



UNIVERSITÀ DEGLI STUDI DI MILANO

SCUOLA DI DOTTORATO IN INFORMATICA

DIPARTIMENTO DI INFORMATICA E COMUNICAZIONE

DOTTORATO IN INFORMATICA XXIV CICLO

ON BINAURAL SPATIALIZATION AND THE USE OF GPGPU FOR AUDIO PROCESSING

INFORMATICA (INF/01)

Candidato:

Davide Andrea MAURO

R08168

Supervisore: Prof. Goffredo HAUS

Coordinatore del Dottorato: Prof. Ernesto DAMIANI

A.A. 2010/2011

“A Ph.D. thesis is never finished; it’s abandoned”

Modified quote from Gene Fowler

Contents

Abstract	x
0.1 Abstract	x
0.2 Structure of this text	xi
1 An Introduction to Sound Perception and 3D Audio	1
1.1 Glossary and Spatial Coordinates	1
1.2 Anatomy of the Auditory System	5
1.3 Sound Localization: Localization Cues	8
1.4 Minimum Audible Angle (MAA)	14
1.5 Distance Perception	14
1.6 Listening through headphones and the Inside the Head Localization (IHL)	17
1.7 3D Audio and Binaural Spatialization Techniques	18
1.8 Binaural Spatialization	18
2 General Purpose computing on Graphic Processing Units (GPGPU): An Overview	20
2.1 Available Architectures	21

2.1.1	CUDA	21
2.1.2	OpenCL	22
2.2	The choice of an architecture	22
2.3	The state of the art in GPGPU for Audio	23
3	A Model for a Binaural Spatialization System	28
3.1	Convolution Engines	28
3.1.1	State of the Art	30
3.1.2	Convolution in the Time Domain	31
3.1.3	Convolution in the Frequency Domain	32
3.2	Reference CPU implementations	35
3.2.1	A CUDA convolution engine	36
3.2.2	An OpenCL convolution engine	36
3.3	The CGPUconv prototype	36
3.3.1	Performance Comparisons	37
3.4	Summary and Discussion of the results	42
4	A Head-Tracking based Binaural Spatialization Tool	43
4.1	Related Works	44
4.2	An overview of MAX/MSP	46
4.3	Integrating a Head-tracking System into MAX	49
4.4	The “Head In Space” Application	50
4.4.1	Coordinates Extraction	52
4.4.2	The Convolution Process	59
4.4.3	Interpolation and Crossfade Among Samples	61
4.4.4	Simulation of Distance	63
4.4.5	The Graphical User Interface	63
4.5	Multiple Webcams Head-tracking	64

4.6	Summary and Discussion of the results	66
5	Psychoacoustics and Perceptual Evaluation of Binaural Sounds	68
5.1	Experimental Design	70
5.1.1	Room Acoustics	70
5.1.2	Spatial Coordinates	71
5.1.3	Classification of the Stimuli	72
5.1.4	Binaural Recordings	76
5.1.5	Classification of subjects	76
5.1.6	Task and Questionnaire	77
5.2	Results	78
5.3	Summary and Discussion of the results	84
6	Conclusions and Future Works	85
6.1	Future Works	86
6.1.1	Improvements in the GPU implementation of a convolution engine	86
6.1.2	Use of different transforms beside FFT	86
6.1.3	OpenCL implementation for radix n	86
6.1.4	Partitioned Convolution Algorithm	87
6.1.5	BRIRs (Binaural Reverb Impulse Responses)	87
6.1.6	Further perceptual tests on binaurally spatialized signals	87
6.1.7	Binaural spatialization in VR applications for the blind	88
A	Convolution Implementations	89
B	Source Code for Head-tracking external module	93
C	Questionnaire for Perceptual Test and Results	100

Acknowledgement	110
Bibliography	111

List of Figures

1.1	<i>Coordinates system used to determine the position of a sound source with respect to head of the listener (adapted from [10]).</i>	4
1.2	<i>External ear adapted from [29].</i>	6
1.3	<i>This graph show ILD for varying azimuths and for varying frequencies. (Graph from [45])</i>	9
1.4	<i>This graph show ITD variations. (Graph from [45])</i>	10
1.5	<i>An analytical model for the effects of pinnae. (Adapted from [7]) . . .</i>	12
1.6	<i>Distribution of the sound pressure, for different resonance typologies, inside an external ear model with a high impedance end. The dotted lines indicate the nodal points. (Adapted from [10])</i>	13
1.7	<i>Minimum Audible Angle for sine waves at varying frequencies and azimuths. (Adapted from [45])</i>	15
1.8	<i>Influence of humidity on attenuation. (ISO 9613-1 [1])</i>	16
2.1	<i>Throughput, with memory overhead.</i>	25
2.2	<i>Throughput, no overhead.</i>	26
2.3	<i>Throughput vs. Segment size.</i>	27

3.1	The workflow diagram of the system.	29
3.2	A scheme of convolution in frequency domain.	33
3.3	Schematic view of the overlap-add convolution method.	34
3.4	Execution time for Direct mode depending on input size.	40
3.5	Execution time for Overlap-add depending on input size.	41
4.1	The evolution of the MAX family.	48
4.2	The workflow diagram of the system.	51
4.3	An overview of the patch.	53
4.4	The translation system.	54
4.5	The detail of the MAX subpatch for the convolution process via CPU.	60
4.6	The detail of the MAX subpatch for the crossfade system.	61
4.7	The graphical user interface of the program.	65
5.1	Coordinates of sound objects.	73
5.2	Two different types of envelope.	75
5.3	The portion of the questionnaire where subjects report about the position of sound objects.	77
5.4	Mean values grouped by sound type and by subject class.	79
5.5	Overall values for artificial and natural sound classes.	80
5.6	Mean values for different angles and distances.	82
C.1	First page of the questionnaire.	101
C.2	Second page of the questionnaire.	102
C.3	Third page of the questionnaire.	103
C.4	Fourth page of the questionnaire.	104
C.5	Fifth page of the questionnaire.	105
C.6	Results, Page 1/3.	106

C.7 Results, Page 2/3.	107
C.8 Results, Page 3/3.	108

List of Tables

3.1	Performance comparisons. Time in ms.	39
3.2	Performance comparisons. Time in ms.	42
5.1	The sound stimuli grouped by types used in the experiment.	75
5.2	Clusters for voice sound (<i>so5</i>).	83

Abstract

This thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at the Università degli Studi di Milano. The supervisor of this thesis is Prof. Goffredo Haus, Head of the Laboratorio di Informatica Musicale (LIM), Università degli Studi di Milano, University Board member and Chair of the IEEE Computer Society Technical Committee on Computer Generated Music (TCCGM).

0.1 Abstract

3D recordings and audio, namely techniques that aim to create the perception of sound sources placed anywhere in 3 dimensional space, are becoming an interesting resource for composers, live performances and augmented reality. This thesis focuses on binaural spatialization techniques.

We will tackle the problem from three different perspectives. The first one is related to the implementation of an engine for audio convolution, this is a real implementation problem where we will confront with a number of already available systems trying to achieve better results in terms of performances. General Purpose computing on Graphic Processing Units (GPGPU) is a promising approach to problems where a high parallelization of tasks is desirable. In this thesis the GPGPU approach is applied to both offline and real-time convolution having in mind the spatialization of multiple

sound sources which is one of the critical problems in the field. Comparisons between this approach and typical CPU implementations are presented as well as between FFT and time domain approaches.

The second aspect is related to the implementation of an augmented reality system having in mind an “off the shelf” system available to most home computers without the need of specialized hardware. A system capable of detecting the position of the listener through a head-tracking system and rendering a 3D audio environment by binaural spatialization is presented. Head tracking is performed through face tracking algorithms that use a standard webcam, and the result is presented over headphones, like in other typical binaural applications. With this system users can choose audio files to play, provide virtual positions for sources in an Euclidean space, and then listen as if they are coming from that position. If users move their head, the signals provided by the system change accordingly in real-time, thus providing the realistic effect of a coherent scene.

The last aspect covered by this work is within the field of psychoacoustic, long term research where we are interested in understanding how binaural audio and recordings are perceived and how then auralization systems can be efficiently designed. Considerations with regard to the quality and the realism of such sounds in the context of ASA (Auditory Scene Analysis) are proposed.

0.2 Structure of this text

This work is organized as follows.

- *Chapter 1 — An Introduction to Sound Perception and 3D Audio.* The goal of the first part (*Chapters 1,2*) is to provide a solid background and context for the remainder of the work. In this Chapter the fundamental concepts of hearing and sound perception are defined. These serve as a basis for the development

of 3D audio techniques and in particular to binaural spatialization. Besides the general introduction to hearing provided we focus on the spatial hearing and the development of techniques for 3D audio rendering, giving emphasis to Binaural Spatialization and detailing the available implementations and what we choose for our work.

- *Chapter 2 — General Purpose computing on Graphic Processing Units (GPGPU): An Overview.* This Chapter briefly sketches the opportunities granted by the application of GPUs (traditionally devoted to graphics) to other kind of computations. This field is experiencing an always increasing interest due the nature of GPUs; they have a highly parallelized architecture that can suit problems that can not be efficiently solved by traditional CPUs or require complex, dedicated and expensive architectures.
- *Chapter 3 — A Model for a Binaural Spatialization System.* This is the “core” chapter of the work and presents results that aim at the creation of a suitable convolution engine. Both the CPU and the novel GPU implementations are described, emphasizing differences in performance, complexity and memory use.
- *Chapter 4 — A Head-Tracking based Binaural Spatialization Tool.* In this chapter we present the results of a number of prototypes that aim at the creation of a suitable tool for real time spatialization of sounds that accounts for the position of the listener. A description of such a system as well as an overview of the employed techniques is given.
- *Chapter 5 — Psychoacoustics and Perceptual Evaluation of Binaural Sounds.* The perception of acoustical phenomena is still a not-completely-known field so the evaluation of procedures, methodologies and results needs to be carried out with psychoacoustics tests. We present the results of a subjective evaluation of

binaural sounds that can serve as a basis for optimized version of spatialization algorithms where some sounds are regarded (perceived) as more important with respect to others.

- *Chapter 6 — Conclusions and Future Works.* Finally, this chapter provides a summary of all the concepts discussed. Relevant results and further works are presented as well.

An Introduction to Sound Perception and 3D Audio

In this Chapter, an introduction will be given in terms of concepts of acoustics and psychoacoustics, focusing on localization. The discussion will start with a preliminary consideration that led us to analyze the expressions “Localization” and “Binaural Localization.” We indicate the capability of our perceptive system, thus including not only the hearing system,¹ to locate a sound source, that could lead to the perception of an auditory event, in an Euclidean space.

1.1 Glossary and Spatial Coordinates

- Auditory Event: Everything perceived by the hearing system.
- Sound Event: A physical phenomenon. Please note that there is not a bijective relationship between auditory events and sound events. The former could exist without the latter. For example in some diseases such as tinnitus (ringing or

¹Interaction between audio-visual systems is well known and studied. See for example [44].

buzzing in the ears), sound events may not be perceived if under the audibility threshold or masked by louder sounds.

- **Localization:** For Moore ([45]), this is the judgement on the location and distance of an auditory event produced by a sound source. Blauert ([10]) uses instead a definition related to laws and rules that lead an auditory event to be in relationship with one or more specific attributes of the sound event or any other event related to the auditory one.
- **Localization Cues:** Specific attributes of the sound event that are used by the hearing system in order to locate the position of a sound source in a Euclidean space. See next Sections for details.
- **Localization Blur:** According to Blauert ([10]), this is the smallest variation of one specific attribute of a sound event, or any event related to an auditory event, that is sufficient to induce a variation on the judgement of the position of auditory event.
- **Lateralization:** Auditory events perceived inside the head normally on an imaginary line that goes from one ear to the other. This is quite common while listening through headphones.
- **Monaural:** of or involving a sound stimulus presented to one ear only.
- **Binaural:** of or involving a sound stimulus presented to both ears simultaneously. The word is commonly used also for sounds recorded using two microphones and usually transmitted separately to the two ears of the listener.
- **Diotic:** involving or relating to the simultaneous stimulation of both ears with the same sound.

- Dichotic: involving or relating to the simultaneous stimulation of the right and left ear by different sounds.
- HRIR (head related impulse response): It is the impulse response of the “head system” (head, pinna and torso), measured at the beginning of the ear canal for a given angle of azimuth and elevation, and for a given distance.
- BRIR (binaural room impulse response or binaural reverb impulse response): According to Picinali ([53]), this is the impulse response of the “head system” measured inside a room, or any other environment. It is basically the combination of a HRIR with a room impulse response.
- HRTF (head related transfer function): The terms HRTF and HRIR are often used as pseudo-synonyms, where HRTF stands for the transfer function represented in the frequency domain while HRIR stands for the same representation in the time domain. As any Linear Time-Invariant (LTI) system it can be described by impulse responses. Body Related Transfer Functions (BRTF) is an extension of the aforementioned concept that takes into account the whole human body [4].

In order to localize a sound source in a 3-dimensional space it is necessary to establish a coordinate system. We can distinguish between three different planes having a common origin in the center of the head (more precisely laying on the segment that goes from one ear to the other), see Figure 1.1).

- Horizontal plane: placed at the superior margins of the two ear canals and at the inferior part of the ocular cavity.
- Vertical plane (or Frontal): placed at an angle of 90° to the horizontal plane, it intersects with this at the upper margins of the two ear canals.

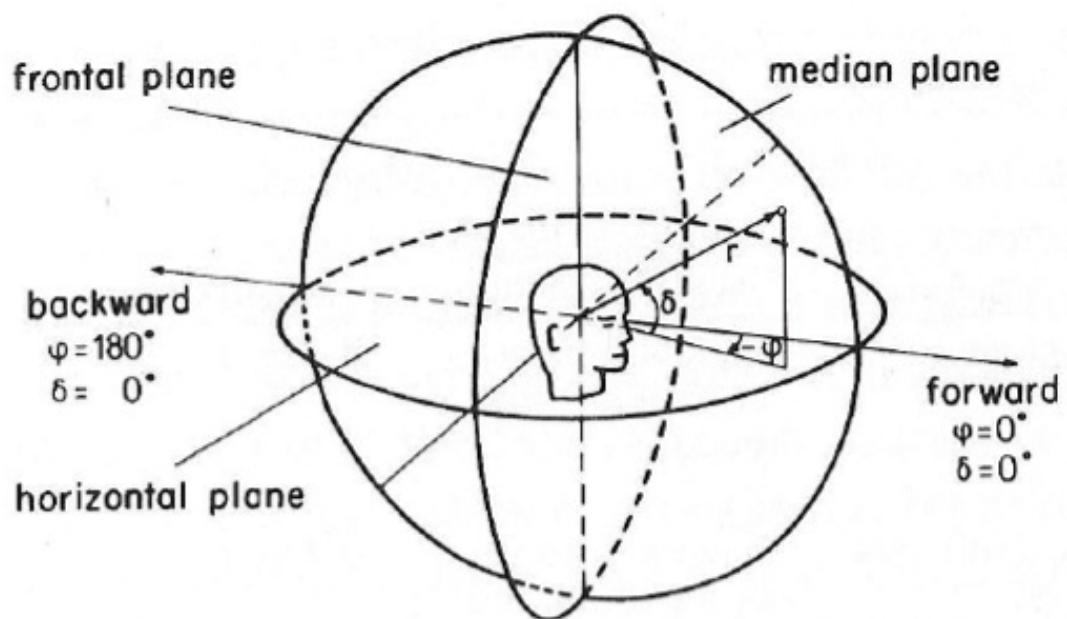


Figure 1.1: *Coordinates system used to determine the position of a sound source with respect to head of the listener (adapted from [10]).*

- Median plane: placed at an angle of 90° to both the horizontal and the frontal planes, it is the plane of symmetry of the head.

The position can now be defined in terms of azimuth ϕ (angle on the horizontal plane), elevation δ (angle on vertical plane) and distance r (in meters, from the sound source to the center of the listeners head). As an example a sound with 0° of azimuth and 0° of elevation is in front of the listener, while one having 180° of azimuth and 0° of elevation is behind the listener.

1.2 Anatomy of the Auditory System

The auditory system is the sensory system for the sense of hearing. The external ear, depicted in Figure 1.2 can be conventionally subdivided into three sections: outer ear, middle ear, and inner ear. The most interesting part for sound localization is the outer ear but it is useful to give an overview of the entire system.

Outer Ear

The outer ear is the external portion of the ear, which consists of the pinna, concha (*cavum conchae*), and external auditory meatus. It gathers sound energy and focuses it on the eardrum (tympanic membrane).

The visible part is called the pinna. It is composed of a thin plate of cartilage, covered with skin, and connected to the surrounding parts by ligaments and muscles; and to the commencement of the external acoustic meatus by fibrous tissue. It is attached with an angle varying from 25° to 45° . It exhibits great variabilities among subjects (this will lead to the problem of individualization of HRTFs). The pinna acts as a sound gatherer and sound waves are reflected and attenuated when they hit the pinna. These interactions provide additional information that will help the brain determine the

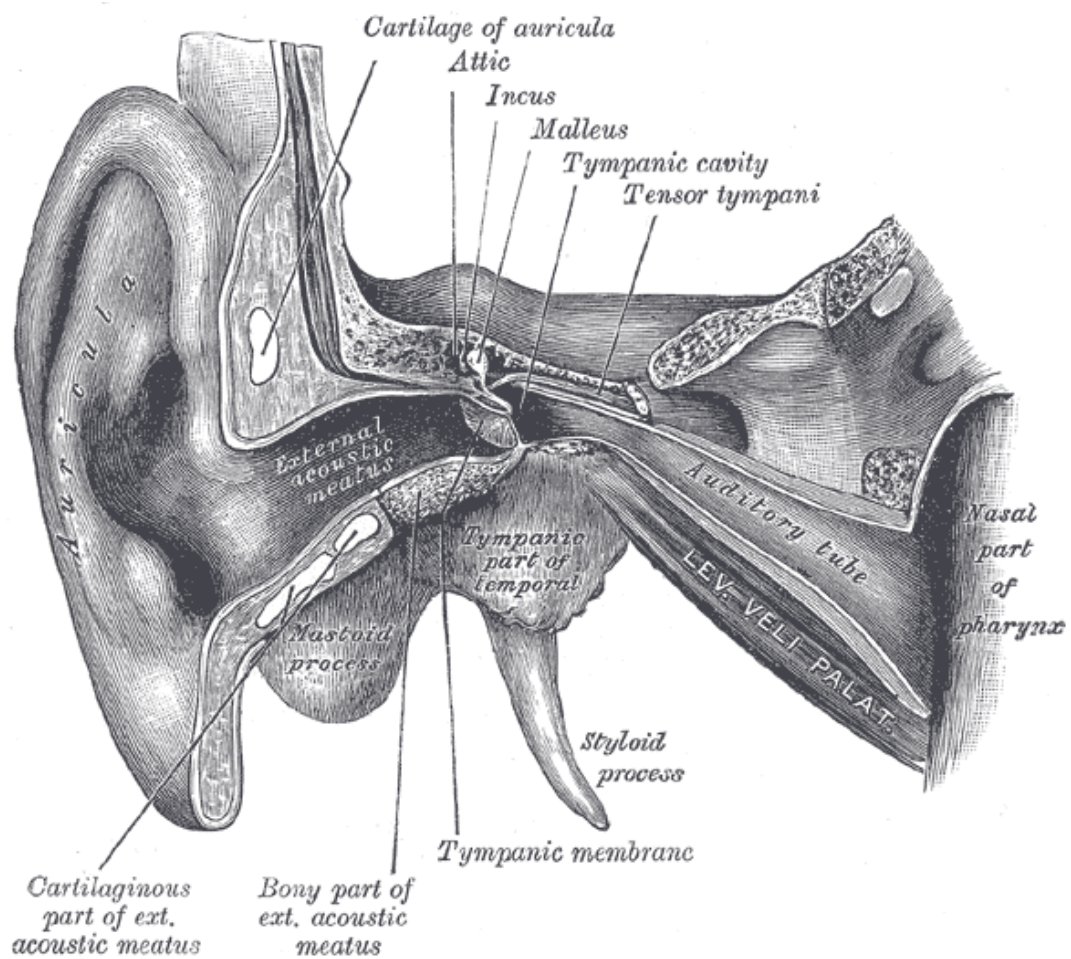


Figure 1.2: External ear adapted from [29].

direction that the sounds arrive from (See section on Direction Dependent Filtering).

The auditory canal is a slightly curved tube fully covered by skin. At the entrance it has a diameter of 5–7 mm, which then rises to 9–11 mm and diminishes again to 7–9 mm; its length is approximately 25 mm.

Middle Ear

The middle ear is the portion of the ear internal to the eardrum, and external to the oval window of the cochlea. The middle ear contains three ossicles, which couple vibration of the eardrum into waves in the fluid and membranes of the inner ear. The hollow space of the middle ear has also been called the tympanic cavity, or *cavum tympani*. The eustachian tube joins the tympanic cavity with the nasal cavity (nasopharynx), allowing pressure to equalize between the middle ear and throat. The primary function of the middle ear is to efficiently transfer acoustic energy from compression waves in air to fluidmembrane waves within the cochlea.

The eardrum is an elliptical membrane (10–11 mm measured at the long angle, and 8.5–9 mm on the shorter), approximately 0.1 mm thick, positioned at the end of the auditory canal with an angle of 40–50°. It can be considered a pressure sensitive receiver. The middle ear contains three tiny bones known as the ossicles: malleus (*hammer*), incus (*anvil*), and stapes (*stirrup*). The ossicles mechanically convert the vibrations of the eardrum into amplified pressure waves in the fluid of the cochlea (or inner ear) with a lever arm factor of 1.3. Since the area of the eardrum is about 17 fold larger than that of the oval window, the sound pressure is concentrated and amplified, leading to a pressure gain of at least 22. The eardrum is attached to the malleus, which connects to the incus, which in turn connects to the stapes. Vibrations of the stapes footplate introduce pressure waves in the inner ear. There is a steadily increasing body of evidence that shows that the lever arm ratio is actually variable, depending on frequency. Between 0.1 and 1 kHz it is approximately 2, it then rises to around 5 at 2 kHz and then falls off steadily above this frequency (see [38] for details). The impedance of the eardrum varies with frequencies, and may increase up to 100%, thanks to the “Acoustic Reflex” phenomenon (see pp. 55-63, [10]), i.e., the contraction of two small muscles, located within the ossicles chain, activated when the sound

pressure level reaches 90–100 dB. The middle ear efficiency peaks at a frequency of around 1 kHz. The combined transfer function of the outer ear and middle ear gives humans a peak sensitivity to frequencies between 1 kHz and 3 kHz.

Inner Ear

The inner ear consists of the cochlea and a non-auditory structure, the vestibular system, that is dedicated to balance. The cochlea has three fluid-filled sections, and supports a fluid wave driven by pressure across the basilar membrane separating two of the sections. Strikingly, one section, called the cochlear duct or scala media, contains endolymph, a fluid similar in composition to the intracellular fluid found inside cells. The organ of Corti is located in this duct on the basilar membrane, and transforms mechanical waves to electric signals in neurons. The other two sections are known as the scala tympani and the scala vestibuli; these are located within the bony labyrinth, which is filled with fluid called perilymph, similar in composition to cerebrospinal fluid. The chemical difference between the two fluids (endolymph & perilymph) is important for the function of the inner ear due to electrical potential differences between potassium and calcium ions.

1.3 Sound Localization: Localization Cues

It is now time to ask how our auditory system can locate a sound in space; the position of the human ears on the horizontal plane supports the perception of interaural differences from sound events that occurs around us more than events above or under the head of the listener.

There exist mainly three different so called “localization cues”; ILD (Interaural level difference), ITD (Interaural time difference), and DDF (Direction dependent filtering) that is a filtering effect with respect to the position of the sound source. While

the first two are regarded as interaural differences the latter is essentially a monaural attribute that still works using only one ear.

Interaural Level Difference (ILD) and Interaural Time Difference (ITD)

- ILD (Interaural Level Difference) [10] represents the difference in intensity between the ears and it is usually expressed in dB. It is most effective for high frequency (above 1 kHz) where the head act as an obstacle generating an acoustic shadow and diffraction on its surface. It is depicted in Figure 1.3 as a function of frequency and azimuth.

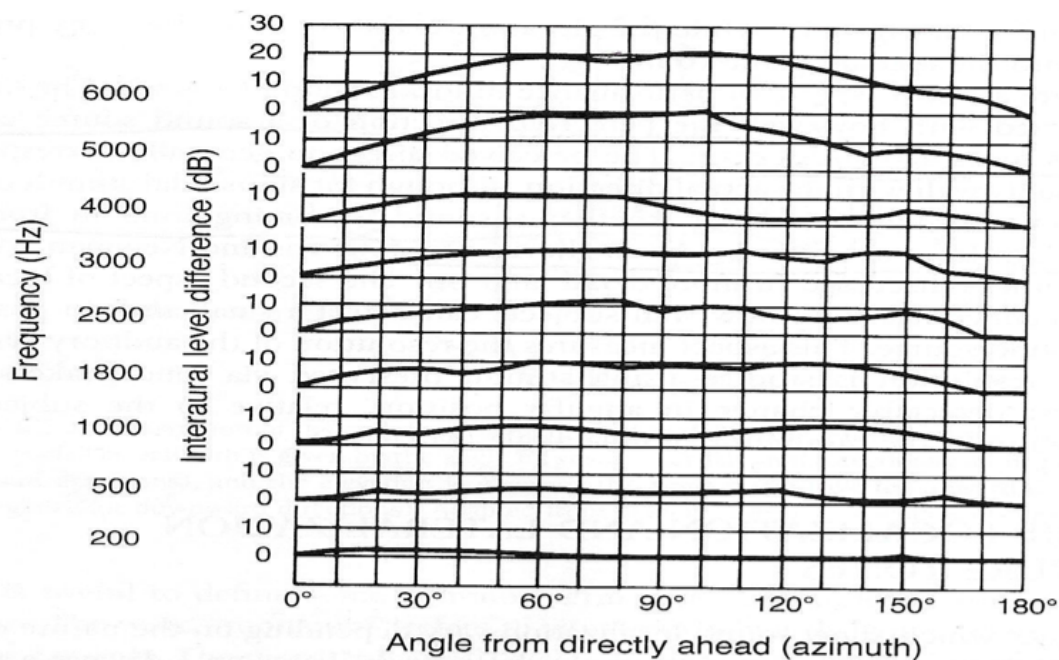


Figure 1.3: This graph show ILD for varying azimuths and for varying frequencies. (Graph from [45])

- ITD (Interaural Time Difference) [10] represents the delay of arrival of the sound

between the two ears (usually expressed in ms). In a real context both the cues cooperate in order to get a correct localization (even if these two parameters alone generate the so called cone of confusion [33]) of sound but they tend to work on different parts of the spectrum (according to the Duplex Theory originally proposed by Lord Rayleigh in 1907 [40].) It is depicted in Figure 1.4 as a function of azimuth.

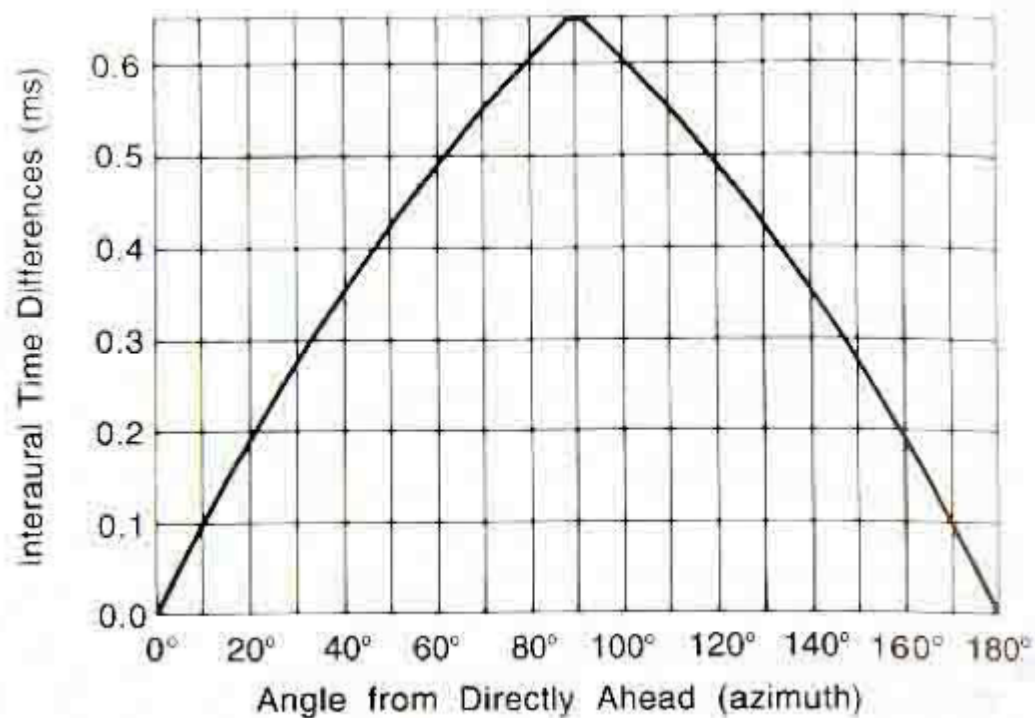


Figure 1.4: This graph show ITD variations. (Graph from [45])

For low frequencies, whose wavelength is bigger than the radius of the head, the head itself does not act as an obstacle giving no significative intensity variations as the wave diffracts around the head. For this reason our hearing system exploits the use of ITD. While the frequency increases the period of the signal becomes comparable with the

ITD itself giving no opportunity to distinguish i.e. between a sound in phase because arrived at the same timing or shifted by one period. So ITD becomes less useful for frequencies greater than 800 Hz, while some evidence suggests that is still possible to analyze changes in the spectral envelope up to 1.6 kHz [45].

Direction-Dependent Filtering (DDF)

The previous two interaural differences alone cannot account to explain how it is possible to distinguish between: e.g. a sound located at an azimuth of 30° and a sound with an azimuth of 150° , because they will yield to identical interaural differences.

In this case a new effect arise caused by a selective filter due to the different position of sounds. This effect known as direction-dependent filtering is caused by the shape and the position of the pinna.

The dimensions of the pinna are too small compared with wavelengths of many audible frequencies for it to function as a sound reflector. The dimensions of its cavities, instead, are comparable to $\lambda/4$ (where λ is the wavelength of a given frequency) for a large number of frequencies, and these can become sound resonators for sound waves coming from specific directions. Therefore, inside the pinna the sound is modified by reflections, refractions, interferences, and resonances activated for specific frequencies and, for the incident angles of specific sound waves, hence the name direction-dependent filtering. Here, several experiments that examined DDF and the effects of the pinna on sound localization are described.

Batteau in [7] made an extensive study on pinna reflections and on the ratio between direct sound and reflections at the entrance of ear canal. He developed an analytical model depicted in Figure 1.5 that take into account azimuth and elevation perception. He also suggested that two distinct delay lines exists (plus the direct signal).

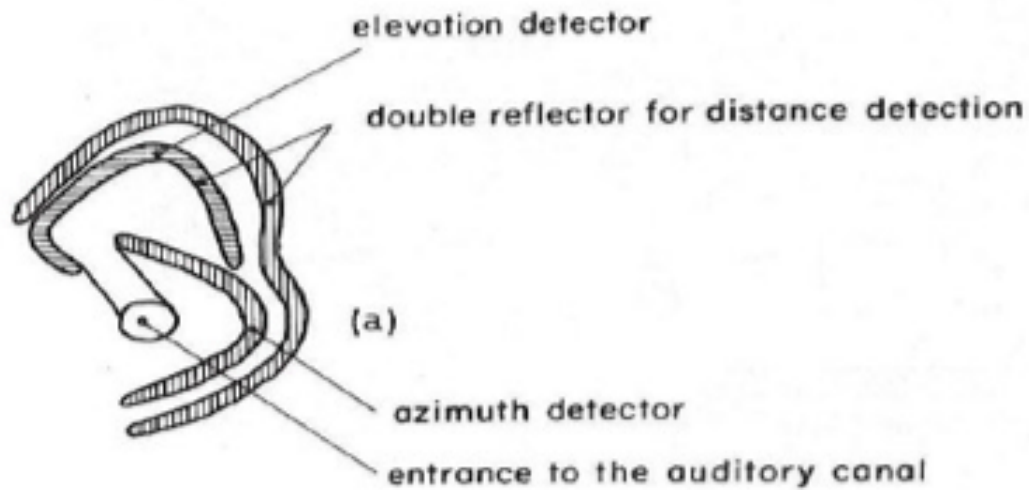


Figure 1.5: An analytical model for the effects of pinnae. (Adapted from [7])

Shaw and Teranishi [63] used a silicone model of the ear plus studies on six subjects placing a probe microphone and a moving sound source at 8 cm from the entrance of the ear canal to measure resonance frequencies for a wide variety of incidence angles. They developed a model with 5 main resonance frequencies (see Figure 1.6).

- F1 around 3 kHz: it is a $\lambda/4$ resonance of a tube closed at one end, with a length of 30 mm, therefore approximately 33% more than the real length of the ear canal (in this case, the pinna seems to act as an extension of the ear canal).
- F2 around 5 kHz: the maximum pressure of this oscillation entirely fills the ear canal; the distribution of the pressure is therefore the same as that with the eardrum occluded. The ear canal and the cavum conchae are involved in this resonance, which can be modified in frequency through inserting material inside the concha (see [10]), and does not depend on the incidence angle of the signal.
- F3 around 9 kHz, F4 around 11 kHz and F5 around 13 kHz: they are stationary

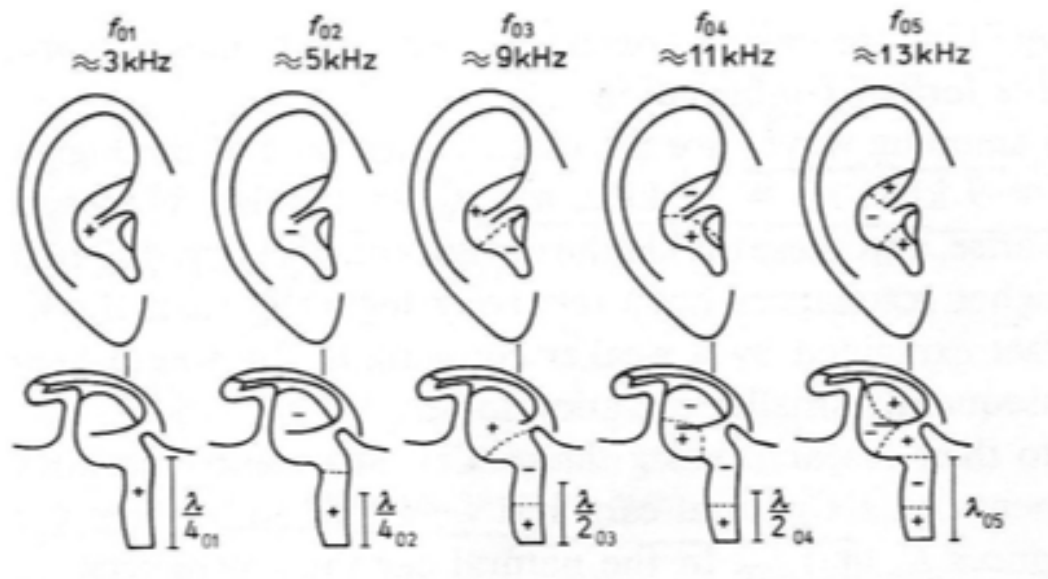


Figure 1.6: *Distribution of the sound pressure, for different resonance typologies, inside an external ear model with a high impedance end. The dotted lines indicate the nodal points. (Adapted from [10])*

longitudinal waves of $\lambda/2$ and λ .

All these resonances may vary between subjects especially with the incidence angle of the sound stimulus (except for F2). This can be explained as the result of interferences between parts of the pinna, refraction and diffraction phenomena. Blauert performed new experiments on the basis of the one from Shaw and Teranishi adding an artificial extension to the ear canal. He observed that different magnitude. As an example F2 (5 kHz) remains constant up to 90° then drops 15–20 dB between 90° and 110° . Through these experiments, Blauert was also able to demonstrate that the resonance F2 is not activated for sound sources coming from behind, and the resonances inside the ear canal are independent of the azimuth and elevation variations. We can now conclude that pinna, along with the ear canal, act as a complex system of acoustic resonators. The energy depends on the direction and the distance of the sound source.

Please note that we now limited ourselves to analyze just part of the auditory system but other parts of the body contribute to this process: for example the entire head , shoulder and torso. While some of the HRTF parameters may be considered constant for everyone, certain others need to be considered individually, thus leading to the individualization of HRTFs (see [27] for further details).

1.4 Minimum Audible Angle (MAA)

The minimum audible angle Is the minimum angular variation distinguishable [45]. In Figure 1.7, the MAA is depicted as a function of φ and frequency. At 0° it is possible to discriminate angles of 1° while the performances drastically reduce for sound sources tending towards lateral positions. MAA also varies with the frequency of the stimulus: for lower frequencies small angles are detectable while above 1500 Hz it is not measurable. This is consistent with the mechanisms of the duplex theory previously cited.

This data are usually taken into account when sampling HRIRs to choose the angles to be sampled.

1.5 Distance Perception

The distance estimation of an auditory event is presumed from the center of the head. When a sound is perceived “within” the head (IHL — Inside the head localization) it means that the distance itself is less than the radius of the head. This occurs happens usually while listening through headphones. It is worth to noting that our auditory system is far less capable at estimating the distance of a sound event than the direction. Therefore, studies on distance perception and IHL usually focus on sounds coming from the median plane (diotic stimuli) or monaural attributes of the signals. All the

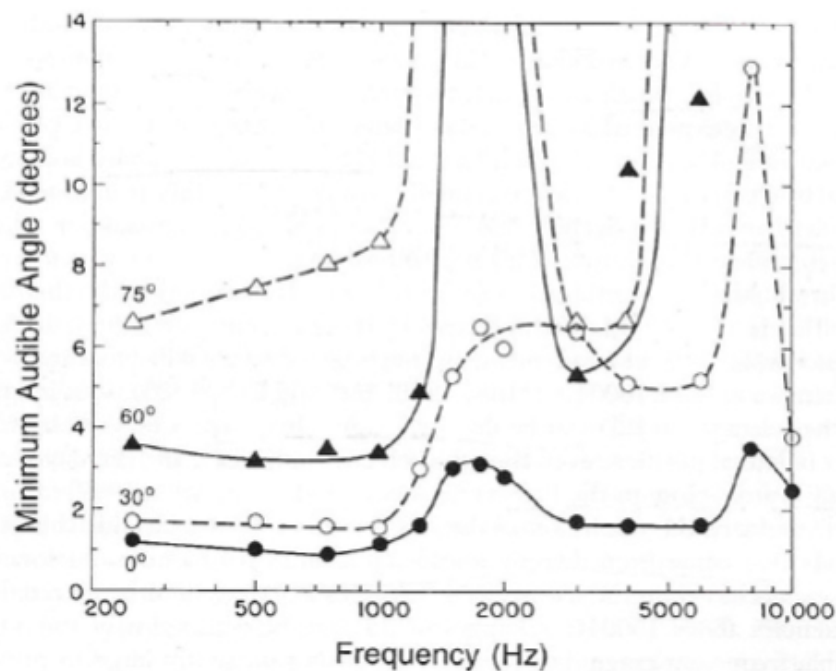


Figure 1.7: *Minimum Audible Angle for sine waves at varying frequencies and azimuths. (Adapted from [45])*

attributes, such as the overall level of the signal, are useful to determine the distance even if normally we do not have an absolute perception but discrimination by comparisons, with both other stimuli that arrives at our ear, with a known distance (maybe related to a visual cue) or with something stored in memory. As presented by Blauert in [10] we can make the following classification scheme of the acoustic environment:

1. At intermediate distances from the sound source, approximately from 3 to 15 m, the sound pressure level at the eardrum depends on distance following the inverse square law ($1/r$). This law states that in a free sound field the pressure halves (-6dB SPL) when the distance doubles.
2. At greater distances from the sound source, more than approximately 15 m, the air path between the sound source and the subject can no longer be regarded as

distortion-free. The inverse square law, that is frequency independent, is still valid but a new effect of high frequencies attenuation appears. This effect is analytically described in ISO 9613-1 (see Figure 1.8). It depends on the humidity and temperature of air and is evaluated through the absorption coefficient of air. It represents the sound attenuation produced by viscosity and heat during a single period of pressure variation.

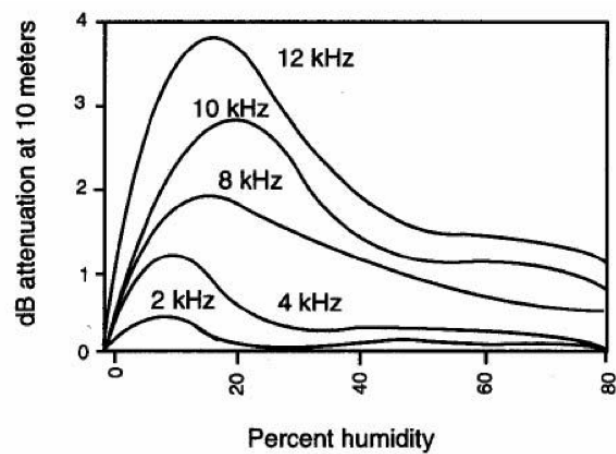


Figure 1.8: *Influence of humidity on attenuation. (ISO 9613-1 [1])*

3. Close to the sound source, within 3 meters from the listener, the effects of curvature of the wave fronts arriving at the head can no longer be neglected. The linear distortions of the signals due to the head and the external ears vary with the distance from the sound source. Close to the sound source the sound pressure level changes with distance, and the shape of the spectrum changes too [7].

We have to take into account that all these cues refer to a free sound field, where our discrimination of distances is dramatically lower than in real environment with reverberations.

These reasons lead to consider works where HRIRs are sampled at different distances in order to evaluate the differences (see [41], [53]) that lead to different distance perceptions.

1.6 Listening through headphones and the Inside the Head Localization (IHL)

Listening through headphones is a common situation, and it is also common to perceive the sound as coming from a source located within the head even if the sounds are actually coming from outside the head (the headphones are placed around the head or inside the ear canal). With headphones, the effect of the head and the pinna (with earplugs) is bypassed and normally lead to the perception of a sound localized within the segment that link the two ears. For this situation we use the term “lateralization”.

Normally presenting a diotic stimulus with headphones, the elicited sensation is the aforementioned; and even if a phase inversion is applied to one of the signal, the “virtual” sound source is perceived on the back of the head. This phenomenon can create issues in the context of binaural spatialization, for which the signals need to be reproduced over headphones. For works related to the perception of the IHL see [37], [24], [62], [64], [57], [39], [75], [32], [13]. As a result, IHL is not present when signals are exactly as they are likely to be in a real scenario; reverb plays a central role in this situation giving significantly better results in “externalization” if applied to the sounds. Also recordings made with dummy heads with an accurate reconstruction of both pinnae normally do not lead to IHL. IHL is present also with some other configurations: with a loudspeakers array on the median plane. Plenge [55] hypothesized that a certain “acquaintance” level can play a central role in terms of IHL; when a subject has been previously exposed to a source located outside the head (e.g. with loudspeakers) then

the same signals presented over headphones seemed not to be affected. This information are stored in short-time memory that can be reorganized during experiments.

1.7 3D Audio and Binaural Spatialization Techniques

3D sound is becoming a prominent part of entertainment applications. The degree of involvement reached by movies and video-games is also due to realistic sound effects, which can be considered a virtual simulation of a real sound environment.

Unlike surround sound refers to the use of multiple audio tracks and multiple loudspeakers to envelop the audiences watching a film or listening to music, causing the perception they are in the middle of a complex sound field that may, in the case of the movie or the music, represent the action or the concert. The surround sound formats rely on dedicated loudspeaker systems that physically surround the audience. The position of the different speakers and the format of the audio tracks vary among the commercial companies specializing in this specific surround format.

For further details into the vast field of 3d audio see [28] [11] [31] [34] [58] [68] [70].

1.8 Binaural Spatialization

Binaural spatialization is a technique that aims at reproducing a real sound environment using only two channels (for example a stereo recording). It is based on the assumption that our auditory system has only two receivers, namely the ears. If it is possible to deliver a signal equal (or nearly equal) to the one which a subject would receive in a real environment, this will lead to the same perception. Our auditory system performs various tasks to obtain a representation of the acoustic environment; most of them are based on the physical parameters of the signal of interest and are called

“cues” [79][10]. It is well suited by headphones where each channel can reach only the required ear but also a pair of loudspeakers can be used taking into account crosstalk and facing it with cancelation mechanisms (e.g. TRADIS [22], BACCH [17]).

Binaural spatialization can be achieved through various processes, such as: equalizations and delays, or convolution with the impulse response of the head (HRIR). The latter approach is the one we have followed in our work. In order to obtain these impulses, many experiments involving the use of a dummy head² have been made (see i.e. [3]), thus creating databases of impulse responses. Most of them use a fixed distance (usually 1 m) from the source (S) to the listener (L), which constitutes a potential limitation.

²A dummy head is a mannequin that reproduces the human head.

Chapter 2

General Purpose computing on Graphic Processing Units (GPGPU): An Overview

The idea of exploiting the capabilities of Graphic Processing Units (GPU) is not new, as well as the use of GPUs for audio processing (see [76] for details). But now it is becoming easier and easier to work with GPUs since the development of architectures that use GPUs but are not “graphic-oriented”. This means that programmer can benefit from the highly parallelized structures of such architectures without having knowledge of the video pipeline and without the need to use pixel and vertex shaders to encapsulate datas not originally meant to be graphic. GPU manufacturers are exploiting the peculiarity of graphic computations, that are highly data-parallel by nature, by creating an affordable processor model capable of a great computational power. However GPUs are not superseding CPUs in every kind of computation; there are some tasks that still fits best in CPUs. GPUs have smaller caches and ALUs (Arithmetic Logic Unit) (although in a higher number), than the CPUs. The smaller cache can be explained since highly arithmetic independent operations, running in parallel on different data trunks

(different threads), can easily hide memory latency, while simpler ALUs can be explained by the fact that they have a restricted set of functions and basically have to be fast floating-point arithmetic units [14]. For a survey on the use of GPU for computing see [50].

2.1 Available Architectures

In the past some software projects have tried to use standard graphics libraries like OpenGL (Open Graphics Library) to use the graphics functions provided to execute non graphics computations on GPUs. Anyway, such an approach did not spread, since not all computational problems that may benefit from running on GPUs can be translated into “graphical problems” solvable by the use of graphical functions. The main available architectures are essentially two. One is associated with hardware from NVIDIA inc. The other project is an open standard developed by a consortium of producers.

2.1.1 CUDA

CUDA or Compute Unified Device Architecture [48] is a parallel computing architecture developed by NVIDIA. CUDA is the computing engine with NVIDIA graphics processing units (GPUs) that is accessible to software developers through variants of industry standard programming languages. Programmers use “C for CUDA” (C with NVIDIA extensions and certain restrictions), compiled through a PathScale Open64 C compiler, to code algorithms for execution on the GPU. The CUDA architecture shares a range of computational interfaces with two competitors - The Khronos Group’s OpenCL - and Microsoft’s DirectCompute. Third party wrappers are also available for Python, Perl, Fortran, Java, Ruby, Lua, MATLAB and IDL, and native support exists in Mathematica. One of the drawbacks of this architecture is the use of

a different compiler called `nvcc` so programs that need to use CUDA APIs can not be compiled with `gcc` or `llvm/clang`.

2.1.2 OpenCL

OpenCL (Open Computing Language) [46] is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors. OpenCL includes a language (based on C99) for writing kernels (functions that execute on OpenCL devices), plus APIs that are used to define and then control the platforms. OpenCL provides parallel computing using task-based and data-based parallelism. It has been adopted into graphics card drivers by both AMD/ATI (which made it its sole GPGPU offering, branded as Stream SDK) and NVIDIA, which offers OpenCL as alternative to its Compute Unified Device Architecture (CUDA) in its drivers. OpenCL's architecture shares a range of computational interfaces with both CUDA and Microsoft's competing DirectCompute. OpenCL is analogous to the open industry standards OpenGL and OpenAL, for 3D graphics and computer audio, respectively. OpenCL is managed by the non-profit technology consortium Khronos Group.

2.2 The choice of an architecture

Coming to the conclusion we need to say that we choose to develop using both architectures while focusing the optimization on the CUDA architecture. This can mainly be explained by the lack of support of the competitor, AMD (formerly known as ATI graphic card manufacturer), in drivers for any platform. The AMD architecture went through a continuous set of changes leading them from CTM (Close to Metal) to the new production version of AMD's GPGPU technology that is now called Stream SDK to AMD Accelerated Parallel Processing (APP) SDK. APP SDK lacks, at the moment, Apple OS X support while CUDA is well supported by Linux distributions.

For this reason, since we are interested in exploiting the capabilities of GPUs, even if the OpenCL initiative suggests a future of interoperability with both manufacturers now it lacks, an efficient FFT implementation for radix n .

2.3 The state of the art in GPGPU for Audio

As previously stated, the idea of using GPGPU for audio processing is not completely new even if it is not largely widespread. A number of works can be highlighted pointing out their novelty.

- Gallo and Tsingos in [26] give an introduction to the use of GPUs for 3D audio. This was one of the first articles in the field. They conducted a first feasibility study investigating the use of GPUs for efficient audio processing.
- Cowan and Kapralos in [20] and [21] show the effectiveness of this approach for the convolution task. In particular, in [20], a GPU-based convolution method was developed that allowed for real-time convolution between an arbitrarily sized auditory signal and a filter. Despite the large computational savings, that GPU based method introduced noise/artifacts to the lower-order bytes of the resulting output signal which may have resulted in a number of perceptual consequences. This was caused by the need of translating the audio samples into a RGB pixel map and then exploiting OpenGL capabilities that are mainly intended for graphics. In more recent work, they employed a superior GPU that eliminated the noise/artifacts of the previous method and provides further computational saving [21].
- Sosnick [65] used GPUs to solve problems for physics-based music instrument models. They describe an implementation of an FD-based (Finite Difference) simulation of a two-dimensional membrane that runs efficiently on mid-range

GPUs; this will form a framework for constructing a variety of realistic software percussion instruments. For selected problem sizes, realtime sound generation was demonstrated on a mid-range test system, with speedups of up to 2.9 over pure CPU execution.

- Fabritius in his master thesis [23] presented an overview of Audio processing algorithms using GPUs. He faces the problem from a wide perspective giving implementations for some common processes. He analyzes and implements four basic audio processing algorithms used in music production for the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) comparing in which situations it is better to perform the computations on the GPU instead of the CPU. By comparing the performance of the audio processing algorithm implementations, the running times are analyzed for typically used parameter values in a music production setting.
- Rush in [59] provides an implementation of a convolution engine with the NVIDIA G80 architecture. This is an implementation that makes use of CUDA capabilities. It exploit partitioned convolution in the frequency domain for long filters and make use of the CUFFT library (FFT library for CUDA) for efficient GPU fast Fourier transform (FFT) implementation. It provides comparisons in terms of execution time with respect to a CPU implementation depicted in Figures 2.1, 2.2, 2.3.

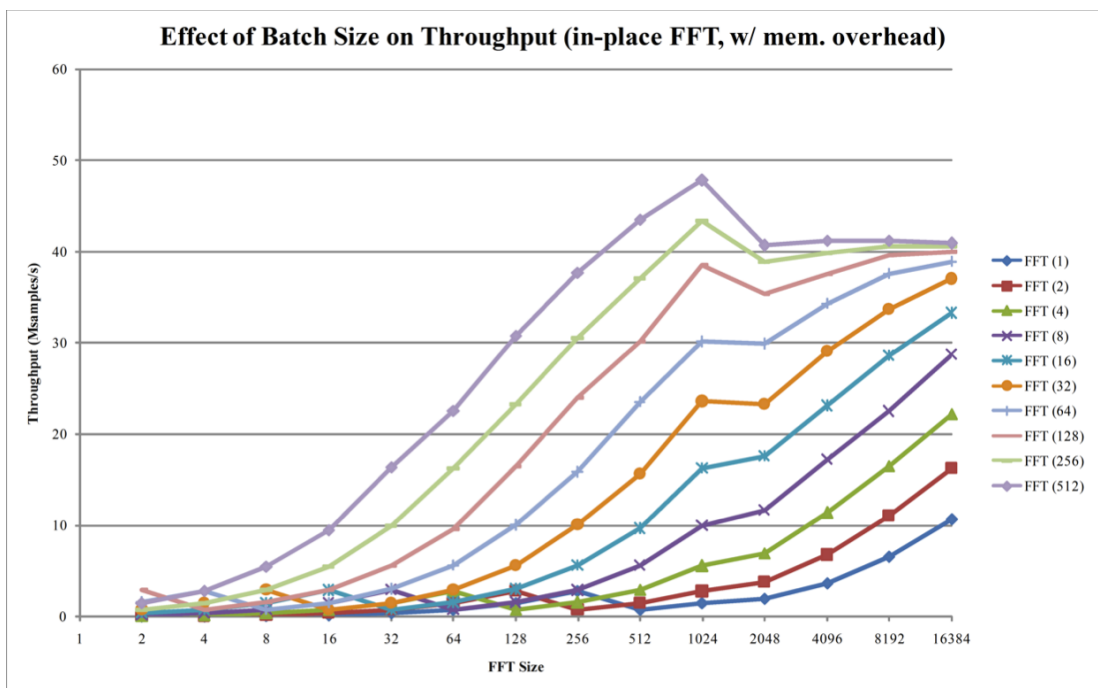


Figure 2.1: *Throughput, with memory overhead.*

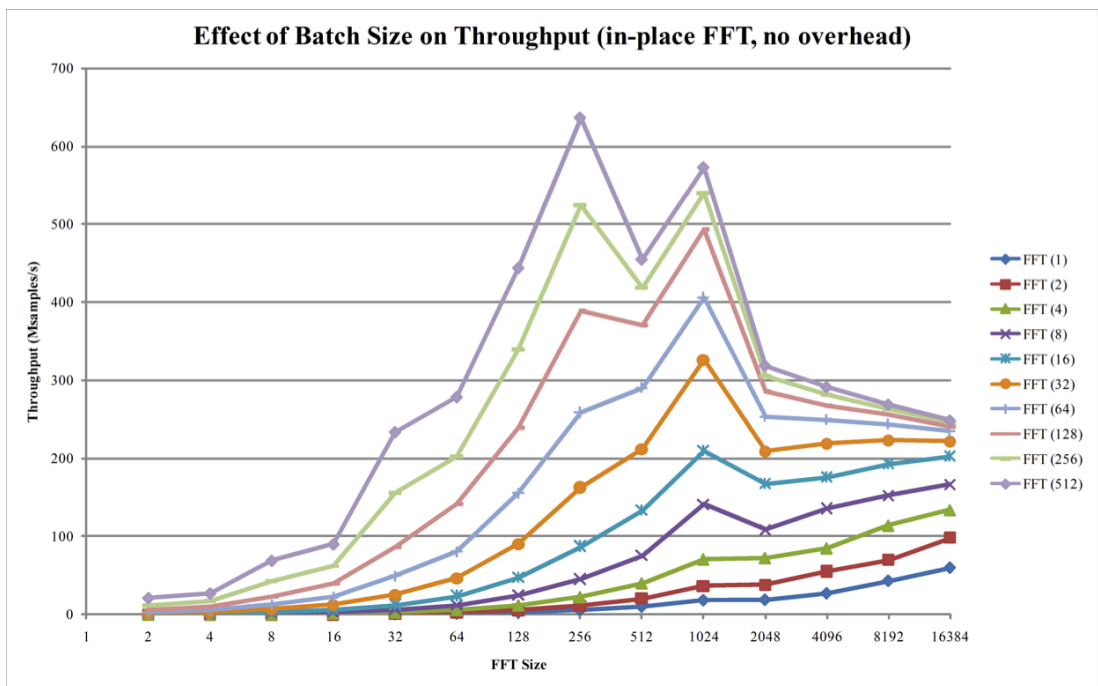


Figure 2.2: Throughput, no overhead.

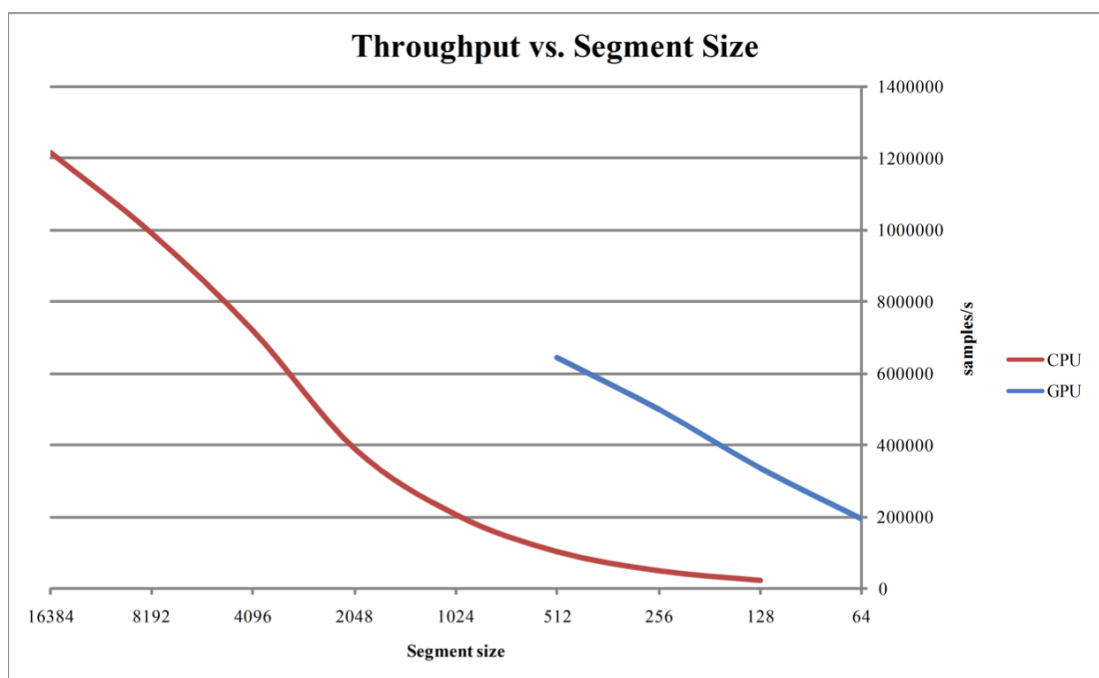


Figure 2.3: *Throughput vs. Segment size.*

Chapter 3

A Model for a Binaural Spatialization System

In this Chapter we first introduce the core of the work in terms of conceptualization and development of a model. Even if the process is well known and understood in terms of mathematics, the realization of implementations that work in real-life scenarios is not trivial. One of the greatest obstacle is the computational complexity that convolution requires both in the time and frequency domain approaches. This means that the problem could be theoretically solved but the computer architecture does not allow it to be solved in a reasonable time for some practical cases of interest.

3.1 Convolution Engines

As shown in Figure 3.1 the system requires as input an anechoic signal (monophonic) and a impulse response (stereo) and the overall output will be two channel spatialized sound that can feed both headphones or loudspeakers (with crosstalk cancelation algorithms [17]).

We will focus on implementations of this system thanks to modern GPGPU tech-

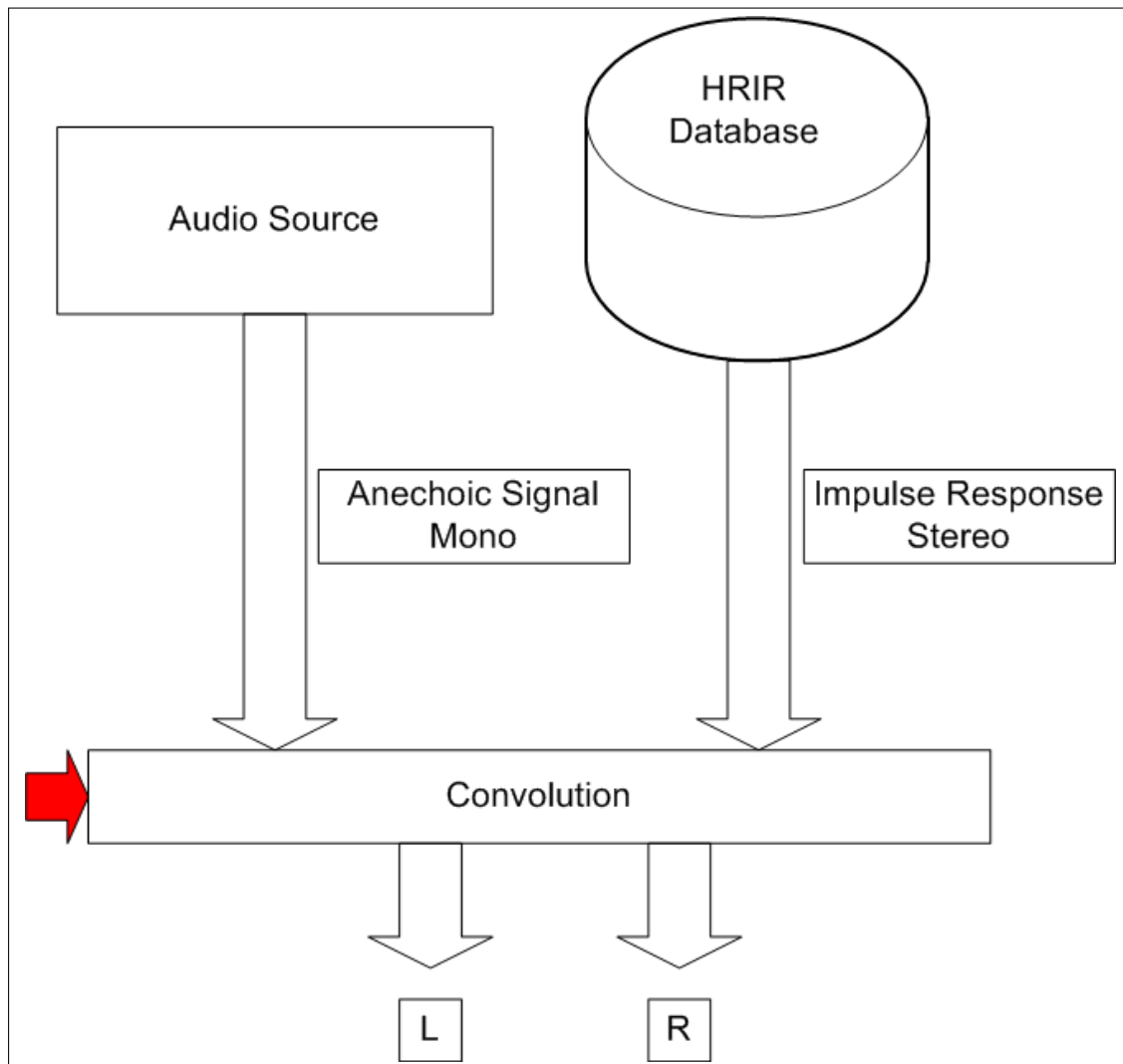


Figure 3.1: The workflow diagram of the system.

niques.

3.1.1 State of the Art

In the literature there are other systems that aim at realizing systems that achieve real-time auralization, or augmented reality. We present a brief sketch of the opportunities and the techniques employed.

- **TConvolutionUB~**: A Max/MSP external patch from Thomas Resch that extends the possibilities given by the *buffir~* object allowing convolution with a filter that has more than 255 points.
- **SIR2**: An easy to use native audio-plugin to use for high quality reverberation. It's available for the plugin formats VST and AudioUnit. Its use can be stretched from a convolution reverb to a convolution engine for auralization given the flexibility of the program itself.
- **djbfft**: A library for floating-point convolution. The current version provides power-of-2 complex FFTs, real FFTs at twice the speed, and fast multiplication of complex arrays. Single precision and double precision are equally supported.
- **BruteFIR**: An open-source convolution engine, a program for applying long FIR filters to multi-channel digital audio, either offline or in realtime, by Anders Torger [71]. Its basic operation is specified through a configuration file, and filters, attenuation and delay can be changed at runtime through a simple command line interface. The author states that the FIR filter algorithm used is an optimized frequency domain algorithm, partly implemented in hand-coded assembler, thus throughput is extremely high. In real-time, a standard computer can typically run more than 10 channels with more than 60000 filter taps each. It makes use

of the partitioned convolution and overlap-save methods that are introduced in the following subsection.

- **AlmusVCU:** From the author of BruteFIR this is a complete system that aims at an integrated environment for sound spatialization. It has been designed primarily with Ambiophonics in mind and contains all processing needed for a complete Ambiophonics system.
- **Aurora Plugin:** From Angelo Farina, is a suite of plug-ins for Adobe Audition: room acoustical impulse responses can be measured and manipulated, for the recreation of audible, three-dimensional simulations of the acoustical space.

3.1.2 Convolution in the Time Domain

This approach can be mathematically described by the formula:

$$y(k) = \sum_{j=1} x_1(j)x_2(k-j+1) \quad (3.1)$$

Where x_1 and x_2 are the input sequences of length m and n and y is the output sequence of length $k = m + n - 1$.

When $m = n$, which is the normal case for other implementations, this gives:

$$\begin{aligned}
 w(1) &= u(1)v(1) \\
 w(2) &= u(1)v(2) + u(2)v(1) \\
 w(3) &= u(1)v(3) + u(2)v(2) + u(3)v(1) \\
 &\dots \\
 w(n) &= u(1)v(n) + u(2)v(n-1) + \dots + u(n)v(1) \\
 &\dots \\
 w(2n-1) &= u(n)v(n)
 \end{aligned}
 \tag{3.2}$$

The computational complexity for the time domain approach is $O(n^2)$.

This is the underlying approach to every other method. Implementing a FIR (Finite Impulse Response) filter is obviously the easiest idea but as can be seen from the complexity as the input size increase it could become impossible to process data in real-time.

3.1.3 Convolution in the Frequency Domain

Thanks to the convolution theorem we can express the convolution of two sequences as the multiplication of their Fourier transforms. Here the general layout for the frequency domain approach is introduced. The approach that can be schematized as follows (see Figure 3.2:

- Zero-Pad input vectors x_1 and x_2 of length m and n so the length of the sequences becomes $m + n - 1$
- Perform FFT of the input vectors;

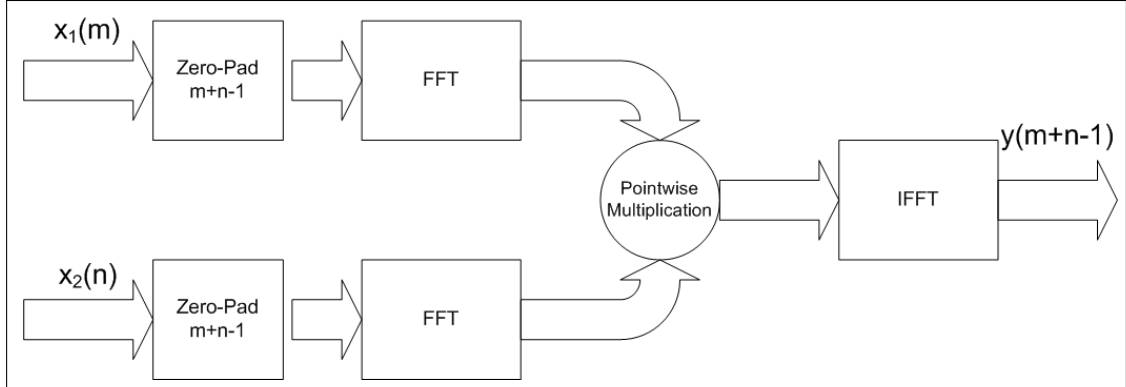


Figure 3.2: A scheme of convolution in frequency domain.

- Perform the pointwise multiplication of the two sequences;
- Perform the IFFT of the obtained sequence.

The computational complexity for the frequency domain approach is $O(n \log(n))$.

3.1.3.1 Overlap-add algorithm

Since the size of the filter kernel can become very high, it is not convenient to use a single window to transform the entire signal so a number of methods can be implemented to overcome this. We choose to use a method called Overlap-add (OA, OLA). It is an efficient way to evaluate the discrete convolution of a very long signal $x[n]$ with a finite impulse response (FIR) filter $h[n]$. The concept is to divide the problem into multiple convolutions of $h[n]$ with short segments of $x[n]$:

$$y[n] = x[n] * h[n] := \sum_{m=-\infty}^{\infty} h[m]x[n-m] = \sum_{m=1}^M h[m]x[n-m] \quad (3.3)$$

where $h[m] = 0$ for m outside the region $[1, M]$.

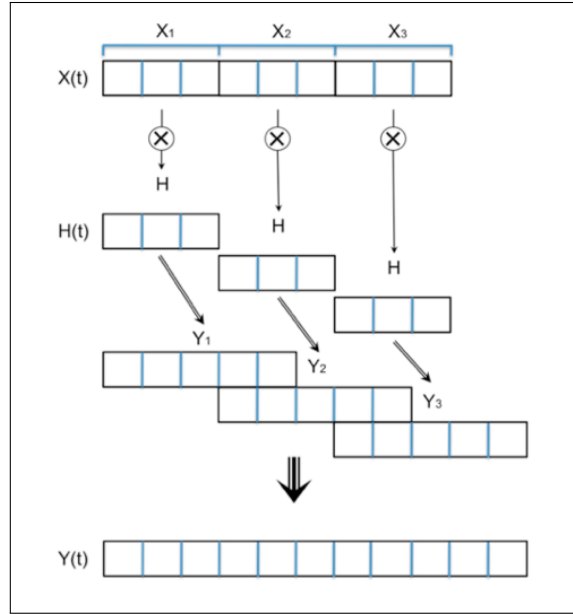


Figure 3.3: Schematic view of the overlap-add convolution method.

$$x_k[n] := \begin{cases} x[n + kL] & n = 1, 2, \dots, L \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

where L is an arbitrary segment length.

$$x[n] = \sum_k x_k[n - kL] \quad (3.5)$$

So $y[n]$ can be written as a sum of convolutions:

$$y[n] = \left(\sum_k x_k[n - kL] \right) * h[n] = \sum_k (x_k[n - kL] * h[n]) \quad (3.6)$$

The method is depicted in Figure 3.3

It is particularly useful for our tasks since it works on independent pieces of input and thus is well suited for a parallelized approach such as one that employs a GPU.

3.2 Reference CPU implementations

In order to make comparisons with the GPU implementations that we will present we need a reference implementation that can serve as a basis in terms of execution time and bitwise precision. For this reason three different prototypes have been developed that use different algorithms.

The first two prototypes are Matlab scripts that use both a Time Domain and a Frequency Domain approach. Since the computational complexity for the Time Domain approach is $O(n^2)$ this can not be used when the filter kernels are big. In our experiments, according to a Max/MSP implementation that will be introduced in the following section, we choose to limit the size to 256 samples.

The frequency domain implementation (presented in [43]) will be used to validate the results in terms of bitwise precision. Since Matlab is mainly intended as a prototyping environment there is no focus on performance and every other implementation can outperform our Matlab testbase by orders of magnitude. Moreover, this implementation works only in “direct mode”; this implies that a single FFT is performed for the entire signal and therefore the algorithm may not be applicable for long sequences due to memory constraints or implementation limits. Source code for both the Matlab implementations are presented in Appendix A.

The last CPU implementation is written in C++ and is based on the FFTW3 library (see [25]). It is based on the architecture presented in Figure 3.2 and implements both modalities (Direct and OLA) previously discussed.

The FFTW library itself is based on Cooley-Tukey algorithm [19]. As presented by the authors, the interaction of the user with FFTW occurs in two stages: planning, in which FFTW adapts to the hardware, and execution, in which FFTW performs useful work for the user. To compute a DFT, the user first invokes the FFTW planner, specifying the problem to be solved. The problem is a data structure that describes

the “shape” of the input data - array sizes and memory layouts - but does not contain the data itself. In return, the planner yields a plan, an executable data structure that accepts the input data and computes the desired DFT. Afterwards, the user can execute the plan as many times as desired.

3.2.1 A CUDA convolution engine

For the CPU implementation with CUDA we were able to implement both Direct and OLA algorithm. We consider the benefits of both approaches in the following section while presenting performance comparisons. For FFT we use a library called CUFFT which is actually based on FFTW3 library with some other optimizations specifically designed for GPUs. One of the current issue is the CUFFT limit of 64 millions of points.

3.2.2 An OpenCL convolution engine

One of the current limitations is that the factorization algorithms works only for powers of 2 (radix-2). So the payload should be adapted to make the sum with the length of the filter kernel to be the closest greater power of 2.

3.3 The CGPUconv prototype

From a number of the previously cited prototypes we derived a single application that allows the user to choose between a CPU- or a GPU-based algorithm and between a direct mode (a single window for the entire signal) and an Overlap-add mode. It is structured as a “wrapper” around the single module that has the capability of opening audio files and writing them back to disk thanks to libsndfile (see [15]). It is a command line tool that compiles and executes both on Microsoft Windows, Apple OSX,

and Linux applications as long as they have, or there exists a version of:

- Libsndfile for I/O;
- FFTW3 library for CPU implementation;
- CUDA Framework;
- OpenCL driver.

The program can be adapted by removing functionalities provided by any subset of the previous requirements by removing the components that make use of that prerequisite.

The source code is available from the author at

<http://www.lim.dico.unimi.it/CGPUconv>.

3.3.1 Performance Comparisons

Performances of these algorithms depends on the size of input. Therefore, to characterize the “trade-off”, we tested them with different input sizes. To make a reliable comparison we choose to use as input signals a logarithmic sine sweep and its TRM (time reversal mirror) so the output should be the δ function (Dirac delta function) or, to be more precise, the limited bandwidth approximation of the sinc (*sinus cardinalis*) function.

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \quad (3.7)$$

$$\int_{-\infty}^{\infty} \delta(x) dx = 1 \quad (3.8)$$

$$\text{sinc}(x) = \frac{\sin(x)}{x} \quad (3.9)$$

We then compute the time spent on the convolution procedure, excluding the load procedure that reads from audio files and the write to disk procedure for the results,

which are collateral to our primary goal. A special case is represented by the first execution for both the CUDA and OpenCL implementation where for the former there exists some extra time devoted to the load of the environment while for the latter, apart from the aforementioned setup, we have to take into account the time that the driver allocate to compile kernel functions.

The algorithms were executed on an OS X 10.6.8 equipped Apple Macbook Pro 13.3" (MacBookPro5,5), Intel Core 2 Duo processor @2,53 GHz, 8 GB Ram, NVIDIA GeForce 9400GM VRAM 256 MB shared memory. OpenCL drivers are provided by the operating system (1.1 compatible), and the CUDA framework is version 4.0.

All the audio files are high quality PCM uncompressed files and have a sample rate of 96 kHz and a quantization word of 24 bit. With this bit depth the theoretical dynamic range is ~ 144 dB.

For each algorithm we measured the difference computed between the signal under test and the reference (coming from the Matlab implementation) with a phase inversion. So the difference on a sample by sample basis gives us a new signal that can be used as a degree of similarity between the two original signals. For each and every proposed approach this signal is below -122 dB FS (dB on the full scale) meaning there is no practical difference, and the result is in the order of magnitude of the noise floor.

Coming to the execution time of the algorithms we propose a summary of the results presented in Figures 3.4, 3.5 and Table 3.1. Results are depicted as a function of the number of input samples, averaged over 100 runs.

We also present in Table 3.2 results for a "real-case scenario". We have a violin sound that is three minutes long and a reverberant impulse response of 1 second (sample rate 96kHz)

- Input: 17703123 samples ($\sim 3'10''$)

	Direct Mode			Overlap-add		
	<i>CPU</i>	<i>CUDA</i>	<i>OpenCL</i>	<i>CPU</i>	<i>CUDA</i>	<i>OpenCL</i>
128	7	-	160	15	88	160
256	3	95	180	16	95	180
512	5	95	263	17	93	265
1024	11	97	363	2	93	361
2048	3	94	300	3	97	290
4096	11	98	293	4	100	290
8192	32	100	248	9	95	340
16384	47	115	388	17	97	393
32768	80	145	392	36	107	395
65536	44	194	407	90	140	439
131072	315	320	435	186	167	430
262144	2450	562	511	398	240	504
524288	1215	970	572	915	385	599
1048576	2676	1728	845	1871	680	948
2097152	8066	-	1303	3901	1197	1534
4194304	24146	-	2409	8487	-	2967
8388608	22976	-	2926	-	-	2608
16777216	-	-	5836	-	-	-

Table 3.1: Performance comparisons. Time in ms.

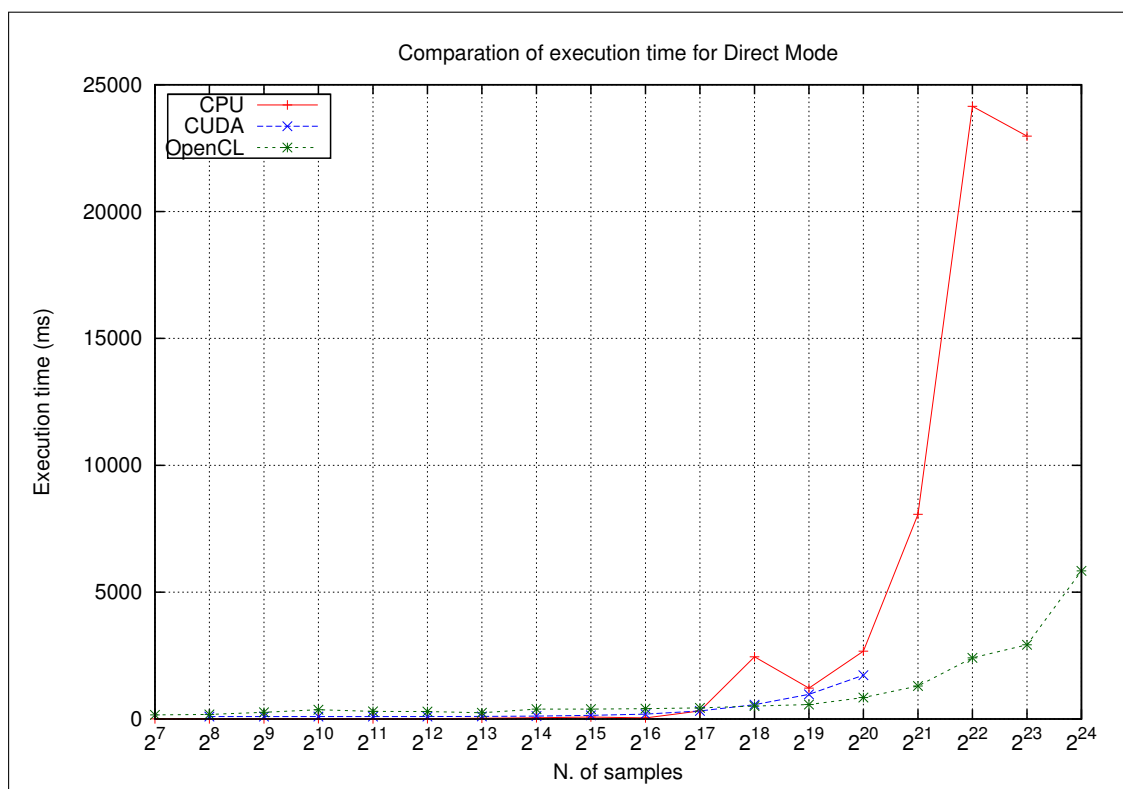


Figure 3.4: Execution time for Direct mode depending on input size.

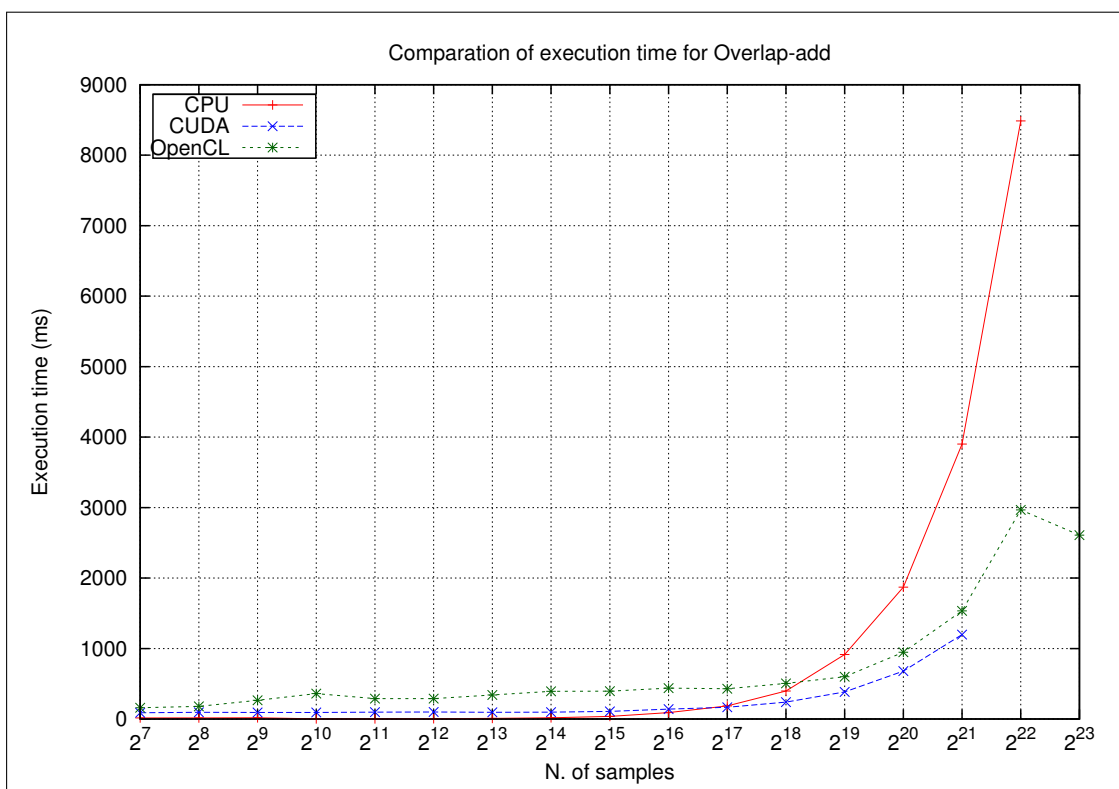


Figure 3.5: Execution time for Overlap-add depending on input size.

	Direct	OLA
CPU	-	9699
CUDA	-	6181
OpenCL	7486	6699

Table 3.2: Performance comparisons. Time in ms.

- Kernel: 96000 (~1”)

Please note that “-” occurs when there is not enough free video RAM to handle the data. The idea here is to have a system that can run on most home computer so the relatively old and low powerful graphic card is a good example of what can be achieved with standard equipment. There are difference between implementations and this can be explained by the different way of encoding real and complex numbers. Also note that there does not exist a concept of “paging” for video RAM so if a structure is too big to fit in memory there is no automatic way to handle the situation.

3.4 Summary and Discussion of the results

In this Chapter we presented a number of prototypes that are suitable for real time spatialization of sounds. Some issues are still present but we want to point out that the basic concepts here expressed are valid and mark a profitable direction.

Performance results suggest that for a number of real case applications there are benefits that can be at least of 1/3 of the execution time (compared to the reference CPU implementation) and can be further improved with other GPU-specific, but not hardware specific, optimizations. Benefits are increasingly evident as the size of the filter kernel grows and this is particularly useful for convolution with long reverberant impulse responses (e.g. BRIRs) that can be employed in order to render real environments.

A Head-Tracking based Binaural Spatialization Tool

In one of the definitions of Virtual Reality, simulation does not involve only a virtual environment but also an immersive experience (see [66]); according to another author, instead of perception based on reality, Virtual Reality is an alternate reality based on perception (see [49]). An immersive experience takes advantage of environments that realistically reproduce the worlds to be simulated.

In our work, we are mainly interested in audio aspects. Even limiting our goals to a realistic reproduction of a single (or multiple) sound source for a single listener, the problem of recreating an immersive experience is not trivial. With a standard headphones system, sound seems to have its origin inside the listener's head. This problem is solved by binaural spatialization, described in Chapter 1, which gives a realistic 3D perception of a sound source S located somewhere around a listener L . Currently, most projects using binaural spatialization aim at animating S while keeping the position of L fixed. Thanks to well known techniques, such a result is quite easy to achieve. However, for an immersive experience this is not sufficient: it is necessary to know the position and the orientation of the listener within the virtual space in order

to provide a consistent signal [51], so that sound sources can remain fixed in virtual space independently of head movement, as they are in natural hearing [9].

As a consequence, we introduce a head-tracking system to detect the position of L within the space and modify the signal delivered through headphones accordingly. The system can now compare the position of S with respect to L and respond to his/her movements.

At the moment, audio systems typically employ magnetic head trackers thanks both to their capability of handling a complete 360° rotation and to their good performances. Unfortunately, due to the necessity of complex dedicated hardware, those systems are suitable only for experimental or research applications. But the increasing power of home computers is supporting a new generation of optical head trackers, based primarily on webcams.

This work proposes a low cost, “off the shelf”, spatialization system which only relies on resources available to most personal computers. Our solution, developed with MAX/MSP, is based on a webcam head-tracking system and binaural spatialization implemented via convolution.

This chapter is structured as follows. First we provide a short review of related literature and similar systems. Then we will describe the integration of a head-tracking system via MAX/MSP externals - namely the multi platform, realtime programming environment for graphical, audio, and video processing used to implement our approach - and the realtime algorithms involved in the processing of audio and video streams.

4.1 Related Works

We present here other similar approaches and projects which served as a basis in the development process.

- **Binaural Tools:** A MAX/MSP patch from the author of the CIPIC database that performs binaural panning using head related transfer function (HRTF) measurements. The panner takes an input sound file and convolves it with a measured sound response recorded from a selectable angle and elevation. The output can optionally be recorded to a sound file. The program was created based on some parts of Vincent Choqueuse's binaural spatializer for Max/MSP [16]. We started from these works to develop our approach. They are inspiring as they do not use external libraries and rely solely on MAX capabilities. This approach also has some drawbacks. For example, in order to perform spatialization efficiently, other techniques could be used but they must be separately implemented.
- **Spat~:** A spatial processor for musicians and sound engineers [35]. Spat~ is a real-time spatial processing software which runs on the Ircam Music Workstation in the MAX graphical signal processing environment. It provides a library of elementary modules (pan-pots, equalizers, reverberators, etc.) linkable into a compact processor that integrates the localization of sound events together with the manipulation of room acoustical quality. This processor can be configured for various reproduction formats over loudspeakers or headphones, and controlled through a higher-level user interface including perceptual attributes derived from psychoacoustical research. Applications include studio recording and computer music, virtual reality, and auralization. The stability and quality of this library could be useful to redesign some structures of our spatializer and achieve better quality and performances.
- **bin_ambi:** A real-time rendering engine for virtual (binaural) sound reproduction [47]. This library is intended for the use with Miller Puckette's open source computer music software Pure Data (Pd). The library is freely downloadable and can be used under the terms of GNU General Public License. It provides

a simple API, easy to use for scientific as well as artistic projects. In this implementation there is a room simulation with 2 sound objects and a listener. One direct signal and 24 early reflections are calculated and rendered per sound object. Sound rendering, based on mirror image sources, is used for the early reflections. Each reflection is encoded into the Ambisonics domain (4th order 3-D) and added to the Ambisonics bus. The listener rotates the whole Ambisonics field, the Ambisonics decoder renders the field into 32 discrete signals of 32 virtual loudspeakers. All 32 speaker signals are filtered by its HRTF in relation to the left and to the right ear (binaural decoding). Interpolation is one of the critical points of such applications. We can choose an approach like the one proposed here that could give a theoretical better interpolation and sound quality but increases the computational complexity of the system.

- 3D-Panner [69]: A SuperCollider-based spatialization tool for creative musical applications. The program spatializes monaural sounds through HRTF convolution, allowing the user to create 3D paths in which a sound source travels. In 3D-Panner the user can easily create unique paths that can range from very simple to very complex. These paths can be saved independently of the sound file and applied to any other monaural source. During playback, the sound source is convolved with the interpolated HRTFs in real-time to follow the user-defined spatial trajectory. This project is inspiring for our work because we plan to introduce new features, such as moving sound sources, and we need a way to describe and handle trajectories.

4.2 An overview of MAX/MSP

In this section we briefly introduce MAX/MSP, a software system originally designed and implemented by Miller Puckette [56] and then developed by David Zicarelli.

MAX/MSP is an integrated platform designed for multimedia, and specifically for musical applications [18]. This graphical real-time data-flow environment can be used by programmers, live performers, “traditional” musicians, and composers. As shown in Figure 4.1, the environment has evolved in a significant manner since its authors started the development process in the 1980s. Some of the key concepts have not changed over time, such as the overall flexibility and modularity of the system.

MAX/MSP basic functions can be extended by the use of:

- *patchers*, i.e. sub-patches recalled by the user under other patches,
- *externals*, i.e. newly created objects implemented usually in C/C++ via the MAX/MSP framework and its API.

MAX/MSP can run on Microsoft Windows and Apple OS X.

The program interface consists primarily of two window types: *MAX* and *patcher window*. The former gives access to the program settings and visualization of system messages, allowing control of the workflow. The latter is the place where the user creates and interacts with the application by placing objects and linking them together.

Patches present two different states: *edit mode* and *performance mode*. In *edit mode*, the user can add *objects*, modify and link them. In *performance mode* the patch follows its workflow and the user can interact with it in real-time.

Objects are represented like “black boxes” which accept input through their *inlets* and return output data through their *outlets*. Programs are built by arranging these entities on a canvas (the *patch*) and creating a data flow by linking them together through *patchcords*. Data are typed; as a consequence, not every arbitrary combination of links is valid.

Choosing the linking order influences the scheduler priority. The rule is right-to-left execution of links. MAX/MSP implements two ways to control the priority of both messages and events: a standard *parallel execution*, and *overdrive*. When overdrive is

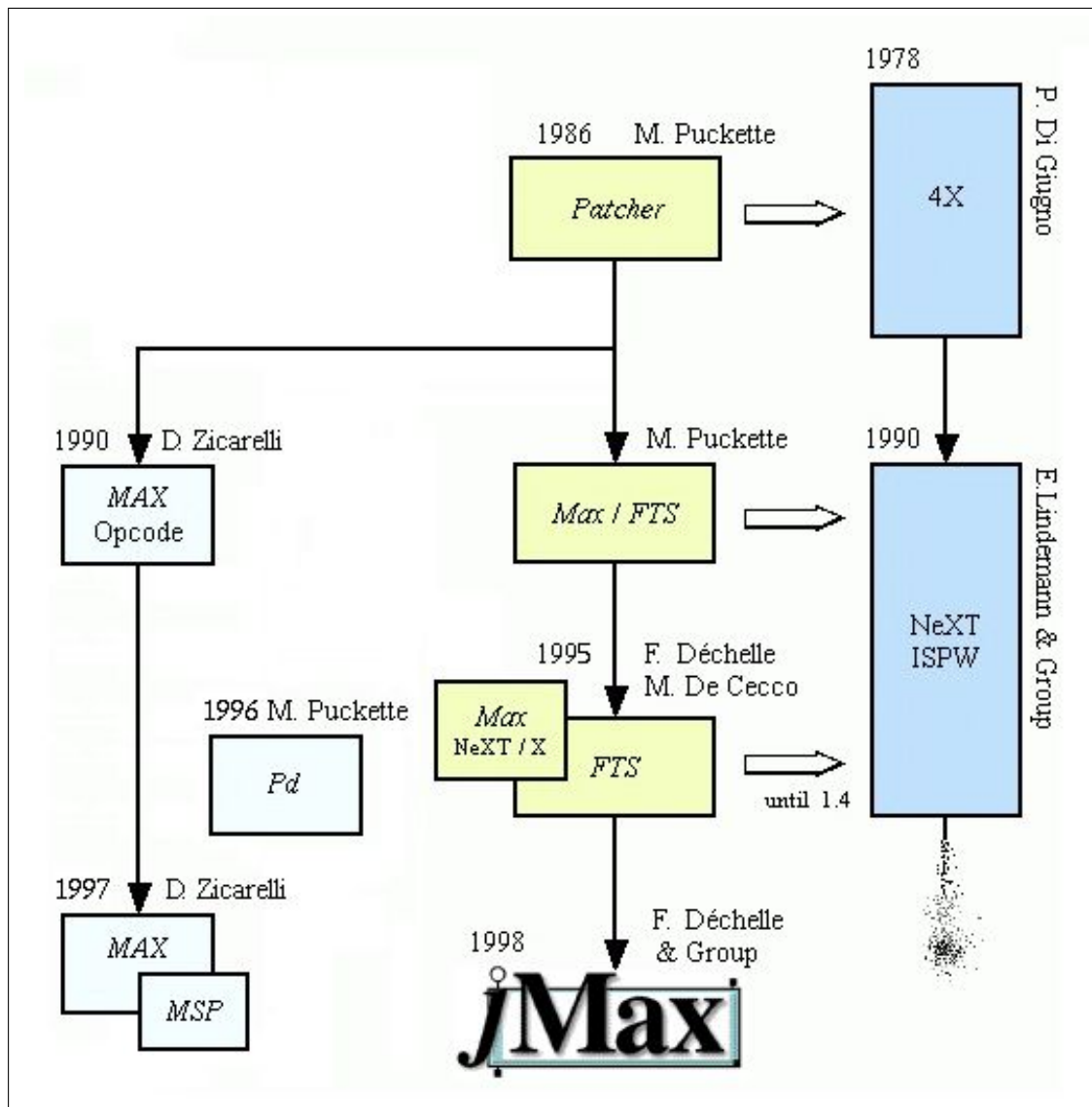


Figure 4.1: The evolution of the MAX family.

enabled, high priority events are actually given priority over low priority events. In this case, the software engine uses two threads for the execution of events so that high priority events can raise an interrupt and be executed before a low priority event has finished.

4.3 Integrating a Head-tracking System into MAX

In our work, we choose to adopt `faceAPI`, namely an optical face tracking system developed by Seeing Machines [2] that provides a suite of functions for image processing and face detection encapsulated in a tracking engine. It is a commercial product — freely usable only for research purposes — that implements a head tracker with six degrees of freedom. It can be seen as a “black box” which grants access to tracking data through a simple interface oriented to programming tasks. Basically the engine receives frames from a webcam, processes them and then returns information about the position of the head with respect to the camera.

MAX provides developers with a collection of APIs to create external objects and extend its own standard library [80]. The integration of the head tracker requires to create a base project for MAX (we used the so called “minimum project”) and then add references to `faceAPI` to start developing the external.

When MAX loads an external module, it calls its `main()` function which provides initialization features. Once loaded, the object needs to be instantiated by placing it inside a patch. Then the external module allocates memory, defines inlets and outlets and configures the webcam. Finally, the `faceAPI` engine starts sending data capturing the position of the head. In our implementation, the external module reacts only to `bang` messages:¹ as soon as a message is generated, a function of `faceAPI` is invoked to return the position of the head through `float` variables.

¹A bang is a MAX special message that causes other objects to trigger their output.

Each MAX object must be defined in terms of a C structure, i.e. a structured type which aggregates a fixed set of labelled objects, possibly of different types, into a single object. Our implementation presents only pointers to the object outlets in order to directly pass variables to the tracking engine.

```
typedef struct _head {
    t_object c_box;
    void *tx_outlet, *ty_outlet, *tz_outlet;
    void *rx_outlet, *ry_outlet, *rz_outlet;
    void *c_outlet;
} t_head;
```

Such values represent translation along three axes (tx, ty, tz), orientation of the head in radians (rx, ry, rz), and a confidence value. After their detection, values are sent to their corresponding *outlets* and they are available to the MAX environment. In brief, the headtracker external presents only one inlet that receives bang messages and seven outlets that represent the values computed by the tracking engine.

4.4 The “Head In Space” Application

This section aims at introducing the Head in Space (HiS) application for MAX. As discussed in Section 4.3, we assume that our head-tracking external acts as a black box that returns a set of parameters regarding the position of the head.

In Figure 4.2 a workflow diagram of the system is shown. This is a “specialized” version of the one proposed in Figure 3.1. It adds a module for tracking the user position and for the interpolation of impulse responses. The “Convolution” box will be presented in the following sections using a traditional CPU approach and also exploiting GPUs capabilities.

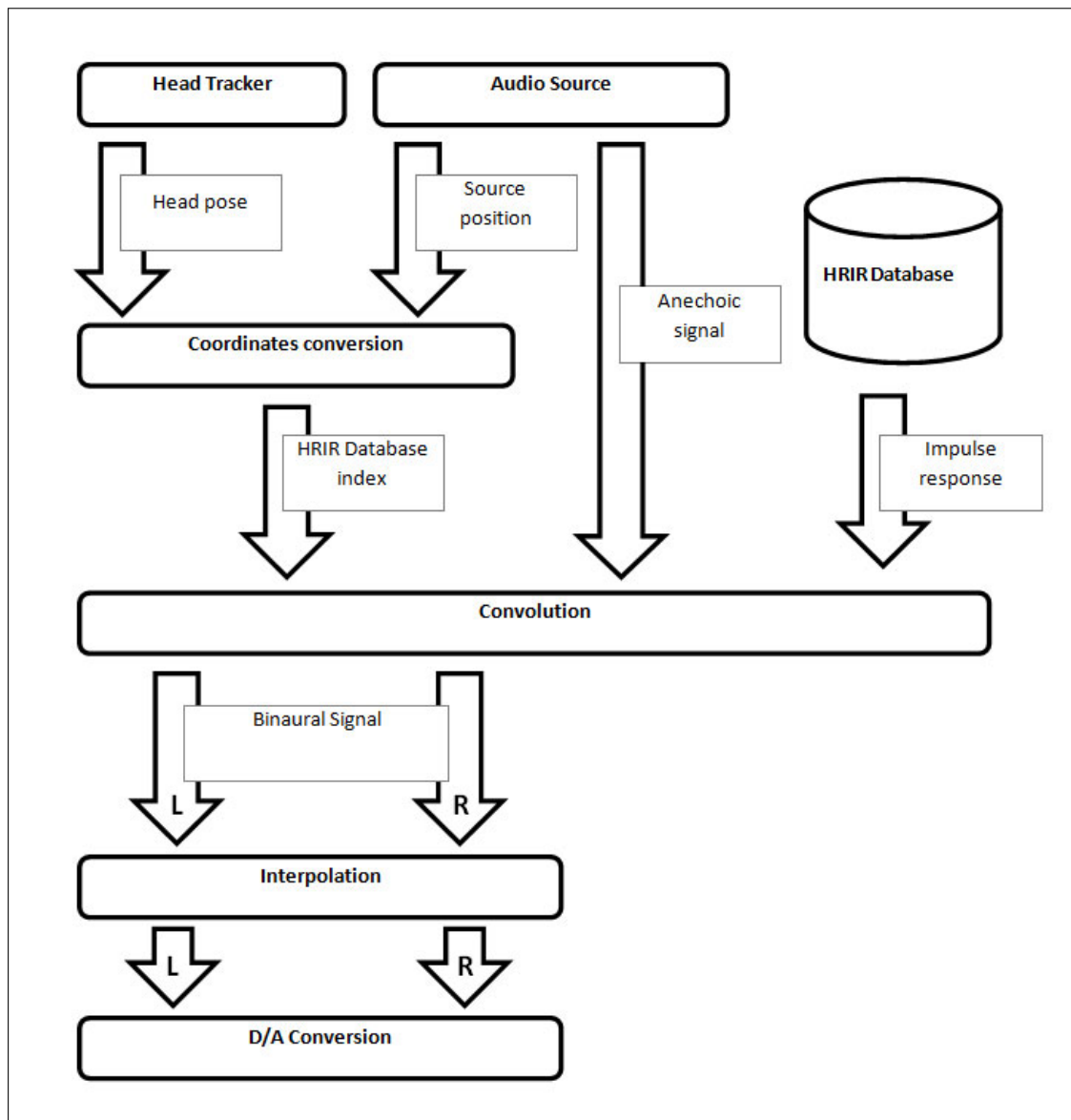


Figure 4.2: The workflow diagram of the system.

In input, two sets of parameters are available to the system, in order to define: 1. the position of the listener, and 2. the position of the audio source. Given this information, and taking into account also the position of the camera, it is possible to calculate the relative position of the listener with respect to the source in terms of azimuth, elevation and distance. This is what the system needs to choose which impulse response to use for spatialization. Once the correct HRIR is obtained from the database, it is possible to perform convolution between a mono audio signal in input and the stereo impulse response. Since the position both of the listener and of the source can change over time, an interpolation mechanism to switch between two different HRIRs has been implemented.

4.4.1 Coordinates Extraction

The spatializer uses a spherical-coordinates system that has its origin in the center of the listener's head. The sound source is identified by a distance measure and two angles, namely azimuth on the horizontal plane and elevation on the median plane. Angular distances are expressed in degrees and stored in the patch through integer variables, whereas the distance is expressed in meters and is stored as a floating point number.

Please note that the head tracker presents coordinates in a cartesian form that has its origin in projection cone of the camera. Thus the representation of coordinates of the spatializer and the one of the head tracker are different and a conversion procedure is needed. The conversion process first performs a roto-translation of the system in order to provide the new coordinates of translation both of the source and of the head inside a rectangular reference system (the patch realizing these functions is depicted in Figure 4.3).

Referring to Figure 4.4, given the coordinates for a generic point P , representing

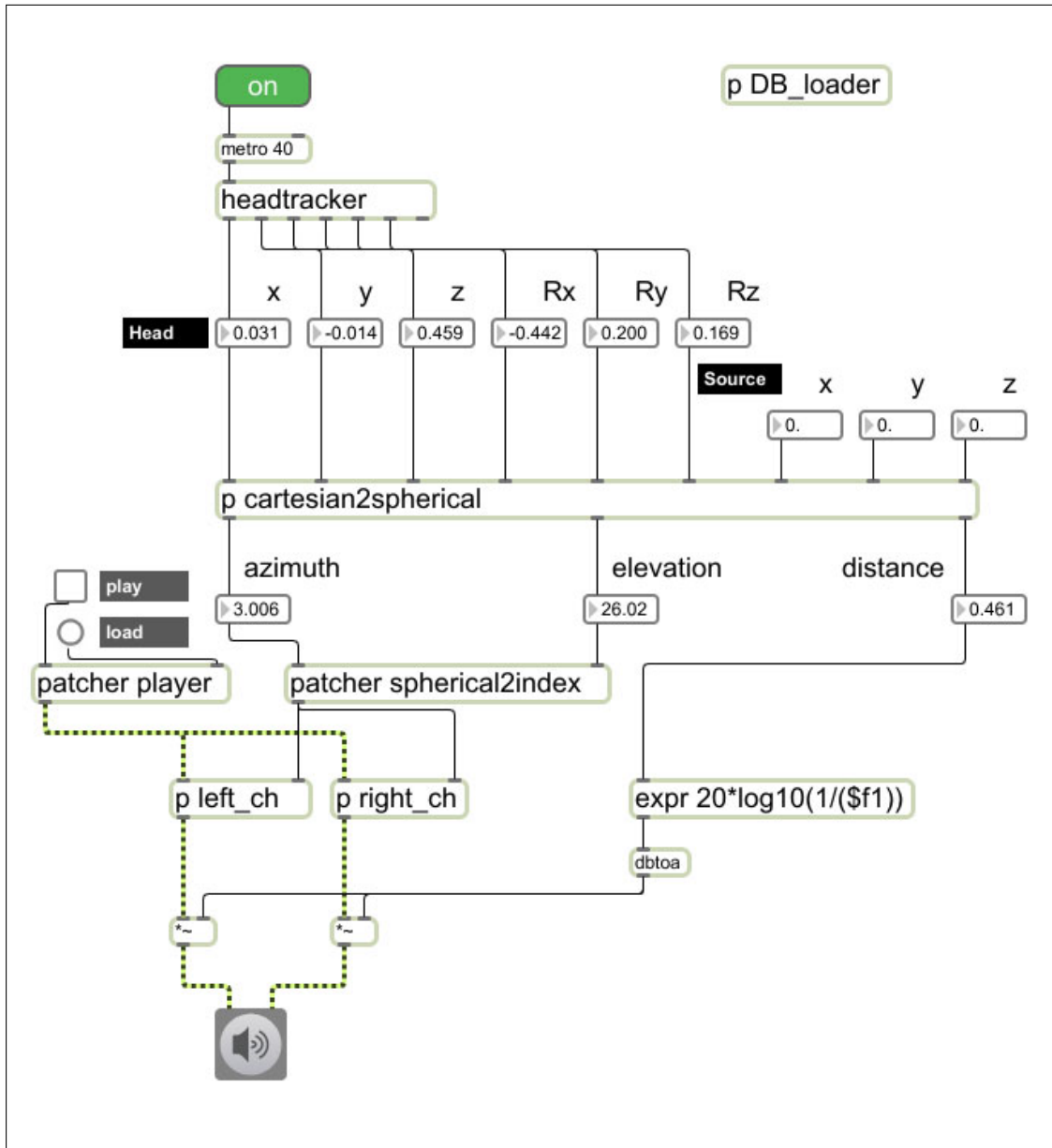


Figure 4.3: An overview of the patch.

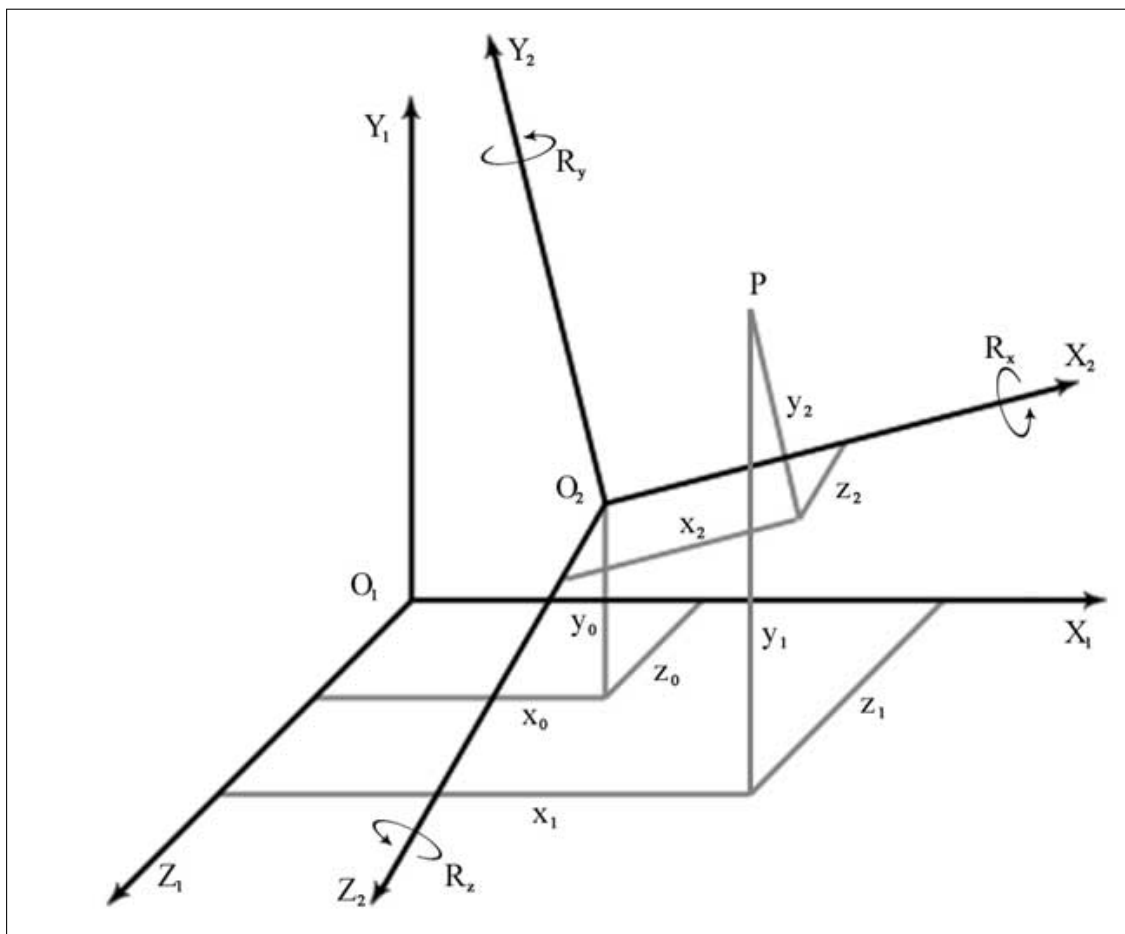


Figure 4.4: The translation system.

the source in a system $(O_1; X_1, Y_1, Z_1)$, we can determine a set of coordinates in a new cartesian plane $(O_2; X_2, Y_2, Z_2)$ that refers to the position of the head through the relation:

$$V_2 = V_0 + (1+k) \cdot R \cdot V_1 \quad (4.1)$$

where:

$$\begin{array}{l}
 V_0 = \begin{array}{|c|} \hline x_0 \\ \hline y_0 \\ \hline z_0 \\ \hline \end{array} \quad \text{translation components} \\
 V_1 = \begin{array}{|c|} \hline x_1 \\ \hline y_1 \\ \hline z_1 \\ \hline \end{array} \quad \text{known coordinates of } P \text{ in } O_1 \\
 V_2 = \begin{array}{|c|} \hline x_2 \\ \hline y_2 \\ \hline z_2 \\ \hline \end{array} \quad \text{unknown coordinates of } P \text{ in } O_2 \\
 k = 0 \quad \text{scale factor} \\
 R = R_x \cdot R_y \cdot R_z \quad \text{rotation matrix} \quad (4.2)
 \end{array}$$

R is the matrix obtained by rotating each cartesian triplet with subscript 1 along its axes X_1, Y_1, Z_1 with rotation of R_x, R_y, R_z to displace it parallel to X_2, Y_2, Z_2 . Rotation

matrixes are:

$$R_x = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(R_x) & \sin(R_x) \\ 0 & -\sin(R_x) & \cos(R_x) \end{vmatrix} \quad (4.3a)$$

$$R_y = \begin{vmatrix} \cos(R_y) & 0 & -\sin(R_y) \\ 0 & 1 & 0 \\ \sin(R_y) & 0 & \cos(R_y) \end{vmatrix} \quad (4.3b)$$

$$R_z = \begin{vmatrix} \cos(R_z) & \sin(R_z) & 0 \\ -\sin(R_z) & \cos(R_z) & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad (4.3c)$$

the product $R_x \cdot R_y \cdot R_z$ is calculated with (4).

$$\begin{aligned}
 R = & \begin{array}{c} \cos(R_y) \cos(R_z) \quad \cos(R_x) \sin(R_z) + \sin(R_x) \sin(R_y) \cos(R_z) \quad \sin(R_x) \sin(R_z) - \cos(R_x) \sin(R_y) \sin(R_z) \\ - \cos(R_y) \sin(R_z) \quad \cos(R_x) \cos(R_z) - \sin(R_x) \sin(R_y) \sin(R_z) \quad \sin(R_x) \cos(R_z) + \cos(R_x) \sin(R_y) \sin(R_z) \\ \sin(R_y) \quad \quad \quad - \sin(R_x) \cos(R_y) \quad \quad \quad \cos(R_x) \cos(R_y) \end{array} \\
 & \tag{4.4}
 \end{aligned}$$

We can now derive formulas to calculate the position in the new system:

$$\begin{aligned}
 x_2 &= (x_0 + x_1) [\cos(R_y) \cos(R_z)] \\
 &\quad + (y_0 + y_1) [\cos(R_x) \sin(R_z)] \\
 &\quad + \sin(R_x) \sin(R_z) \cos(R_z) \\
 &\quad + (z_0 + z_1) [\sin(R_x) \sin(R_z) \\
 &\quad \quad - \cos(R_x) \sin(R_z) \sin(R_z)]
 \end{aligned} \tag{4.5}$$

$$\begin{aligned}
 y_2 &= (x_0 + x_1) [\cos(R_y) \sin(R_z)] \\
 &\quad + y_0 + y_1 [\cos(R_x) \cos(R_z) \\
 &\quad \quad - \sin(R_x) \sin(R_z) \sin(R_z)] \\
 &\quad + (z_0 + z_1) [\sin(R_x) \cos(R_z) \\
 &\quad \quad + \cos(R_x) \sin(R_z) \sin(R_z)]
 \end{aligned} \tag{4.6}$$

$$\begin{aligned}
 z_2 &= (x_0 + x_1) \sin(R_y) \\
 &\quad + (y_0 + y_1) [\sin(R_x) \cos(R_y)] \\
 &\quad + (z_0 + z_1) [\cos(R_x) \cos(R_y)]
 \end{aligned} \tag{4.7}$$

Now we can calculate spherical coordinates using the following formulas:

$$\text{distance } \rho = \sqrt{x^2 + y^2 + z^2} \tag{4.8}$$

$$\text{azimuth } \varphi = \arctan\left(\frac{z}{x}\right)^2 \tag{4.9}$$

$$\text{elevation } \theta = \left(\frac{y}{\sqrt{x^2 + y^2 + z^2}} \right) \tag{4.10}$$

The new set of coordinates can be employed to retrieve the right HRIR from the database. Since our database includes only HRIRs measured at a given distance, we

²arg function is used instead of arctan to cover the entire range.

only use azimuth and elevation. How to use the distance value to simulate the perception of distance will be explained in Section 4.4.4. Since not all the possible pairs of azimuth and elevation have a corresponding measured HRIR within the database, we choose the database candidate that minimizes the euclidean distance.

4.4.2 The Convolution Process

This section describes the convolution process between an anechoic signal and a binaural HRIR. We use the CIPIC database [3], consisting of a set of responses measured for 45 subjects at 25 different values for azimuth and 50 different values for elevation. Each impulse consists of 200 samples.

For the sake of simplicity we present here a system that, since the relatively small length of impulses, exploit a time-domain approach. This approach can be extended with the results of the convolution engines provided in previous chapters with little effort. It is worth to cite that some limitation, such as a maximum number of 255 channels, may still be present but they are intrinsic to the MAX/MSP environment.

Figure 4.5 illustrates the detail of the subpatch for one channel. From its first inlet it receives the anechoic signal, while from the second it gets the index for HRIR within a *buffer~* object. HRIRs are stored in a single file that concatenates all the impulses. The process is performed one time for left channel and another one for right channel. Inside the database, azimuth and elevation values are numbered through an ad hoc mapping. Given an azimuth position *naz* and an elevation position *nel* we can calculate the starting point within the buffer with the formula:

$$[((naz - 1) \cdot 50) + (nel - 1)] \cdot ir_{length} \quad (4.11)$$

A *buffir~* object is a finite impulse response (FIR) filter that loads both coefficients from the buffer and an audio signal, and then performs the convolution in the time do-

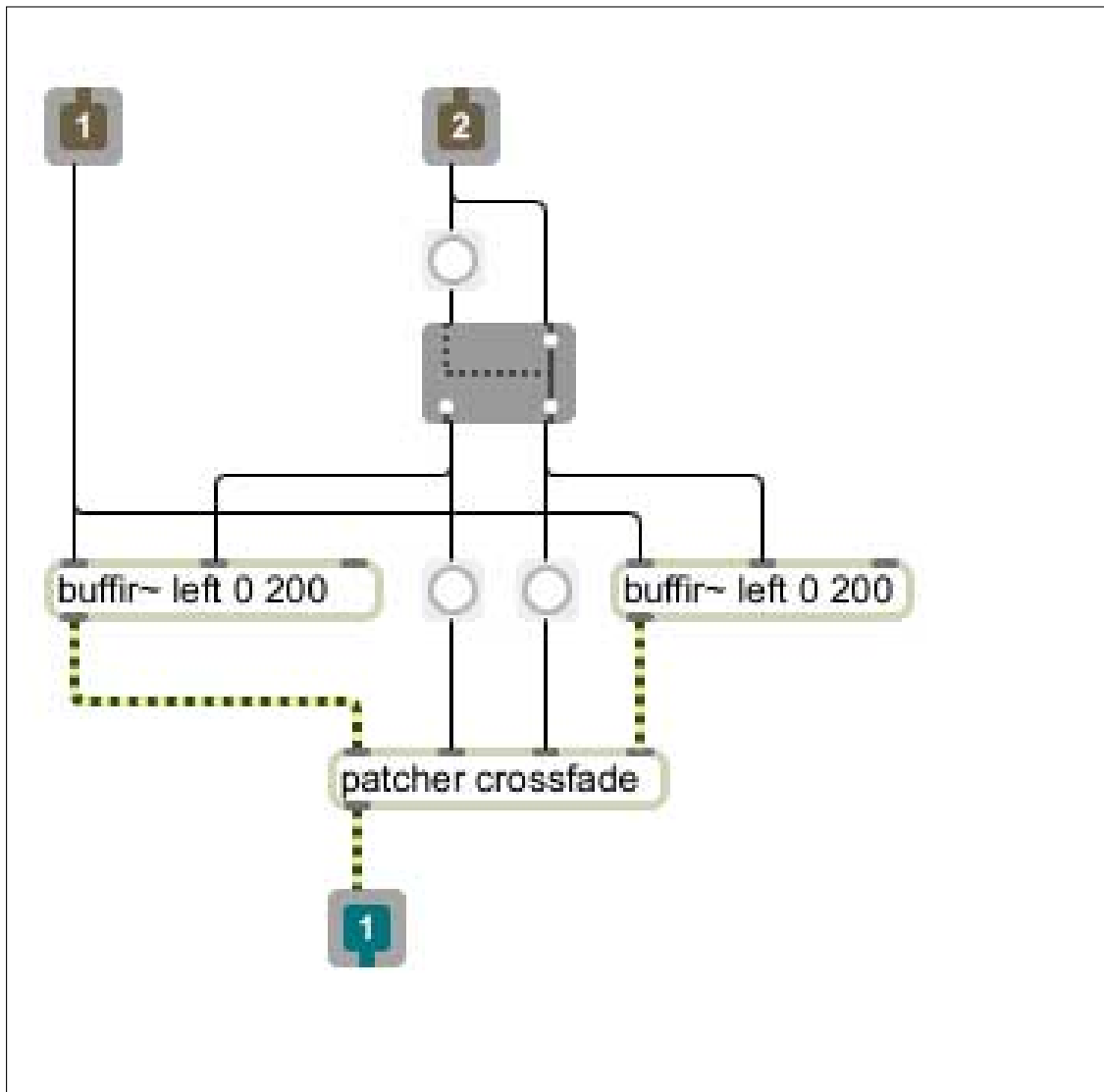


Figure 4.5: The detail of the MAX subpatch for the convolution process via CPU.

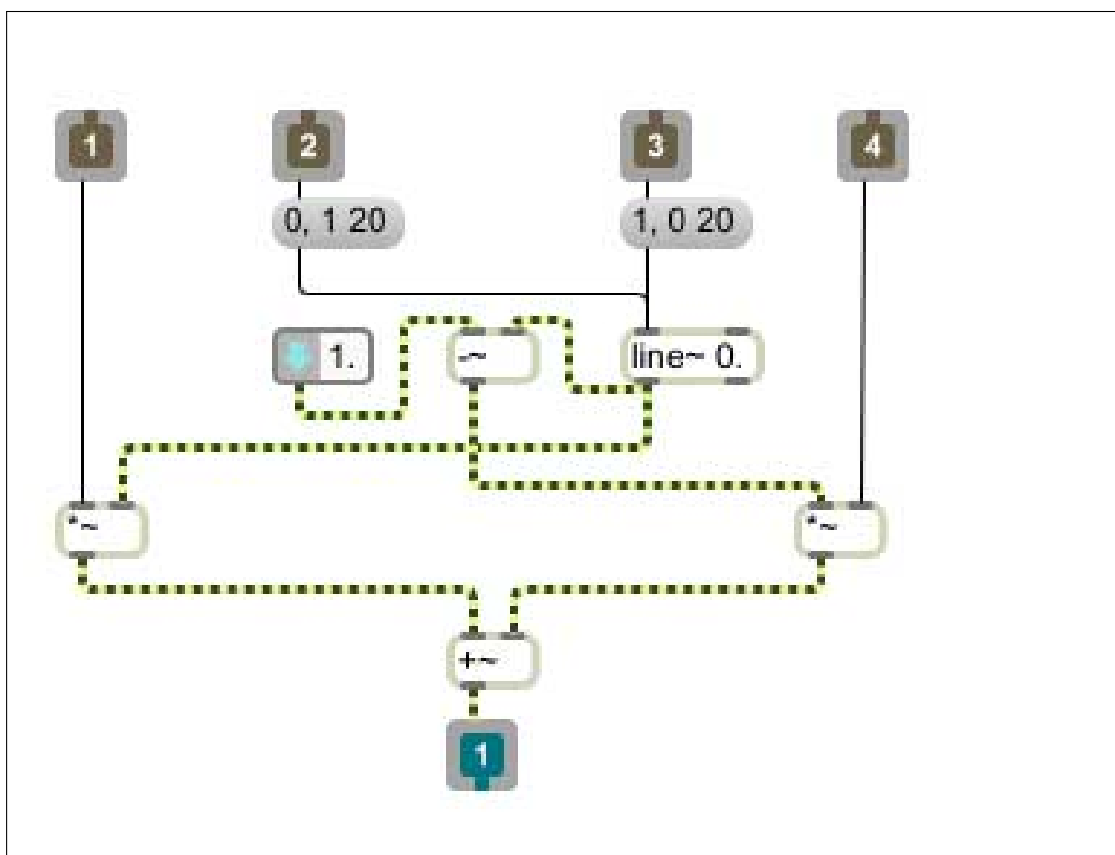


Figure 4.6: The detail of the MAX subpatch for the crossfade system.

main. Convolution is implemented through a FIR filter since the small number of samples of HRIRs makes it computationally convenient to perform it in the time domain instead of frequency domain. *buffir~* object allows to store up to 256 coefficients.

4.4.3 Interpolation and Crossfade Among Samples

One of the known problems related to the use of the HRIR for spatialization is the interpolation between two signals convolved with two different impulses. This is a very common case for such real-time applications because when moving from one azimuth value to another impulses are very dissimilar. As a consequence, output signals can change abruptly, thus affecting negatively the perceived quality of the system. We

have designed a simple yet performing interpolation procedure based on crossfade to limit the artifacts produced by the switch between impulses. For further information, regarding the simulation of moving sound sources see [42], [74].

The approach replicates the audio stream for each channel that leads to changes to the convolution subpatch (Figure 4.6). For the CPU approach we add a second *buffir~* object so now the first filter will produce signals convolved with the current impulse and the second filter will be loaded with the new HRIR provided by the new position. Then new signal will gradually overcome the signal from the other filter with a crossfade function. Once done, the role of the two filters is switched. This behaviour is achieved through a *ggate~* object.

As a performance issue it should be noted that in a real time environment every redundant operation should be avoided. In our implementation this means that a crossfade between impulse responses is needed only if a switch has been detected by a *change* object that gives a value in output only if it is not equal to its previous value. This avoids unnecessary computations by the CPU that is useless if applied to the same impulse response and could lead to a degradation in terms of quality. Another improvement is given by the use of *speedlim~* object that establishes the frequency of messages in terms of the minimum number of milliseconds between each consecutive message. It could happen that changing azimuth and elevation at the same time will result in two different new messages being generated in a rapid sequence. This could lead to a premature refresh in the filter coefficients leading to a loss of quality. With this component, they are spaced by at least 40 ms. This value is chosen according to the typical refresh rate of a video stream (25 fps). This value is also used to define the crossfade duration between samples, and in our implementation the crossfade is linear. The user can define a value between 5 ms and 20 ms. Through experimentation, depending on the CPU power, it is possible to achieve a good quality even at 5 ms. So

the overall delay between changes is:

$$20 \text{ ms} + \frac{200 \text{ samples}}{44100 \frac{\text{samples}}{\text{sec}}} \quad (4.12)$$

4.4.4 Simulation of Distance

One of the limitations of the CIPIC database is presenting measures only at one given distance. In order to simulate the distance effect, our patch contains a simple procedure based on the inverse square law. The function is implemented by an *expr~* object³ with the expression:

$$20 \log_{10} \left(\frac{1}{\text{distance}} \right) \text{ dB} \quad (4.13)$$

We limit the range of the distance value, which is a relative value, produced by the head-tracking system between 0.1 and 2. Conventionally a value of 1 identifies the reference distance of the impulse response, and in this case no gain is applied. The mentioned distance value is employed to feed the gain of each channel. The process could be enhanced by adding a filter which simulates air absorption or using a database where HRIRs are measured at various distances or adding BRIRs (Binaural Room Impulse Responses).

4.4.5 The Graphical User Interface

The software application that implements the algorithms previously described is a standard patch for MAX/MSP. The patch uses an ad hoc external to implement the head-tracking function.

After launching it, the software presents a main window comprised of a number of panels and a floating window containing the image coming from the webcam after

³An *expr~* object evaluates C-like expressions.

faceAPI processing. In the latter window, when a face is recognized, a wireframe contour is superimposed over the face image.

In Figure 4.7 we present the user interface of the application. Regarding the main window, it is organized in several panels. First, it allows one to switch on and off the processing engine. In addition, a number of text boxes and buttons are used to set the position of the camera and of the source. Other controls provide feedback about the derived position of the listener and the corresponding translation into azimuth, elevation, and distance. A 3D representation (with the use of the OpenGL support of Jitter) of the system made of the listener (dark cube) and the source (white sphere) is also provided and updated in real time.

The bottom right panel contains the controls to choose an audio file to be played and to start the playback.

4.5 Multiple Webcams Head-tracking

The system described in the previous section can be enhanced to support multiple webcams for extending the range covered by the engine and/or for improving the precision of the system. In order to achieve this goal the external object needs to be modified. We decide to implement the head-tracking engine as an external software that sends OSC (Open Sound Control [77] [78]) messages over network to MAX/MSP.

We define a protocol for communication structured as follows:

- WEBCAM_MSG: */webcam*, *ifffffff* webcam_id tx ty tz rx ry rz confidence

The first parameter is an integer value used for identifying each webcam. Then seven floating point number are used to represent translation along three axes (tx, ty, tz), orientation of the head in radians (rx, ry, rz), and a confidence value. The application allows the user to decide the identifier associated with the webcam and specify an IP address and a port.

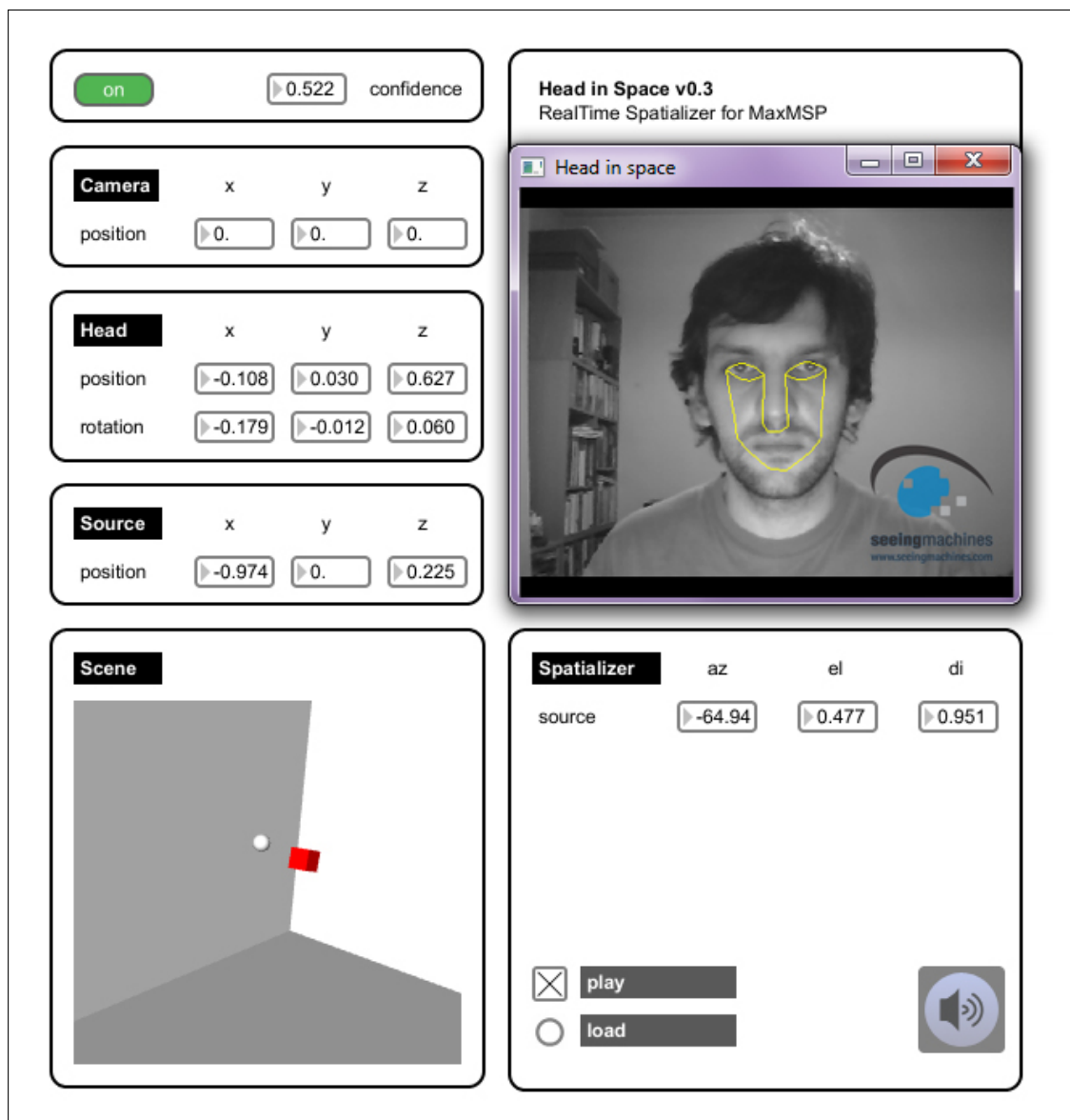


Figure 4.7: The graphical user interface of the program.

The workflow of the system is the same but now, if more than a webcam is used, a procedure to merge the results coming from each webcam is needed. A simple weighted mean using the confidence value as weight can solve the problem taking into account also the possibility that one or more webcams are not sending any useful data (*confidence* = 0).

A calibration procedure is performed at the startup of the system. Even if the position of the webcams is unknown, as long as all of them are tracking the user, the user can decide the reference position.

The source code of the application is presented in Appendix B.

4.6 Summary and Discussion of the results

This Chapter has described a working application that performs real-time spatialization of an audio signal based on the position of the listener.

The system can be improved in several manners. The use of a single webcam corresponds to a limited resolution of azimuths and elevations (± 90 azimuth, $-30/+60$ elevation, data coming from faceAPI specifications). It could be possible to combine more cameras in order to fully represent the space and choose the camera with the highest confidence value.

Another possible improvement is adding support for more than one source in order to render a richer environment. It could also be interesting to take into account moving sound sources; this implies that a way to describe trajectories needs to be implemented.

The use of CIPIC database limits the number of possible measured distances and led us to implement a distance simulation mechanism, whereas it would be desirable to switch among HRIRs measured at various distances. Also the 200-samples HRIRs do not account for reverberation, so a reverberation tool is needed.

Since the application is structured in modules, it can be easily extended in order to

support the future changes we have mentioned.

The source code and application are freely available at:

<http://www.lim.dico.unimi.it/HiS>.

Psychoacoustics and Perceptual Evaluation of Binaural Sounds

The use of binaural recordings as well as the increasing availability of auralization tools (see previous chapter for further details) is pushing the need of research on the perception of such materials. What we search is a possible influence of “schizophonia,” literally the fracture between a soundscape and its reproduction (as defined by R.M. Schafer in [61]), on the performance of this peculiar type of recordings. Binaural recordings and binaural spatialized sounds are in general more effective, in terms of spatial rendering, when listened to through headphones, so they are well suited for mobile platforms, which are by definition context-independent. If a binaurally spatialized sound was proven to be particularly schizophonic with respect to the surrounding context, it could somehow lose its perceptive effectiveness and be recorded by a listener’s auditory system as “unlikely,” making all the computational effort required to perform binaural spatialization nearly useless. On the other hand, knowing which parameters are less dependent on context could lead to better engineered binaural spatialization systems.

As stated by Tsingos in [73] “With increasingly complex environments, the cost of

auralization can quickly become a significant bottleneck for interactive applications, such as video games or simulators. While limitations of the human auditory perception system have been successfully leveraged for lossy audio compression, real-time auralization pipelines still implement brute-force processing, independent of the content to process and perceptive capabilities of a human listener” and we then want to investigate if differences exists between various types of sound so it could be possible to differentiate the spatialization quality with respect to their criticality.

The research area of sound spatial perception ([9]) received a lot of attention and there are important sets of experimental results (e.g. [10]). Most experiments deal with source localization and very few with movement recognition ([52]). Most of the experiments use only artificial sound stimuli and fixed listening conditions in order to obtain measurable results. The adoption of such a small set of stimuli (e.g. pure sines, narrow-band noises, and trains of pulses) is not representative of the richness and complexity of the typical sounds used for composition of musical pieces, installations, or audiovisual productions. The application of results only from this type of experiment to real world conditions could lead to misinterpretations of some phenomena and could lead to poor performances. Regarding the set of stimuli we then choose to add natural sounds.

We obviously need to introduce some limitations, so as we are working on binaural recordings, we chose to fix the listening conditions to headphones only. These experiments are primarily a consequence of the experience gathered from a set of “binaural concerts” proposed in Italy by the Crackerjack collective (<http://www.crackerjack.it>), and produced by AGON in collaboration with LIM, where the musicians play around a dummy-head placed on stage, and the audience listens to the performance through headphones.

5.1 Experimental Design

This section is intended to describe the underlying process that led us to choose a combination of sound objects, spatial coordinates, and rooms that were presented during the test to the subjects. The formulation of a questionnaire was also one of the critical parts as it could influence the way the subjects perceive the proposed test.

In this experiment, the perception of realism of a binaural recording is assumed to be related to the difference between the listening context and the context in which the recording has been performed. By “context” we basically mean the acoustic features of the room in which the recording is performed or listened to.

The stimuli that subjects are asked to compare are pairs of binaural recordings of the same sound, which differ only by the room in which they have been recorded. The former version of the sound was recorded in the same room where the subjects are located during the test while the latter was recorded in another room (room B) with different acoustic features.

Each couple is presented to the subject in random order (A, B or B, A), and the subject is asked to state which version of the sound is perceived as more realistic. The subject is also asked to record how much difference is perceived between the two versions of each sound, on a scale ranging between 1 (very subtle) and 5 (very different). Subjects are also allowed to express no preference if no difference is perceived.

5.1.1 Room Acoustics

Binaural recordings are performed in two different rooms with different acoustic features and the test must be set in one of the two rooms (reference room, or room A) and the subject must sit in the same position where the dummy head was placed when the recording was performed.

We have chosen to maximize the difference between the two rooms because prelim-

inary tests with a small number of subjects showed that subtle differences are unlikely to influence the perception of realism.

Objective measures are not considered as critical for our aim: we principally focus on context discrimination, so rooms have been chosen first of all according to the difference between them, regardless their peculiar acoustic features. Thus we do not deeply focus on materials and acoustics, and to define each room here we substantially use size and reverberation time. The reference room (room A) is an editing studio with acoustically treated walls, plasterboard ceiling and tiled floor, namely a “dry” room with a very short reverberation time. It measures 7.4 x 5.9 m in width and 3.1 m in height, with a T_{30} of 260 ms. The stairwell of a two-storey building with concrete walls and ceiling and tiled floor serves as room B. Although it is less wide than room A, being 7.3 x 4.3 m wide, it is considerably higher (10.31 m), This gives it a remarkably long reverberation time, with a T_{30} of 3320 ms.

5.1.2 Spatial Coordinates

The influence of context on perceived realism could be related to the position of the sound source in space. In our experiment only static sound sources are considered: trajectories and movement are not regarded as relevant for our scope.

Each sound source can be thus located in a tridimensional space using a set of three spatial coordinates (see Chapter 1). The origin of the coordinate system is placed in the center of the head, at the intersection between an imaginary plane placed between the top of the ear canals (horizontal plane), and the vertical symmetry axis of the head (median plane) [10]:

$$p_n = \{\varphi_n, \delta_n, r_n\} \quad (5.1)$$

where:

- φ is the angle on the horizontal plane (clockwise);

- δ is the angle on the median plane;
- r is the distance.

In order to reduce the size of the test, the value of δ was always set to 0° , so only sound sources located on the horizontal plane are considered in the test. Values of φ are a subset of multiples of 45° . To further reduce the size of the experiment, some of the values are also omitted. Values of φ are:

$$\varphi = \{45^\circ, 90^\circ, 225^\circ, 270^\circ\} \quad (5.2)$$

Values $\varphi = 0^\circ$ and $\varphi = 180^\circ$ (front and rear) were discarded as we are focusing on cases where interaural differences play a significant role in localization, since localization of frontal and rear sound sources is mainly influenced by pinna-related filtering [7]. Values $\varphi = 135^\circ$ and $\varphi = 315^\circ$ are regarded as redundant to, respectively, $\varphi = 225^\circ$ and $\varphi = 90^\circ$, being symmetrical to these values. For each value of φ , two different values of r are considered, one for a “close” sound source, one for a “far” sound source:

$$r = \{50 \text{ cm}, 150 \text{ cm}\} \quad (5.3)$$

The set of chosen angles and distances is depicted in Figure 5.1.

We would have set the second value of r to at least $r = 200\text{cm}$, but due to space constraints caused by the small size of the rooms this was not possible.

5.1.3 Classification of the Stimuli

The distinction between natural and artificial sounds, and the consequent role their peculiar features play in the perception of realism of spatialization, has been considered as one of the main topics in our experimental work. The stimuli we used in the test have were divided into two major classes, depending on whether they are artificial or

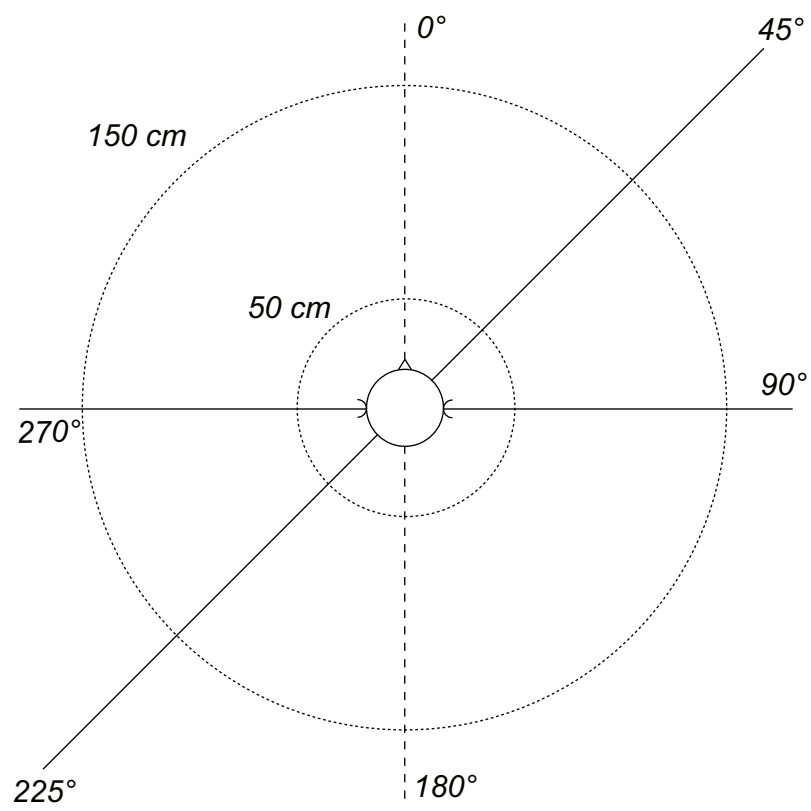


Figure 5.1: Coordinates of sound objects.

natural sounds. To better clarify this intuitive distinction, we consider a sound object (so_n) as defined by the tuple

$$so_n = \{t, i, b, e_i, e_b\}, \quad (5.4)$$

where:

- t is time extension (duration) of the sound object;
- i is intensity;
- b is spectral richness;
- e_i is the amplitude envelope, considered as a function of i over t ;
- e_b is the spectral envelope, considered as a function of b over t .

According to this definition, which is based upon the work of Pierre Schaeffer on theory of sound and musical objects [60], we classify a sound object as natural if its spectral richness is relevant and its amplitude and spectral envelopes both evolve in a complex way. On the other hand, a sound object is considered artificial if spectral richness is very low, or if its envelopes don't show a particular level of complexity. This distinction has been used to define the "semantic relevance" of a stimulus. We assume natural sounds to be perceived as more significant than artificial ones, consequently drawing more attention of our perceptual system in the process of auditory scene analysis [12]. Six types of stimuli were used in the experiment, four of which (stimuli s_1 , s_2 , s_3 and s_4) are classified as artificial (less semantically relevant sounds), while two (s_5 and s_6) are classified as natural (more semantically relevant)¹. Stimuli s_1 to s_4 were synthesized with CSound, using a noise generator for s_1 and s_2 and a sine wave generator set to a frequency of 1000 Hz for s_3 and s_4 . A percussive amplitude

¹Values were chosen also according to opinions reported by subjects during test.

envelope has been applied to s_1 and s_3 , while a slowly evolving attack-sustain-release was applied to s_2 and s_4 (both envelopes are presented in Figure 5.2).

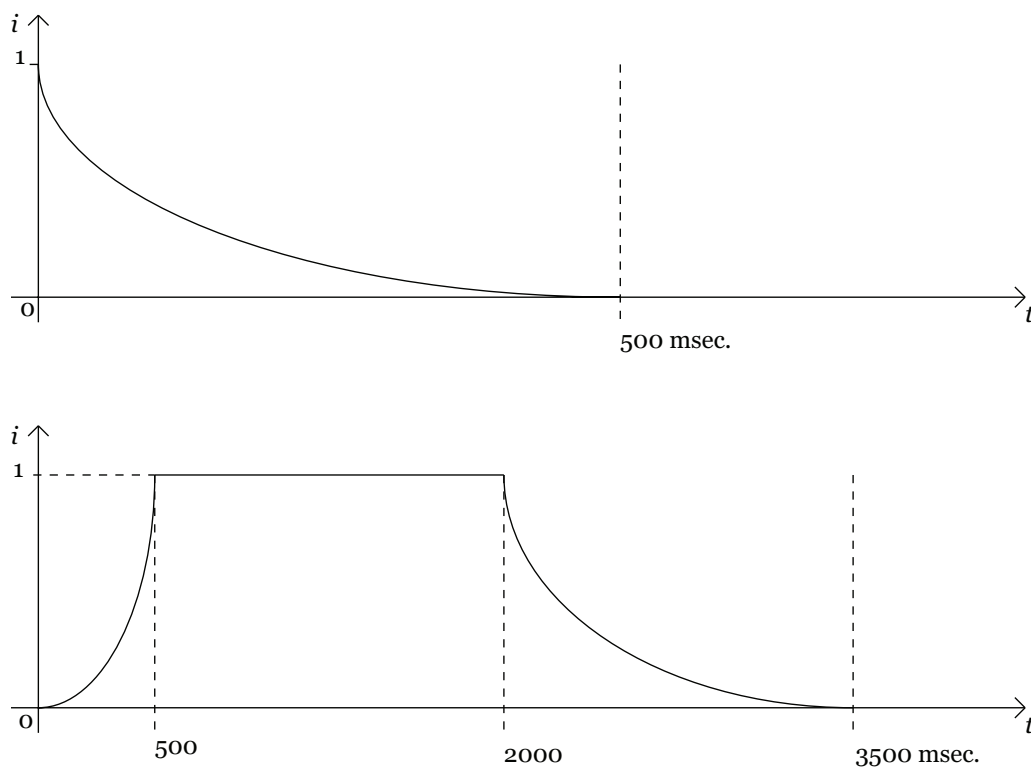


Figure 5.2: Two different types of envelope.

For stimuli s_5 and s_6 an anechoic recording of a male voice and a musical phrase played by sampled flute were used. We present a compact visualization of sounds in Table 5.1.

	Artificial				Natural	
	so_1	so_2	so_3	so_4	so_5	so_6
b	Rich	Rich	Poor	Poor	Rich	Rich
e_i	Percussive	Slow	Percussive	Slow	Articulated	Articulated
e_b	Static	Static	Static	Static	Articulated	Articulated

Table 5.1: The sound stimuli grouped by types used in the experiment.

5.1.4 Binaural Recordings

All the available sounds were recorded through a self-made dummy-head [41]. We used the plastic head of a mannequin filled with polyurethane spray, varnished with a rubber-based paint, that roughly imitates the absorption of skin. We use reproductions of the pinnae made of rubber (produced by GNRsound) and two lavalier condenser microphones from Sennheiser (MKE 2P) placed at the end of the cavum-conchae.

The dummy head was placed on a fixed stand at the height of 120 cm measured from the center of the pinna, to reproduce the height of a sitting listener.

The loudspeaker was a Fostex 6301B with coaxial woofer and tweeter.

As soundcard we used a MOTU Traveler firewire interface with integrated preamplifiers controlled by an Apple laptop and a custom Max/MSP patch for playback and recording.

5.1.5 Classification of subjects

Overall, $N = 26$ listeners were involved in the experiment. The subjects were divided into two classes, in order to determine whether the influence of the listening context on perception depends somehow on the musical training of the subject. Before taking the test, the subjects were asked to complete a short questionnaire, in which they were asked to report their musical training, their familiarity with sound and music technology, signal processing and music production, their profession (if related to music or audio) and their listening habits. These data have been then used to categorize the listeners as “naive,” if they had no musical training nor familiarity with audio technology, or as “expert” if they had some musical training or familiarity with audio technology. Collected data would have allowed more detailed classification, but the limited number of subjects prevented us from performing further subdivisions. As a result, we had a perfect split into two groups of $N_1 = 13$ naive listeners and $N_2 = 13$ expert listeners.

5.1.6 Task and Questionnaire

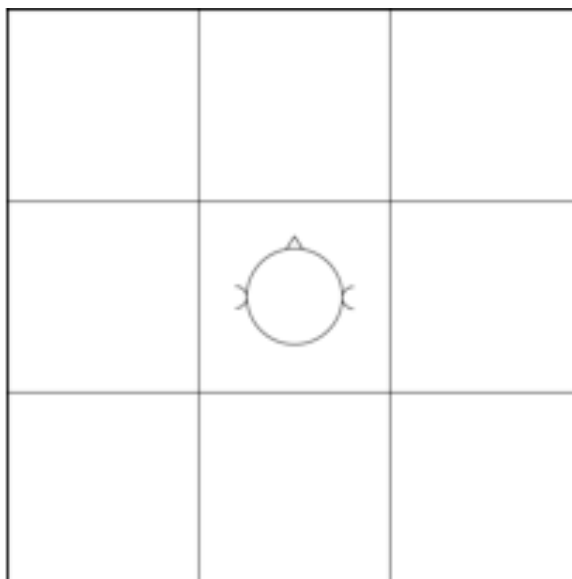


Figure 5.3: The portion of the questionnaire where subjects report about the position of sound objects.

A two-part questionnaire was prepared to collect and then process the results. Part 1 is described in Section 5.1.5, and is intended to collect informations about the subject's musical training, in order to perform the classification. Part 2 is a multiple choice form, where subjects are asked to state for each pair of binaural recordings:

1. which one, between the two sounds, is perceived as more realistic;
2. how much difference is perceived between the two sounds, on a scale ranging from 1 (very subtle) to 5 (very different);
3. where the sound source is located, on a 9-quadrant graphic form depicting a head in the central square, as shown in Figure 5.3. This field serves as a quality check: results where this answer is incorrect are discarded (set to 0) before performing the analysis.

Each subject is asked to evaluate an overall number of $N_c = 48$ pairs of sounds (one for each couple of $\{\varphi_n, r_n\}$ space coordinates). Listening condition was fixed with headphones, which have been calibrated to the same level of acoustic pressure measured during the recording. Subjects are not allowed to adjust the volume of the headphones.

The test is run by a supervisor, who plays the pairs of binaural recordings on a Max/MSP patch that generates random couples of recordings, keeping track of their order to then correctly pair each stimulus with its corresponding answer in the form.

In order to get each listener's ears acquainted with the acoustic features of the reference room, subjects are brought in the room in which the test is performed, where they are instructed by a supervisor, and then asked to fill the part of the questionnaire about their musical training right before taking the test. Subjects are placed in the same position where the dummy head was placed during the recordings. This operation usually takes a few minutes. The subject can ask questions about the task to the supervisor. The subject is not aware of the real aim of the test, to avoid the answers to be affected by the listener's expectations.

The test is not strictly timed, but subjects are asked to answer as quickly as possible: too reasoned answers are indeed unlikely to be useful for our goal, as conscious analysis of the perceived stimulus could be very misleading. For the same reason, subjects could only listen to each pair of recordings once. We take note of the time spent on task and this is used during the analysis.

5.2 Results

For each subject an Excel DataSheet that encompasses both the answers given in the preliminary questionnaire and the proper test was obtained.

The responses of the listeners were processed to create matrices of values ranging

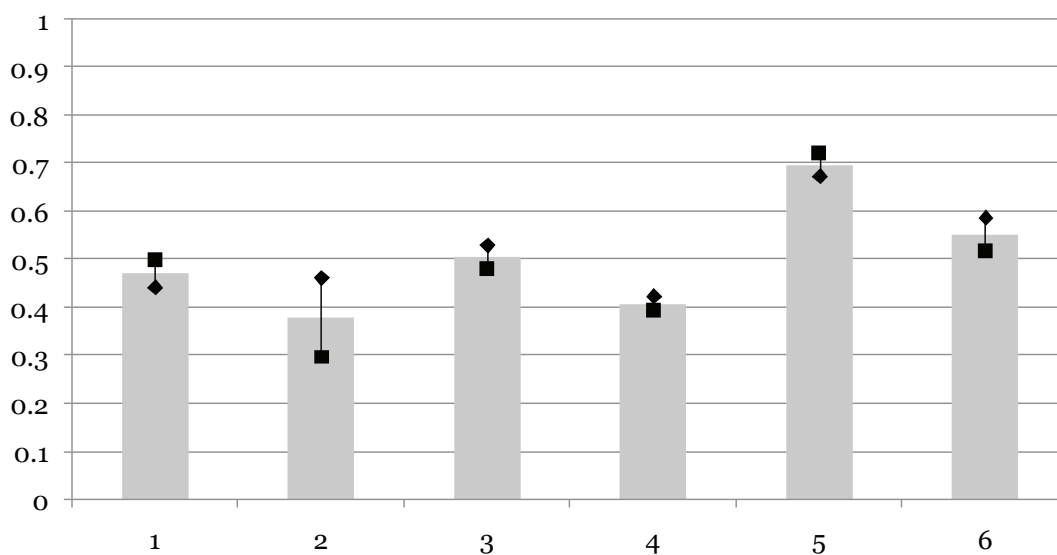
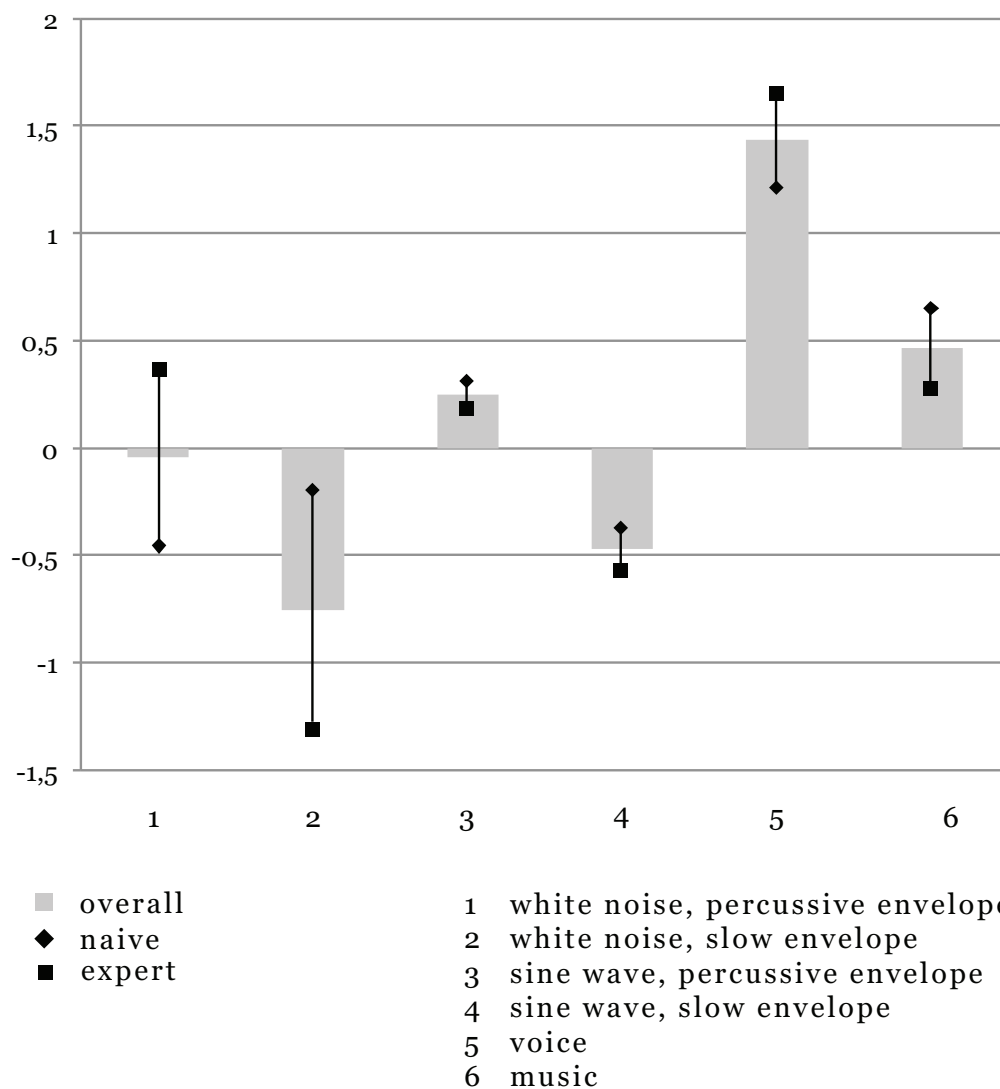


Figure 5.4: Mean values grouped by sound type and by subject class.

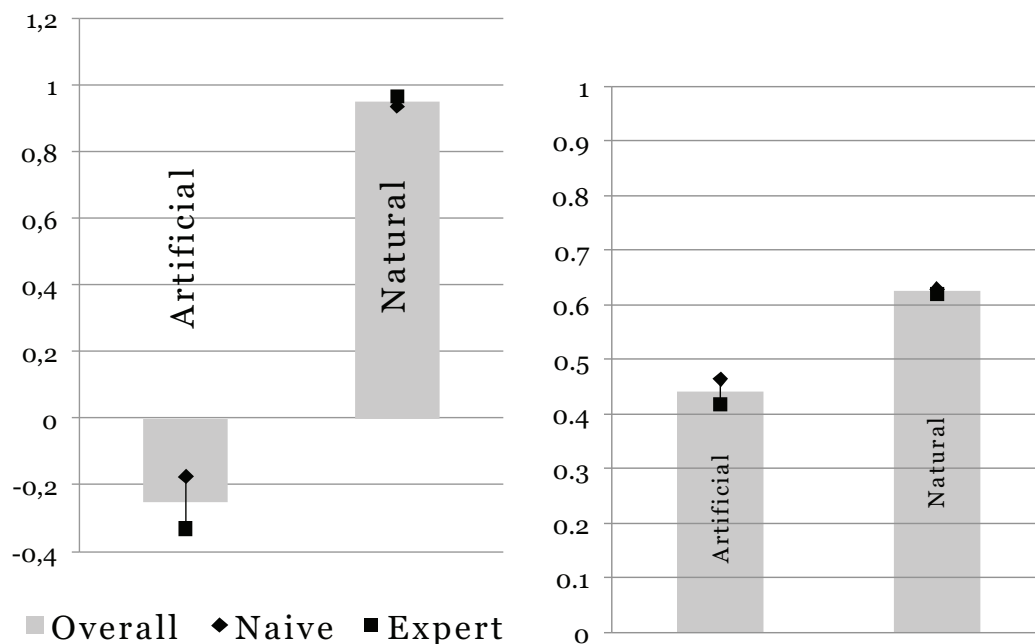


Figure 5.5: Overall values for artificial and natural sound classes.

between -5 and $+5$. These values are computed assigning 1 is to correct answers, -1 to incorrect answers, and 0 if no answer is given at all, and then multiplying by the values given in the ordinal scale. The answers could then be analyzed to compare the results of each listener, grouped by listener “type” or they could be processed according to one or more sound characteristics described previously.

We then used the well-known SPSS software for statistical analysis in order to process our data. We ran a number of analysis specially focusing on clustering. We performed also TwoStep Clustering ([6]).

In Figure 5.4 values of arithmetic mean for all subjects are depicted according to sound object, then in Figure 5.5 the same values are grouped into just two sound classes: Artificial (Mean:0,441 StDev:0,496 CI:0,033) and Natural (Mean:0,625 StDev:0,484 CI:0,046). Values are also presented for Naive and Expert subjects (Mean:0,485/0,519 StDev:0,500 CI:0,039). As conjectured, better results are obtained for natural sounds.

The perception of realism for voice seems specially sensitive to schizophonia, while music seems less influenced by the different contexts. Both graphs are structured as follows: on the upper (or left) parts mean values calculated from the original matrix are presented while in the lower (or right) we used the matrix with only correct/wrong answer (1 or 0).

For sound position and distance (Figure 5.6) overall results are extremely low, and for distance no significative difference exists. For the position even if the values are low could be the case that localization and context discrimination are more effective for “off-axis” sounds.

With regard to clustering we use expert/naive as the categorical variable and we chose to perform each analysis with different values for the *maximum number of clusters* options. We let the program automatically choose this value, we fixed it to two, to four, and to the maximum number of clusters allowed by the software. In no case were more than four clusters produced except for the case of maximum the number where one cluster for each subject is produced. For artificial sounds SPSS automatically generated one cluster while for natural sounds two clusters were generated. This result shows that, while in the answers for artificial sounds no trend seems to exist, for natural sounds discrimination actually exists.

As an example we provide an in-depth analysis for the clusters obtained with so_5 . This situation, depicted in Table 5.2, can be summarized as follows: the automatic option returned the same two clusters as the fixed option. Cluster number 2 has most of the elements.² This subdivision reflects the mean value of answer given by subjects. This coincides with the hypothesis that natural sounds are easier to discriminate and this is especially true for voice.

Both the raw and the processed data are available.

²The condition of a single cluster grouping a vast majority of observations is sometimes called “elephant cluster.”

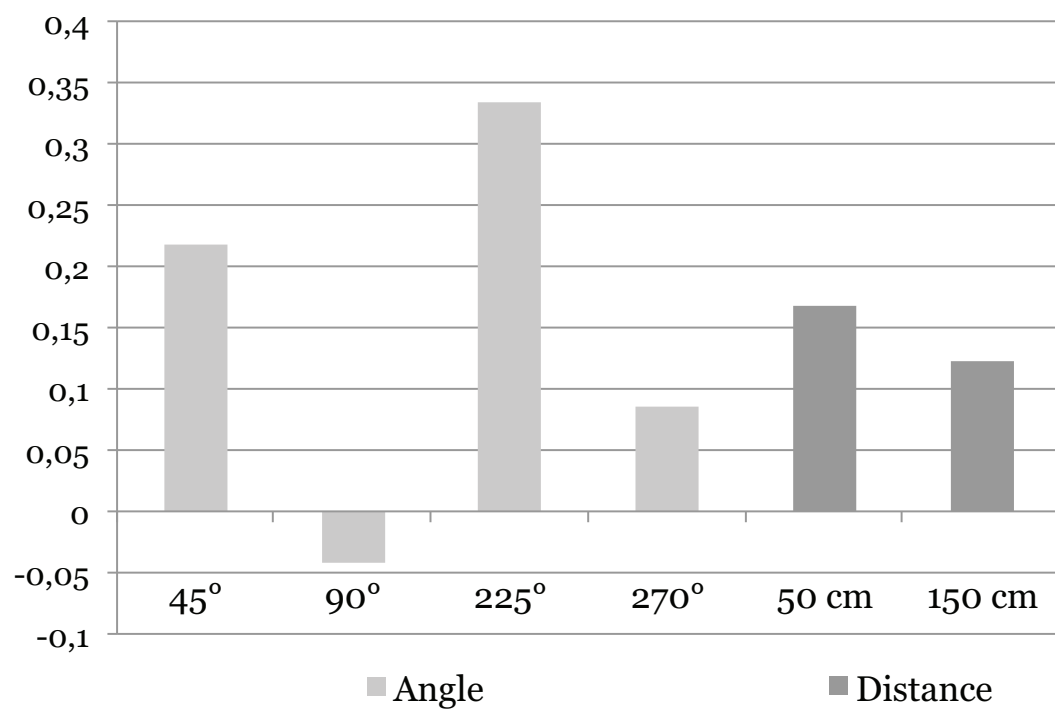


Figure 5.6: Mean values for different angles and distances.

Distribution					
Cluster	N	Combined %	Total %		
1	5	19,2%	19,2%		
2	21	80,8%	80,8%		
Combined	26	100%	100%		

Frequency for groups of subject	Naive		Expert		
	Cluster	Freq.	%	Freq.	%
	1	3	23,1	2	15,4
	2	10	76,9	11	84,6
	Combined	13	100	13	100

Centroids				
Cluster	<i>so</i> ₅ 45N		<i>so</i> ₅ 45F	
	Mean	Std. dev.	Mean	Std. dev.
1	-2,40	1,517	-1,40	3,507
2	2,86	2,351	2,00	3,114
Combined	1,85	3,042	1,35	3,405
Cluster	<i>so</i> ₅ 90N		<i>so</i> ₅ 90F	
	Mean	Std. dev.	Mean	Std. dev.
1	-3,40	1,140	-3,60	1,342
2	2,10	2,791	2,00	2,915
Combined	1,04	3,364	0,92	3,486
Cluster	<i>so</i> ₅ 225N		<i>so</i> ₅ 225F	
	Mean	Std. dev.	Mean	Std. dev.
1	-3,60	1,140	-3,60	0,894
2	2,62	2,819	2,90	2,606
Combined	1,42	3,580	1,65	3,521
Cluster	<i>so</i> ₅ 270N		<i>so</i> ₅ 270F	
	Mean	Std. dev.	Mean	Std. dev.
1	-3,20	1,095	-2,80	2,775
2	1,81	3,124	3,67	2,915
Combined	0,85	3,472	2,42	2,942

Table 5.2: Clusters for voice sound (*so*₅).

5.3 Summary and Discussion of the results

The results of our experiment can be summarized in various ways. Some evidence seems more convincing than others especially considering the relative small number of subjects that attended our experiment. We want to point out that our work can serve as a basis for future work intended to enlarge the number of subject or to investigate some other aspects related to perception of binaural sounds.

One of the salient results of our experiments is that the position and localization of sound objects is not a relevant factor in determining the overall quality, even if some evidence indicate that further experiments with larger groups of subjects could confirm off-axis sounds to be more influenced by context.

As expected and confirmed by other studies (see e.g. [52]), artificial sounds, as well as the other classified with low semantic relevance, give significantly lower results compared to music and voice.

We have focused only on single sound objects, so the task is highly simplified with respect to real conditions with competing sounds. Even with this condition, the discrimination rate is generally low.

Another result is that an being expert does not improve discrimination, probably because this is not an usual musical task. In our case, the naive subject even had better results in some scenarios. This could be explained by the small number of subjects but also with the lack of knowledge of specific phenomena related to sound propagation. As noted by the time spent on the task, expert subjects tried to apply their specific knowledge to find a possible mechanism of solution.

Chapter 6

Conclusions and Future Works

This concluding chapter summarizes the whole work; the outcomes of the developmental and evaluation stages of this research are recapitulated. Finally we present possible future improvements and additions to this research, from the development of new functions and/or applications to the setup and performing of further perceptual tests.

In this work we have developed tools and systems for both offline and real-time binaural spatialization. We have shown that GPUs are suitable also for this kind of audio computation and can solve some of the computational problems that arise when an interactive environment needs to be rendered in real-time with multiple sound sources. We have provided a system that takes into account the position of the listener via head-tracking to render a scenario compatible with augmented reality requirements. Finally we have started research in the field of psychoacoustics that can lead to a better understanding of the underlying processes of sound perception for binaural spatialized sounds.

6.1 Future Works

We want to detail here some of the possibilities opened by this research that could be addressed.

6.1.1 Improvements in the GPU implementation of a convolution engine

In this work we did not exploit all the possible benefits of a GPU implementation. Improvements such as the extensive use of coalescing memory or other architecture-specific optimizations can lead to further benefits in terms of execution time and memory use.

6.1.2 Use of different transforms beside FFT

Even if it is proved that FFT is the only transform that have a “convolution theorem” (see [67] for further details), it is worth citing one of the possibilities detailed in that paper. The authors state that a potential benefit could emerge if the data are already stored in a transformed representation. As an example, in [30] the authors show that under certain conditions the use of UDWT (Undecimated Discrete Wavelet Transform) is much more efficient than traditional convolution algorithms.

6.1.3 OpenCL implementation for radix n

One of the problem reported in Chapter 3 is the lack of support for radix-n signals. This can be achieved by extending the original procedure with the results used in the CUDA implementation.

6.1.4 Partitioned Convolution Algorithm

Partitioned convolution is not a widely known algorithm. Description of its implementation came from a paper by Armelloni, Giottoli, and Farina [5]. An earlier paper by Torger and Farina [72] also gives some insight. The first paper describes in detail the implementation of uniformly partitioned convolution on a DSP board. It is also possible to partition in a non-uniform manner (different sized partitions) for latency reduction purposes. Since this was an algorithm that aimed at splitting very long filters for real time time domain implementations it could be interesting to make a comparison with our frequency based approach. [59] also used this algorithm for an efficient FFT implementation on GPU.

6.1.5 BRIRs (Binaural Reverb Impulse Responses)

Our model can be extended to support the use of BRIRs. Adding a reverb model to an “anechoic-like” HRIR contributes to increase the perceived realism of the system. Models of BRIRs can be found in literature (see [53]). Adding a BRIR in real time can be a challenging task since they are normally longer than commonly used HRIRs (~ 100000 vs. ~ 500 samples). This problem can be solved with our implementation which helps to reduce the computational time for auralization.

6.1.6 Further perceptual tests on binaurally spatialized signals

In Chapter 5 we presented some preliminary results of a perceptual test. The experiment could be rearranged in various ways: an interesting opportunity involves fixing all other variables, to have a second group of subjects that will have the test in the other room used for binaural recordings. There could exist as a threshold of “difference” between rooms. We have purposely chosen rooms with very noticeable differences while someone could be concerned by very subtle ones even if these preliminary results

suggest that such differences will not be perceived. This consideration could induce researchers to take into account further investigation in determining threshold for perceptual discrimination between different rooms. This research goes into the direction of “Auditory Scene Analysis” ([12]). Also a salience map [36] for audio can be the output of this process as it is possible for the video cue.

Such investigations can find important counterparts in the design and planning phase of music pieces as well as in determining important steps in development of related hardware/software techniques by giving priority to some critical aspects as proposed in [8] and [27].

6.1.7 Binaural spatialization in VR applications for the blind

This idea comes from the work of Picinali & Katz (e.g. [54]) to create a wide research project linked with binaural spatialization applied to Virtual Reality applications for the blind. We show that our system can be used in augmented reality and virtual reality environments and so it will be possible to adopt it as a whole or in parts (e.g. the head-tracking system or the convolution engine) in order to determine if non-sighted individuals could show different performances in terms of spatial hearing as compared with sighted persons.

Appendix **A**

Convolution Implementations

We present here different code snippets for the different implementations of the convolution engine.

The first implementation is the reference frequency-domain implementation developed in Matlab and presented for the first time in [43].

```
function fastconvo(primofile, secondofile, nomefile)

siz1=wavread(primofile,'size');
siz2=wavread(secondofile, 'size');
chan1=siz1(2);
chan2=siz2(2);

if ((chan1>=2) & (chan2>=2))
    disp('Errore, unable to perform convolution with two stereo
        files or with multidimensional ones');
else
    [c,Sr1,bit1]=wavread(primofile);
    [d,Sr2,bit2]=wavread(secondofile);
```

```
if ((Sr1==Sr2) && (bit1==bit2))
    if ((chan1==1) & (chan2==1))
        disp('Performing convolution with two mono files');
        C=fft([c' zeros(1,length(d)-1)]);
        D=fft([d' zeros(1,length(c)-1)]);
        Y=ifft(C.*D);
        mxx=max(Y);
        YY=Y/mxx;
        wavwrite(YY,Sr1,bit1,nomefile);
    else
        if ((chan1==1) & (chan2==2))
            disp('Flipping channels');
            e=c;
            c=d;
            d=e;
        end;
        disp('Performing convolution with one mono file
            and one stereo impulse');
        cs=c';
        //Convolution
        C1=fft([cs(1,:) zeros(1,length(d)-1)]);
        C2=fft([cs(2,:) zeros(1,length(d)-1)]);
        D=fft([d' zeros(1,length(c)-1)]);
        Y1=ifft(C1.*D);
        Y2=ifft(C2.*D);
        //End
        mxx1=max(Y1);
        YY1=Y1/mxx1;
        mxx2=max(Y2);
```



```

        YY2=Y2/mxx2;
        YYt=[YY1;YY2];
        YYYt=YYt';
        wavwrite(YYYt,Sr1,bit1,nomefile);
    end;
else
    disp('Error, type mismatch');
    end;
end;

```

This implementation can be modified into a time-domain one modifying the highlighted lines between comments with these ones:

```

Y1=conv(C1,D)
Y2=conv(C2,D)

```

As a comparison we present the CUDA and OpenCL kernel functions for complex pointwise multiplication of two vectors. CUDA:

```

/////////////////////////////////////////////////////////////////
// Complex operations
/////////////////////////////////////////////////////////////////
// Complex multiplication
static __device__ __host__ inline cufftComplex
ComplexMul(cufftComplex a, cufftComplex b)
{
    cufftComplex c;
    c.x = a.x * b.x - a.y * b.y;
    c.y = a.x * b.y + a.y * b.x;
    return c;
}

```

```
}

// Complex pointwise multiplication
static __global__ void ComplexPointwiseMul
(cufftComplex* a, const cufftComplex* b, int size)
{
    const int numThreads = blockDim.x * gridDim.x;
    const int threadID = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = threadID; i < size; i += numThreads)
        a[i] = ComplexMul(a[i], b[i]);
}
```

OpenCL:

```
// OpenCL Kernel Function for element by element vector
// Complex Pointwise Multiplication
__kernel void ComplexPointwiseMul
(__global float* a, __global float* b,
 __global const float* c, __global const float* d, int size)
{
    const int numThreads = get_local_size(0)
        * get_num_groups(0);
    for (int i = get_global_id(0); i < size;
         i += numThreads){
        float k = a[i];
        a[i] = (a[i] * c[i]) - (b[i] * d[i]);
        b[i] = (k * d[i]) + (b[i] * c[i]);
    }
}
```

Appendix B

Source Code for Head-tracking external module

We provide here the source code for an OSC-capable headtracking software.

```
// OSCHT.cpp : A console application.
// This creates a head-tracker using the first WDM driver supported camera found on the system,
// and sends the tracking data with OSC to a network interface.
// Copyright: Davide Andrea 'Murivan' Mauro developed at LIMSI
// Precompiled header file
#include "stdafx.h"
// Some utilities for error handling, printing and console key-press interpreting etc.
#include "utils.h"
//liblo: Lightweight OSC implementation
#include "lo/lo.h"
// For CTRL-C handling
#include <signal>
// Here you can toggle behaviour to use either the synchronous head-pose callback
// (called when head-pose is changed)
// or an asynchronous direct-call mechanism that reads the latest head-pose value.
// Comment out this define to use the direct-call mechanism.
#define USE_HEADPOSE_CALLBACK

using namespace std;
using namespace sm::faceapi::samplecode;
lo_address t;
int webcamid=0;
// -----
// Notes on Callback Routines
// -----
//
// The following callbacks are called by internal engine worker threads.
//
```

```

// These example routines lock a mutex to protect any data structures against read-write race-conditions,
// or in the case of the receiveLogMessage() function, to serialize the calling threads to avoid
// garbled messages from being printed.
//
// In these examples the mutex is just protecting a boolean which is a POD type and atomically
// read / written by the CPU, so it is there for example purposes only. In your real application code
// you will want to copy the contents of the head-tracking data structures into your own data structures.
// These will not copy in one instruction so you will need to use a mutex to avoid weird noise from appearing
// in the data (and then you will blame it on the tracker!)
//
// The code below also illustrates that those faceAPI functions marked as "reentrant" can be called from
// within the callback routines. All other faceAPI non-reentrant functions will incur deadlocks, BEWARE!

// Callback function for messages generated by API routines
void STDCALL receiveLogMessage(void *, const char *buf, int /*buf_len*/)
{
    Lock lock(g_mutex); // serialize logging calls from different threads to avoid garbled output.
    cout << string(buf);
}

// Callback function for face-data
void STDCALL receiveFaceData(void *, smEngineFaceData face_data, smCameraVideoFrame video_frame)
{
    Lock lock(g_mutex);

    // Get info including data pointer to original image from camera
    smImageInfo video_frame_image_info;
    THROW_ON_ERROR(smImageGetInfo(video_frame.image_handle, &video_frame_image_info)); // reentrant, so ok

    // video_frame_image_info.plane_addr[*] now point to the image memory planes.
    // The memory is only valid until the end of this routine unless you call
    // smImageAddRef(video_frame.image_handle).
    // So you can deep copy the image data here, or use smImageAddRef() and just copy the pointer.
    // If you use smImageAddRef() you are responsible for calling smImageDestroy() to avoid a memory leak later.
    // In this callback you will typically want to copy the smEngineFaceData data into your own data-structure.
    // Since the smEngineFaceData contains multiple pod types copying it is not atomic and
    // a mutex is required to avoid the race-condition with any thread simultaneously
    // reading from your data-structure.
    // Such a race condition will not crash your code but will create weird noise in the tracking data.

    if (g_do_face_data_printing)
    {
        cout << video_frame << " " << face_data;

        // Save any face texture to a PNG file
        if (face_data.texture)
        {
            // Create a unique filename
            std::stringstream filename;
            filename << "face_" << video_frame.frame_num << ".png";
            // Try saving to a file
            if (saveToPNGFile(filename.str(), face_data.texture->image_info) == SM_API_OK)
            {
                cout << "Saved face-texture to " << filename.str() << std::endl;
            }
        }
    }
}

```

```

        else
        {
            cout << "Error saving face-texture to " << filename.str() << std::endl;
        }
    }
}

// Callback function for head-pose
void STDCALL receiveHeadPose(void *,smEngineHeadPoseData head_pose, smCameraVideoFrame video_frame)
{
    Lock lock(g_mutex);

    // Get info including data pointer to original image from camera
    smImageInfo video_frame_image_info;
    THROW_ON_ERROR(smImageGetInfo(video_frame.image_handle, &video_frame_image_info)); // reentrant, so ok

    // video_frame_image_info.plane_addr[*] now point to the image memory planes.
    // The memory is only valid until the end of this routine unless you call
    // smImageAddRef(video_frame.image_handle).
    // So you can deep copy the image data here, or use smImageAddRef() and just copy the pointer.
    // If you use smImageAddRef() you are responsible for calling smImageDestroy() to avoid a memory leak later.
    // In this callback you will typically want to copy the smEngineFaceData data into your own data-structure.
    // Since the smEngineFaceData contains multiple pod types copying it is not atomic and
    // a mutex is required to avoid the race-condition with any thread simultaneously
    // reading from your data-structure.
    // Such a race condition will not crash your code but will create weird noise in the tracking data.
    /*
    if (g_do_head_pose_printing)
    {
        cout << video_frame << " " << head_pose << std::endl;
    }
    */

    /* send a OSC Message */
    //if (head_pose.confidence>=0.20) to send messages only if the webcam is actually receiving some datas
    lo_send(t, "/webcam", "ifffffff", webcamid, (head_pose.head_rot.x_rads),
    (head_pose.head_rot.y_rads), (head_pose.head_rot.z_rads),
    head_pose.head_pos.x, head_pose.head_pos.y ,head_pose.head_pos.z, head_pose.confidence);
}

// Create the first available camera detected on the system, and return its handle
smCameraHandle createFirstCamera()
{
    // Detect cameras
    smCameraInfoList info_list;
    THROW_ON_ERROR(smCameraCreateInfoList(&info_list));

    if (info_list.num_cameras == 0)
    {
        throw runtime_error("No cameras were detected");
    }
    else
    {
        cout << "The followings cameras were detected: " << endl;
        for (int i=0; i<info_list.num_cameras; ++i)

```

```

    {
        char buf[1024];
        cout << "    " << i << ". Type: " << info_list.info[i].type;
        THROW_ON_ERROR(smStringWriteBuffer(info_list.info[i].model,buf,1024));
        cout << " Model: " << string(buf);
        cout << " Instance: " << info_list.info[i].instance_index << endl;
        // Print all the possible formats for the camera
        for (int j=0; j<info_list.info[i].num_formats; j++)
        {
            smCameraVideoFormat video_format = info_list.info[i].formats[j];
            cout << "        - Format: ";
            cout << " res (" << video_format.res.w << ", " << video_format.res.h << ")";
            cout << " image code " << video_format.format;
            cout << " framerate " << video_format.framerate << "(hz)";
            cout << " upside-down? " << (video_format.is_upside_down ? "y":"n") << endl;
        }
    }
}

// Create the first camera detected on the system
smCameraHandle camera_handle = 0;
THROW_ON_ERROR(smCameraCreate(&info_list.info[0], // Use first camera
                             0, // Use default settings for lens
                             &camera_handle));

// Destroy the info list
smCameraDestroyInfoList(&info_list);

return camera_handle;
}

// The main function: setup a tracking engine and show a video window, then loop on the keyboard.
void run()
{
    // Capture control-C
    signal(SIGINT, CtrlCHandler);

    // Make the console window a bit bigger (see utils.h)
    initConsole();

#ifdef _DEBUG
    // Log API debugging information to a file
    THROW_ON_ERROR(smLoggingSetFileOutputEnable(SM_API_TRUE));

    // Hook up log message callback
    THROW_ON_ERROR(smLoggingRegisterCallback(0, receiveLogMessage));
#endif

    // Get the version
    int major, minor, maint;
    THROW_ON_ERROR(smAPIVersion(&major, &minor, &maint));
    cout << endl << "API VERSION: " << major << "." << minor << "." << maint << "." << endl << endl;
    // Print detailed license info
    char *buff;
    int size;

```

```

THROW_ON_ERROR(smAPILicenseInfoString(0, &size, SM_API_TRUE));
buff = new char[size];
THROW_ON_ERROR(smAPILicenseInfoString(buff, &size, SM_API_TRUE));
cout << "LICENSE: " << buff << endl << endl;
// Determine if non-commercial restrictions apply
const bool non_commercial_license = smAPINonCommercialLicense() == SM_API_TRUE;

// Initialize the API
THROW_ON_ERROR(smAPIInit());

# ifdef _DEBUG
// Get the path to the logfile
smStringHandle logfile_path_handle = 0;
THROW_ON_ERROR(smStringCreate(&logfile_path_handle));
THROW_ON_ERROR(smLoggingGetPath(logfile_path_handle));
int buf_len = 0;
unsigned short *buf = 0;
THROW_ON_ERROR(smStringGetBufferW(logfile_path_handle, (wchar_t **)&buf, &buf_len));
wcout << "Writing log to file: " << wstring((wchar_t *)buf) << endl;
THROW_ON_ERROR(smStringDestroy(&logfile_path_handle));
# endif

// Register the WDM category of cameras
THROW_ON_ERROR(smCameraRegisterType(SM_API_CAMERA_TYPE_WDM));

smEngineHandle engine_handle = 0;
smCameraHandle camera_handle = 0;
if (non_commercial_license)
{
    // Create a new Head-Tracker engine that uses the camera
    THROW_ON_ERROR(smEngineCreate(SM_API_ENGINE_LATEST_HEAD_TRACKER, &engine_handle));
}
else
{
    // Print out a list of connected cameras, and choose the first camera on the system
    camera_handle = createFirstCamera();
    // Create a new Head-Tracker engine that uses the camera
    THROW_ON_ERROR(smEngineCreateWithCamera(SM_API_ENGINE_LATEST_HEAD_TRACKER, camera_handle, &engine_handle));
}

// Check license for particular engine version (always ok for non-commercial license)
const bool engine_licensed = smEngineIsLicensed(engine_handle) == SM_API_OK;

cout << "-----" << endl;
cout << "Press 'r' to restart tracking" << endl;
cout << "Press 'a' to toggle auto-restart mode" << endl;
if (!non_commercial_license)
{
    cout << "Press 'l' to toggle lip-tracking" << endl;
    cout << "Press 'e' to toggle eyebrow-tracking" << endl;
}
if (engine_licensed)
{
    cout << "Press 'h' to toggle printing of head-pose data" << endl;
    cout << "Press 'f' to toggle printing of face-landmark data" << endl;
}

```

```

}
cout << "Press '1' to toggle face coordinate frame axes" << endl;
cout << "Press '2' to toggle performance info" << endl;
cout << "Press '3' to toggle face mask" << endl;
cout << "Press '4' to toggle face landmarks" << endl;
cout << "Press CTRL-C or 'q' to quit" << endl;
cout << "-----" << endl;

// Hook up callbacks to receive output data from engine.
// These functions will return errors if the engine is not licensed.
if (engine_licensed)
{
#   ifdef USE_HEADPOSE_CALLBACK
    THROW_ON_ERROR(smHTRegisterHeadPoseCallback(engine_handle,0,receiveHeadPose));
#   endif
    if (!non_commercial_license)
    {
        THROW_ON_ERROR(smHTRegisterFaceDataCallback(engine_handle,0,receiveFaceData));
    }
}
else
{
    cout << "Engine is not licensed, cannot obtain any output data." << endl;
}

if (!non_commercial_license)
{
    // Enable lip and eyebrow tracking
    THROW_ON_ERROR(smHTSetLipTrackingEnabled(engine_handle,SM_API_TRUE));
    THROW_ON_ERROR(smHTSetEyebrowTrackingEnabled(engine_handle,SM_API_TRUE));
}

// Create and show a video-display window
smVideoDisplayHandle video_display_handle = 0;
THROW_ON_ERROR(smVideoDisplayCreate(engine_handle,&video_display_handle,0,TRUE));

// Setup the VideoDisplay
THROW_ON_ERROR(smVideoDisplaySetFlags(video_display_handle,g_overlay_flags));

// Get the handle to the window and change the title to "Hello World"
smWindowHandle win_handle = 0;
THROW_ON_ERROR(smVideoDisplayGetWindowHandle(video_display_handle,&win_handle));
SetWindowText(win_handle, _T("OSC Head Tracker"));

// Start tracking
THROW_ON_ERROR(smEngineStart(engine_handle));

// Loop on the keyboard
while (processKeyPress(engine_handle, video_display_handle))
{
    // Read and print the current head-pose (if not using the callback mechanism)
#   ifndef USE_HEADPOSE_CALLBACK
        if (engine_licensed)
        {
            smEngineHeadPoseData head_pose;

```



```

        THROW_ON_ERROR(smHTCurrentHeadPose(engine_handle, &head_pose));
        Lock lock(g_mutex);
        if (g_do_head_pose_printing)
        {
            std::cout << head_pose << std::endl;
        }
    }
}
#   endif

// NOTE: If you have a windows event loop in your program you
// will not need to call smAPIProcessEvents(). This manually redraws the video window.
THROW_ON_ERROR(smAPIProcessEvents());

// Prevent CPU overload in our simple loop.
const int frame_period_ms = 33;
Sleep(frame_period_ms);
}

// Destroy engine
THROW_ON_ERROR(smEngineDestroy(&engine_handle));
// Destroy video display
THROW_ON_ERROR(smVideoDisplayDestroy(&video_display_handle));
} // run()

// Application entry point
int main(int argc, char** argv)
{
    if (argc==1){
        printf("Starting with default settings id 0 localhost:7770.\n");
        t = lo_address_new(NULL, "7770");
    }
    else if (argc!=4)
    {
        printf("Wrong number of arguments. Must be 3, a WebcamID, a host and a port number.\n");
        return smAPIQuit();
    }
    if (argc!=1){
        char *address= new char[1024];
        char *port= new char[1024];
        address=argv[2];
        port=argv[3];
        t = lo_address_new(address, port);
        webcamid=strtol(argv[1], NULL, 10);
        printf("Starting with settings id %d %s:%s.\n",webcamid,address,port);
    }

    try
    {
        {
            run();
        }
        catch (exception &e)
        {
            {
                cerr << e.what() << endl;
            }
            return smAPIQuit();
        }
    }
}

```

Questionnaire for Perceptual Test and Results

Please note that since the test had been submitted to Italian speaking persons it is not translated in english.

Results are showed in Figure C.6, C.7, C.8.

Legend for rows: e.g. RP90V

- Sound type: R for Noise, T for Tone, V for Voice, M for Music.
- Envelope: P for Percussive L for Slow
- Angle: 45°, 90°, 225°, 270°.
- Distance: V for Near L for Far.

For each subject the second column also states if they are “Expert” or “Naive”. Results are computed as follows: 0 for no answer, +1 for correct answer and -1 for wrong answer then multiplied for the value expressed in ordinal scale (1...5).

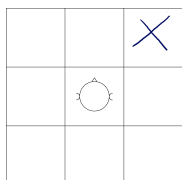
ISTRUZIONI:

Ai suoni che ascolterai sono stati applicati degli effetti di spazializzazione binaurale, una tecnica che consente all'ascoltatore di cogliere la direzione di provenienza di un suono.

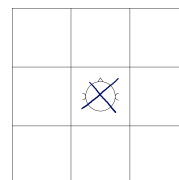
Quando si ascolta la musica in cuffia, infatti, l'ascoltatore la percepisce come se i suoni si trovassero all'interno della testa. Con la spazializzazione binaurale, invece, i suoni vengono percepiti come se si trovassero all'esterno della testa, a distanze e angoli differenti.

Ascolterai ogni suono due volte, trattato ogni volta con un effetto di spazializzazione binaurale leggermente diverso. Ti chiediamo di indicare:

- 1) quale dei due suoni è secondo te più realistico, ovvero quale dei due suoni ti dà maggiormente l'impressione che la sorgente sonora si trovi nello spazio intorno a te;
- 2) quanta differenza avverti, su una scala da 1 (pochissima/nessuna differenza) a 5 (molta differenza) tra i due suoni;
- 3) da dove senti provenire il suono rispetto alla tua testa, segnando una crocetta in uno dei quadranti come nella figura sottostante e scrivendo se senti il suono provenire da vicino o da lontano.



Es.1: suono proveniente da davanti a destra



Es. 2: suono percepito dentro la testa

Il test non è cronometrato, ma ti preghiamo di non impiegare troppo tempo a ragionare sulle risposte: quello che ci interessa è la tua impressione immediata.

Se noti qualcosa di particolare, sull'ultimo foglio del questionario è presente uno spazio per annotare le tue osservazioni.

Grazie della tua disponibilità!

Figure C.1: First page of the questionnaire.

QUESTIONARIO NUMERO

Età Sesso M F

Il tuo lavoro ha a che fare con la musica, le tecnologie audio o altri argomenti relativi al suono?
Sì No
Se sì, qual'è il tuo lavoro??

Sei musicista, suoni qualche strumento o canti?
Sì No
Se sì, che strumenti suoni?

Per quanto tempo al giorno ascolti musica?
Meno di mezz'ora Circa un'ora Circa due ore
Più di due ore

Come ascolti normalmente la musica? (è possibile selezionare più di una casella)
Impianto stereo Lettore portatile Autoradio
Computer (casse) Computer (cuffie)

Sei a conoscenza e/o hai mai utilizzato strumenti di registrazione o spazializzazione binaurale del suono?
Sì No

Figure C.2: Second page of the questionnaire.

	Quale suono è più realistico?		Avverti molta differenza tra i due suoni?						
Coppia 1	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 2	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 3	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 4	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 5	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 6	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 7	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 8	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 9	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 10	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 11	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 12	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 13	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 14	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 15	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 16	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 17	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 18	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 19	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 20	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 21	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 22	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 23	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 24	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 25	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 26	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 27	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 28	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 29	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 30	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 31	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 32	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 33	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 34	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 35	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 36	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 37	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 38	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 39	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 40	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 41	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 42	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 43	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 44	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 45	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 46	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 47	A	B	Poco/per niente	1	2	3	4	5	Molto
Coppia 48	A	B	Poco/per niente	1	2	3	4	5	Molto

Figure C.3: Third page of the questionnaire.

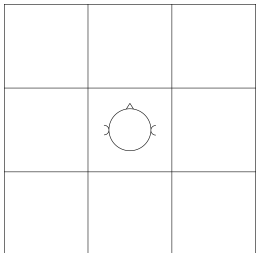
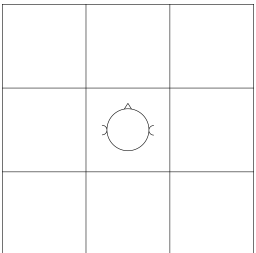
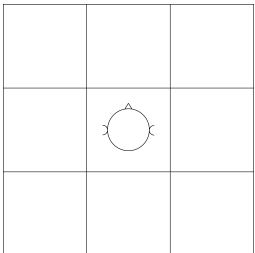
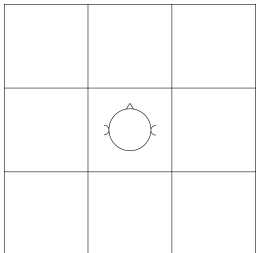
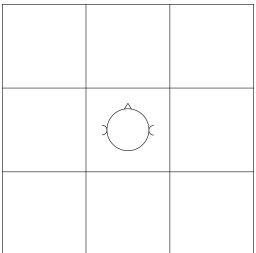
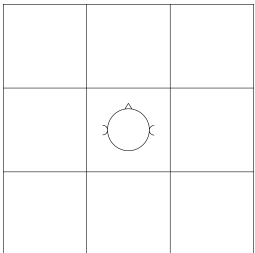
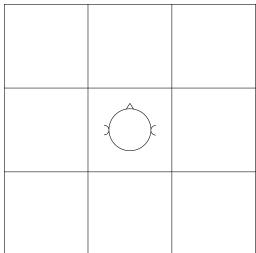
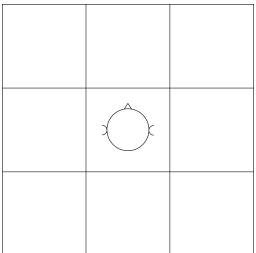
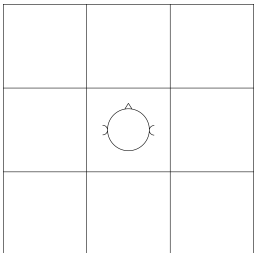
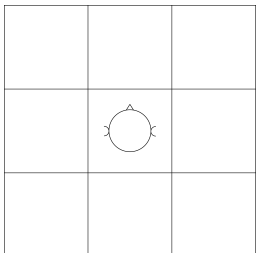
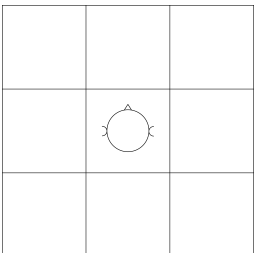
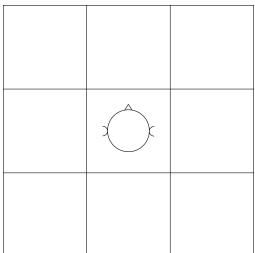
Coppia 1	Vicino	Lontano	Coppia 2	Vicino	Lontano	Coppia 3	Vicino	Lontano
								
Coppia 4	Vicino	Lontano	Coppia 5	Vicino	Lontano	Coppia 6	Vicino	Lontano
								
Coppia 7	Vicino	Lontano	Coppia 8	Vicino	Lontano	Coppia 9	Vicino	Lontano
								
Coppia 10	Vicino	Lontano	Coppia 11	Vicino	Lontano	Coppia 12	Vicino	Lontano
								

Figure C.4: Fourth page of the questionnaire.

ID	ESPERTO	RP45V	RP45L	RP90V	RP90L	RP225V	RP225L	RP270V	RP270L	RL45V	RL45L	RL90V	RL90L	RL225V	RL225L
1	TRUE	-4	-4	-4	-4	-4	-4	-3	-4	-4	-4	-4	-4	-4	4
2	TRUE	-4	0	4	4	-4	4	4	4	-3	-3	-3	-3	-1	3
3	TRUE	-4	0	-4	-4	0	-4	-4	-4	-3	-4	-4	4	-4	-4
4	TRUE	3	-2	-3	-3	-3	-3	-3	-4	-3	-4	-2	-3	-4	-4
5	FALSE	-4	1	-4	-4	-2	-5	-4	3	4	2	4	3	2	-2
6	FALSE	-3	4	-4	-4	-4	-1	-4	-4	-3	4	-4	-4	-4	4
7	TRUE	5	-3	-3	2	5	-3	-4	-3	-3	-1	-1	2	-1	-3
8	FALSE	-5	-4	5	4	5	4	-3	4	4	-4	5	5	4	4
9	TRUE	-3	-4	-4	-3	-3	-3	-3	-3	-2	-2	-2	-3	-1	2
10	TRUE	4	3	5	4	4	4	4	4	-4	-5	-4	-5	-4	-5
11	TRUE	4	2	4	3	4	4	4	4	0	-3	-3	2	5	4
12	FALSE	-3	-3	-2	-2	-3	-3	-2	-3	-3	-3	-3	-3	-3	2
13	FALSE	3	-3	4	-4	4	3	2	4	-4	-5	2	4	3	-3
14	TRUE	4	3	4	3	2	-3	3	3	3	2	4	3	-3	3
15	TRUE	4	4	0	5	4	4	5	4	-3	-3	1	-3	-3	-3
16	FALSE	-4	-3	-4	-7	4	-4	-2	-4	2	-1	-1	-2	2	-3
17	FALSE	3	2	-5	-4	3	3	5	4	4	3	5	3	-4	4
18	TRUE	5	-1	-3	-1	4	3	2	-1	-2	3	-1	-3	3	2
19	FALSE	3	3	-4	2	-3	2	4	2	-1	3	-3	-3	1	-3
20	FALSE	4	4	2	3	5	4	4	-3	5	5	-5	5	2	-3
21	FALSE	-5	-3	-4	-4	-4	-3	-5	-4	-3	-3	3	-4	-3	-3
22	FALSE	-4	-4	-4	-4	-3	-3	4	-4	4	3	-3	3	2	-4
23	TRUE	1	-3	-3	-2	2	3	2	-2	1	3	-1	-3	1	-3
24	TRUE	-5	2	5	5	-5	5	5	5	4	-5	4	-4	5	5
25	FALSE	3	3	0	3	-3	3	3	3	3	3	2	3	-2	3
26	FALSE	-2	3	3	3	-3	3	4	-4	-3	2	2	-4	-3	2

Figure C.6: Results, Page 1/3.

	RL270V	RL270L	TP45V	TP45L	TP90V	TP90L	TP225V	TP225L	TP270V	TP270L	TL45V	TL45L	TL90V	TL90L	TL225V	TL225L
5	-4	3	-4	-3	3	-4	4	4	4	4	-1	-1	1	-2	1	1
-2	-3	-4	3	4	-4	4	-3	-4	-4	-4	-2	2	-3	-3	-3	-3
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	0	-3	-3	-4	-4	-5
-2	-2	-3	-3	-3	-3	-3	-3	-3	-3	-3	3	-3	-3	-3	1	-3
4	3	3	2	-2	-2	3	2	2	2	0	-4	-4	-3	-5	3	-3
-2	-4	-5	-2	-3	-4	-5	0	-4	-5	-3	-3	-3	-4	-3	-3	-3
-2	-3	-3	2	0	4	5	5	3	3	-3	-1	-2	-4	-4	-1	-1
-5	4	5	5	4	4	3	4	4	4	5	0	3	4	3	4	4
-2	-2	-3	-4	-3	-3	-2	-3	-2	-3	-3	1	-2	-2	-2	-1	2
-4	-5	4	4	3	-4	4	4	4	4	-4	-3	4	3	-4	4	-3
2	-4	4	0	-4	3	3	3	3	2	0	-3	2	-1	1	1	3
2	-3	-2	-2	-2	2	-3	-2	-2	-3	-3	-3	-3	-3	-3	-3	2
-4	-3	3	-4	4	-3	4	4	3	-2	1	-2	5	-1	-3	-3	-3
3	-4	3	2	-4	3	3	-3	3	3	-2	-3	1	-3	1	-2	-2
-3	-3	4	4	4	4	4	4	4	4	4	3	-1	1	1	2	3
-2	-5	-4	-2	-3	-3	0	-4	-4	-4	-1	-1	-2	0	1	3	3
-5	-3	3	3	2	4	-3	3	-3	3	4	3	3	4	3	3	3
-3	5	-1	4	-5	-4	-3	-4	-4	5	2	3	-3	-2	2	-1	-1
0	-2	4	4	3	3	-3	2	2	2	3	1	2	1	-1	-3	-3
-3	-2	5	4	5	5	0	5	5	5	-4	1	-2	-1	1	3	4
-5	0	-5	-5	-4	-4	-5	-5	-4	-4	-2	-2	3	-2	4	-4	-3
-4	-3	2	3	3	-3	-3	3	3	3	-2	3	-4	-4	-4	-4	-3
1	1	-2	2	-2	-3	-2	-2	-3	-2	2	2	-1	1	1	1	-1
-5	-5	5	4	5	4	5	5	5	5	5	1	3	3	-4	-4	-1
-4	3	3	3	3	4	3	3	3	3	-3	1	1	-3	3	-2	3
-2	2	2	-3	2	4	-3	-2	2	3	1	-1	-2	-1	1	1	-2

Figure C.7: Results, Page 2/3.

	TL270V	TL270L	V45V	V45L	V90V	V90L	V225V	V225L	V270V	V270L	M45V	M45L	M90V	M90L	M225V	M225L	M270V	M270L
-1	-1	3	4	3	3	4	4	3	3	4	-4	-4	-4	-4	-4	-4	4	3
3	-4	4	4	3	4	3	4	-3	5	-3	4	-4	3	3	3	3	3	4
-4	-2	4	-4	0	-4	4	4	4	3	-4	3	0	-4	-4	5	-4	3	4
-2	-3	-3	-3	-3	-3	-3	-3	-2	-4	-3	-3	-3	-3	-3	-3	-3	-3	-3
-5	-4	-3	4	-3	-3	-4	-4	-4	-3	5	-2	2	3	5	3	1	3	3
2	-3	-4	-3	4	-3	4	-5	3	3	-2	-4	-3	-4	-4	-3	4	-3	3
-1	-2	5	5	-5	1	5	-4	-4	5	4	-5	2	5	3	5	4	3	3
4	4	5	4	5	4	5	5	5	3	5	4	5	4	5	5	5	5	5
-1	-3	3	3	3	3	3	3	3	3	-3	-2	-3	-3	-3	-3	-3	-4	-3
4	-2	-4	0	-4	-5	-5	-3	-4	-5	-5	-4	-5	-5	-4	-5	-5	-5	-5
5	2	2	3	3	3	3	3	4	5	3	-4	4	4	4	4	4	1	4
-2	-4	0	-3	-2	-2	-2	-3	-2	2	-3	-2	-3	-3	-3	-3	-3	-3	3
-3	2	3	-3	5	3	3	5	3	3	4	4	2	3	-3	4	4	3	3
-2	-3	4	2	3	4	3	3	4	3	3	3	3	4	3	3	3	3	3
-2	2	5	4	4	4	5	4	4	5	4	4	4	4	4	4	4	4	4
-4	-3	3	4	2	4	-4	3	-3	4	3	-4	2	-3	2	-4	-1	-4	-4
4	4	3	2	3	4	3	3	2	2	4	-3	-2	3	-3	2	3	-3	-3
2	2	3	4	-4	3	5	3	-2	3	-1	-5	3	3	-4	-4	-4	-3	-3
-3	2	3	2	2	2	2	4	3	3	3	-3	4	5	3	3	3	3	3
0	-1	5	5	4	-5	5	5	-5	4	-4	5	3	5	3	5	4	5	5
3	-4	-2	-5	-5	-5	-4	-5	-4	-4	-4	-5	-4	-5	-4	-4	-5	-5	-5
-4	-4	-3	3	-3	2	3	3	3	4	-3	-3	3	3	-4	4	-3	3	3
1	-1	2	-3	2	-2	-2	2	2	3	3	-2	-2	-2	-1	-2	-2	2	2
4	-2	2	5	4	5	-5	5	5	5	4	5	5	5	5	5	5	5	5
3	0	3	4	3	3	3	4	3	4	3	3	3	3	0	3	3	3	3
3	-2	5	-3	3	4	3	3	4	4	2	-3	-2	2	1	1	2	2	-2

Figure C.8: Results, Page 3/3.

Acknowledgement

This section is more than the sum of the counterparts in my B.A. and M.Sc. theses. This section represents the result of an ongoing work begun in 2006. Baron Jean Baptiste Joseph Fourier; Prof. Goffredo Haus; Dr. Lorenzo Picinali and the Fused-Media Laboratory, De Montfort University (Leicester, UK); Dr. Brian F.G. Katz and the LIMSI Laboratory, Universités UPMC et Paris-Sud 11 (Orsay, France); Dr. Ing. Luca A. Ludovico; Dr. Adriano Baratè; Dr. Antonello D’Aguanno; Other members of LIM: Dr. Maurizio Longari, Dr. Alberto Pinto, Dr. Elisa Russo, Dr. Giancarlo Vercellesi and many others. Prof. Ottavio D’Antona, above all for the quote from Edison “Genius is 1% inspiration and 99% ... perspiration” (and I want to thank him again and again for this quote; helpful in every thesis!); Prof. Alberto Bertoni; Prof. Sergio Cingolani; Ferrara University; Prof. Nicola Prodi and all the researchers from the Acoustics Laboratory; Dr. Andrea Capra and Prof. Angelo Farina from Parma University; LIM for more than six years spent at the same desk; The first floor of via Comelico 39 building that patiently put up with me and my experiments; GN Resound (especially Gianluca Vivarelli); Lorenzo Valerio and Alessio Orlandi for this amazing L^AT_EX template; Michele Voltini-Napolitano as the materials science expert; Ing. Paoletti and “consider this horse as a sphere”; Federica Mirabelli for codename OLGA; the first dummy head; Filotico’s; Vezzola/Zanini’s house; Dr. Ieri; Saló; Paolo Agnelli

a.k.a. “Paul Lambs” for pasta with tuna; “Virgola”; The Magnificent Seven of SILab; Who makes my sob go away (specially Teo); Kebabs; XCode, really... I mean version 4; The undergrads I supervised during these years...; The Sound and Music Computing Community; Random rooms in Leicester; VERY random rooms in Paris; Quarto Oggiaro!; Random friends from all over the world; Colleagues from LIMSI; Escape through arts and music; Whoever may care; “L’interrogativo ha la forma di una chiave che entra nelle porte dei dubbi piú sinceri.”

Bibliography

- [1] Acoustics — Attenuation of sound during propagation outdoors — Part 1: Calculation of the absorption of sound by the atmosphere. *ISO 9613-1*, 1993.
- [2] Seeing Machines Face Tracking API Documentation. <http://www.seeingmachines.com/product/faceapi/>, 2009.
- [3] V. R. Algazi, R. O. Duda, D. M. Thompson, and C. Avedano. The CIPIC HRTF Database. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages W2001–1/W2001–4, October 2001.
- [4] M. Altinsoy and S. Merchel. BRTF-Body related transfer functions for whole-body vibration reproduction systems. *DAGA, Rotterdam, Netherlands*, 2009.
- [5] E. Armelloni, C. Giottoli, and A. Farina. Implementation of real-time partitioned convolution on a DSP board. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 71–74. IEEE, 2003.
- [6] J. Bacher, K. Wenzig, and M. Vogler. SPSS TwoStep Clustering—A First Evaluation. In *Recent Developments and Applications in Social Research Methodology. Proceedings of the RC33 Sixth International Conference on Social Science Methodology, Amsterdam*, 2004.

-
- [7] D. Batteau. The role of the pinna in human localization. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 168(1011):158, 1967.
- [8] D. C. Begault, E. M. Wenzel, and M. R. Anderson. Direct comparison of the impact of head tracking, reverberation, and individualized head-related transfer function on the spatial perception of a virtual speech source. *Journal of Audio Engineering Society*, 49(10), October 2001.
- [9] D. R. Begault. *3-D sound for virtual reality and multimedia*. Academic Press Professional, Cambridge, MA, 1994.
- [10] J. Blauert. *Spatial Hearing: The Psychophysics of Human Sound Localization*. MIT Press, Cambridge, MA, revised edition, 1996.
- [11] J. Breebaart and C. Faller. *Spatial audio processing: Mpeg surround and other applications*, 2008.
- [12] A. Bregman. *Auditory scene analysis: The perceptual organization of sound*. The MIT Press, 1994.
- [13] T. Brookes and C. Treble. The effect of non-symmetrical left/right recording pinnae on the perceived externalisation of binaural recordings. In *Proceedings of the 118th Audio Engineering Society Convention*.
- [14] E. Calore. Optimization of the AGATA pulse shape analysis algorithm using graphics processing units. Master's thesis, Università degli Studi di Padova, 2010.
- [15] E. Castro Lopo. Libsndfile [computer software]. Retrieved December, 28:2005, 2005.

-
- [16] V. Choqueuse. Binaural Spatializer (For Max/MSP). <http://vincent.choqueuse.free.fr/>, 2007.
- [17] E. Y. Choueiri. Optimal crosstalk cancellation for binaural audio with two loudspeakers. 2011.
- [18] A. Cipriani and M. Giri. *Musica Elettronica e Sound Design*, volume 1. Con-TempoNet, 2009.
- [19] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*, 19(90):297–301, 1965.
- [20] B. Cowan and B. Kapralos. Spatial sound for video games and virtual environments utilizing real-time GPU-based convolution. In *Proceedings of the ACM FuturePlay 2008 International Conference on the Future of Game Design and Technology*, pages 166–172, Toronto, Ontario, Canada, November 3-5 2008.
- [21] B. Cowan and B. Kapralos. Real-time GPU-based convolution: a follow-up. In *Proceedings of the ACM FuturePlay @ GDC Canada 2009 International Conference on the Future of Game Design and Technology*, pages 25–26, Vancouver, British Columbia, Canada, May 12–13 2009.
- [22] P. Damaske. Head related two-channel stereophony with loudspeaker reproduction. *Journal of the Acoustical Society of America*, 50, 1971.
- [23] F. Fabritius. Audio processing algorithms on the GPU. Master’s thesis, Technical University of Denmark, 2009.
- [24] N. V. Franssen. *Some considerations of the mechanism of directional hearing*. Institute of Technology, Delft, 1960. Dissertation.

-
- [25] M. Frigo and S. Johnson. The design and implementation of FFTW3. *Proc. IEEE (Special Issue on Program Generation, Optimization, and Platform Adaptation)*, 93:216–231, 2005.
- [26] E. Gallo and N. Tsingos. Efficient 3D audio processing with the GPU. In *GP2, ACM Workshop on General Purpose Computing on Graphics Processors*, 2004.
- [27] M. Geronazzo, S. Spagnol, and F. Avanzini. Estimation and modeling of pinna-related transfer functions. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx-10)*, Graz, Austria, September 6–10 2010.
- [28] R. H. Gilkey and T. R. Anderson. Binaural and spatial hearing in real and virtual environments, 1997.
- [29] H. Gray. *Anatomy of the human body*. Lea & Febiger, 1918.
- [30] H. Guo and C. Burrus. Convolution using the undecimated discrete wavelet transform. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 3, pages 1291–1294. IEEE, 1996.
- [31] S. Handel. *Listening: An introduction to the perception of auditory events*, 1989.
- [32] W. Hartmann and A. Wittenberg. On the externalization of sound images. *Journal of the Acoustical Society of America*, 99(6):3678–3688, 1996.
- [33] E. Hornbostel and M. Wertheimer. Über die wahrnehmung der schallrichtung. *Sitzungsberichte der Preusslichen Akademie der Wissenschaften*, 20:388–396, 1920.
- [34] D. M. Howard and J. A. S. Angus. *Acoustics and psychoacoustics*, 2009.

- [35] J. Jot and O. Warusfel. Spat~: A spatial processor for musicians and sound engineers. In *CIARM: International Conference on Acoustics and Musical Research*, 1995.
- [36] C. Kayser, C. Petkov, M. Lippert, and N. Logothetis. Mechanisms for allocating auditory attention: An auditory saliency map. *Current Biology*, 15(21):1943–1947, 2005.
- [37] H. Kietz. Spatial hearing. *Acustica*, 3:73–86, 1953.
- [38] T. Koike, H. Wada, and T. Kobayashi. Modeling of the human middle ear using the finite-element method. *The Journal of the Acoustical Society of America*, 111(3):1306–1317, 2002.
- [39] P. Laws. On the problem of distance hearing and the localitazion of auditory event inside the head. *Dissertation, Technische Hochschule, Aachen*, 1972.
- [40] O. Lord Rayleigh. XII. On our perception of sound direction. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 13(74):214–232, 1907.
- [41] A. Mancuso, D. A. Mauro, and G. Vercellesi. Distance effects of the auditory event in binaural spatialization. In *DSP Application Day*, Milan, Italy, September 17 2007.
- [42] M. Matsumoto, M. Tohyama, and H. Yanagawa. A method of interpolating binaural impulse responses for moving sound images. *Acoustical Science and Technology*, 24(5):284–292, 2003.
- [43] D. A. Mauro. Effetti della distanza nella spazializzazione e localizzazione binurale. Master’s thesis, Università degli Studi di Milano, July 2006.

-
- [44] H. McGurk and J. MacDonald. Hearing lips and seeing voices. *Nature*, 264(5588):746–748, 1976.
- [45] B. C. J. Moore. *An introduction to the Psychology of Hearing*. Elsevier academic press, fifth edition, 2004.
- [46] A. Munshi et al. The OpenCL Specification. *Khronos OpenCL Working Group*, pages 11–15, 2009.
- [47] M. Noisternig, T. Musil, A. Sontacchi, and R. Hoeldrich. 3D binaural sound reproduction using a virtual Ambisonics approach. In *VECIMS - International Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, Lugano, Switzerland, 2003.
- [48] NVIDIA. Compute unified device architecture programming guide. *NVIDIA: Santa Clara, CA*, 83:129, 2007.
- [49] K. Osberg. *But what's behind door number 4? Ethics and virtual reality: A discussion*. Human Interface Technology Lab Technical Report R-97-16, 1997.
- [50] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.
- [51] S. P. Parker, G. Eberle, R. L. Martin, and K. I. McAnally. Construction of 3-D audio systems: background, research and general requirements. Technical report, Victoria: Defence Science and Technology Organisation, 2000.
- [52] B. Payri. Limitations in the recognition of sound trajectories as musical patterns. In *7th Sound and Music Computing Conference Proceedings*, pages 67–73, July 2010.

- [53] L. Picinali. *The creation of a binaural spatialization tool*. PhD thesis, De Montfort University, 2011.
- [54] L. Picinali, B. Menelas, B. Katz, and P. Bourdot. Evaluation of a Haptic/Audio System for 3-D Targeting Tasks. *Audio Engineering Society Convention 128*, 2010.
- [55] G. Plenge. On the problem of inside-the-head-locatedness. *Acustica*, 26:241–252, 1972.
- [56] M. Puckette. *The Theory and Technique of Electronic Music*. World Scientific Publishing Co. Pte. Ltd., 2007.
- [57] W. Reichardt and B. G. Haustein. On the case of the inside-the-head-locatedness effect. *Hochfrequenz-tech. U. Electroakustik*, 77:183–189, 1968.
- [58] F. Rumsey. Spatial audio (music technology) (music technology), 2001.
- [59] M. Rush. Modeling a GPU-based Convolution Engine EEC 289Q.
- [60] P. Schaeffer. *Traité des objets musicaux: essai interdisciplines*. Editions du Seuil, 1977.
- [61] R. M. Schafer. *The New Soundscape: a handbook for the modern music teacher*. BMI Canada, 1969.
- [62] W. Schirmer. On the explanation of errors in head-related stereophonic and monophonic reproduction. *Acustica*, 17:228–233, 1966.
- [63] E. A. G. Shaw and R. Teranishi. Sound pressure generated in an external-ear replica and real human ears by a nearby sound source. *Journal of Audio Engineering Society*, 44, 1968.

- [64] T. Sone, E. M, and N. Tadamoto. On the difference between localization and lateralization. In *6th International Congress on Acoustics*, pages A-3-6, Tokyo, 1968.
- [65] M. Sosnick and W. Hsu. Efficient finite difference-based sound synthesis using GPUs. In *Proceedings of the Sound and Music Computing Conference (SMC 2010)*, Barcelona, Spain, July 2010.
- [66] J. Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication*, 42(4):73-93, 1992.
- [67] H. Stone and L. Williams. On the uniqueness of the convolution theorem for the Fourier transform. *NEC Labs. Amer. Princeton, NJ. Online: <http://citeseer.ist.psu.edu/176038.html>*, 19, 2008.
- [68] R. Streicher and F. A. Everest. *New stereo soundbook*, 1998.
- [69] T. Tatli. 3D Panner: A Compositional Tool for Binaural Sound Synthesis. In *International Computer Music Conference (ICMC 2009)*, Montreal, Quebec, Canada, August 16-21 2009.
- [70] M. Tohyama, H. Suzuki, and Y. Ando. *The nature and technology of acoustic space*, 1995.
- [71] A. Torger. BruteFIR - an open-source general-purpose audio convolver. <http://www.ludd.luth.se/torger/brutefir.html>.
- [72] A. Torger and A. Farina. Real-time partitioned convolution for Ambiophonics surround sound. In *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*, pages 195-198. IEEE, 2001.

-
- [73] N. Tsingos. Perceptually-based auralization. In *19th International Congress on Acoustics*, Madrid, Spain, September 2–7 2007.
- [74] K. Watanabe, S. Takane, and Y. Suzuki. A novel interpolation method of HRTFs based on the common-acoustical-pole and zero model. *Acta acustica united with acustica*, 91(6):958–966, 2005.
- [75] S. Weinrich. Improved externalization and frontal perception of headphone signals. In *Proceedings of 92nd AES Convention, Preprint*, volume 3291, 1992.
- [76] S. Whalen. Audio and the graphics processing unit. In *IEEE Vis*, 2004.
- [77] M. Wright. Open sound control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200, 2005.
- [78] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, pages 101–104. International Computer Music Association San Francisco, 1997.
- [79] W. A. Yost. *Foundamentals of hearing: An introduction*. Academic press London, third edition, 1994.
- [80] D. Zicarelli, J. Clayton, and R. Sussman. *Writing External Objects for Max and MSP 4.3*, 2001.