2.3 Orientation measurement

Measurement of the orientation, or attitude, of an object in three dimensions was one of the biggest challenges while implementing the concept of this thesis. Each user's orientation in the co-located environment needs to be obtained in order to supply meaningful directional audio.

Orientation in 3D can be stored in multiple ways, two of which are the most common: Euler rotation, consisting of three rotations around pre-specified orthogonal axes, and quaternions, encoding an axis vector and a rotation angle. Both of these can be converted from one to another.

To measure the absolute orientation of an object in three dimensions, at least two reference directions are needed. If a vector in both these directions can be obtained, the orientation can be calculated. This can be achieved by using two different sensors.

Motion sensors, or *accelerometers*, can measure the total force acting on an object. At rest, the only force acting on an object is gravitational force directed to Earth's centre, which can be used as the first reference direction. Magnetic sensors, or *magnetometers*, can measure the strength and direction of the magnetic field exerted on an object. When no artificial magnetic fields are present, the only magnetic field is one directed to Earth's magnetic North or South Pole, which can be used as the second reference direction.

Since both these kinds of sensors tend to be less accurate than desired for a proper calculation of orientation, a third sensor can be used to improve the accuracy: Rotation sensors, or *gyroscopes*, can measure the change of rotation over time. Using algorithms called sensor fusion, the measurements of all three kinds of sensors can be combined to obtain a very accurate result of attitude, if one of each kind is present on the same object. Such a setup of three sensors to calculate an object's attitude is called attitude heading and reference system (AHRS).

Several different such algorithms to perform sensor fusion have been developed and can be used on the raw results of each sensor [24]. Most algorithms can also perform if not all three kinds of sensors are available, however with less accurate results. For example, if only a gyroscope sensor is available, the attitude of an object in relation to an arbitrary initial attitude can be estimated by integrating the change over time. Errors introduced by inaccuracies of the gyroscope will accumulate though, causing the result to

Figure 2.1: Some users playing a fish game in Ars Electronica's Deep Space.[1]

become more and more inaccurate. This problem is called *drift* and can cause the attitude to change even if the object is at rest. Only if all three sensors are available, an absolute orientation without drift can be acquired.

As part of this thesis, a custom module able to determine its own orientation in space has been developed. It can be attached to a pair of headphones and send sensor readings to a receiver device via Wi-Fi. The implementation of this module is discussed in section 5.3.

2.4 Existing systems

Co-located systems are used for multiplayer environments in some existing projects and installations.

2.4.1 Ars Electronica Deep Space

The *Ars Electronica Center* (AEC) is a scientific, educational and research institute in Linz, Austria. There is an installation called *Deep Space* [17], which contains co-located multiplayer applications. *Deep Space* features a laser-tracking system which provides the positions of each person standing or moving inside a 16×9 metre area. Further, images can be projected on the floor of this area and on a wall next to it using four 4K projectors, resulting in a resolution of up to 8192×4320.

It features lots of games and applications enabling the persons inside to interact using their positions in the room. Figure 2.1 shows users playing a

[1] Image from the AEC website: http://www.aec.at/feature/en/deep-space-8k/.

Figure 2.2: Images of *the Void*'s head-mounted display. It features a 2K display and headphones for three-dimensional visuals and audio.[2]

game in the *Deep Space* installation. Their positions are used to control fish on the floor projection.

Some parts of this system, mainly the position measurement system, are also used for the implementation of the thesis project.

2.4.2 The Void

The Void[3] is a concept for virtual reality games created by *The Void LLC*, Utah, USA. Advertised as "hyper reality" by its creators, *the Void* provides game installations, the first of which at Madame Tussauds in New York City.

Using a virtual reality suit called *Rapture Gear*, an obstacle parcour is turned into a virtual world. The suit features motion tracking of the whole body, haptic feedback and three-dimensional audio and visuals using a headset with a display and headphones, as displayed in Figure 2.2.

2.4.3 OptiTrack

Optitrack[4] is a motion tracking system by *NaturalPoint*, based in Oregon, USA. It uses high-speed cameras to track position and rotation of marker objects. Multiple cameras can be used together to track many objects. It can be easily integrated with VR applications, it is for example also used by *the Void* to track users moving inside the VR parcour. Marker objects can be attached to the users' clothes to track their positions and their individual movements. There are different motion tracking solutions available by *OptiTrack*; Figure 2.3 shows one of the cameras used for it.

[3] *The Void* website: https://thevoid.com

[3] Image by *the Void* website: https://thevoid.com/Press

[4] *OptiTrack* website: http://www.optitrack.com/

Figure 2.3: One of the cameras used by *OptiTrack* for motion tracking.[5]

Figure 2.4: Four people using the *iGameFloor* platform. Image from [11].

2.4.4 iGameFloor

iGameFloor [11] is a system for co-located multiplayer applications developed in the Center for Interactive Spaces at the University of Aarhus, Denmark. It features a glass floor of $12m^2$ with image projection from the bottom. Interaction is possible through tracking of user's limbs with a camera. Figure 2.4 shows what the system looks like.

[5]Image by the *OptiTrack* website: http://www.optitrack.com/products/prime-41/

2.4.5 Conclusion

There are some existing systems creating co-located multiplayer environments. All of them feature some kind of tracking of the users' positions, some even more complex tracking like motion of arms, hands, etc. Most systems are not able to determine a user's orientation.

Visuals, either in form of head-mounted displays (HMD) or projections on floor or the wall, are also used in most of them. Sounds are, if used, global and the same for each user most of the time. The only system to use customized and directional audio is *the Void*.

The system created along with this thesis is supposed to contain all the above mentioned features, with the exception of motion tracking and HMDs.

Chapter 3

Binaural Audio and Spatial Sound Perception

3.1 Overview

The word binaural literally means "regarding both ears". When used in the context of audio, it usually means deliberately delivering specific sound signals to both ears of a listener. This means, for each ear, a separate acoustic signal is generated and should then be the only sound heard in this ear. This can most simply be achieved by using stereo headphones, though other methods exist to achieve binaural audio using for example stereo speakers and crosstalk cancellation techniques [5, 6].

Using headphones, audio data can be generated by computer programs using two audio channels, each of which is then played back by one of the headphone's speakers. Ideally, there should be no transformations of sound while propagating from the headphone output to the inner ear. This, of course, heavily depends on the type and quality of the headphones used.

The human brain can localize the direction of sounds remarkably well in the real world. When we hear something, most of the time we instantly know if the sound source is to the left or right, is above, below, in front or behind us. This works because both ears do not perceive one sound exactly the same due to various factors [4].

Binaural audio can be used to recreate what both ears of a listener heard at another time and place, or even simulate sound sources at virtual positions inside a virtual world [7]. To achieve this, it is important to know which kinds of differences between the signals received in both ears the brain uses to localize a sound, which is explained in section 3.2. Various techniques to create directional audio virtual have been invented; the ones relevant to this work are introduced in section 3.3. Finally, problems and imperfections of these techniques as well as their (partial) solutions are presented in section 3.4.

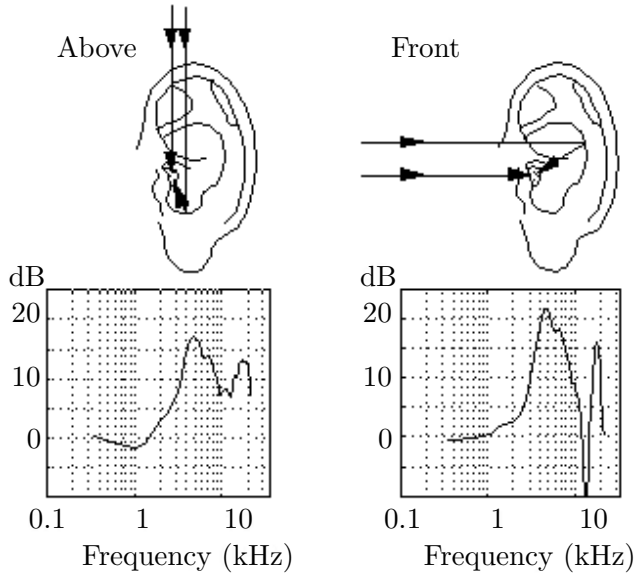


Figure 3.1: How monaural cues change the perceived spectrum of a sound, depending on the incoming direction. Image from [32].

3.2 Basics of spatial sound perception

Sound localization is automatically performed by the human brain when hearing a sound. It is possible for a listener to tell if a sound source is above, below or on the same level as the their ears, which is called *elevation* detection. Furthermore, the horizontal direction or *azimuth* can be determined, i.e. if the sound source is left, right, in front or behind the listener.

The accuracy of sound localization depends on various properties of the perceived sound(s). To be able to localize the direction of a sound, the brain automatically analyses various cues present in the perceived sound. There are two types of cues: monaural cues, which can be detected using only one ear, and binaural cues, which arise from the difference between what is perceived simultaneously at each ear. Binaural cues appear as the interaural time difference (ITD) and the interaural level difference (ILD) between the two ears, where interaural means "between the ears".

3.2.1 Monaural cues

Even though binaural audio is important for localizing of sound sources, there are cues which can be sensed with just a single ear. These cues are formed by the shape of the outer ear, the pinna. The pinna is formed in a way that causes sound waves to bounce off elevations and thus cause positive and negative interferences with themselves. This leads to certain frequencies

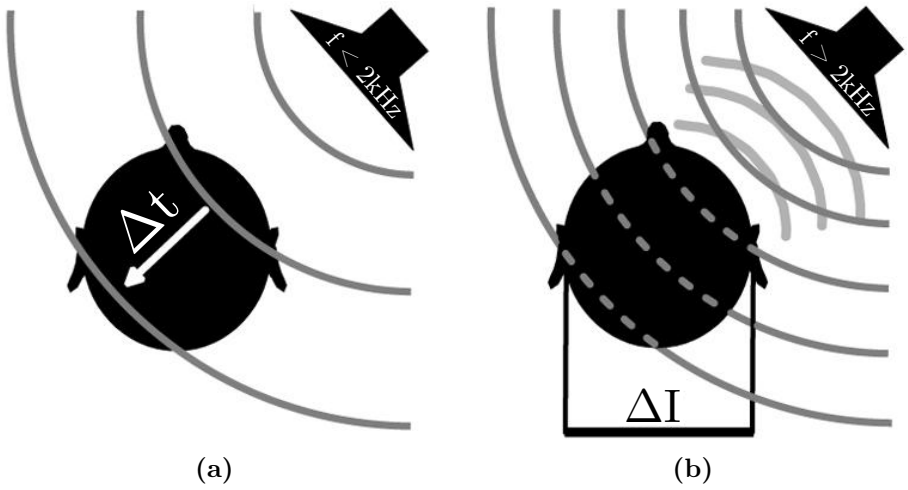


Figure 3.2: Illustration of interaural time difference (ITD, left) and interaural level difference (ILD, right). Images from [12].

being amplified while others are attenuated. In other words, the pinna acts as a filter on the signal whose frequency response depends on the direction of the incoming sound waves. These cues are most important in detecting the elevation of a sound source and contribute only little to azimuth detection [32]. Figure 3.1 shows an example of this effect for the same sound perceived from different elevations.

3.2.2 Interaural time difference

When a sound source is not located directly in front or behind a listener, it will be perceived at a slightly different time in each ear. This time difference is used as a cue to localize sounds. Figure 3.2a illustrates the time difference of a sound source located to the front right of a listener.

The time difference depends on three factors: the propagation speed of sound waves, which is roughly the same in air everywhere on the planet, the distance between the ears of a listener, which does not change over time, and the direction of the incoming sound wave. This makes it possible for the brain to localize a sound based on the time difference.

Yet, since sounds are wave forms, a time difference can only be perceived as a shift in phase. To unambiguously infer the actual time difference from this phase difference, it needs to be less than half the period of the wave. As a result, ITD can only help localize sounds with relatively low frequencies, since these have longer periods [19]. However, for very low frequencies, the time and thus the phase difference are also very small, resulting in less accurate localization [3].

3.2.3 Interaural level difference

Similar to ITD, there are also differences in the perceived volume of sounds at each ear. These differences are caused by a longer path length to the ear farther away from the sound source and because the head acts as an obstacle for the sound waves. Figure 3.2b illustrates the level difference of a sound source located to the front right of a listener.

This effect is also used to localize sound sources; the strength of the level difference between the two ears corresponds to incident angle from this direction. The attenuation is stronger for short wavelengths and almost non-existent if the wavelength becomes longer than the part of the sound wave shadowed by the head, which makes ILD more apparent for high frequencies.

3.2.4 Implications

Even though localization is performed automatically inside the brain when hearing a sound and there is no need to create a system to localize sounds computationally as part of this thesis, knowledge of how the localization works is still essential. To be able to recreate sounds that can be localized by a listener, important cues need to be included.

Importance of different cues

Both monaural and binaural cues need to be included when recreating sounds using binaural audio. Monaural cues are most important for inferring elevation, binaural cues for azimuth. To be able to localize sounds of all frequencies, both ITD and ILD are essential.

Virtual environment design

If binaural spatial audio is to be used to help users interact in a virtual environment, it is also important to select the sounds that can be heard in this environment with respect to this knowledge. Those sounds should be ones which can be localized more easily. Since ILD and ITD perform more effectively at different frequencies, these are sounds which spread throughout the whole frequency spectrum.

Cones of confusion

As already stated in this section, the accuracy of sound localization depends on various properties of the sound and sound source. Since ITD and ILD are used to detect the direction between a listener's head and the sound source, ambiguous points exist where both of them are roughly the same. These are points lying on slices of a cone along the axis between the two ears. For each point on a circle like this, the distance to both ears is the same [4, 14, 19]. Figure 3.3 shows this cone of confusion at the left ear of a listener.

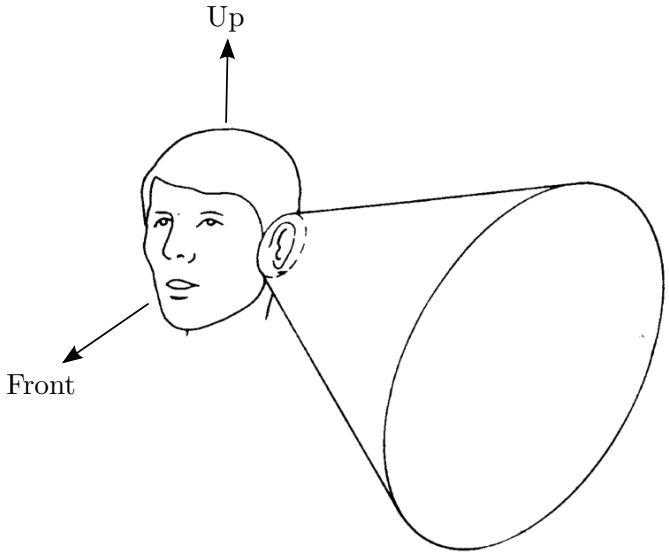


Figure 3.3: Illustration of the cone of confusion to the left of a listener. For each point on the circle, the distances to both ears are the same, resulting in ambiguous ITD and ILD. Image from [4].

If a sound is heard from only one direction, a listener can often not determine if it is coming from the top, bottom, front or back. A solution to this problem is slightly changing the incoming direction, since this context can be used to solve the ambiguity. This can either be done by using moving sound sources or by the listener tilting and moving their head slightly [33].

This means that it is important for a virtual system using spatial audio to allow its users to turn their heads and instantly adapt to the simulated sound direction, for them to be able to localize even still standing sound sources.

3.3 Spatial sound techniques

Various different techniques exist to add directional cues to sounds to make them appear to originate from some point in a virtual world. The aim is to recreate exactly what would reach the ears of a person if this virtual world and sound sources were real. Several techniques exist to achieve this task, and only a subset of those are suitable for use in a co-located environment.

3.3.1 Positioning of real sound sources

The most obvious way to create directional sounds is to actually position real sound sources, e.g. loudspeakers, at physical positions relative to a listener

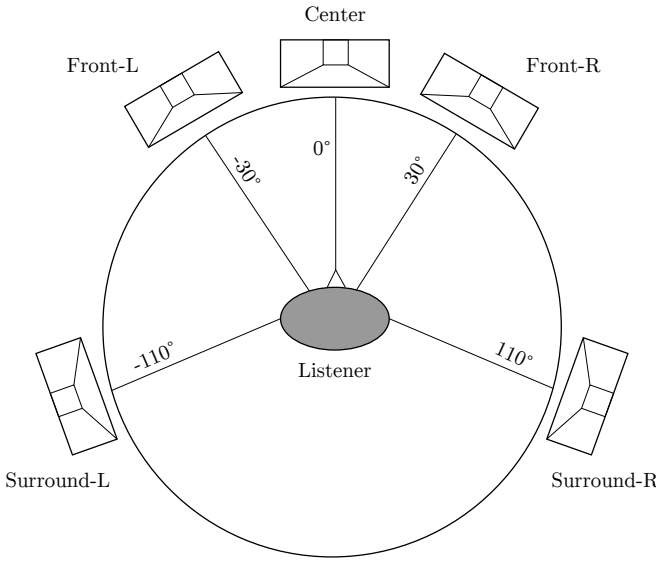


Figure 3.4: 5.1 surround sound set-up using five high-frequency speakers and 1 low-frequency speaker (not shown).[1]

and having them produce sounds. Usually this is done by having two or more speakers and panning sounds between them to be able to also simulate sound positions between the sound sources, which has been shown to be possible [10]. This includes simple stereo sound set-ups as present in most desktop computer installations and more complex surround sound set-ups using more than two speakers. Figure 3.4 shows one possible surround sound configuration.

The benefit of this solution is that listeners do not need to be wearing headphones and thus multiple persons can listen to the same sounds at the same time, which makes it suitable for environments with many listeners, e.g. cinemas. However, the possible positions of simulated sound sources and the resulting accuracy of localization is very limited using these techniques, which makes them unusable as part of this thesis project.

3.3.2 Monaural recording

Monaural recording, or mono, simply means recording sounds with one microphone and then listening to the record using headphones. Since both ears receive the exact same sounds at the same time, ITD and ILD at both ears is the same and the sound will be perceived to originate inside the listener's head. To produce directional sounds, two different sound channels for each ear are needed.

[1]Image by Kamina (Own work) https://commons.wikimedia.org/wiki/File:5-1-surround-sound.svg, CC BY-SA 3.0, via Wikimedia Commons.

3.3.3 Binaural recording

To binaurally record a sound, two microphones are needed, one for each ear. These two microphones need to capture exactly what the respective ear of a potential listener would hear. This is possible by using small microphones inside the inner ear of either a real human or a dedicated dummy head for binaural recording.

Such dummy heads are created specifically for binaural recordings and aim to simulate the acoustic properties of a human head with high accuracy, including shape and material of the head, ears and torso. One example of such a dummy head is the KEMAR[2] (Knowles Electronic Manikin for Acoustic Research) head and torso simulator by Danish company G.R.A.S., which has existed since 1972.

When a sound is recorded binaurally, it is saved as a two-channel sound file, one for each signal recorded. When listened to with headphones, the listener will be able to localize the sound the same way as when hearing the original sound from its original direction in relation to where it was recorded.

Binaural recording can be used to provide realistic three-dimensional sound experience for pre-defined scenarios, such as video recordings. Yet it cannot be used in a co-located environment, since the relative direction between the users and the virtual sound sources can not be known beforehand.

3.3.4 Head-related transfer function (HRTF)

The transformations which occur to sound waves as they travel from a sound source to the ears of a listener can be modelled by mathematical functions. If two such functions are known, one for each ear, they can be applied to any sound in order to add directional cues. These functions are called head-related transfer functions, or HRTFs [28].

To model ITD, ILD and monaural cues, such a function must be able to perform frequency-dependent phase and magnitude changes on a signal. Fortunately, this can be done by modelling HRTFs as LTI (linear, time-invariant) systems [18, Chapter 4]. LTI systems are relatively easy to implement and relatively fast to apply to digital signals. They can be implemented as digital filters by convolving the original signal with a digital impulse response signal, which has been shown to be possible to even perform in real time on mobile devices [26].

Since HRTFs can be used to make any sound directional, they are suitable for creating three-dimensional sounds in virtual environments. For each virtual sound source, two HRTFs, one for each ear of the listener, need to be obtained and applied to the original sound. These HRTFs need to correspond to the direction of the virtual sound. Thus, whenever the relative sound direction changes, either when the virtual sound source's position or

[2]KEMAR website: http://kemar.us/.

the position or rotation of the user changes, a new pair of HRTFs is needed. There are multiple ways to obtain the HRTFs, as described below.

Physical modeling

It is possible to create mathematical models of the transformation of sounds between a sound source and a listener's inner ear. Using such a model, the HRTFs for one specific sound direction can be calculated and applied to an input signal [2, 6, 9]. This way, a virtual environment containing virtual sound sources can be simulated by calculating two HRTFs for each of those sound sources and applying them to their original sound at each simulated time interval.

However, these mathematical models tend to be very complex and thus take a lot of processing power and time to compute, which makes these solutions rather impractical to simulate virtual environments in real time.

HRTF database

In contrast to modelling a virtual acoustic environment and calculating sound transformations as HRTFs, they can also be recorded in real environments and saved in databases. These databases contain two HRTF entries, one for each ear, per desired sound direction. This means that HRTF databases can grow rather large, dependent on the included number of directions. The more HRTFs are included, the more accurate the result since more individual direction can be simulated, but also the more data is needed for storing the database.

To simulate a sound originating from a specific direction, the pair of HRTFs closest to this direction is selected from the database and applied to the simulated sound. There are also techniques to interpolate between multiple HRTF pairs to improve the result's accuracy [20].

HRTFs can be recorded using binaural recording. To do so, the transformations between the original sound and the recorded sounds need to be calculated. This can, for example, be done by sending only a very short impulse, and using what is recorded as impulse response of the transfer function. There are also better techniques to obtain the HRTFs with higher precision. Fortunately, various such databases are available online for free usage [1, 8, 16, 30], thus no manual recording was necessary as part of this thesis.

3.4 Problems and limitations of binaural audio

Binaural audio techniques aim to provide sounds which can be localized naturally by a listener. For this to be possible, the sound waves which reach the inner ear of the listener need to be recreate real sounds originating

from a desired direction as close as possible. Using pairs of HRTFs, the transformations of sound waves travelling between the sound source and the inner ear are modelled and can be applied to virtual sounds. However, the results can never perfectly simulate real directional sounds for a number of reasons.

3.4.1 Differing physical conditions

Since most sets of HRTFs are measured using microphones inside the ear of a human or mechanical dummy listener, the resulting simulated directional sounds will only be the same for the exact same listener. The shape and physical properties of ears, head and body differ from person to person, making the results of using HRTFs recorded with any other head than the listener's non-optimal. These differences can result in significant problems in sound localization [23]. The same is true for binaural recordings.

This problem can be solved by using the same physical environment for binaural recording or HRTF recording as for sound generation. This would mean that each person using a co-located environment featuring binaural audio would first have to acquire their own set of HRTFs, which is an expensive and time-consuming process.

A more simple way to not eliminate, but reduce the effects of these problems is to automatically create individualized HRTFs for each user. This can be done using a mix of recording HRTF databases and using physical modelling to adapt these databases for a user. For example, Spagnol *et al.* [29] created a device capable of capturing physical properties of the listener's pinnae. These properties are then used with the structural HRTF synthesis model from Brown *et al.* [27] to obtain customized HRTFs. Other solutions to create customized HRTFs exist, some even incorporating not only the shape of the pinnae but also head and torso properties [21, 22, 31].

3.4.2 Sound generation inaccuracies

Additional problems exist, even if assuming a digital representation of the perfect sound for one specific listener and direction could be obtained by eliminating the previously mentioned problems. It is still nearly impossible to have this exact sound be heard by the listener, since the sound still needs to be shaped into actual physical sound waves by headphones and then reach the inner ears. The severity of errors introduced in this step highly depends on the type and quality of used headphones.

3.4.3 Implications

For the above stated reasons, binaural audio will most likely never reach the same quality for localization as sounds heard in the real world. It is important to be aware of this fact when using binaural audio provide directional

cues for users in a co-located environment. Sounds which can be localized perfectly by one user might not even be locatable at all by another user.

Thus, directional audio should probably only be used alongside other directional cues such as visuals, so every user is able to interact properly with the application. If using one of the many existing binaural audio systems for a co-located environment, the quality of sound localization can and should be evaluated, like it is done by Guastavino *et al.* [13], before deciding for a single one of these systems.

Chapter 4

Concept—Binaural Audio in a Co-Located Environment

This thesis creates a system to use binaural audio methods for realistic directional audio in a co-located multiplayer environment. This chapter introduces the concept of the implemented system, the goal of the project; and it discusses which problems are being solved by it and with which methods.

4.1 Goal

Currently existing co-located multiplayer games mainly use, as discussed in chapter 2, global visuals and audio to inform the players about the state of the virtual world. Global means that each player receives exactly the same information. Some systems already use customized output via head-mounted displays and headphones, so each user gets the information from their perspective.

The goal of this project is to use an existing co-located multiplayer environment, namely the Ars Electronica Deep Space, and enhance the experience of users by adding customized, realistic three-dimensional audio. This is expected to increase the feeling of immersion into the virtual world for the players, by being able to hear virtual sound sources just as if they would actually exist and being able to determine the relative direction to them just by listening.

This new feature of directional audio is supposed to act as an additional way to convey information to the users. Its quality should be high enough that users can identify the direction to a virtual sound source by listening. To test if this has actually been achieved, a game using this feature has been be implemented. The game contains some kind of interaction between the players and the virtual world which is only possible by locating sound sources. By collecting data about users' performance in the game, it can then be evaluated if this goal has been achieved.

Figure 4.1: Photographs[1] of one of the *Sick* laser scanner devices. The black ring is where the laser is emitted, meaning it tracks the legs of the players at this height.

4.2 Utilized methods

As discussed in section 2.2, the implemented system requires user interaction to handle input and output. The following are the methods which have been used to achieve this goal.

4.2.1 User position—Deep Space laser tracking

To receive information about each co-located user's position, an existing part of the *Deep Space* system has been used. A pre-defined, rectangular area of 16 x 9 metres is surrounded by 2D laser scanner devices created by the company *Sick AG*[2]. Using these devices, the positions of multiple users inside the area can be calculated. The maximum number of users is proportional to the number of used laser-tracking devices.

These devices are installed just above the ground, about $10-20$ cm high. They produce horizontal laser light, and by measuring at which angle and distance the light gets reflected, the position of objects inside the area can be calculated. These objects are, under normal circumstances, the legs of people located inside. When two legs are close enough to each other, their centre point is reported as the position of a user. Figure 4.1 shows two photographs

[1] Images by the *Sick* website: https://www.sick.com/at/en/detection-and-ranging-solutions/2d-laser-scanners/lms1xx/lms100-10000/p/p109841

[2] Sick website: https://www.sick.com/

of one of the laser tracking devices.

These positions are calculated and gathered by a server application called *Pharus* [25] and then transmitted to the consumer application. This whole process was already installed and available for usage as part of this project. Section 5.1 discusses how the interaction with this system is implemented.

4.2.2 Global visual output—Deep Space projections

For global visual output, the existing projection capabilities of the *Deep Space* system are used. It features two 8K projector setups, which means an output resolution of 8192 x 4320 pixels. Each of these setups is actually made up of four 4K projectors. One of them projects an image directly onto the co-located 16 x 9 metre area; the other projects an image onto the wall next to it. Figure 2.1 in chapter 2 shows the Deep Space system using both of those projections for a game. This visual output can be used to display information available to all users.

4.2.3 Customized visual and audio output—smartphone

It is also be important to be able to deliver information to each user specifically. As the goal is to have realistic directional audio, each user must be supplied a different audio stream. Also, having the possibility to display custom visuals per user can help creating more interesting games.

This is implemented using one smartphone and one pair of headphones per user. The smartphone runs a client application communicating with the game server, from which it receives information about the virtual world, which entities should be displayed and which sounds should be played back.

The actual smartphone devices used for this can be selected from a wide range of possible devices. The device is required to be able to run applications created with the *Unity* game engine. Also, a way to connect stereo headphones is necessary. Furthermore, about 200 megabytes of free storage for the application is needed. This includes almost every modern smartphone device capable of running the *Android* or *iOS* operating system.

4.2.4 Directional audio generation—RealSpace3D engine

To create realistic directional audio, spatial audio techniques as described in section 3.3 have to be used. Fortunately, there are various software products available capable of performing this task, so no custom solutions had to be implemented.

In this project, the *RealSpace3D*[3] audio engine by *VisiSonics*[4] has been used. It uses HRTF databases to create binaural directional audio. As in-

[3] *RealSpace3D* audio engine website: http://realspace3daudio.com/

[4] *VisiSonics* website: http://visisonics.com/

put, the engine needs the positions of one or more sound sources and one sound listener inside the virtual world. Each sound source can play back an audio file. In response, it generates two audio channels containing the binaural audio. When listened to with headphones, the virtual sounds seem to originate from their relative virtual positions. There are also many other parameters to configure to fine-tune the result. Details about the integration of *RealSpace3D* into the game are given in section 5.2.

4.2.5 User orientation—two solutions

As described in section 2.3, orientation measurement methods to calculate the co-located users' orientations in relation to the virtual sound sources are needed. Only if the system responds to users turning around inside the co-located environment by changing the simulated sound direction, the users will be able to locate the virtual sound sources by listening. To achieve this, two different solutions have been implemented and used in the project.

Smartphone sensors

Smartphones are used for customized visual and audio output. Most current smartphones feature integrated sensors for orientation measurement, namely accelerometer, gyroscope and in most cases also a magnetometer. This means, the devices which are already used for output can also be used for orientation input.

There are, however, two problems with this approach. The first problem is that the quality and accuracy of these sensors in smartphones vary vastly between different devices. Some might be very accurate, but others might not and be subject to inaccuracies and sensor drift. Secondly, since smartphones are held in the users' hands, the sensors do not automatically turn when users turn their heads in another direction. The directional audio bound to the sensor readings will only feel realistic if users always turn the smartphone the same way they turn their heads.

Custom orientation sensor module

Since there are obvious problems with using smartphones for orientation measurement, another solution for this task has been developed. A custom orientation sensor module that can be attached to a pair of headphones has been developed. This way, the two previous problems can be eliminated. The sensor module always turns the same way the user's head is turned. Also, since the sensors used can be evaluated beforehand, suitably accurate ones can be selected.

This module consists of various pieces of hardware that can operate autonomously and transmit sensor readings wirelessly. Details about the hardware used and the software implementation are given in section 5.3.

4.3 Multiplayer game framework

All the above mentioned methods have been combined into one system which enables using binaural directional audio in a co-located multiplayer environment. It is both a software and hardware concept which allows the development of application using these methods.

4.3.1 Application server

A central application server runs a main program that communicates with all other parts of the system. This application contains a representation of the virtual environment and is solely responsible for changing this environment's state based on user interaction and other factors. A representation of the virtual world that should be visible to all users can be displayed via two connected video projectors, one projecting an image onto the floor of the co-located area, another one on a wall next to it.

The server receives information about each user's position from a positioning application, which may or may not run on the same computer. This positioning application receives data from the laser tracking devices and computes the positions, which are provided by a software framework as a conveniently accessible list.

4.3.2 Client application and mobile devices

A client application can be installed on different smartphones. When these devices are connected to the same Wi-Fi network as the server machine, users carrying them can enter the co-located area and begin interacting with the application. The client devices are linked to the corresponding user's positions by a mapping algorithm. This way, multiple users can enter the co-located area, with or without client devices. The client application receives information about the virtual environment from the server and can both display it on screen and play back three-dimensional audio using headphones and the *RealSpace3D* audio engine.

The client devices also allow interaction via touchscreen and various sensors, e.g. orientation sensors. These interactions are sent from the client application to the server, which is responsible for responding to them by changing the state of the virtual environment.

Optionally, the custom orientation sensor module can be attached to the headphones worn by a user. The module transmits the sensor readings to the client device via Wi-Fi, which can then use it for interaction instead of its own sensor readings.

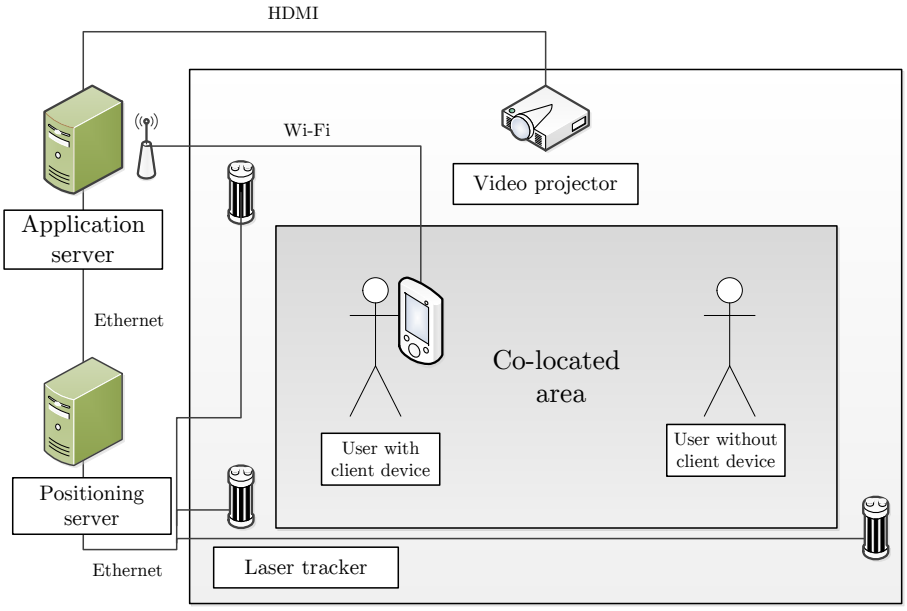


Figure 4.2: Overview of how the different parts of the game framework communicate with each other.

4.3.3 System overview

Figure 4.2 shows an overview over all the components of the multiplayer game framework. Each line symbolizes communication between parts, either wired or wireless. The type of communication is stated next to the lines.

4.3.4 Application development

To develop an application or a game using this system, a software framework for the Unity game engine is provided. Both the server and the client application can be developed using this framework. It supplies the list of user positions and, if a client is connected, user orientations, to be used in any desired way.

Virtual sound sources can be placed in the game world, which are played back by the client application to appear from the correct relative direction automatically. Further, interactions like touchscreen presses and gestures can be sent from the client to the server application via a command system.

As an example of what this system is capable of, a fully-featured example game has been developed, explained in the next section.

4.4 Example use case—binaural Pac-Man

4.4.1 Overview

Alongside the concept of this thesis, a video game has been implemented that uses the developed system discussed above. One or more players can enter the co-located area, using a smartphone running a client application and a pair of headphones connected to the smartphone. These headphones can either be connected via cable or wireless, e.g. using bluetooth. Optionally, the orientation measurement module can be attached to the headphones. If not, the smartphone's orientation sensors are used. If players enter the area without a client device, they are represented as "passive" and can not interact with the virtual environment.

4.4.2 Game rules

Players are represented by a Pac-Man like visual at their position via the video projector and the display on the held smartphone. The player icon appears looking in the direction that the orientation sensor measures. If the orientation seems wrong, it can be calibrated via an option in the client application.

After some time, the game server spawns enemy entities. They are represented by ghost images and chase a randomly selected player. On the projector, enemies are not visible. On the client devices, the players only see a small area of the virtual world, in the direction they are currently looking. So enemies far away from the players, to their sides or behind them are invisible.

The enemies also act as virtual sound sources, producing one of various enemy sounds at their position. Players can hear these sounds via their headphones and are to localize the relative direction from their position to the enemy's position. There are also variants of the enemies that are completely invisible, so users can only find them by listening to their sounds.

Once spawned, an enemy will try to reach its target player's position. They move at a pre-defined speed using one of different chasing algorithms. There are currently two of those algorithms: *direct chase*, which follows the straight line from the enemies to the players, and *chase from behind*, which first tries to get behind the players and then reach them without being seen.

Enemies continue to chase the players until a player is looking directly at an enemy. It then stops moving and disappears after a certain time. This means, it has been defeated. If, on the other hand, an enemy reaches a defined minimum distance to a player without being seen, the player is "hit" and loses one point. In both cases, the enemy entity gets removed.

Furthermore, the game server spawns so-called collectible items, represented by a white dot. These items are also only visible when a player is near

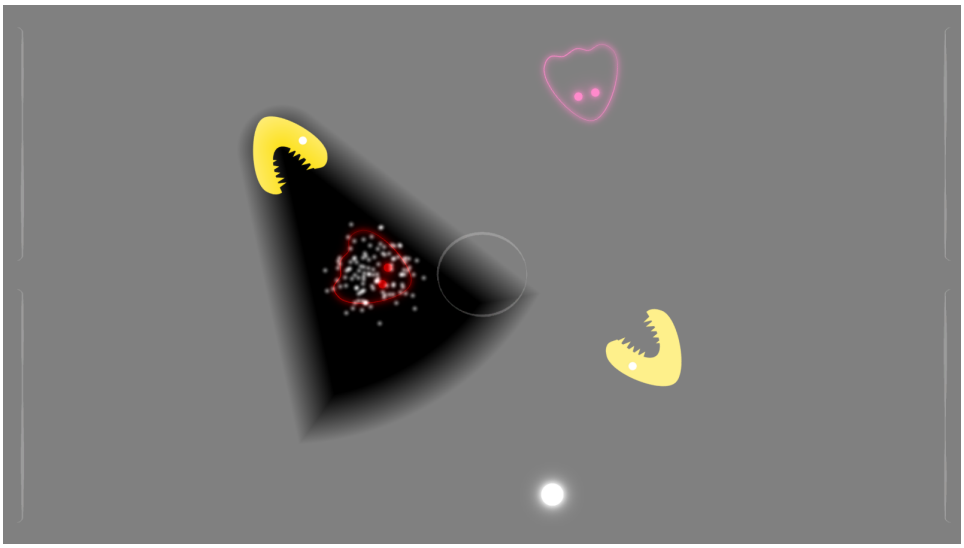


Figure 4.3: Example situation of the implemented game. Two players (yellow) are chased by two enemies (red and purple). The objective is to avoid the enemies and collect the white dots. The purple enemy receives damage because one player is looking at it. Everything outside the black area will be invisible to the player.

enough and looking directly at them. They also produce a virtual sound. The difference to the enemy entities is that these items do not change their position. If a player can find the collectible and reach its position, it disappears and the player is awarded one point. At this moment, a new collectible item is spawned at a random location.

Players' points are associated to the smartphone devices as long as the client application is running. Thus, a player can exit and enter the co-located area at any time and does not lose the progress made in the game.

4.4.3 Game example

Figure 4.3 displays a screenshot of this game with two players, two enemies and one collectible item. It shows what the player in the top left would see on the client device's screen. The invisible area is shown half-transparent here; in the actual game, everything outside of the black triangle in front of the player would be completely invisible. Thus, all entities except the red enemy would not be visible to the player until moving near enough. On the projection display, nothing would be greyed out, but the enemies and collectible items would not be visible at all. Players need to try to locate the enemies by hearing instead of by seeing.

Chapter 5

Implementation

The concept introduced in chapter 4 has been implemented as part of this thesis. It includes the example game described in section 4.4 as well as a software library to create new games or applications using the concept and also the required hardware.

As this project uses features of the *Ars Electronica Deep Space* system (see section 2.4), some parts of it did not have to be developed newly. This includes the laser tracking system *Pharus* for user position detection and the video projectors for visual output. These systems have been available both in hardware and software. Furthermore, a software library for creating three-dimensional audio using binaural techniques via headphones, called *RealSpace3D*, was used, so no additional binaural methods had to be implemented manually.

This chapter describes the various parts of the system which were developed with it. This includes the main server application, the client application running on mobile devices (smartphones) and the custom orientation measurement module.

5.1 The server application

The server application is the central part of the project. It is executed on a personal computer and communicates with all other parts of the system. It is responsible for maintaining the game state, reacting to user input and producing output. It is developed with the *Unity* game engine, since software libraries for interacting with the *Pharus* system are available to integrate with *Unity* projects. The following describes the responsibilities and tasks of the server application.

5.1.1 Game state and game logic

The server application is responsible for maintaining the state of the virtual world, or game state. This depends on the actual use case, but in most cases the game state is made up of different types of entity objects, like players, enemies, items etc. These entities interact with each other and some can be controlled by the users or interacted with by some way from the outside. These interactions are referred to as game logic.

Visual output

Normally the entities have a visual representation, either in 2D or 3D. In the example game described in section 4.4, the player, enemy and collectible objects are visualized as 2D sprites. A global representation of the game state is displayed with the video projectors provided by the Deep Space system. The player objects are displayed at the actual positions of the users using floor projection, and another or the exact same image of the game state can be displayed using the wall projection.

Audio output

The game server itself does not output any sounds, since each user is provided customized audio by the client application, and using global audio might have a negative impact on the perception of the customized directional audio.

5.1.2 Communication with the laser tracking system

The laser tracking devices send their raw sensor data to a server application via Ethernet. This server application, called *Pharus* [25], handles the raw data and computes the positions of each user inside the co-located area. The *Pharus* server can run on the same or a different machine as the game server application. It sends data about the user positions via UDP in a format called *TUIO* (tangible user interface objects) [15] and can be configured to transmit it to one or more receiving devices.

To receive the *TUIO* data provided by *Pharus*, a software framework for the Unity game engine is provided. It listens to a specified UDP port, processes incoming data and provides a list of user objects containing their positions relative to the boundaries of the co-located area. It also provides methods for reacting to *TUIO* objects being added, moved and removed, which can be overridden by the game application. The application is then responsible for using this data to interact with the game state in some way. The following code showcases how to create, maintain and display game objects representing the co-located users at each of their positions using the provided *TUIO* framework.

```
1 abstract public class ATuioPlayerManager :
2     UnitySingleton<ATuioPlayerManager> {
3
4   protected Dictionary<long, ATuioPlayer> m_playerMap =
5     new Dictionary<long, ATuioPlayer>();
6
7   public virtual void AddPlayer (long sessionID,
8     Vector2 position, ATuioGameManager currentGameManager) {
9
10    GameObject playerObj = GameObject.Instantiate(
11      currentGameManager.m_PlayerPrefab);
12
13    ATuioPlayer player = playerObj.GetComponent<ATuioPlayer>());
14
15    player.transform.position = new Vector3(
16      position.x, position.y, 0);
17      m_playerMap.Add(sessionId, player);
18  }
19
20  public virtual void UpdatePlayerPosition (
21      long sessionID, Vector2 position) {
22    m_playerMap[sessionID].MoveTo(position);
23  }
24
25  public virtual void RemovePlayer (long sessionID) {
26    ATuioPlayer player = m_playerMap[sessionID];
27    GameObject.Destroy(player.gameObject);
28    m_playerList.Remove(sessionID);
29  }
30 }
```

The code assumes a prefab (prefabricated game object, a template for new game objects in Unity) for player objects exists and is set in the *GameManager* object. The method *AddPlayer* is called when a new user enters the co-located area. The method *UpdatePlayerPosition* is called whenever a new position is calculated and transmitted by the *Pharus* server for one of the users. The method *RemovePlayer* is called when a user exits the area. The provided parameter *sessionID* always represents one of the uers. All classes referenced in this code are provided by the TUIO game framework and the default behaviour can also be changed freely.

5.1.3 Communication with the client application

The client application is responsible for providing customized visuals and audio for each user, and for allowing users to interact with the system in ways additional to their co-located positions. To achieve this, the relevant parts of the game state needs to be transmitted to the clients and interactions need to be sent back.

Game state synchronization

To synchronize the game state from the server to the clients, Unity's built in game multiplayer network system, called *UNet*, is used. Using *UNet*, game entities can be marked as network entities. This causes their positions, rotations and more to be synchronized automatically from the server to the client applications. These network entities can only be created, updated and removed by the game server. In the example game, the types of network entities are the TUIO users, called *TuioPlayer*, the enemies and the collectible items.

Additionally to this automatic entity synchronization, method calls can be sent from the server to the client. For this, a method on a Unity script needs to be tagged with the *ClientRpc* (short for client remote procedure call) attribute. When such a method is called on the server, it is automatically called on all clients with the same arguments at the same time. This can, for example, be used to trigger the playback of a sound effect on all clients. The following code shows an example of such an RPC method. Note that any such method must start with the *"Rpc"* prefix.

```
1 [ClientRpc]
2 public void RpcPlayEnemyKilledSound(bool killed) {
3   GetComponent<AudioSource>().LoadAudioClip(
4     killed ? enemyKilledSound : killedByEnemySound);
5   GetComponent<AudioSource>().PlaySound();
6 }
```

Client interactions

There are two ways to send interactions the other way around, from the client to the server. Both ways involve a network object with a "player authority". This authority is associated with one of the connected clients. The authorized client then takes control of this object, and changes to position, rotation etc. are automatically sent from this client to the server, which then updates the game state and transmits it to all other clients.

For each connected client, a network object of type *ClientPlayer* is created. These automatically get their authorities set to the respective client by the game engine. This is done by setting the *ClientPlayer* prefab as player prefab in Unity's network manager object and enabling the "auto create player" option. Figure 5.1 shows a screenshot of the example game in the Unity editor with the relevant section for this configuration. It also shows the three types of network entities available. The system automatically associates each *ClientPlayer* object, representing a connected client application with one *TuioPlayer* object, representing one co-located user. This association is not always present the other way around, since users can enter the co-located area without a client device.

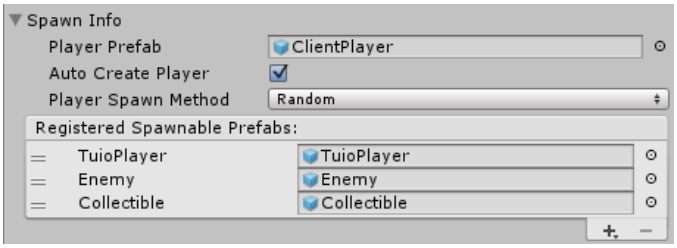


Figure 5.1: Spawn info section of Unity's network manager in the example game. The three types of network entities are configured, and the *ClientPlayer* is set as the automatically created player prefab.

Additionally, so-called command methods can be used, which are the exact opposite to the previously mentioned RPC methods. A command method can be called from a client, and is then automatically invoked on the server. These methods can also only be called on objects marked with player authority. All command methods must start with the *"Cmd"* prefix. This code shows one such command method to update some player preferences from the client:

```
1 public string UserID { get; private set; }
2 public string GyroMode { get; private set; }
3
4 [Command]
5 public void CmdSetUserClientPrefs(string userID, string gyroMode) {
6   UserID = userID;
7   GyroMode = gyroMode;
8 }
```

5.2 The client application

The client application mainly acts as a means of distributing the directional audio to each user. It is also used to deliver customized visual output to each user and enables users to interact with the system. This application is also developed with the Unity game engine and runnable on most modern smartphone devices capable of Wi-Fi. This section describes the responsibilities of the client application and how they are implemented.

5.2.1 Connecting to the game server

To be able to interact with the game server, a connection has to be established. For this, the device running the client application must be in the same Wi-Fi network as the server machine. When the client application is started, an input field to enter the server's IP address is provided. Once the connection is established, the client device is ready to use.

5.2.2 Receiving game state

As described in the previous section, the game server is responsible for maintaining the game state and running the game logic. The clients also have a copy of the game state, which is permanently transmitted from the server when some part of it changes. This way, a representation of the game state can be presented to the user of the client device in form of visual and audio output.

5.2.3 Delivering directional audio

The directional audio is the most important part of the client application, since it is the main goal of this thesis. Directional audio is created using the *RealSpace3D* audio engine and a pair of headphones connected to the client device. It is an extension to Unity's built in audio system.

Exactly one *AudioListener* object can be present in the game world. It can be attached to any game object. The directional sounds will appear to be perceived at this game object's position. Any number of *AudioSource* objects can then be placed in the scene. They are virtual sound sources that can also be attached to game objects. The sounds they produce will appear from the relative direction from the audio source to the audio listener.

The client application automatically attaches one audio listener to the *TuioPlayer* object associated with its client authority object. This way, the virtual sounds will be perceived from the user's current position inside the co-located area. The following lines of code are used inside the *TuioPlayer* class to attach the audio listener to the player entity:

```
1 var audioListener = FindObjectOfType<
2     RealSpace3D_AudioListener>().gameObject;
3 audioListener.transform.parent = this.transform;
4 audioListener.transform.position = this.transform.position;
```

The *RealSpace3D* engine uses a database of HRTFs for creating the directional audio. For each sound source, the relative direction to the sound listener is calculated, and the best matching set of HRTFs for each ear is selected from the database. These two HRTFs are then applied to the sound source's audio signal (a one-channel audio file). The two created signals are finally output via the stereo headphones. The engine also provides the option to select a different database of HRTFs. It provides five such databases measured under different conditions. The documentation however claims that the default setting works for 80% of users, which is what has been used when testing the implementation.

5.2.4 Reading orientation

Finally, the client is responsible for obtaining the user's orientation inside the co-located area and transmitting it to the server. For this, the orienta-

tion is applied to the client's authority game entity. The server then rotates the associated player entity accordingly. There are two different ways implemented to receive the orientation, one of which can be selected when starting the client application.

Smartphone sensors

The first and simple way is to use the client device's built-in orientation sensors. Most smartphones contain sensors for measuring their orientation, i.e. accelerometer, gyroscope and magnetometer. The Unity engine provides a simple way of accessing this orientation as a quaternion. This quaternion however uses a right-handed coordinate system, while Unity uses a left-handed one for all their entities. This can be converted by exchanging the z- and the y-axis of the corresponding Euler rotation:

```
1 Quaternion getBuiltinOrientation() {
2   //gyro rotation uses right handed coordinate system, unity left handed
3   var gyro = Input.gyro.attitude.eulerAngles;
4   return Quaternion.Euler(gyro.x, gyro.z, gyro.y);
5 }
```

Custom orientation module

A different way to obtain the orientation of a user has been implemented using a custom sensor module. The implementation of this module is explained in section 5.3. This module broadcasts the calculated orientation as a quaternion via UDP in 16-byte data packages. Each of these packages contains four 4-byte floating point values making up the quaternion (w, x, y, z). The quaternion can be reconstructed the following way:

```
1 Quaternion getCustomOrientation() {
2   //receive broadcast from any device at port 5741
3   IPEndPoint receiveEndPoint = new IPEndPoint(IPAddress.Any, 5741);
4   UdpClient receiveClient = new UdpClient(receiveEndPoint);
5
6   byte[] recvData = receiveClient.Receive(ref receiveEndPoint);
7
8   int i = 0;
9   float w = BitConverter.ToSingle(recvData, i);
10  float x = BitConverter.ToSingle(recvData, i += 4);
11  float y = BitConverter.ToSingle(recvData, i += 4);
12  float z = BitConverter.ToSingle(recvData, i += 4);
13
14  return new Quaternion(x, y, z, w); //w comes last here
15 }
```

5.3 Custom orientation measurement module

To provide realistic directional audio, a co-located system is required to be able to determine the direction its users are looking at. When a user turns around, the virtual sounds need to change direction as well. To solve this problem, an orientation sensor device is needed. One solution is to use the sensors included in the mobile devices on which the client application is running.

However, as explained in section 4.2, there are some problems with this approach. These problems can be solved by instead attaching an orientation sensor device onto the headphones worn by the users. Since no devices which seemed suitable for this task were available when this project was implemented, a custom solution has been created.

5.3.1 Goal and requirements

The goal of this task was to be able to measure which direction a user's head is facing, in order for the system to react to users turning their heads by adjusting the direction of the virtual sounds. For this, a device was needed which could measure its absolute orientation. This device would have to be small and versatile enough so it could be attached to regular headphones. Furthermore, it should be able to operate on its own so no additional cables are required. The values measured by its sensors have to be transmitted wirelessly to the consumer application.

The orientation measured by this device needs to be absolute in relation to some reference orientation. It should always provide the same results when used under the same circumstances, i.e. in the same position and with the same rotation. The orientation should not be subject to drift, which means it should stay accurate over time and not change if the sensor does not change orientation.

5.3.2 Hardware

Various pieces of consumer hardware have been used in combination to create a device fulfilling the above mentioned requirements.

The orientation sensor

At least two reference directions are needed to measure absolute orientation. This can be achieved by using two different sensors. An accelerometer can provide a reference direction to Earth's center by measuring the force vector of gravity. A magnetometer can measure the magnetic field directed to Earth's magnetic North or South Pole.

There are different solutions combining such sensors available on the market. Most of these use a third sensor, namely a gyroscope, to improve

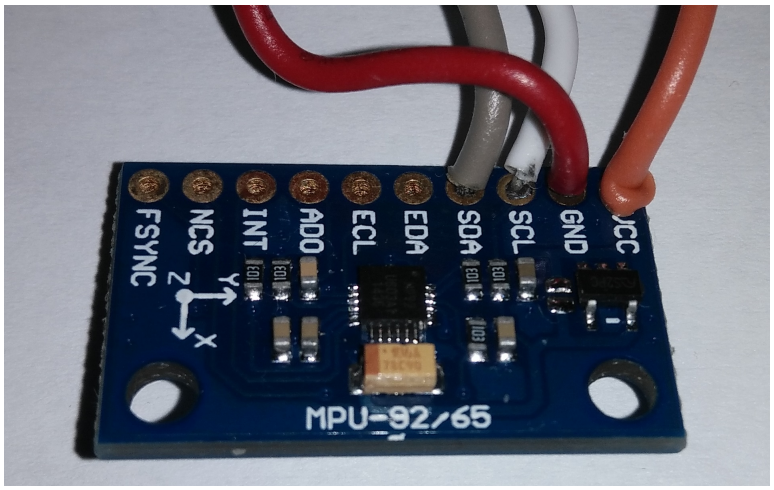


Figure 5.2: The *MPU-9250* orientation sensor containing accelerometer, gyroscope and magnetometer on a circuity board connected via IIC interface.

the accuracy of the result. A gyroscope measures the change in orientation over time and can be used to mitigate inaccuracies of the other sensors.

One such multi-sensor device by the company *InvenSense*[1], called *MPU-9250*, was used to create the orientation measurement module. It is available on a small circuity board, about 1 cm × 2 cm in size. The board contains 10 connector pins to be able to communicate with it, either via the inter-integrated circuit (IIC) or serial peripheral interface (SPI) protocol. To communicate via IIC, only four of those pins have to be connected for default operation. Figure 5.2 shows a photograph of the MPU-9250 circuity board with the four IIC pins connected.

The microcontroller

A microcontroller unit (MCU) is necessary to read orientation sensor measurements and transmit them wirelessly. This microcontroller is required to be able to communicate with other devices via IIC interface and to support Wi-Fi for transmission.

The most suitable hardware for this task was the "*D1 mini*" by *WeMos*[2]. It is a relatively small, about 2 cm × 4 cm, MCU board capable of IIC, Wi-Fi and more. It can operate with voltage as low as 3.3 V.

Figure 5.3 shows the top and bottom side of the *WeMos D1 mini* board. In the top view, the 2.4 GHz Wi-Fi chip is visible. On the left and right are 16 connector ports which can be used for various interfaces, including IIC.

[1] *InvenSense* website: https://www.invensense.com/
[2] *WeMos* website: http://www.wemos.cc/

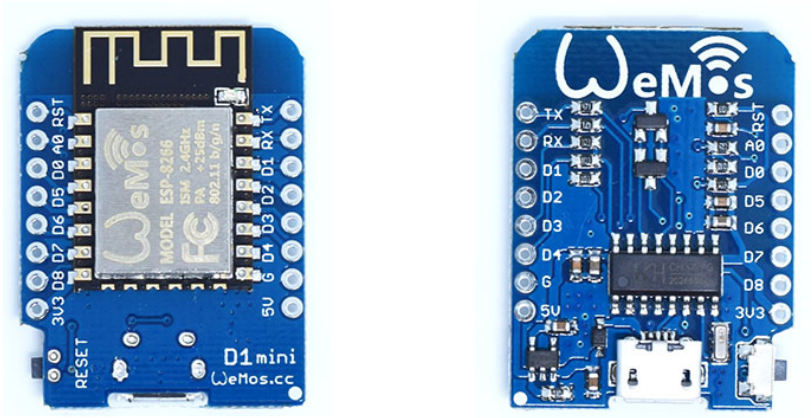


Figure 5.3: The *WeMos D1 mini* MCU board from above (left) and from below (right).[3]

Power supply

In order for the hardware module to be truly wireless, a power supply must be included. The WeMos D1 can operate with a voltage between 3.3 V and 4.2 V. A module called "battery shield" is available that can be attached to the *D1 mini*. It allows connecting a rechargeable lithium battery via a JST-XH 2.54 inch connector. The battery can be charged by connecting a micro-USB cable to the battery shield. When the USB cable is not connected, it acts as power supply for the MCU by supplying the required voltage.

As actual power supply, a lithium-ion battery in default AA format has been used. It supplies a voltage 3.7 V with a capacity 800 mAh. According to the *WeMos* website, the *D1 mini* consumes about 70 mA when fully operating, which would allow for more than 11 hours of operation.

Overview

All the previously described pieces of hardware together allow the orientation measurement module to fully operate on its own. The *MPU-9250* sensor board is connected via four cables to the *WeMos D1 mini* using IIC interface. The battery shield is attached to the MCU and connected to a rechargeable lithium-ion battery. A photograph of the complete orientation sensor module on its own is shown in Figure 5.4. The module can be attached to any headphones to be used with in the co-located environment. In Figure 5.5, it is attached to a pair of *Philips SHB-7250* headphones.

[3]Images from *WeMos* website: http://www.wemos.cc/Products/d1_mini.html.

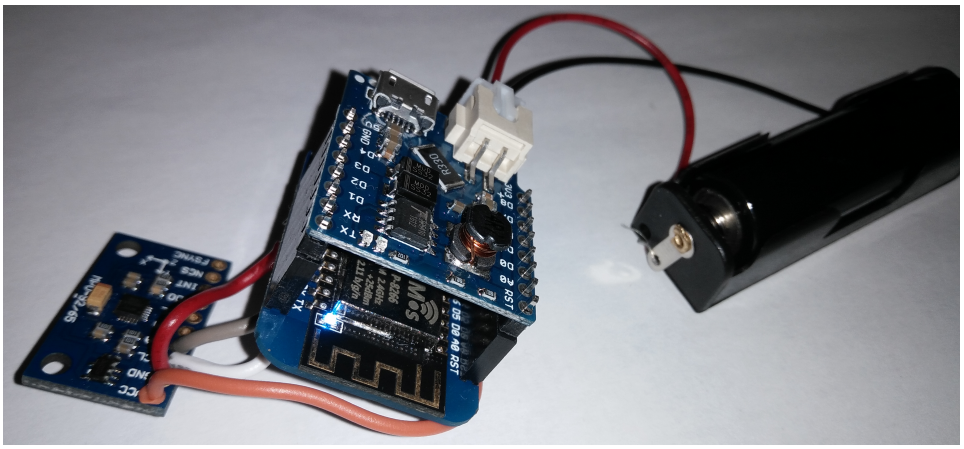


Figure 5.4: The custom orientation sensor module consisting of the *MPU-9250* accelerometer, gyroscope and magnetometer sensor unit (bottom left), the *WeMos D1 mini* microcontroller including battery shield (centre) and a rechargeable Lithium-ion battery (right).

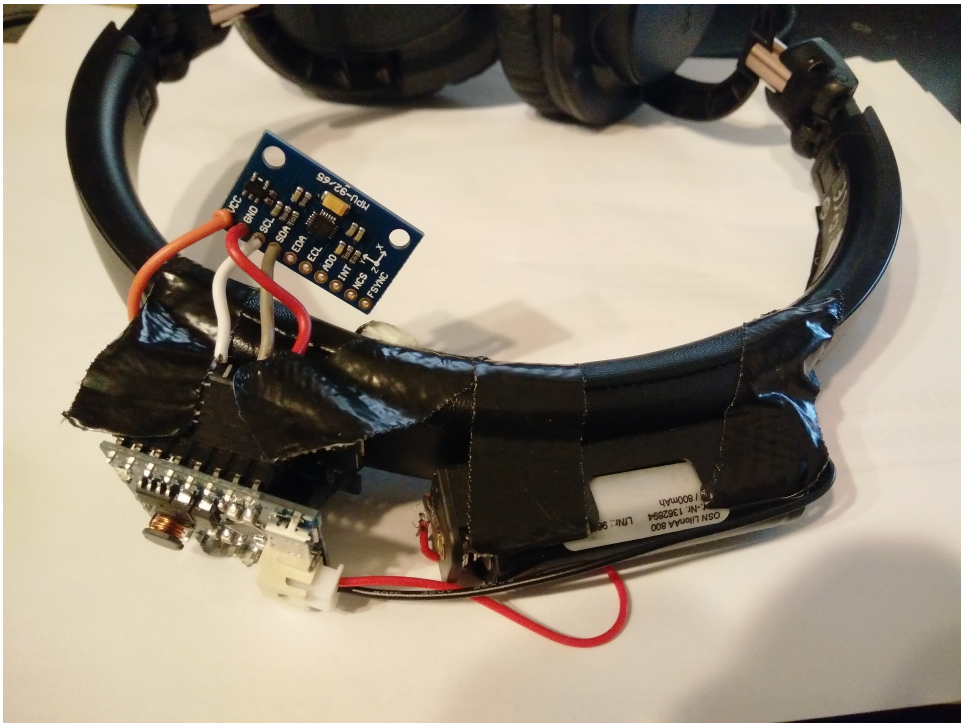


Figure 5.5: The custom orientation sensor module attached to a pair of *Philips SHB-7250* headphones.

5.3.3 Software

To complete the autonomous gyroscope module, a program for the MCU was implemented. It reads sensor data via the IIC port, processes it into a suitable format and transmits it via Wi-Fi to the consumer application.

The program is developed using the *Arduino*[4] platform in the programming language C++. It consists of four general parts:

- *Setup*: connect to Wi-Fi network, initialize IIC connection and check for potential problems.
- *Read sensor data*: read raw data provided by *MPU-9250* via IIC.
- *Calculate orientation*: get absolute orientation by combining readings from accelerometer, magnetometer and gyroscope.
- *Transmit orientation*: transmit calculated orientation in a suitable format via Wi-Fi.

The first of these parts is executed once when the MCU is turned on, the other three are called consecutively in an endless loop afterwards. At the end of each loop, the runtime of the loop is measured. Then the MCU is put into wait mode for an amount of time so the number of orientation broadcasts per second does not exceed a defined limit. This can also reduce battery usage of the module. The implementation of the four parts will be elaborated in the following sections.

Setup

In this step, the Wi-Fi connection is established. Credentials for the Wi-Fi network are stored in program memory, so when the sensor module is to be used in a new network, they need to be modified and program needs to be re-deployed onto the hardware. An Arduino code library for this task is available, and connecting to a Wi-Fi network can be done by a single call, where the variables *ssid* and *pass* are character-arrays containing the network credentials:

```
1     WiFi.begin(ssid, pass);
```

Additionally, the MPU-9250 device is initialized in this step. First, the IIC connection is established and verified. If no communication is possible, an error is reported and the MCU will restart. In this case, the IIC wires need to be examined for problems.

Next, there are a number of configurations that have to take place before the sensor data can be acquired. Each of the three individual sensors must be enabled specifically. Also, the scale of the output values can be set to one of various pre-defined options for each sensor. This will affect the quality of the orientation result.

[4] Arduino project website: https://www.arduino.cc/

Read raw sensor data

The *MPU-9250* sensor board provides raw sensor readings via IIC interface. Using IIC, data can be requested from hardware register numbers on devices identified by a 10-bit address. The *MPU-9250* actually contains two such devices. The accelerometer and gyroscope readings can be accessed at device address `0x68` as vectors consisting of each three elements. The accelerometer data vector starts at register address `0x3B`, the gyroscope data vector at `0x43`. The magnetometer readings can be accessed the same way at device address `0x0C` starting at register address `0x03`. The reason for there being two different device addresses is that there are actually two different MCUs embedded on the *MPU-9250* board.

Each element of such a raw vector is a 2-byte numbers with an effective range of -32768 to 32767. These values need to be scaled to the actual range of the sensors, which have been configured separately from a list of available ranges for each sensor in the setup step. For example, the following piece of C++ code calculates the acceleration vector in multiples of $g = 9.81\,\mathrm{m/s^2}$, assuming the accelerometer sensor is configured to a full range of $16g$:

```
1  void readAccelData() {
2      uint8_t rawData[6];    //raw bytes from I2C
3      int16_t rawValues[3]; //actual values between -32768 and 32767
4      float accel[3];        //actual acceleration in g
5
6      //full range of acceleration sensor is 16g
7      const float sensorRange = 16.0;
8      const float factor = sensorRange / 32768.0;
9
10     //read six bytes via I2C from device 0x68 at address 0x3B
11     I2CreadBytes(0x68, 0x3B, 6, &rawData[0]);
12
13     //combine each 2 bytes of raw data into one 16-bit integer (little-endian format)
14     rawValues[0] = ((int16_t) rawData[0] << 8) | rawData[1];
15     rawValues[1] = ((int16_t) rawData[2] << 8) | rawData[3];
16     rawValues[2] = ((int16_t) rawData[4] << 8) | rawData[5];
17
18     //calculate actual acceleration in g
19     accel[0] = rawValues[0] * factor;
20     accel[1] = rawValues[1] * factor;
21     accel[2] = rawValues[2] * factor;
22
23     //do something with acceleration values...
24 }
```

Calculate absolute orientation

To calculate the absolute orientation of the sensor device, the readings from all three different sensors can be combined. Each sensor provides three-dimensional element vectors representing their current readings.

For the accelerometer, this is a vector $A = (a_x, a_y, a_z)$ pointing to Earth's center in multiples of $g = 9.81\,\mathrm{m/s^2}$, the magnitude of gravity, as long as the sensor experiences no additional acceleration. The coordinate system used is relative to the sensor circuity board with the z-axis pointing upwards from the board. This coordinate system is printed on the board as visible in Figure 5.2.

The gyroscope returns a vector $G = (g_x, g_y, g_z)$ containing the current rate of change in orientation around the x, y and z-axis in degrees per second, using the same coordinate system as before.

For the magnetometer, the raw values are the vector $M = (m_x, m_y, m_z)$ containing the strength of magnetic field in x, y and z-direction in mG (milli Gauss). The magnetic field in M points towards Earth's magnetic North Pole, if no artificial magnetic fields are present. The coordinate system used here has its x- and y-axis switched in comparison to the other two sensors, so m_x and m_y should be exchanged in all following calculations.

Additionally, though, the magnetometer readings will always include constant values, called bias. So the actual reading returned from the sensor will be

$$M_b = (m_x + b_x, m_y + b_y, m_z + b_z). \tag{5.1}$$

Including the bias, the magnetometer readings are rather useless. Fortunately, the bias values do not change with sensor orientation. Thus, a calibration procedure can calculate the bias and the actual magnetic field vector can be approximated. To do this calibration, the sensor needs to be rotated a few times in all directions, so the maximum and minimum values in all three axes can be acquired. Then, the bias vector can be calculated as

$$B = (b_x, b_y, b_z) \approx \left(\frac{x_{\max} - m_{\min}}{2}, \frac{y_{\max} - y_{\min}}{2}, \frac{z_{\max} - z_{\min}}{2} \right). \tag{5.2}$$

The actual magnetic field can then be approximated as $M = M_b - B$.

After all three sensor vectors A, G and M have been acquired, the absolute orientation can be calculated by combining them using a so-called sensor fusion algorithm. There are many different algorithms available for this task. Two of which, called *Madgwick's* and *Mahony's* sensor fusion algorithm [24] have been used in this project, since they claim to be able to perform in real time even on microcontrollers and still produce viable results. Both of these algorithms perform the same task, in different ways: They combine the three vectors to approximate the current absolute orientation of the sensor. In general, they use the acceleration and magnetic field vectors as reference direction to approximate the current orientation. The rate of change in orientation from the gyroscope is then used to smoothen inaccuracies of the other sensors, at the cost of response time to fast orientation changes.

The absolute orientation in three-dimensional space can not be represented as a single 3D-vector, since there would still be one degree of freedom

available: rotating the vector around itself. Thus, quaternions are used to represent orientations. Quaternions are of the form

$$q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}. \tag{5.3}$$

A quaternion can contain an axis (x, y, z) and an angle θ of rotation. From these it can be calculated as

$$q = \cos\left(\frac{\theta}{2}\right) + x\sin\left(\frac{\theta}{2}\right)\mathrm{i} + y\sin\left(\frac{\theta}{2}\right)\mathrm{j} + z\sin\left(\frac{\theta}{2}\right)\mathrm{k}. \tag{5.4}$$

Both *Madgwick's* and *Mahony's* sensor fusion algorithm can calculate such a quaternion from the three sensor vectors. They are both only effective when called many times per second with new sensor readings and the time interval, or delta time, between the current and the last sensor reading, since the gyroscope's readings are integrated numerically over time. There are also free parameters that can be tuned to change the responsiveness to orientation changes in contrast to the error correction efficiency.

An implementation in C++ of both of these algorithms was available and did not have to be implemented newly.[5] Both these implementations have been used with the *MPU-9250*, and *Madgwick's* seemed produce the more accurate results.

Transmit orientation

To transmit the calculated orientation values to a consumer device, the user datagram protocol (UDP) and broadcast is used. This way, data can be sent continuously to all devices in the same network. Any device can open a UDP socket and read it. UDP is known for being fast at the cost of not guaranteeing that every sent datagram will be received by all clients in the original order or at all. Since new sensor readings are being sent in constant intervals, not receiving all of them will not be a big problem.

To communicate with UDP, 16-bit a port number needs to be agreed upon. The port used for communication with the orientation measurement module has been arbitrarily set to 5471. Using this port number, a UDP client can be opened in the MCU program the following way:

```
1 //global:
2 WiFiUDP Udp;
3 const int udpPort = 5741;
4
5 //in setup method:
6     Udp.begin(udpPort);
```

[5] Sensor fusion algorithms provided by Kris Winer on Github: https://github.com/kriswiner

Once the client is open, datagram packages can be built out of any number of bytes and then sent to a 32-bit destination IP-address. The IP-address 255.255.255.255 can be used instead to send the datagram to all clients in the current network. The following code can be used to send a quaternion object consisting of four 4-byte floating-point values to all other devices via the previously opened UDP client:

```
1 void sendGyroData(const quaternion &att_quat) {
2     //~0 = 255.255.255.255, broadcast on this network
3     unsigned int sendIP = ~0;
4     Udp.beginPacket(sendIP, udpPort);
5
6     //send quaternion data in byte form:
7     Udp.write(reinterpret_cast<const uint8_t *>
8         (att_quat.quatData()), 4 * sizeof(float));
9
10    Udp.endPacket();
11 }
```

5.4 Problems and challenges

As described in this chapter, the whole concept from chapter 4 has been technically implemented and a fully working game using binaural audio in a co-located multiplayer environment has been developed. However, this whole process was not without problems, and there were several challenges to be faced throughout it.

5.4.1 Orientation drift

Even though, in theory, an absolute orientation can be calculated using accelerometer and magnetometer sensor, and even a gyroscope sensor has been used to improve the result's accuracy, the measured orientation of the custom orientation measurement module turned out to still be subject to drift. When it was used in the example game, the player entity's rotation would sometimes change slowly even when the users were not rotating their heads. Thus a method to re-align the in-game orientation to the user's real orientation was still necessary.

One possible cause for this problem is that a custom solution for the whole process of obtaining the absolute orientation, including reading raw sensor data and performing sensor fusion, had to be implemented. There actually exists proprietary software performing sensor fusion with the *MPU-9250* sensor, it however was not able to be executed on an *Arduino* microcontroller, thus also not on the used *WeMos D1 mini* MCU. There might be small calibration errors in the custom code causing the drift.

Another possibility, assuming the orientation calculation code is flawless, is that sensor readings were inaccurate. As mentioned in section 2.3, readings

from accelerometer and magnetometer are only accurate if the sensors are at rest and no artificial magnetic fields are present. If, for example, magnetic fields exerted by electronic devices were present in the testing environment, they could cause the sensor readings to be inaccurate and consequently cause the drift.

5.4.2 Instability of the sensor module

The orientation measurement module was developed using MCU breakout boards and was then attached to headphones without being contained in a protective case. This caused the soldered wires between the *MPU-9250* sensor board and the *WeMos D1 mini* MCU board to loosen and detach a few times when transporting the headphones. Thus, they had to be re-soldered a few times.

Chapter 6

Empirical study and evaluation

Using the implementation of the example game from section 4.4, an empirical study was performed by allowing different users to play through a pre-defined scenario using parts of the game's features and recording data about their performance. From this data, statistical evaluations were performed to find out which parts of the game worked well and which ones presented more challenges to the users.

Section 3.2 stated that sounds which spread over big parts of the frequency spectrum can be localized better than sounds which only occupy a small part of it. The test scenario contained different sounds that were to be localized by the users in order to be able to find out if this assumption is actually valid.

Furthermore, the study should also show which relative directions are best and worst suitable for localizing sound sources. Also, since there are visible and invisible enemies in this game, the study can statistically show which ones were easier to find and defeat for the players. Finally, the study can evaluate if using the custom orientation measurement module can actually improve the players' performance in finding enemies by listening.

6.1 Setup

In this study, users were asked to play through a pre-defined scenario of the example game described in section 4.4. The main parts of the game remained the same, with a few differences. The goal was to provide a test scenario that is exactly the same for each user, and that can be used to statistically evaluate data about their performances. For this, the following limitations to the game have been made:

1. Players are asked to only stay inside a relatively small rectangle in the

center of the co-located area. Players should just focus on localizing the direction incoming enemies instead of walking around in the game. When a player leaves this rectangle, the game is paused.

2. Only one enemy attacks the players at a time, and only one player can play at a time.
3. Enemies are always spawned in the same order of combinations of the following properties: relative direction (angle from the player to the enemy), the sound effect used, enemy colour, chasing algorithm and visibility. They are always spawned at the same distance from the player. This way, each player is faced with the exact same challenge. In total, there are 32 enemies spawned in order, one with each combination of four relative directions, four colours and either visible or invisible. The sound effect and chasing algorithm are tied to the colour.
4. Collectible items are not used, since players are not allowed to walk around and should only concentrate on locating the enemies.

Instead of collecting points, the goal of the study game is to play through the whole scenario of 32 enemies, regardless of how many enemies are defeated. Players are shown a progress bar on the screen of the mobile device so they are aware of how many enemies are still to be defeated.

6.2 Recorded data

The study procedure described above was performed with 16 test users, 5 of which using the custom orientation measurement module and 11 using the default mobile device orientation sensors. Each test user was asked to complete the study procedure of 32 enemies. For each user and enemy, data about the parameters, performance and efficiency was recorded, resulting in 512 data sets of various properties that can be grouped into two kinds.

6.2.1 Configuration and enemy properties

These properties describe the circumstances of the test and appearance and behaviour of the enemies:

- *Time stamp*: Global time on the game server when the enemy was spawned.
- *User ID*: Anonymous identification code for the user, to be able to select all data sets of a single user.
- *Orientation mode*: Flag indicating the used orientation measurement, either mobile device orientation or custom orientation measurement module.
- *Sound effect name*: Name of the enemy's sound effect.
- *Colour*: Enemy colour.

- *Spawn angle*: Angle between the player and the enemy at which it was spawned.
- *Invisible*: Boolean flag indicating if the enemy was invisible or visible.

6.2.2 Player performance properties

These properties give information about the player's efficiency in the game:

- *Number of enemy deactivations*: Number describing how many times the player looked at the enemy, deactivating its movement, before the enemy was defeated or hit the player.
- *Time until first deactivation*: Real time between spawning the enemy and the player first finding and looking at it. If the enemy was not found at all, this property will be zero.
- *Time until last deactivation*: Real time between spawning the enemy and the player last looking at it before it was removed. If the enemy was not found at all, this property will also be zero.
- *Time until removed*: Real time between spawning the enemy and removing it, either when it was defeated or hit the player.
- *Defeated by player*: Boolean flag indicating if the enemy was defeated by the player, or if it did hit the player.

6.3 Results and data analysis

The data was recorded as a CSV (comma-separated values) file by the game server while the test users were playing the game. These files were then imported into an *SQLite* database, from which statistical operations can be performed to evaluate it from various perspectives.

To find out which game parameters worked better or worse for players in general, the player performance properties can be evaluated. Table 6.1 shows the general boundaries, averages and variances for those properties, as calculated over the complete testing data set for all enemies that were defeated by the players. It shows, for example, that it took the players on average 7.91 seconds to find each enemy and 13.4 seconds to defeat them. It is worth noting, however, that in the test game, it takes the enemies 6.25 seconds to reach a distance to the player where they can first be seen, and 4 seconds of looking directly at the enemies until they are defeated. Assuming the players do not change position, this means on average players find enemies 1.66 seconds ($7.91\,\mathrm{s} - 6.25\,\mathrm{s}$) after them being in the visible range. The table also shows the proportion of enemies that were defeated by the players, which is 92.86%, thus only 7.14% of all enemies could not be found until they reached the players.

Property name	Min value	Max value	Mean value	Median value	Variance
Number of deactivations	1	11	2.05	1	2.25
Time until first deactivation	2.19 s	16.51 s	7.91 s	7.40 s	6.19 s
Time until last deactivation	2.65 s	25.69 s	9.40 s	8.61 s	10.35 s
Time until removed	6.66 s	29.69 s	13.40 s	12.61 s	10.35 s
Defeated by player			92.86%		

Table 6.1: Statistical analysis of the recorded player performance properties of all game modes and enemy configurations, but only for enemies which were defeated by the players.

6.3.1 Direction analysis

In the study game scenario, enemies were spawned from four different directions, relative to the player: 0°, 90°, 180° and 270°, where an angle of zero degrees means directly in the front of where the player is currently looking. Assuming this angle impacts the efficiency of players finding the enemies by listening to their directional sounds, statistical conclusions can be drawn by analysing the test data set.

Figure 6.1 shows the average time it took players in the study scenario to find and defeat each enemy, grouped by the initial direction. These values are calculated as the median value of the corresponding properties of defeated enemies that were spawned with the specified initial angle. The time it takes enemy objects to enter the visible range has been subtracted. The median was used here instead of the mean to give less weight to extreme outlier values, which can occur due to players getting distracted or connection between the client and server devices being temporarily slow.

The graph highlights that enemies getting spawned directly in front of the players are found the quickest, with a median value of only 0.38 seconds. This is most likely because players do not have to move at all to find those enemies. Objects being spawned to the right (90°) and left (270°) seem to be harder to find, with a median time of 1.11 and 1.02 seconds, respectively. This is an increase in time of 192% and 168%. The most difficult objects to find turn out to be objects spawned directly behind the players with 180°, which took them 1.94 seconds, or 410% longer, on average.

This increase in difficulty is most likely due to the fact that sounds directly in the front or back are generally harder to localize than ones to

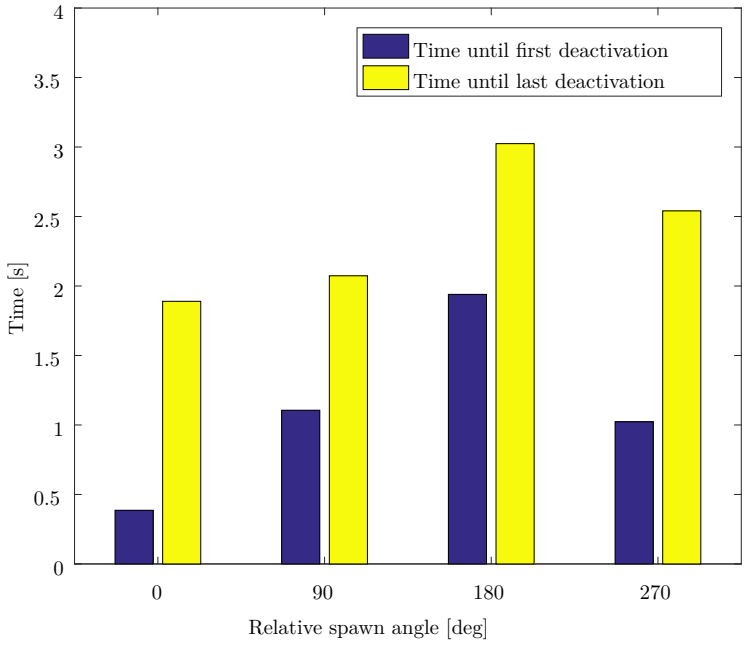


Figure 6.1: Comparison of the time it took players to find and defeat enemies, depending on the initial angle between the player and the enemy.

the left or right, since the intensity and time difference is the same for those sounds for each ear and only monaural cues can be used for localization (see also section 3.2).

Another interesting aspect shown by this analysis is that it seemed to take players more time to defeat enemies to their left than those to their right. Objects at 90 degrees took them 1.93 seconds to defeat, and those at 270 degrees took 2.39 seconds, or 24% longer. This might be caused by players more often turning clockwise than counterclockwise when searching for enemies.

6.3.2 Sound effect analysis

In the evaluated game scenario, four different sound effects with different spectral properties were used for the enemy objects. Two sound effects, which will be referred to as *All #1* and *All #2* later, were selected that span over most parts of the frequency spectrum. One sound effect, called *High*, has been filtered so that it mostly contains frequencies between 500 Hz and 1,500 Hz. The last sound effect, called *Low*, has also been filtered and mostly contains frequencies between 100 Hz and 400 Hz. This selection has been made because as stated in section 3.2, sounds that span over most of the

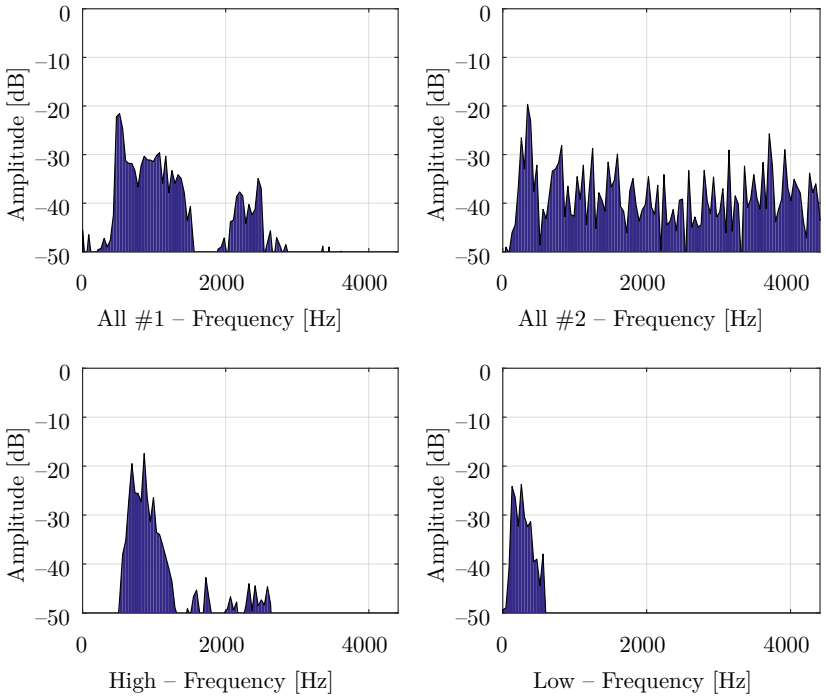


Figure 6.2: Spectral analysis of the four sound effects used in the test game.

spectrum are easier to localize than ones with a narrow spectrum. If this assumption is true, the sounds *All #1* and *All #2* should perform better than *Low* and *High*. Figure 6.2 shows a spectral analysis of these four sound effects.

As in the previous section, an analysis of the players' times in finding and defeating enemies was performed, this time grouping by the four different sound effects. Figure 6.3 shows a bar graph of the median values of these times for each of the four sound effects. The difference in these values between the sound effects is much smaller than when grouped by initial direction. Sound effects *All #1* and *All #2* performed almost equally well, with players finding those enemies after 1.04 and 1.02 seconds, on average. Enemies producing the sound effect *High* were found after 1.17 seconds on average, which is only 14.70% longer than the best performing sound. The average of this time for the sound effect *Low* was 1.30 seconds, or 27.45% longer.

From these findings it can be inferred that the actual sound effect used for objects which are to be found by localizing the sounds has a much smaller influence on the players' performances than the initial position of the virtual sound sources. Still, as assumed, sound effects with a more narrow frequency

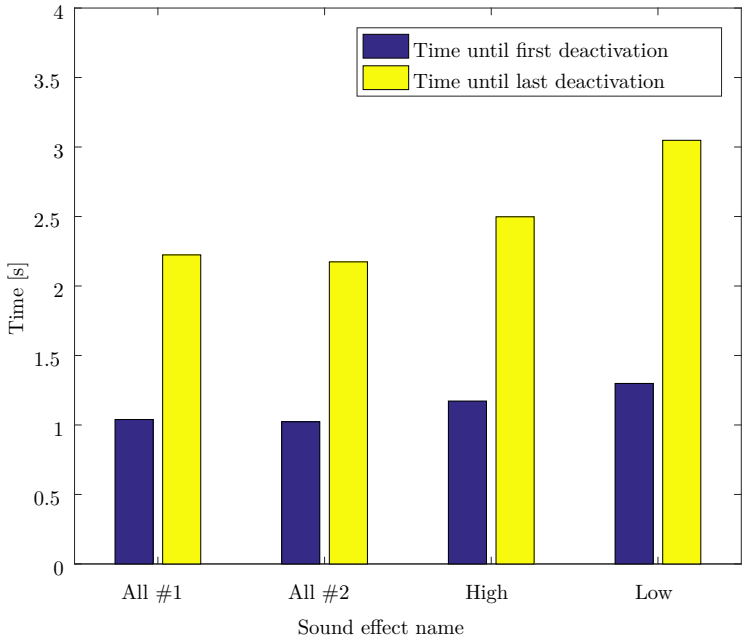


Figure 6.3: Comparison of the time it took players to find and defeat enemies, depending on the sound effect used.

spectrum performed worse, but the impact of this was not as critical as expected. However, when asked which sounds felt easier to localize after playing through the study scenario, all but one player also stated that the sound effect *Low* was the hardest one to locate.

6.3.3 Analysis of other properties

The same analysis as above can be performed focussing on other parameters of the testing process. By grouping by orientation mode, it can be calculated that users using the mobile device orientation sensors need 1.19 seconds to find the enemies, on average, compared to 0.97 seconds when using the custom orientation measurement module. This shows that the custom solution provides an average improvement of 18.49%.

The last aspect that was analysed is the enemy object's visibility. Calculating the median values of players' times until the enemies are first found results in 1.18 seconds for visible enemies and 1.10 seconds for invisible ones. This means invisible enemies were found 7.27% faster than visible ones, on average. This small improvement could originate from players looking at the screen less when the enemies are invisible and instead focussing on listening and locating the enemies' sound effects.

6.4 Summary

The calculated results of this empirical study greatly correspond to the expectations. Directional sound works better when the sound effects used span over most parts of the frequency spectrum. The detrimental effect of using sound effects that were filtered to only a small part of the spectrum was weaker than expected though.

Virtual sound sources that are created directly behind the users are harder to localize than ones to their left or right. The effect of this initial direction on the localization of the sound source turned out significant. The customly built orientation measurement module provided an improvement in player performance, most likely because the virtual player entity turning the same way users turn their heads results in a more realistic experience. Enemies being invisible helped the players in locating them, but only by a rather small amount.

Chapter 7

Conclusions

This thesis presented an overview of how binaural directional audio can improve the immersiveness of co-located multiplayer games. Existing co-located environments have been examined, if and how directional audio is already used in them. The goal was to increase the realism of an existing co-located environment, namely the *Deep Space* system, by using binaural audio to create virtual sound sources that players can locate just by listening. Users only need to wear some type of conventional stereo headphones; no special equipment is necessary.

To solve this problem, the theoretical basics of sound localization have been covered as well as methods to create locatable sounds in a virtual environment. Using this knowledge, a concept to include binaural audio techniques in an existing co-located environment has been built and technically implemented. The implemented solution supports developing applications that can use binaural audio in the co-located multiplayer environment. Multiple users can interact in this environment, and multiple virtual sound sources can be placed at any positions inside it. Users carrying a smartphone and wearing stereo headphones will hear those virtual sound sources just as if they would actually exist in this environment. Also, a custom extension for measuring head rotations that can be attached to any headphones has been developed both in hardware and software. It improves the experience by letting the virtual player entities instantly rotate when users rotate their heads. Without this extension, the player entities can only be rotated by rotating the smartphone held.

Finally, an example game using the implemented methods was created. Using this game, an empirical study was performed by collecting data from users playing a test scenario of it, to find out if and how well the developed methods are working. The study proved that the directional audio does work as expected, users were able to locate the direction of virtual sound sources by listening. It further confirmed assumptions made before development about what kinds of sounds would work better or worse for this

task. The study also showed that players using the custom-built orientation measurement module improved their performance in finding virtual sound sources, as expected.

A solution in both hardware and software that can be used to create co-located multiplayer games featuring binaural audio has been developed and presented. Using it, any kinds of applications and games that are playable in the *Deep Space* can be created. Only one such game exists as of now, which is the one discussed in section 4.4. In theory, it supports a large number of co-located users interacting with each other at the same time, yet it has only been tested with up to two users until now. Tests with more users can be performed in the future, and improvements to the experience and stability may be necessary. Furthermore, more such games can, and hopefully will, be implemented by using the outcome of this thesis as a starting point.

The custom orientation measurement module has been proven to improve the game experience. Only one prototype of this module exists as of now, and while it is fully working, there are still some problems with it, as described in section 5.4. Future works can hopefully solve these problems and produce more stable versions of that module.

Appendix A

Contents of CD-ROM/DVD

Format: CD-ROM, Single Layer, UDF-Format

A.1 PDF-files

Path: /

Bartsch2016.pdf	Master thesis

A.2 Literature

Path: /literature

*.pdf	Copies of literature; each file is named by the last name of the first author and the year published.

A.3 Source code

Path: /src

BinauralPacMan/ . . .	*Unity* project of the client and server application of the binaural Pac-Man game
arduino/	*Arduino* source code of the custom orientation measurement module

A.4 Compiled binaries

Path: /bin

BinauralPacMan.zip . .	ZIP-file containing the client and server application compiled for *Windows* 64-bit

BinauralPacMan.apk . .	APK-file of the client and server application compiled for *Android*

A.5 Study data

Path: /data

data.db	*SQLite* database file containing all data collected for the empirical study
*.csv	Original data collected for the empirical study
*.sql	*SQL* source files used for building and analyzing the database
*.m	*MATLAB* source files used for generating the graphs in this document
sqlite_extensions/* . .	Extension library for *SQLite* to be able to use the median function[1]

A.6 Images

Path: /images

*.svg, *.pdf(_tex) . . .	Vector graphics used in this document
*.png, *.jpg	Pixel graphics used in this document
*.vsd	*Microsoft Visio* diagrams used in this document

[1] SQLite extension functions:
http://www.sqlite.org/contrib/download/extension-functions.c?get=25

References

Literature

[1] V. R. Algazi et al. "The CIPIC HRTF database". In: *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*. October. New York, 2001, pp. 99–102 (cit. on p. 17).

[2] Jeroen Breebaart. "Analysis and Synthesis of Binaural Parameters for Efficient 3D Audio Rendering in MPEG Surround". In: *2007 IEEE International Conference on Multimedia and Expo*. Beijing, 2007, pp. 1878–1881 (cit. on p. 17).

[3] Andrew Brughera, Larisa Dunai, and William M Hartmann. "Human interaural time difference thresholds for sine tones: the high-frequency limit." *The Journal of the Acoustical Society of America* 133.5 (2013), pp. 2839–2855 (cit. on p. 12).

[4] Simon Carlile. *The physical and psychophysical basis of sound localization*. Springer, 1996 (cit. on pp. 10, 13, 14).

[5] Edgar Y. Choueiri. "Optimal crosstalk cancellation for binaural audio with two loudspeakers". *Princeton University Journal* (2008), pp. 28–51 (cit. on p. 10).

[6] William G. Gardner. *3D audio and acoustic environment modeling*. Tech. rep. Arlington, MA: Wave Arts, Inc, 1999 (cit. on pp. 10, 17).

[7] William G Gardner. "Spatial Audio Reproduction: Toward Individualized Binaural Sound". In: *Reports on Leading-Edge Engineering from the 2004 NAE Symposium on Frontiers of Engineering*. Vol. 34. 2005, pp. 37–42 (cit. on p. 10).

[8] William G. Gardner and Keith Martin. *HRTF Measurements of a KEMAR Dummy-Head Microphone*. Tech. rep. Cambridge, MA: MIT Media Lab Perceptual Computing, 1994 (cit. on p. 17).

[9] Michele Geronazzo, Simone Spagnol, and Federico Avanzini. "Mixed structural modeling of head-related transfer functions for customized binaural audio delivery". *18th International Conference on Digital Signal Processing (DSP)* (2013), pp. 1–8 (cit. on p. 17).

[10] Michael Gerzon. "Surround-sound psychoacoustics". *Wireless World 80* 1468 (1974), pp. 483–486 (cit. on p. 15).

[11] Kaj Grønbæk and Os Iversen. "IGameFloor: a platform for co-located collaborative games". In: *Proceedings of the International Conference on Advances in Computer Entertainment Technology*. 2007, pp. 64–71 (cit. on p. 8).

[12] Benedikt Grothe and Michael Pecka. "The natural history of sound localization in mammals – a story of neuronal inhibition". *Frontiers in Neural Circuits* 8.116 (2014), pp. 1–19 (cit. on p. 12).

[13] Catherine Guastavino et al. "Spatial Audio Quality Evaluation: Comparing Transaural , Ambisonics and Stereo". In: *Proceedings of the 13th International Conference on Auditory Display*. Montreal, Canada, 2007, pp. 53–58 (cit. on p. 19).

[14] Craig T. Jin and Simon Carlile. "Spectral Cues in Human Sound Localization". *Advances in Neural Information Processing Systems* 12 (2000), pp. 768–774 (cit. on p. 13).

[15] Martin Kaltenbrunner et al. "TUIO: A protocol for table-top tangible user interfaces". In: *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*. 2005, pp. 1–5 (cit. on p. 29).

[16] H Kayser et al. "Database of Multichannel In-Ear and Behind-the-Ear Head-Related and Binaural Room Impulse Responses". *EURASIP Journal on Advances in Signal Processing* 2009.1 (2009), pp. 1–10 (cit. on p. 17).

[17] Daniela Kuka et al. "Deep space: high resolution VR platform for multi-user interactive narratives". In: *Interactive Storytelling*. Springer, 2009, pp. 185–196 (cit. on p. 6).

[18] Alexander M. Lowe. "Signal Theory". In: *Electrical Engineering - Volume I*. Ed. by Kit Po Wong. Paris, France: Encyclopedia of Life Support Systems (EOLSS), 2004. Chap. 10, pp. 262–297 (cit. on p. 16).

[19] Keith Dana Martin. "A Computational Model of Spatial Hearing". MA thesis. Cambridge, MA: Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1995 (cit. on pp. 12, 13).

[20] Russell Martin and Ken McAnally. *Interpolation of head-related transfer functions*. Tech. rep. Victora, Australia: Defence Science and Technology Organisation, 2007 (cit. on p. 17).

[21] Alok Meshram et al. “P-HRTF: Efficient personalized HRTF computation for high-fidelity spatial sound”. In: *ISMAR 2014 - IEEE International Symposium on Mixed and Augmented Reality - Science and Technology 2014, Proceedings*. Chapel Hill, NC, 2014, pp. 53–61 (cit. on p. 18).

[22] Parham Mokhtari, Ryouichi Nishimura, and Hironori Takemoto. “Toward HRTF personalization: an auditory-perceptual evaluation of simulated and measured HRTFs”. In: *Proceedings of the 14th International Conference on Auditory Display*. 2008, pp. 1–8 (cit. on p. 18).

[23] H. Møller and M. F. Sørensen. “Binaural technique: Do we need individual recordings?” *Journal of the Audio Engineering Society* 44.6 (1996), pp. 451–469 (cit. on p. 18).

[24] Estefania Munoz Diaz et al. “Evaluation of AHRS algorithms for inertial personal localization in industrial environments”. *IEEE International Conference on Industrial Technology (ICIT)* June (2015), pp. 3412–3417 (cit. on pp. 5, 41).

[25] Otto Naderer. “Crowd Tracking And Movement Pattern Recognition”. MA thesis. Linz, Austria: Johannes Kepler University, Department of Computational Perception, 2015 (cit. on pp. 4, 22, 29).

[26] Jens Nørnberg Paaske and Søren Krarup Olesen. *3D audio rendering on the Android platform*. Tech. rep. Aalborg: Aalborg University, Department of Electronic Systems, 2011 (cit. on p. 16).

[27] C. Phillip Brown and Richard O. Duda. “A structural model for binaural sound synthesis”. *IEEE Transactions on Speech and Audio Processing* 6.5 (1998), pp. 476–488 (cit. on p. 18).

[28] Tilen Potisk. *Head-Related Transfer Function*. Tech. rep. Ljubljana, Slovenia: University of Ljubljana, 2015 (cit. on p. 16).

[29] Simone Spagnol, Davide Rocchesso, and Federico Avanzini. “Extraction of Pinna Features for Customized Binaural Audio Delivery on Mobile Devices”. In: *International Conference on Advances in Mobile Computing & Multimedia*. 2013, pp. 514–517 (cit. on p. 18).

[30] Hagen Wierstorf et al. “A Free Database of Head-Related Impulse Response Measurements in the Horizontal Plane with Multiple Distances”. In: *Proceedings of the Audio Engineering Society Convention*. Vol. 130. 2011, pp. 3–6 (cit. on p. 17).

[31] Dmitry N. Zotkin, Ramani Duraiswami, and Larry S. Davis. “Customizable auditory displays”. In: *Proceedings of the 2002 International Conference on Auditory Display*. July 2-5. Kyoto, Japan, 2002, pp. 1–10 (cit. on p. 18).

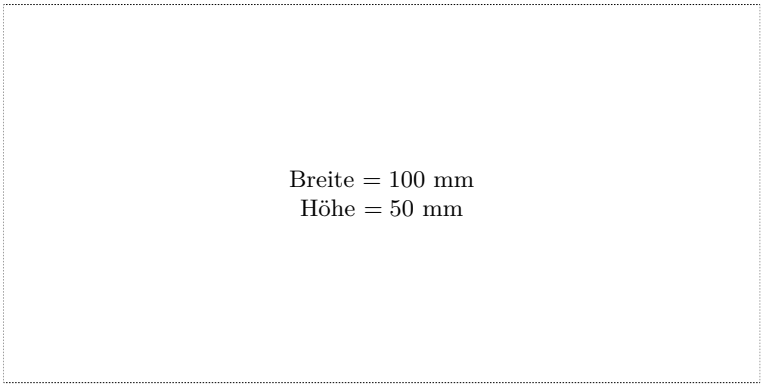
Online sources

[32] V. R. Algazi. *Psychoacoustics of Spatial Hearing*. 2011. URL: http://interface.cipic.ucdavis.edu/sound/tutorial/psych.html (visited on 09/16/2016) (cit. on pp. 11, 12).

[33] P. N. Vassilakis. *Auditory localization cues: Interaural spectral differences & HRTFs*. 2013. URL: http://acousticslab.org/psychoacoustics/PMFiles/Module07b.htm (visited on 09/16/2016) (cit. on p. 14).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —