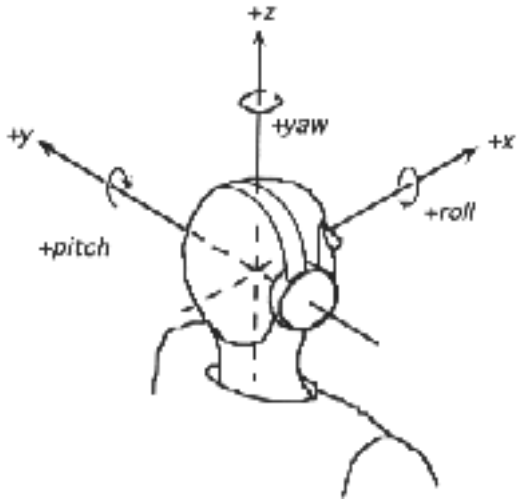
Spatial Audio for the Mobile User

IGNACIO SÁNCHEZ PARDO

KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LCN 2005-01



SPATIAL AUDIO FOR THE MOBILE USER

By

Ignacio Sánchez Pardo

Thesis Written in Partial Fulfillment
of the Requirements for the Double Degree of
Master of Science
In
Telecommunication and Electrical Engineering

Advisor and Examiner: G.Q. Maguire Jr.
Wireless@KTH
Stockholm, 2005
Final Report

ABSTRACT

Voice over the Internet Protocol (VoIP) is one of the latest and most successful Internet services. It takes advantage of Wireless Local Area Networks (WLANs) and broadband connections to provide high quality and low cost telephony over the Internet or an intranet. This project exploits features of VoIP to create a communication scenario where various conversations can be held at the same time, and each of these conversations can be located at a virtual location in space. The report includes theoretical analysis of psychoacoustic parameters and their experimental implementation together with the design of a spatial audio module for the Session Initiation Protocol (SIP) User Agent "minisip". Besides the 3D sound environment this project introduces multitasking as an integrative feature for "minisip", gathering various sound inputs connected by a SIP session to the "minisip" interface, and combining them altogether into a unique output. This later feature is achieved with the use of resampling as a core technology. The effects of traffic increment to and from the user due to the support of multiple streams are also introduced.

SAMMANFATTNING

Röst över Internet Protocol (VoIP) är en av de senaste och mest framgångsrika Internettjänsterna. Det utnyttjar Trådlösa Nätverk och bredband för att erbjuda högkvalitativ och billig telefonering över Internet eller ett Intranät. Det här projektet använder sig av VoIP för att skapa ett kommunikationsscenario där flera olika konversationer kan hållas samtidigt och där varje konversation kan placeras på en virtuell plats i rymden. Rapporten innehåller en teoretisk analys av *psykoakustiska* parametrar och deras experimentella genomförande tillsammans med design av en 3D ljud modul för Session Initiation Protocol (SIP) User Agent ”minisip”. Förutom ljudmiljön i 3D introducerar projektet *multitasking* som en integrerbar del av ”minisip”. Alla tänkbara ljudkällor baserade på SIP förbindelser samlas med ”minisip” interfacet och kombineras till en enda utsignal. Detta uppnås med hjälp av *resampling* som kärnteknologi. Effekterna av att mer trafik når användaren på grund av stödet av *multiple källor* introduceras också.

ACKNOWLEDGEMENTS

I would like to thank everybody that has contributed to this project, sharing their knowledge and devoting some of their time to help me carry out this challenging task. I would like to especially thank the following people:

- Professor Gerald Q. Maguire Jr., because from the moment we met he has motivated me to do my job better, he has been always willing to give a hand in the worse moments, and has led my project into a successful ending. Also wanted to thank him for his amazing talks, the sharing of his never ending experience, and the way he has supervised this project.

- Lalya Gaye, for her previous experience in the spatial audio field that she shared with me. This helped me to achieve my first insight into the problem.

- Professor Arne Leijon, for the conversations on binaural processing, the references he provided me, and the books I borrowed from him on spatial hearing.

- Professor Barbara Shinn-Cunningham, for the email exchange in which she helped me to decide which parameters to use in my spatial audio application.

- Participants in the Friday VoIP meetings, because through the meetings I have learnt and experienced the feeling of working in a development team.

- Johan Billien and Erik Eliasson, for having created “minisip” that has given me the opportunity to work in a developing environment, and for their time devoted to guide me through the tough world of C++ programming and compilation.

- The testers of the spatial audio system, because without them the analysis of the impact of the theoretical parameters in a real world listening test would not have been possible.

- My family, who have always been there, in the good, the bad, and the worse moments, and had always believed in me.

- My friends, the ones here, and the ones there, for their support and their never ending hours of nice moments spent together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

3G	Third Generation Cellular Phone Technology
3G+	Generation Between 3G and 4G
4G	Beyond 3G
API	Application Program Interface
AVI	Audio Video Interleaved
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
IID	Interaural Intensity Difference
ITD	Interaural Time Difference
MAA	Minimum Audible Angle
MLD	Masking Level Difference
MMS	Multimedia Messaging Service
MP3	Moving Picture Experts Group Layer-3 Audio (audio file format/extension)
MPEG	Moving Picture Experts Group (International Standards Organization)
PC	Personal Computer
PDA	Personal Digital Assistant
PHP	PHP: Hypertext Preprocessor
POTS	Plain Old Telephony System
SIP	Session Initiation Protocol
SMS	Short Message Service
SSP	Signal Speech Processing
SSP	Signal Speech Processing
VoIP	Voice over Internet Protocol
WLAN	Wireless Local Area Network

1. Introduction

The introduction of the third generation (3G) cellular telephony has created the need for new services and applications that operators can offer to their customers in order to make their 3G investments profitable. Specific applications for 3G phones exploit the fact that the 3G network can provide a much higher peak speed than the existing GSM and GPRS networks (see Table 1).

Video calling (two way video conferencing), on demand streaming (one way, down to the phone), and real time video broadcast (sports, news, etc.) are some of the newest services being offered by mobile operators. However, it is broadly thought that the real 3G revolution has not been the increase in capacity and the applications directly coupled to this increased bandwidth, but rather that the capabilities of handsets has increased in order to support these new services. The newest devices have large color screens with high resolution, they can handle a number of multimedia file formats (AVI, MP3, MPEG…), they are equipped with high quality digital cameras, they support many different types of Internet (based) formats, they have larger memories, and they are programmable by the end user. This is in addition to the existing features from GSM and GPRS, such as SMS and MMS messaging.

Table 1 – Main Parameters of the Principal Terrestrial Wireless Voice and Data Communication Standards

STANDARD	BANDWIDTH[1]	MAXIMUM LINK SPEED	POWER	RANGE [2]
802.11b (WLAN)	22 MHz	11 Mbps	100 mW	100 m
802.11g (WLAN)	22 MHz	54 Mbps	100 mW	100 m
Bluetooth	1 MHz	1 Mbps	2.5 mW ->100 mW	10 m
GSM	200 kHz	272 kbps	100 mW -> 1W	35 km
WCDMA (3G)	5 MHz	2 Mbps	100 mW -> 1W	10 km
AM RADIO [3]	10 kHz	---	~ W -> ~ kW	10s - > 100s km
FM RADIO [3]	200 kHz	~ 10 kbps (side band)	~ W -> ~ kW	10s - > 100s km

[1] This is the bandwidth of one carrier.

[2] Practical usage.

[3] Generally for broadcast radio, but some low power systems are used for local communication.

Mobile terminals no longer a limit the services that can be provided and one could think that the transition to 3G systems is going to be fast, just a stop and go before 3G+ and Wireless Local Area Network (WLAN) based technologies take the arena. The later could be a strong competitor for 3G, since 3G has not satisfied all the user's expectations, and the wireless technology is developing blindingly fast.

Table 1 shows the major difference between 3G and WLAN capacities. Today, high-end mobile terminals support multiple types of wireless connections and this enables them to utilize an even wider range of applications than 3G alone offers. Moreover, some terminals have AM and FM radio receivers. Some digital services use FM sidebands for one-way data transmission. This provides yet another source of data to be supported by handheld terminals.

The main concern when talking about WLAN access is actually the access itself. Currently, access to WLANs is limited to hotspots and private WLANs, thus service is not always available. However, once this hurdle is reduced sufficiently more complete services will be available to the user beyond simply providing Internet access. Thus if the areas where a user spends most of their time have WLAN access, then the lack of complete coverage will not be a significant barrier.

WLAN has become the link that closes the gap between GSM and POTS telephony and the Internet. The development of Voice over IP (VoIP) is threatening to change the telephony business as we now know it. Calls taking place through the Internet are cheaper, simpler, and allow a number of additional applications unavailable via the GSM system.

This project tries to exploit the characteristics of a VoIP environment over WLAN, by developing a module for the existing open-source SIP User Agent (UA) called "minisip" [5], developed at the Telecommunication Systems Lab in cooperation with researchers at the Center for Wireless Systems (Wireless@KTH) at Royal Institute of Technology (KTH), Kista, Sweden.

This service enables the exploitation of spatial audio, a psychoacoustic field that has been of great interest lately in both games and real-time applications. Spatial audio utilizes the ability of the human hearing system to locate the origin of a sound by the analysis of some of the parameters of the signal received in the ears. In the case of this project the sound is directly delivered to the user through headphones.

The module designed in this thesis utilizes resampling as another of its core features. Users today are not willing to have one single process running for them, as an example they would like to listen to their music at the same time they can hold a phone call, attend to their voice mail, or receive any other type of audio source such the broadcast radio ones mentioned above. Multitasking has become a major requirement for most of our systems, from PCs to PDAs. Additionally, mobile users are the biggest and most demanding targets for vendors. In the case of sound-based applications, a significant issue arises when looking at the different types of data: every sound application has its own preferred encoding system, uses its own choice of the most suitable sampling frequency, and delivers its data in the most appropriate time frame for each application. However, the target system may only be capable of adjusting its audio output to one unique set of parameters, thus making it impossible to reproduce all the incoming sources at the same time. By introducing resampling in this project, all the sources become independent, they can have their own parameters (which better fit their needs), and the final system will adjust and combine all these different incoming streams to suit the requirements of the underlying hardware.

1.1. Organization and contents of the report

The report is divided in four different parts, each concerning a different aspect of the project:

- PART I: Spatial Audio. This part introduces the most relevant parameters and features to be considered concerning spatialized audio. A theoretical base for the concepts and previous work are given. Some experiments analyzing the effect of the basic features for audio spatialization were conducted and the results are shown in order to contrast with the theory. The tests conducted during this phase were used to design the spatial environment for the final application.

- PART II: Programming Tools and Methods. Many tools and applications have been designed for spatial audio and resampling applications. This part of the report deals with the selection of an appropriate tool for programming the implementation; this simplifies most of the programming and provides better performance by building on well-tested ideas.

- PART III: An Example Service and its Implementation. This part describes the process of creating the desired service, from the basic program designed to receiving a call and locating it in space, to a more complex environment with multiple

simultaneous conferences. The development of the program is explained in small steps, so all the different aspects of the application can be observed. Finally, the working system was integrated into the "minisip" application, and some features were optimized for this specific environment.

- PART IV: Conclusions and Further Work. The section gives a perspective of the current situation of the implementation of spatial audio and describes some of the near term activities that could be conducted. Some implications of the use of this service on the underlying communication are given.

1.2. Unique contributions

Spatial audio has mainly been used in a passive-user manner, where the sound is played to give the user the sensation of being immersed in the environment that the sound simulates. This project and this report examine spatial audio from a different perspective: the user is now in charge of spatializing the sound. This facilitate the user multitasking, i.e. exploiting the "cocktail party" effect to allow the user to simultaneously listen to multiple independent audio streams.

Using the latest generation of PDAs with integrated WLAN allows receiving different media streams containing speech and other audio. These streams are managed by a VoIP user agent (here we use "minisip") that has been extended to allow the user to decide where to virtually place each stream in space. The user can organize these separate conversations based on their own preferences and maintain spatially distributed multiconferences just as if they were in a meeting room. This enhances the experience of 3D sound as it gives the users the possibility of interacting with their own listening experience in a simple but useful way.

Moreover, what this project introduces is a new way of understanding multitasking. Since "minisip" will be able to integrate all the incoming sound sources into an integrated output stream, the application is no longer restricted to VoIP related actions, but rather becomes the locus of different applications, each capable of establishing a SIP session. Any audio-based application with the capability of making a SIP connection will be then incorporated into the final sound output, allowing us to have many applications integrated into our multitasking environment. More on SIP sessions can be found in [9].

This report also presents the results obtained when testing different spatial audio environments based on a specific parametric design. This means that the user is not simply

exposed to sounds coming from different loudspeakers or exposed to sounds in their headphones from random positions, but rather the tests use sounds played following different patterns to simulate diverse environments and emphasizing different parameters in each experiment. This allowed the design of the final application to be based upon an experimental determination of the importance of each listening parameter based on actual user experience.

2. Background

"Free Internet Telephony that just works" is how Skype[4] introduces their VoIP client that is available for most major platforms in the market (e.g., Microsoft's Windows and Windows Mobile [38], Linux [39], and Apple's Mac Os [40]). This statement summarizes the basic underlying truth behind the success of VoIP in the last two years: it just works. Users connected to the Internet can hold a conversation using VoIP without any additional cost beyond their existing connection by using their existing computer.

VoIP confronts the old telephony system by offering a high quality voice service delivered via a simple user interface to software that can be easily installed on anyone's laptop or mobile device. Most of the VoIP services offered today are completely free. Recently VoIP companies have begun to deliver new services that allow calls to mobile and fixed telephones at lower rates than the conventional telephony network. Today, the Internet telephony phenomenon seems to be finally finding its way to the mass market, with Skype Technologies announcing one million users connected to their "network" during the last weeks of October 2004. The most significant factor for companies such as Skype Technologies is the fact that these users only exchange signaling traffic via the Skype™ overlay network, as all the voice traffic goes directly between the users' computers, thus their network infrastructure does not have to be enormous to support so many users.

"minisip" [5] an open source alternative to Skype™. Initially implemented on the HP iPAQ h5550 PDAs running Linux, this User Agent allows the user to use a WLAN as the transmission media for calls, thus converting these PDAs into WLAN phones. "minisip" is under constant development and is used in a number of research projects that are extending the capabilities and applications of the original system, e.g., with security via MIKEY and SRTP, Push to Talk [11], …

Spatial audio has been developing over many decades. Entertainment companies first introduced spatial audio early in 1939 with the development of a 3 channel sound system for the Walt Disney film "Fantasia". Perhaps the most known achievement in this field came in the 1990's when sound surround systems were installed in the cinemas [19]. The gaming industry followed this lead and created more realistic game environments by providing the user with 3D sound. Virtual reality games as well as the ones played in first person (Doom and similar) try to

[4] Skype™ is a Skype Technologies S.A. product.

give the user the most affordable sensation of immersion in the environment by means of synchronized visual and auditory information.

Unfortunately the gaming and film industries each developed their own platforms for 3D sound support. As it can be seen in Table 2, Microsoft's Windows has a wide range of development APIs to create sound applications for their operating system.

Table 2 - Microsoft's Windows Sound APIs

API	3D PROCESSING	FEATURES
Windows Multimedia	NO	Basic OS services for playing sounds
Direct Sound	NO	Simple 2D sound processing
Direct Sound 3D	YES	3D positioning, distance attenuation, Doppler effect, Head Related Transfer Functions (HRTF), reverberation, and chorus effect
OpenAL	YES	3D positioning, Doppler effect, and distance attenuation
EAX	YES	Room acoustic enhancement, requires OpenAL or Direct Sound 3D
Sensaurea	YES	HTRF and virtual surround
Aureal 3D	YES	HRTF and extended room characteristics

Spatial audio has been studied for a long time with psychophysics studies and biomedical surveys having been conducted since the 1950's. Aids for the hearing impaired, assisted learning methods, and other applications were all developed prior to the entertainment boom in the 1960's. In fact, hearing aids were one of the first applications of the transistor.

This project tries to connect both fields (VoIP and spatial audio) by means of exploiting the packet-based characteristic of the IP network and the wide variety of possible applications allowed by spatial audio. The goal is to provide the users with entertainment audio, VoIP calls, and other services while allowing them to move about and carry out various activities (specifically multitasking and conferencing). Security issues are also a major consideration while developing these new services; however, this project simply builds upon MIKEY/SRTP and other security work already undertaken in other projects.

3. Goals

By enhancing the original "minisip" to provide support for **spatialized multistream communication** we seek to enable a set of new applications. The objectives were to design, implement, and evaluate a spatial sound "distributor", that receives different streams of audio and assigns them to a virtual location in space based upon the user's pre-established preferences or specific requests. The output interface is assumed to be a **stereo headphone**.

The idea was to design a module to be integrated in "minisip". This module could be used by any application to spatialize sound. This implies for example having an audio application playing audio files in one spatial location (making use of two stereo channels) at the same time that another call occupies another location in space. Many different applications exist: multi-conference [8], push-to-talk [11], videoconferencing, and something that has not been examined in this report: the possibility of sharing the spatial audio experience beyond the limits of a single "minisip" user (i.e., more than two speaker configurations).

The other main goal of the project was to support multiple input sampling rates while having a single output sampling frequency; thus allowing an application to receive different streams with their independent characteristics and play them via hardware, which also has its own requirements. In this way the number and characteristics of the applications that could be integrated with "minisip" are not tightly restricted, but simply depend on the ability to establish a SIP connection. This opens up a new range of applications that can be developed built upon the "minisip" technology.

4. Progress of the Project

4.1. Requirements

Some of the main requirements of the project are related to the environment in which the spatial audio is to be implemented. Since the application has to be designed as a module for "minisip", this required the software to be compatible and preferably developed in the same programming language. This means that the programming language should be C++, which gives the possibility of developing the system both for Linux (the OS used in the lab for design) and Windows (the OS installed in the iPAQ handheld devices to be used for user testing). The portability of the application is important, since one of the requirements was to develop a user-friendly application that can compete with Skype™ and other VoIP clients while providing a greater number of features.

The final implementation of the spatial audio application has to be deployed on and tested in a HP iPAQ h5550 PDA. The test bed is a group of students from one of the KTH Masters Programs. These students are going to be the first users of the application and they will provide feedback about the performance and their personal experiences with this application. However this testing is not part of this thesis.

Since the final implementation of "minisip" should be widely used all the components used in the design of the system must be either licensed or license free. In this case, for simplicity, this requirement leads to the use of open source programs as much as possible. Unfortunately, few open source sound APIs compatible with Windows and Linux have been developed, so finding the appropriate tool was also essential.

It is also crucial that if new libraries have to be installed and added to the project, that they don't interfere with the existing ones; additionally they should provide a basis for new applications (i.e., ideally the library will not only be used for this project, but should also be usable by other applications). This later requirement is due to the limited resources of the PDA.

Since interactive audio applications (such as telephony) require low latency, the system must work in real-time. This implies that the signal processing cannot be too complex and time consuming, as communication delays lead to user annoyance.

Together with all the above requirements, a final requirement was writing appropriate documentation to support further development of the deployed application. The documentation

should allow the reader to understand what has been done and why, and also give him or her the insight necessary to add additional features to the resulting module.

4.2. Accomplishments

Thus far the project has accomplished the following:

- All the relevant parameters (from now on referred to as *cues,* since this is the formal term used to define the relevant acoustic features of a signal that provide information about different aspects such as location, intensity, frequency components, etc.) in spatial sound have been studied and their relevance in source localization analyzed by means of a web-based test open to public access[5]. The test was conducted in order to determine which type of spatial source distribution gave the user better resolution of the spatialized environment. The analysis of the results leads to the design of a virtual environment for the spatial audio module that emphasizes the most important cues and distributes the sources based on user-experience. Theoretical analysis, test background, and results as well as their implications are included later in this report.

- A number of programming tools have been examined, their behavior analyzed, and their features studied relative to how they could be useful for the project. The study has been conducted in two different areas: spatial audio APIs and resampling APIs. The market for spatial audio products is quite broad, but most of the solutions are proprietary. The most suitable API for this thesis is the OpenAL library together with its related wrapper for C++ that is called OpenAL++. Both the library and the wrapper have been tested and considered for the application, but were finally discarded (the reasons for this decision are explained later in this report). In the other hand, not many resampling tools could be found, but their performance and ease of use helped me a lot. “libsamplerate” was the library finally selected to be integrated in the spatial audio application. It provides a set of functions that make it possible to work with streaming sound, and also gives the possibility of time varying conversions, which may be useful in a changing environment such as the wireless environment.

[5] The test and some results can be found at http://usuarios.lycos.es/iggyto/

- A spatial audio module for "minisip" has been implemented. The module gives support for five simultaneous streams located in different positions, and it is based on a resampling structure that provides the necessary high sampling rate input to perform the spatializing operations.

4.3. Limitations

Some of the considerations referred to as requirements could also be viewed as limitations of the project. The main limitation comes due to the issue of licensed vs. license free software and the specific target OSs. These two factors significantly reduced the number of design options to be considered; although at the same time they helped define a specific and more limited programming environment that helped to focus the effort.

The most frequently cited limitation in any project is always time. The limited time available for the project discouraged the analysis of the effects of spectral cues in spatial audio, since the establishment of the models and their implementation is a project itself to which many Masters theses and investigations have already been made. Only limited theoretical results have been used in this experiment, although it provided sufficient knowledge to analyze the relevance of parameters and how to use the information that spectral analysis makes available.

The mentioned test for Master Program students has not been undertaken because at the end of this project there was not official stable and complete release of "minisip" to be installed in their PDAs. The progress of the whole "minisip" development is something to have in consideration when looking at individual goals, since most of the projects are dependent on others.

Last but not least, my limited C++ programming experience and the limited time devoted to acquire the minimum required skills represented an additional hurdle to achieving the aims of this project. Some very useful resources for those who might find themselves in the same programming situation can be found in [37] (documentation in Spanish).

5. Methodology

Each of the phases of the project requires an specific methodology depending on whether the related work is more theoretically or practically oriented.

In the case of the spatial audio study both aspects were taken into consideration. The process consisted of a bibliographic investigation of the field, followed by the reading and comprehension of numerous related articles, papers, books, …. The reading provided the necessary background regarding the *cues* involved in spatializing sound required to develop my own spatial audio test in order to check the theoretical results and to experiment with the effects of the variation of acoustic *cues*.

The test was web-based, using PHP and HTML technologies to develop an interactive client-server interface where the users can test their own spatial skills and at the same time provide the project with crucial results regarding the perceived spatializing process. All the sound sources used were previously processed with MATLAB using Speech Signal Processing (SSP) techniques in order to recreate a specific spatialized environment. More information about the testing process follows in Part I of this report.

For the tool analysis and selection, an initial thorough search of available tools and components was made. Then, with all the options in hand, the features of each alternative were compared, and only those whose interest for the project was easily demonstrated were considered further. The use of these programs together with an analysis of their theoretical and programming basis was required since the results obtained from one product might be appropriate, but the underlying technologies might not suit the requirements of the project. The final selection was made based on simplicity and performance criteria. For this reason no library was used for the spatial audio part, but the specific necessary code was written instead.

The implementation has been divided in two phases: an approach in C to understand the resampling and spatializing routines, and the final integration of C++ code into “minisip”. The first phase consisted of a step-by-step implementation of the spatializing procedure in C, starting by reading sound samples in blocks, and ending with a spatialized distribution of resampled audio. Once the correct behavior was confirmed, the code was integrated into “minisip” by introducing a new class and modifying two of the existing classes.

PART I: SPATIAL AUDIO

A theoretical overview of

spatially located sound

6. Acoustics and Psychophysics

Sound is a pressure wave, and in the case of human speech the vibration of the vocal cords produces the wave. A sound wave travels through the air at approximately 340 m/s depending on temperature and humidity. The most important physical properties of a sound waveform are frequency, intensity, and complexity. These physical properties are directly related to their perceptual analogues: pitch, loudness, and timbre respectively.

Frequency is probably the most important factor in signal speech processing [7]. Human ears are sensitive to sounds with energy in the range of frequencies from 20 Hz to 22 kHz. Humans are most sensitive to frequencies near 2 kHz and less sensitive in upper and lower ranges. From frequency analysis of speech the majority of the relevant features can be extracted and then analyzed to understand the underlying behavior of the human speech production system. If one understands this system, then the use of synthesized parameters allows simulating human speech and modifying it to an extent which human beings are unable to reach by means of conventional speaking.

But acoustics does not simply entail analysis of the physical sound waveform. The listener has an enormous influence on how the emitted sound is perceived. Since every human being is different and their way of thinking is influenced by so many diverse factors, a universal generalization of the speech model is impossible, and so when simulating speech the goal is to achieve a general result that satisfies the majority of the users. It is also very application-dependent, so different features of speech and sound are emphasized depending on the application, the user, and their environment.

Another relevant acoustic factor to consider regarding spatial audio is masking effects. When a listener is presented with various speech signals at the same time, these signals can interact in various ways. When a signal that is perfectly perceived in isolation is inaudible in presence of the other signals a phenomenon called masking occurs [32][13]. Noise masking effects are the most common and undesired effects.

7. Binaural Processing

All human hearing processes that are made affordable or are enhanced by the use of two ears rather than one are referred as *binaural* processing. These binaural functions support the human ability to localize sound sources in three dimensions, identify speech in noisy environments, performing loudness estimations, and headphone-based tasks (the later are the ones of specific interest in this project). The term *interaural* is sometimes used instead of binaural.

The parameters and important features related to binaural processing are called binaural cues. The term cue can be defined as "a component of an incoming signal that can be used in the identification and analysis of a perceptual feature"[6].

7.1. Binaural Cues

The most significant cues for source location depend on the differences appreciated in the signals arriving to the left and right ears. The simplicity of binaural cues resides in the fact that their analysis is done by comparing the signals arriving at each ear, then extracting from this comparison the attributes that arise from source position. There are two central binaural cues: Interaural Time Differences (ITDs) and Interaural Intensity Differences (IIDs). These two cues provide only left and right separation of sound.

Depending on the angle formed by the medial plane of the body and a sound source location, one ear might receive the sound earlier than the other. This time difference (ITD) is considered to be useful up to a frequency where the wavelength is approximately twice the distance between the two ears. Beyond that frequency no difference is appreciated in the sounds arriving to the left and the right ear. Some important features of ITDs are:

- they vary with both azimuth and elevation (see figure 1).
- they grow with the angle of the source to the medial plane with a range of values from 0 to 600-800μs (see figure 2).
- humans are able to detect ITDs from 10-50 μs depending on the listener. This corresponds to a difference in angle of 1-5 degrees. But the sensitivity to changes in ITD varies depending on the location of the source. As ITD increases (i.e., the source moves away from the medial plane) sensitivity deteriorates.

[6] Source: http://www.ling.mq.edu.au/units/sph307/terminology/index.html

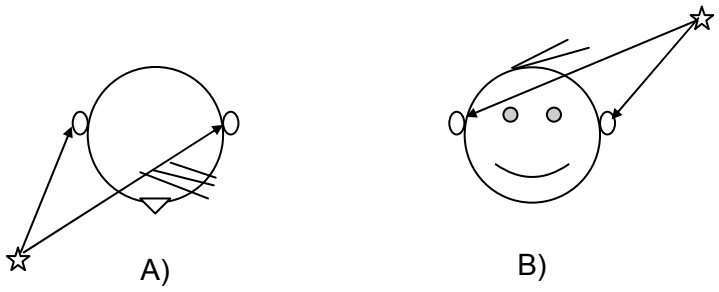


Figure 1 – A) Dependence of the ITD on azimuth; B) Dependence of the ITD on elevation

IIDs are based on the fact that the sound reaching the closest ear is louder that the one reaching the furthest ear. The intensity of the sound drops with the square of the distance (formula 1). However, when taking into account the absorption of sound in the air, it has to be noted that higher frequencies decay faster (i.e., $I \propto 1/d^3$).

$$I = \frac{W}{4\pi d^2} \qquad (1)$$

where W is the sound power [W],

d is the distance to source-ear [m].

The difference in relative intensity of sound (IID) arriving to the two ears varies with the location of the sound; it increases with frequency and the angle between the source and the medial plane.

A third cue called Interaural Loudness Difference (ILD) appears when the source is within reach of the listener. ILDs are extra large IIDs at all frequencies. These ILDs help to express information about the relative distance and direction of the source from the listener [23].

To determine the exact location of a sound, when ITDs and IIDs provide ambiguous information, Head Related Transfer Functions (HRTFs) are used. HTRFs will be explained in section 8.2.

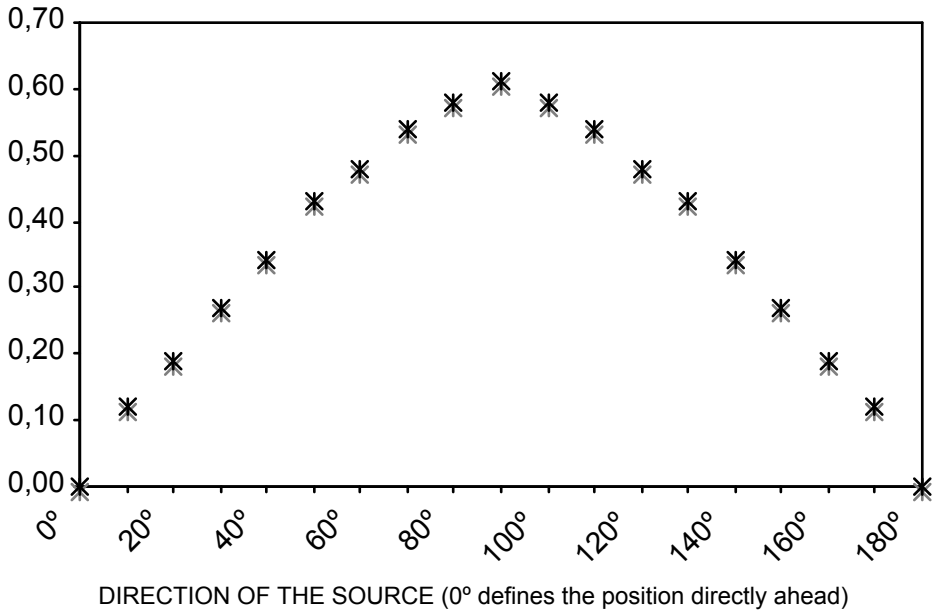


Figure 2 - Computed values of ITD depending on source's azimuth being 0º directly ahead. The curve is an interpolation of theoretically calculated values.

7.1.1. Head-shadow Effect

Both ITDs and IIDs are affected by the head shadow effect. This effect is caused by the reflection and diffraction of signals by the head, causing less energy to arrive to the far side of the head. The head acts as a screen on high frequencies that have small wavelengths compared to the size of the head ($\lambda << r$) as shown in figure 3a. Low frequencies ($\lambda >> r$) are simply diffracted (figure 3b).

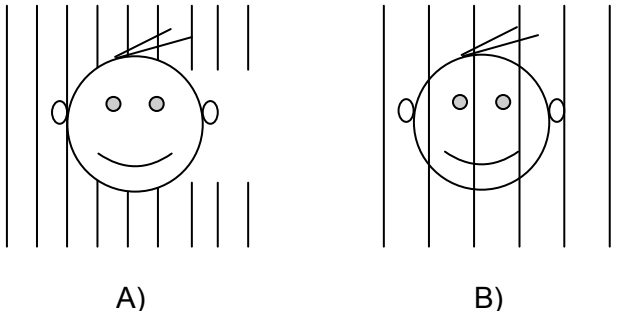


Figure 3 – A) High frequencies are shadowed; B) Low frequencies are diffracted

7.2. Lateralization and Localization

Stereo sound presented through headphones gives the impression of coming from inside the head and has a definite spatial definition. The sound is distributed in the virtual space defined by the line that goes from one ear to the other. Perception of sounds within the head is defined as *internalization* of sound and their location along the imaginary line between the ears is called *lateralization [16]*. In contrast, when a sound is presented to the listener using loudspeakers the sound is considered to be *externalized*, and it is located in a process called *localization*.

In this project a mixture between lateralization and localization is used in the test part. Since the listener generally has experience about this environment and is surrounded by different sounds that can be located by visual confirmation, the process followed to point out source locations when receiving sound through headphones is based on an "externalization of the internalized sounds". This is, the listener receives the sound through the headphones, and internalizes it in one of the positions in the virtual line between the ears. Then, using comparison with location-known sounds and other a priori knowledge, the listener assigns the internal sound to an external location in the range of positions in front of him.

Since the internalized sound has only right and left information the listener could as well define the symmetrical position behind them and it would provide the same information. To obtain a real location based in the exact simulation of the sound perceived by the listener in a free field environment the use of HTRFs is needed.

7.2.1. Horizontal Plane

Localization of sounds in the horizontal plane is based on IID and ITD analysis. A source directly in front of the listener produces almost the same waveforms in both ears (with the same IID and ITD), but when the source moves away from the midline the sound will arrive sooner and be louder in the closer ear than in the far one.

One of the concepts that must be taken into account when discussing localization is *localization blur*. This term refers to the difference existing between the auditory space and the real space where the sound sources exist. A point source produces a sound that is spread in space thus producing a blur in the identification process. In [1], localization blur is defined as "*the amount of displacement of the position of the sound source that is recognized by 50% of experimental subjects as a change in the position of the auditory event"*.

For spatial hearing analysis in the horizontal plane the minimum localization blur occurs in the forward direction while it increases when moving left or right. The maximum is found in the direction orthogonal to the one the listener is facing, this is 90º from the medial plane. Behind the user the blur decreases, but it has higher values than in the front (see figure 4).

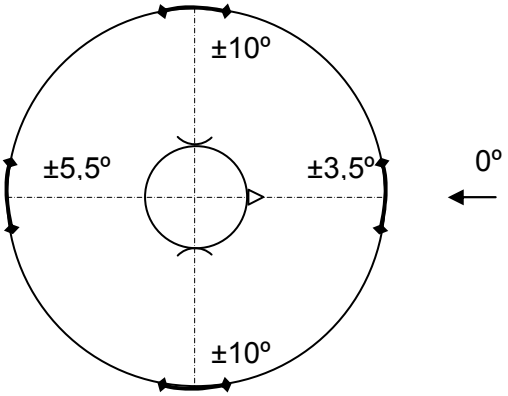


Figure 4 – Localization Blur in the Horizontal Plane (after [1])

As stated in [6], ITD and IID cues provide the main contribution to horizontal-plane localization. ITD is a crucial cue for localization of low frequencies while it is almost useless for high frequency ones. The IID frequency-dependent head shadow effect makes it useful in high frequencies, but is useless with low frequency stimulus.

Since speech is broadband (200 Hz - 8 kHz), the various frequency components of a speech signal in free field conditions are differently affected by ITDs and IIDs, and therefore differently perceived by the listener. During the experimental phase of this report broadband signals have been used in order to obtain a better understanding of the effect of *binaural* cues, although the real bandwidth of speech signals in telephony differs from free field conditions.

Traditional telephone speech signals have a frequency range from 250 Hz to 3.5 kHz, which ensures intelligible communication, but reduces the effects of the IIDs. The wide band suppression in telephony does not cause a big problem for the listener, since he/she actually hears the overtone frequencies of the voice in the range of 250-3500 Hz. The human brain extrapolates the information lost in the upper frequency band, enabling us to understand conversations and to identify speakers.

7.2.2. Vertical Plane

Spatial localization is not as good in the vertical plane as it is in the horizontal plane. As was mentioned before the theoretical minimum audible angle (MAA) in the horizontal plane is about 1º of arc, although the results in figure 4 show that the minimum is actually about 3.5º of arc in the best situation. In the case of the vertical plane this MAA is around 9º of arc [1].

In contrast with horizontal-plane localization, vertical plane localization is not based on ITD nor IID cues, but on pinna[7]-based spectral cues, reflection from the torso, and the interaural pinna disparity cue.

Similar results as the ones shown for the horizontal plane are shown for the vertical plane in figure 5. The main differences observed between the two planes are the higher blur occurring in the vertical plane, and also the big differences in front and back localization accuracy. As we see the vertical plane location in front of the user is much more accurate than behind the user. This is the effect of the already mentioned pinna-based spectral cues. The appearance of these cues explains the main importance of the shape of the ear; since the incoming path of the ear is orientated toward the front, better results are obtained compared to the sound waves that come from the back and must past around the ear to reach the auditory channel.

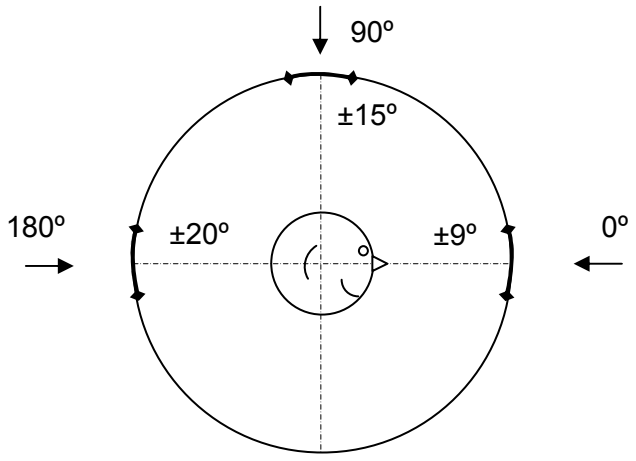


Figure 5 – Localization Blur in the Vertical Plane (after [1])

[7] Pinna is the visible outer part of the ear in charge of sound collection.

7.3. Front-back and up-down confusions

Positioning of sound sources is altered by two main factors. The first is the just mentioned localization blur effect. The other, observed in almost every localization study, is the front-back and up-down confusion. The first refers to forward sources received in the rear hemisphere and the second to sources located above the horizontal plane of sound detection and that are located beyond it.

These confusions are the result of ambiguities caused by the spherical shape of the head and the role of ITDs and IIDs in localization. For example, since ITDs from front and back locations are symmetric, they result in the same perception is we base our analysis only on ITDs. A given interaural difference produces a range of possible sound source locations describing a cone. This phenomenon has been called the "*confusion cone*".

7.4. Reverberation

The acoustic energy that arrives to the listener through indirect paths is referred to as reverberation. Reverberation affects localization in two ways: it degrades the perception of source direction and it enhances the perception of sound distance. Reverberation also provides information about the environment in which the listening experience occurs. It gives for example information about the size and spatial distribution of elements in a room.

Although adding reverberation to a sound simulation provides information about the relative distance it can also decrease directional perception accuracy, interfere with the speech reception, and degrade the ability to attend to more than one source. Thus, reverberation was not considered further for the testing environment designed for this project.

7.5. Benefits of Binaural Hearing

As discussed before the major benefit extracted from binaural hearing is the ability to determine the location of sound sources. But this is not the only advantage we encounter; binaural hearing is of great aid when selectively attending to sources coming from different locations. As explained in [2], this is of great importance when a group of sources are competing in the same environment. Rather than just separating sources, spatial information can be used to disregard signals coming from a direction different from the direction of interest.

A specific benefit from binaural hearing applied to headphone listening is the increase in

the Masking Level Difference (MLD). MLD can be defined as the difference in intensity for the detection of a signal when ITDs and IIDs are present compared to when these cues are not considered.

7.6. Supernormal Auditory Localization

A way of obtaining better results when testing spatial localization is to exaggerate the design parameters. This affects for example the head and ear size cues that are given to the listener. The aim is to provide the user with a better resolution than in the real world. This synthetic sound can be of great help when managing a large number of sources since a greater difference can be appreciated compared to the real world, where a complex environment leads to confusion and difficulty in achieving spatial localization.

Since the purpose of the tests performed in this project was to understand the binaural cues and their effects in spatial audio in a real environment, supernormal localization has not been used, thus putting the user in the most real environment.

8. Spatial Simulation

Spatial simulation can be done by using either headphones or loudspeakers. Headphones allow better control of the interaural cues since the designer has complete control of the two independent signals arriving to the ears.

8.1. Headphone Simulation

The simplest way of simulating sound through headphones is delivering the same signal to both ears (diotic displays). This kind of experiment provides no spatial information; the sources are perceived internally in the midline between the two ears and cannot be externalized.

Dichotic displays make use of ITDs and IIDs to provide spatial information. The result are signals that can be internally located in the imaginary line inside the head that connects both ears, but that can then be externalized by the user as explained in section 7.2. This is not a natural process though. The act of varying ITDs and IIDs causes the sounds to move from right to left inside the listeners' heads, and using the combination of both information sources he/she creates an external image of the sound. This is the kind of localization process used in this project. If a more realistic sound is needed then signal speech processing techniques can be used to provide most of the spatial cues available in the real world. The most widely used technique is Head-Related Transfer Functions.

8.2. Head-Related Transfer Functions

The most effective way of recreating spatial audio in the listeners' ears is to reproduce the exact waveform that would arrive to them from a source in the desired location. This is done by measuring the transfer functions that show how the waveform is affected from when it is produced until it arrives at the ears of the listener. Then, for every position that is simulated, the transfer function that has been previously calculated is used to filter the known source signal.

The filters that define this transformation from the source to the listener are called Head-Related Transfer Functions. They give information on how to simulate directions, but do not include for example reverberation parameters.

Although theoretically HRTFs provide signals that are completely equivalent to the ones provided by natural hearing, there are some limitations to HRTF processing that explain some of the reasons why they have not been used in this project. HRTF processing is a difficult, time

consuming process, and it requires huge amounts of storage space. Apart from that, HRTF are typically calculated only for a few locations and then interpolated and scaled to obtain the whole set of desired positions. Moreover, HRTFs are individual parameters, because the individual differences are very important in source localization; hence the use of non-individualized HRTFs reduces the accuracy and externalization of auditory images.

Together with the theoretical study, during October 2004 I maintained email correspondence with Professor Barbara G. Shinn-Cunningham, from Boston University. She is an expertise in the spatial audio field, and specially in HTRF processing. Her suggestions finally discouraged me from using HRTFs in my project.

" […] I think that there are many, many benefits that you can get from simply adding ITD and IID (interaural time and intensity difference, respectively) cues to the signals. The extra benefit you get from HRTFs is most important when there is high-frequency information, above 5 kHz. For the kind of application you are investigating, using HRTFs may not provide much benefit. Moreover, the benefit of using HRTFs (rather than simply using ITD and IID cues) depends on using individualized HRTFs for each listener. With "generic" HRTFs and a low-pass stimulus like you are probably working with, the HRTFs may not give much benefit over simple interaural difference cues. […] "

B.G. Shinn-Cunningham

15th October 2004

9. Test

During October 2004 a test bed was created in order to observe the phenomena related to binaural processing, in particular the effects of ITD and IID changes in the perception of sound. All the experiments entail data only in the horizontal plane. The goal of these tests was to determine which of the parameters was more relevant in order to design the appropriate virtual environment for the final application.

The tests were web-based in order to make them available to as many people as possible. The users had to enter their names and the final results were stored so a personal record of the results is made.

The different cue-based virtual environments designed for the test are shown in figure 6. The first two grids display the same number of different source positions so a fair comparison can be made. The "Six Position Grid" (SPG) enhances the IID parameter while the "Long Table Grid" (LTG) gives more importance to ITD.

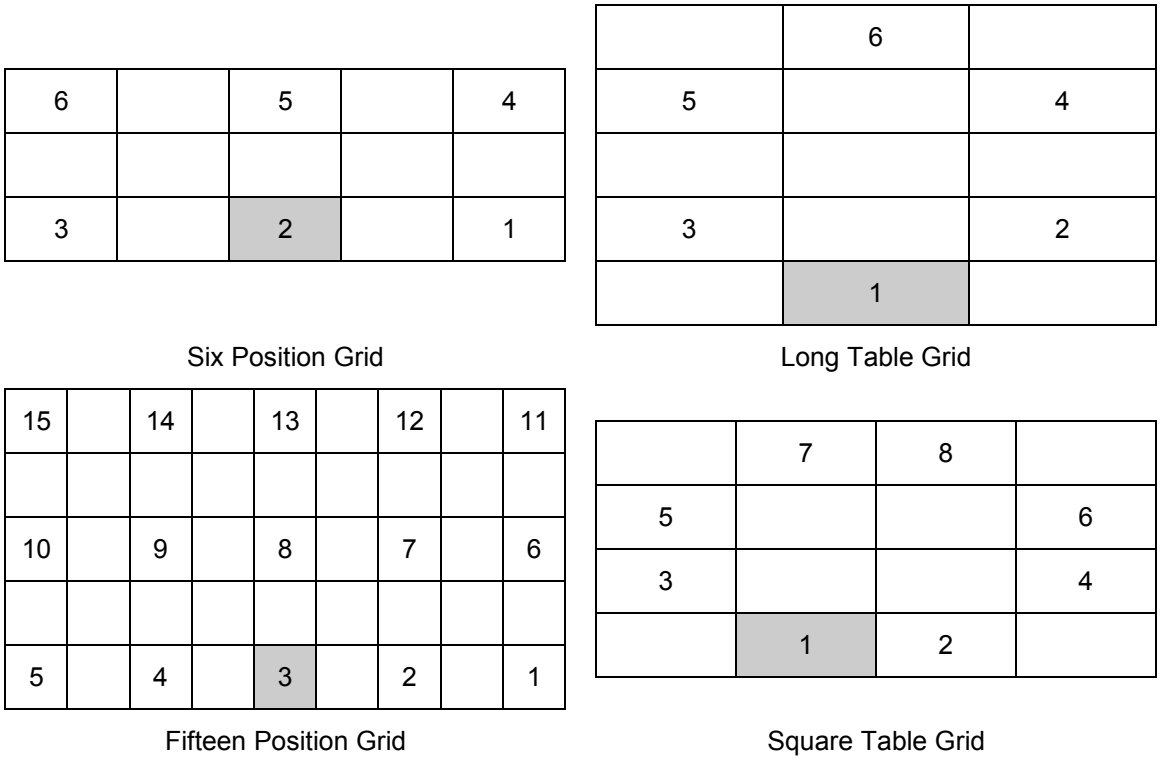


Figure 6 – Different Test Environments; shadowed position denotes the listener

The “Fifteen Position Grid” (FPG) is an experiment in order to determine the ability of the human hearing system to distinguish among very close positions. This test was done to crosscheck the theoretical results on the Minimum Audible Angle (MMA) shown in figure 4.

To conclude, the “Square Table Grid” (STG) mixes both the ITD and IID cues and provides an environment where positions in the same side of the virtual table provide very similar cues. This test provides results about the acuity of small changes in binaural cues.

9.1. Method

9.1.1. Subjects

The tests were public so any one that had access to the web page could experiment with their own spatial audio experience and provide useful data. None of the users had previous experience in psychoacoustic experiments. In order to provide accurate results only the subjects that completed the tests more than three times where considered. Specific subjects were asked to cooperate in order to obtain, apart from a general view, specific individual-based results. All the subjects were adults of ages 18-50.

9.1.2. Stimuli

The subjects were presented with a WAVE format stereo speech audio sample. The sound that was presented to the listener was a sample sentence spoken in English. The original audio file was a monaural (mono) signal, so in order to be converted to stereo and have its ITDs and IIDs modified, MATLAB’s speech signal processing was used. The process followed to create the spatialized stimuli is shown in figure 7.

In this example the sound is simulated to be coming from the left. The mono file is split into two channels to enable control of both the ITDs and IIDs. As the sound comes from the left, the sound received in the right channel will come later and with less energy than the one on the left. To simulate this, the samples from the right ear are shifted and the difference in level is achieved by scaling the samples.

The scaling is done following acoustics; this is, directly scaling by the distance to the source. Since the designed environments have virtual locations expressed in meters, the division is directly done using these values.

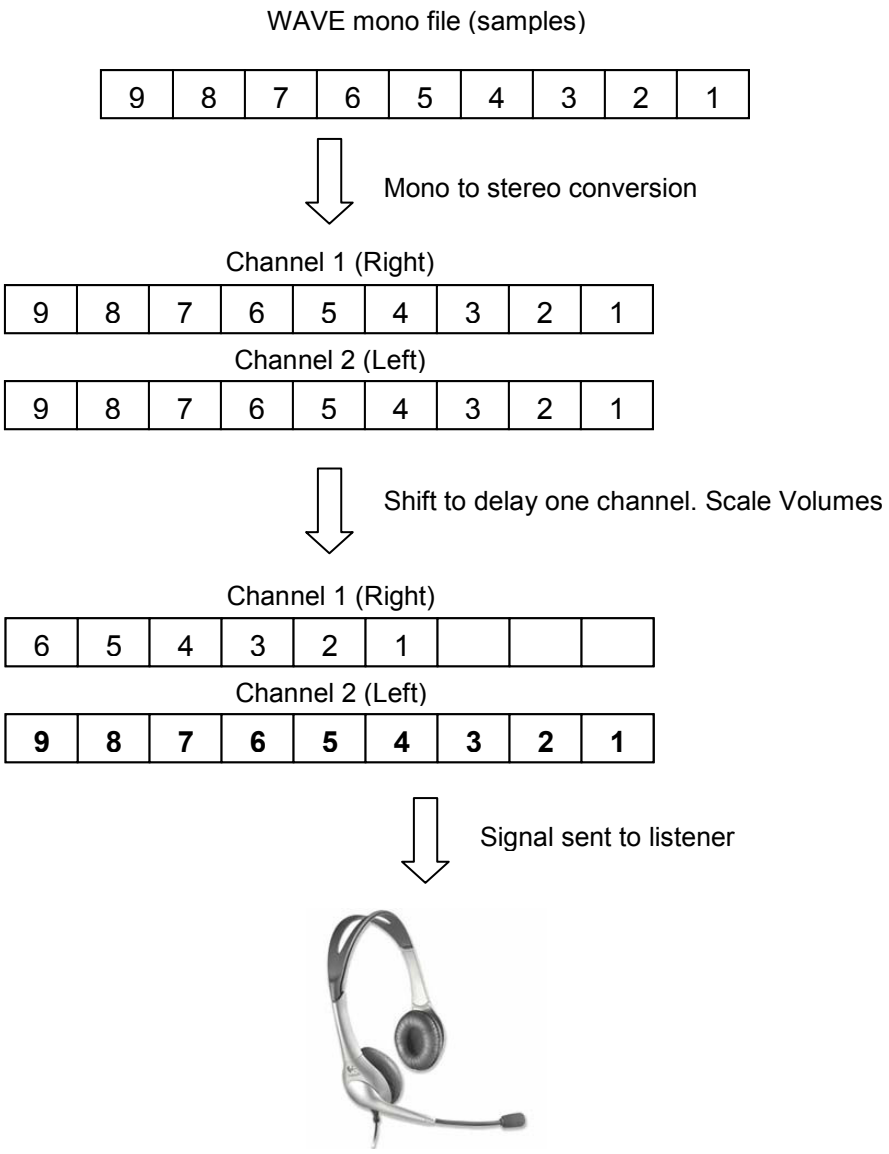


Figure 7 – Stimuli creation process

To obtain the delay values in samples the process is: first, the distances to both right and left ear are computed. Then, the difference in propagation time is obtained (dividing each distance by 340 m/s and then subtracting the values). With the difference in time, the last step is to compute the difference in units of sample time; this number of samples will be then used to shift one of the channels in order to recreate the ITDs. The number of samples to shift by is determined by multiplying the difference in time by the sampling frequency. This introduces one of the main

concerns of the project: resampling.

The first sound signals I used for the test were signals sampled with an 8 kHz sampling rate. All the steps of the spatializing process were followed, but when the sound was delivered to the headphones no significant differences among sounds were observed. This was due to the sampling rate used. If the sampling rate is not high enough, then the temporal resolution of the samples does not provide enough information to spatialize sound. The subsequent stimuli creation was all done with a 44.1 kHz sampling rate, and in this case much better results were achieved. This result means that we need to have high sampling rate signals arriving to the headphones.

Since the CODECs typically used with "minisip" uses an 8 kHz sampling rate, the resampling of these signals is of major importance. Moreover, the actual implementation of "minisip" for the HP iPAQ h5550 already requires the input signals to be sampled at 16 kHz. "minisip" already implements a simple resampling routine to upsample the incoming samples to 16 kHz. The use of a general method that can resample any incoming sampled signal to the desired output sampling rate was used in this project. Alternatives to the resampling process are left for Part II of this report where the tools for the application development are described.

9.1.3. Test Procedure

The test procedure is analogous for all the four grids, so only one case will be presented in detail and the other ones can be inferred from this. For each grid, the user can first practice by listening to learn the possible sound locations. When the user feels that has an understanding of the different spatial positions, the test starts.

During the test, the user is presented with five different and random positions from the grid that is being tested. Each sound can be listened to as many times as required by the user, but must be followed by a decision. The user is told whether their decision was right or wrong. When the five trials finish the user is given a final result and this result is stored.

Via the different grids the user can experiment with different parameters, and although the user is not aware of the exact changes in the signals, their reactions will show how they are interpreting the binaural cues.

9.2. Results

All the results from the test were stored internally in the same server where the test files were hosted. The raw file contained useful data together with other results that could not been considered further. These extraneous results were generated for example when users navigated back and forward in the test using the navigator back and forward functions instead of the appropriate functions within the webpage. The final results after the extraction of extraneous results and averaging are shown in Table 3.

Table 3 – Results of the Test in Number of Correct Choices (1-5)

GRID 1 (SPG)		
1.	Ine	4.60
2.	L.Lo	4.25
3.	Ann	4.18
4.	Ing	4.00
5.	Mar	4.00
6.	Flo	4.00
7.	Car	3.67
8.	Ale	3.67
9.	Chi	3.50
10.	Enr	3.43

GRID 2 (LTG)		
1.	L.Lo	4.00
2.	Ine	3.60
3.	Inm	2.50
4.	Mar	2.33
5.	Jes	2.00
6.	Ann	1.67
7.	Ale	1.50
8.	Uge	1.33
9.	Flo	1.33
10.	Chi	1.33

GRID 3 (FPG)		
1.	L.Lo	4.75
2.	Ine	4.60
3.	Ann	4.40
4.	Mar	3.67
5.	Chi	3.50
6.	Inm	3.00
7.	Mim	2.83
8.	Car	2.67
9.	Flo	2.67
10.	Ale	2.67

GRID 4 (STG)		
1.	L.Lo	4.50
2.	Ann	3.60
3.	Ine	3.60
4.	Mar	3.17
5.	Ale	2.50
6.	Enr	2.50
7.	Flo	2.00
8.	Jos	1.83
9.	Chi	1.83
10.	Inm	1.17

INDIVIDUAL AVERAGES		
1.	L.Lo	4.30
2.	Ine	4.00
3.	Ann	3.10
4.	Mar	3.10
5.	Ale	2.37
6.	Chi	2.27
7.	Inm	2.13
8.	Flo	2.13
9.	Enr	2.09

GRID AVERAGES	
SPG	3.569 (71.38%)
LTG	1.937 (38.74%)
FPG	3.278 (65.56%)
STG	2.670 (53.4%)

The results can be analyzed from three different points of view. First the individual user results for each grid were studied. Through this analysis one can see that the SPG provides better results, since all of the users average more than 50% correct choices. FPG provides similar results, but with worse average results but higher maximum value. The explanation is simple: SPG is the most played, and repetition is a major factor in this kind of tests; thus most of the users achieved a degree of accuracy quite good. Moreover, the majority of the users did the practice for the SPG but skipped the practice of the other three. Subjects with the same level of practice obtained very close results in grids SPG and FPG.

When talking about the FPG something else becomes clear: the individual ability together with the randomness of the process makes the results highly variable. The average results are spread over a wide range of values that impedes the extraction of any clear conclusion, besides that a high number of sound sources leads to a high degree of confusion and does not provide a good spatial audio experience. STG was the one with the least number of participants and so the variance of the results must not be taken too negatively.

From a different perspective, one can analyze the results based on the global user results. Three subjects (Ine, L.Lo, and Ann) were personally asked to perform the tests in a more thorough way, by practicing and reporting their comments on the tests. The fact that these three subjects appear in the first positions of all the grids and therefore in the first positions of the total results reinforces the theory that practice, repetition, and a priori knowledge of the system provide a significant advantage when making choices. Some of the relevant facts that these three subjects reported were: the importance of the proper functioning of the headphones, the creation of their own way of recognizing the sounds once they learned how to make correct choices, the great difficulty of distinguishing between close sounds in grid two, and the fact that practicing at least once to determine a reference position was of major importance.

Finally, the grid average results were compared. This comparison is only valid among the SPG, the LTG, and the STG, since the FPG utilizes different parameters. The results show that grids SPG and LTG provide very similar and good results, thus this kind of spatial scheme should be the one used in the final application.

10. Conclusions About Spatial Audio

ITDs and IIDs give very useful information about how we localize sounds. The most important result is that in well-defined environments with marked ITDs or IIDs, the results obtained are very satisfactory. The "Six Position Grid" and the "Long Table Grid" display an average of correct choices higher than 50%. Both of these environments could be used for the final application. Considering that the conditions of the test presume no previous knowledge of the position of the incoming sound, the results obtained imply that a user-guided process of locating their incoming phone calls in space will provide a very good spatial experience.

In order to enhance the results, Supernatural Auditory Localization (section 7.2) could be used, so that the cues could be varied in an unnatural way that provides better performance.

The "Square Table Grid" has worse results in general; showing that if the cues are not very defined the user easily gets confused, since the externalization process is not precise and the localized positions fall into the blur area.

The other major result extracted from the test is based on the "Fifteen Position Grid". The analysis of the results from this grid show that users have a lot of difficulty in choosing the right source. Most of the users are able to determine whether the sound comes from the right or the left, but when the resolution problem arises the decision becomes really hard. Many of the users achieved **no** correct choices, showing the difficulty of making the correct choice in very busy environments. A virtual environment with such a large number of positions is therefore not recommended for the design of the final application.

11. Future Work on Spatial Audio

This first part of the report gave a theoretical introduction to spatial sound. The experiments are based on simple tests based on two parameters (ITD and IID). In order to obtain more accurate results other parameters such as reverberation, echo, supernormal localization, and HRTFs should be introduced. Including these new variables would lead to a more complex spatial environment, giving a better understanding of the location process, but not necessarily ensuring better results, since the increasing complexity of the environment leads to an increasing difficulty in the localization process. Therefore, the individual contribution of each parameter should be studied to determine which of them could contribute to a better spatial audio system than that provided in this thesis.

PART II: PROGRAMMING TOOLS

Analysis and selection of the tools and methods

involved in realizing a spatial audio module

12. Programming Language

The entire "minisip" application has been developed in C++ following a modular structure where every feature of the program is represented by its own folder containing the header files as well as the code itself. The majority of the libraries used have been adapted for the application and placed in their own folders, with their particular dependencies and compiling instructions. Some of the libraries used are not directly included in the "minisip" files and have to be downloaded from other sources.

Since this project was to be compatible with all the existing code and was to be integrated into the "minisip" application, the code developed for the audio localization process is C++ code in its own folder, containing its own header files with the methods for the created classes. Since there is no need for a complex structure of classes, only one class is defined, containing all the methods that are necessary for the creation of spatial output. There are no new libraries needed, but some of the existing files need to be compiled against the "libsamplerate" library [3], thus this becomes a requirement for users that want to enable the spatial audio features in their installation of the program.

In contrast, files developed for resampling tasks were merged into the existing classes, in particular in SoundSource (found under the SoundIO files in the source code of "minisip" [5]). This class defines a method for audio resampling, so that any sound application that creates a SoundSource instance of the class is then able to perform resampling operations on its streaming data.

For simplicity of understanding, and since the majority of the files developed for the test in the first part of the thesis were written in C, the files written to test the functioning of the locating process were also first written in C before being translated into C++ for their inclusion in "minisip". This allows reusing some of the material developed in the first part of the project as well as quickly gaining a better understanding of the sampling process, since its related library is written in C, and had a series of examples in this language.

13. Spatial Sound Tools

The first intention was to use one of the existing sound applications capable of managing 3D sound (a reference to them can be found on page 6). One of the requirements for the selection of the API was that it had to be possible to use it without any charge, and if necessary that it should also be possible to modify it to better suit the purposes of the thesis. These two major criteria lead to the analysis of open source projects, which can be both used freely and extended given the access to all their source files and often their programmers.

Taking a closer look at the open source options I found that OpenAL [41] was the most appropriate alternative to proprietary solutions. OpenAL is a library that provides 3D sound after a short parameter initialization and with simple function calls. Moreover, a wrapper called OpenAL++ [42] simplifies the task, providing C++ compliance and presenting the API in a transparent way to the user. The application only has to link sound sources (file, stream, etc.) to a position in space specified by Cartesian coordinates.

The drawback of using OpenAL and OpenAL++ is the large number of compiling dependencies with other libraries, which increases the complexity of the already complex process of installing "minisip". In addition, OpenAL tests run in the lab show no better results than those obtained with my own code, but instead they increase the processor usage due to their complex internal operations. The tests also showed that the location process worked well with sound files, but the support for *streaming* sound was not so reliable.

For these reasons, the decision was to create and use my own code, simplifying to the maximum extent the operations, and aiming for fast processing with little processor consumption. One of the methods to obtain these results is the substitution of *software emulated fixed-point multiplications* by *lookup tables (indirect memory references)*.

13.1. Lookup vs. Fixed-Point Arithmetic

A lookup process consists of creating a lookup table containing all the possible results of an operation that is to be constantly or frequently repeated, and using this table to simply lookup the result of that operation instead of explicitly doing it. Instead of having to repeat the whole computation each time the function is called, the result is obtained by looking in the appropriate place in the lookup table. To illustrate this technique I will introduce as an example the process followed to locate a sound at a given position.

As explained in Part I of this report, spatial sound depends on two basic cues: delay and intensity. Intensity is the parameter that will be analyzed now to demonstrate how the use of lookup tables works and the benefits it introduces. When adjusting the volume level of each sound channel of the incoming signal, an output value is determined depending on the position and the incoming samples. A problem arises due to type differences. "minisip" uses *short int*[8] variables to represent the sent and received sound samples. These samples have to be multiplied by a floating-point value *(float)* that is calculated depending on the position. The resulting line of code would look like:

*sample_out = (short int) ((float) sample_in * scaling_factor) (code sample I)*

The input sample must be first type-cast to match the type of the scaling factor, and the result obtained must be again type-cast to obtain the *short int* type we need. For the current block size of samples that "minisip" processes, this operation would have to be repeated 160 times every 20 ms, consuming resources in a non-efficient way.

The proposed alternative is to create a lookup table. Since both sample_out and sample_in have a fixed range determined by their type, the results of the multiplications are limited to this range. Moreover, in a five-position configuration such as the one shown in figure 6 (p. 26), only two of the positions need to have volume scaling (for simplicity, the positions located 90º from the user have no sound in the farther ear, i.e., for a sound 90º on the right, no sound is produced in the left ear). Since they are symmetrical positions, only the two scaling factors from one of the positions need to be considered.

Let us take only one of the channels of one of these positions; then all the possible values of the output will be the result of multiplying all the possible values of the input by the scaling factor and then converting the result to a *short int* so it matches the data type of the sound samples.

The code to create this lookup table (code sample II) is very similar to the one shown in code sample I, but a slight detail must be noted. Since the index of the loop is also the index of the lookup table rows, and this cannot be a negative number, we have to make a correction in the multiplication in order to convert to the appropriate range of numbers.

[8] Short int = [-32768, 32767]

```
for (index=0; index<65536;index++) {
    lookuptable[index] = (short int) ( (float) (index - 32768) * scaling_factor ) ;
}
```
(code sample II)

The result of the lookup table can be seen in Table 4. The range of the output table corresponds to the range of the *short int* variables multiplied by the scaling factor and rounded to an integer value.

Table 4 – Lookup table creation (using code sample II); scaling factor = 0.7

INDEX		INPUT (index-32768)		MULTIPLICATION (factor = 0.7)		LOOKUPTABLE[INDEX]
0	→	- 32768	→	- 22937.6	→	- 22938
1	→	- 32767	→	- 22936.9	→	- 22937
2	→	- 32766	→	- 22936.2	→	- 22937
3	→	- 32765	→	- 22935.5	→	- 22936
…	…	…	…	…	…	…
32768	→	0	→	0	→	0
…	…	…	…	…	…	…
65534	→	32766	→	22936.2	→	22936
65535	→	32767	→	22936.9	→	22936

For every different factor we have a lookup table containing 65536 *short int* (16 bits) samples; thus each lookup table represents a use of 0.125 MB of memory. One simple structure to store all the possible output values is to have two matrices, one for the right channel and the other for the left channel. In our example we would have two matrices of five columns (the positions) and 65536 rows (the possible values) each, consuming a total of 1.25 MB in memory. In order to make the lookup processes efficient, there are different alternatives for the creation of the lookup tables.

The first one is to initialize the lookup tables when "minisip" is started. Since there is no knowledge of the future number of users, the five tables for each channel must be created, using a large amount of memory. The amount of data stored is independent of the number of users connected, thus inefficient. The positive thing is that the process is done before the user starts to talk; consequently no operations must be done when the user starts a new conversation; the

lookup process can be directly started. Moreover, all the users share the same matrices containing the data, thus only one lookup table for each position is created. The total amount of data is the already mentioned 1.25 MB.

A different alternative would be to assign each user their lookup tables depending on the position they are assigned as they establish a new call. In this way for every user we have 0.25 MB of memory space needed, 0.125 MB for each channel, and the use of memory is more efficient. However, when more than one user has the same position there would be duplicity of data.

A third possibility arises taking the positive part of the two already proposed alternatives. The process consists in creating lookup tables for every position that is being used. Whenever a user establishes a conversation and is assigned a position, the lookup tables for that position are created. When more than one user is in the same position, they would utilize the same block of data, consequently using memory in a more efficient way. Furthermore, only the memory of the currently used positions is occupied, hence a better memory allocation. Special treatment should be given to those positions having default scaling factors (zero and one); using this information in a smart way the amount of memory used can be greatly reduced.

Now that the process of creation of the lookup tables has been explained, I will introduce the lookup process itself. We now have samples coming in to our system, each sample being a *short int.* Code sample I showed the way the samples should be processed in each channel in order to obtain the desired output. This expression is replaced in the lookup process by the following piece of code:

sample_out = lookuptable [sample_in + 32768] (code sample III)

Notice again that the index of the lookuptable has to be offset in order to ensure a valid index. The results of the lookup process can be seen in Table 5.

This process has to be done for each channel; as we will see further in this report the way this conversion is done depends on the number of input channels. This means that if we have a stereo input we could have different input samples to the lookuptables, while if we have a mono input, the same input sample would feed the lookuptables for the right and left channel.

In any case the payoff from using lookuptables is high because far more than 65 K samples will be processed. This is the number of computations needed to create each of the

lookuptables for each channel and position. Considering the 8 kHz sampling rate earlier, every 20 ms we must process 160 input samples. Thus is around 5 seconds we will have to process more than 65K samples, thus for calls that last more than 5 seconds we would have to do more than 65 K float to short int conversions and multiples; since few calls last less than 5 seconds this method pays off.

Table 5 – Lookup process example (using code sample III); scaling factor = 0.7

SAMPLE_IN		INDEX (sample_in + 32768)		SAMPLE_OUT (lookuptable [index])
- 32768	→	0	→	- 22938
- 32767	→	1	→	- 22937
- 32766	→	2	→	- 22937
- 32765	→	3	→	- 22936
…	…	…	…	…
0	→	32768	→	0
…	…	…	…	…
32766	→	65534	→	22936
32767	→	65535	→	22936

This section of the report has illustrated how the use of lookup i.e., indirect memory references can help reduce the processor consumption especially when using software emulated floating-point multiplications. Many other applications for this technique can be found, all of them try to lower the processor consumption. A good example can be found in the next section of the report where resampling is considered. The basic function used by the resampling module has an array of *float* data as input parameter, and as we have already seen "minisip" works with *short int* sound samples. This makes it necessary to do a type conversion. This could be done by type-casting each input sample, or as it has just been presented, by creating a table of floats where every element of the array is a *float* within the *short int* range, and then using this lookuptable when required. The process would look like:

1. Creation of the lookup table (float lookuptable[65536]):

   for (i=0; i<65536;i++) { lookuptable[i] = (float) (i – 32768) ; }

2. Lookup process:

   float_input = lookuptable [short_int_input + 32768];

14. Resampling tools

Resampling is not a requirement for spatial audio, but it proves to be an interesting feature to be combined with it. Resampling was introduced in this project due to the constraint introduced by the hardware CODEC of the target HP iPAQ h5550. Since this CODEC requires a single sampling rate for all the sound data that is sent to it, if we want to have the possibility of simultaneously using different CODECs with different sampling rates, and to have multitasking of our "minisip" without having to stop one call to start a new one using a different sampling rate, resampling of signals to a common rate that is accepted by the hardware CODEC is essential.

More discussion of the reasons that lead to the introduction of resampling in the project, the way that this is performed, and the results that have been obtained will be shown in the next section of this report. Here I will introduce the resampling tools that have been examined, and explain in detail the one that has been selected for the spatial audio module.

There are several libraries and resampling products available, but as stated from the beginning the aim of this project was to find open source solutions. Taking a look at these we find "resample" [24], which is considered to be one of the first high quality sample rate converters library available as source code. A lot of converters have been developed taking "resample" as a base, the one presented in [10] probably is one of the best examples. "resample" contains free sampling-rate conversion and filter design utilities written in C, and it is based on the theoretical approach presented in [25] by its author. Other approaches to sampling rate conversion can be found in [14][17].

"libresample" [12] is another of the libraries based on "resample". As stated by its author, "libresample" *"is not the highest-quality resampling library available, nor is it the most flexible, nor is it the fastest. But it is pretty good in all of these regards, and it is quite portable. The best resampling library I am aware of is libsamplerate by Erik de Castro Lopo."[12]*

"libresample" introduced floating-point computation compared to "resample" that used fixed-point, as well as established support for streaming information by handling small chunks of data to be processed and enabled time-varying resampling ratios.

The last two libraries I will introduce are "libsamplerate" [3] and the one presented in [10]. This later is targeted for being used in real-time applications when managing blocks of streaming audio. It is implemented with a time-varying fractional delay filter and uses lookup

tables to store the filter coefficients in order to eliminate the calculation of the coefficients at run-time. As was mentioned earlier in this report, lookup tables are a highly efficient method since they compute the coefficients only once; but as different tables are needed for different rates they can become memory consuming. This library has a code optimization specification in order to enhance the performance of "resample" on which is based. As it can be seen in [10] the optimization can reduce the number of cycles by 64%, as well as improving both processor and memory efficiency.

To conclude I will analyze the library that was finally used for the creation of the spatial audio environment. The library is called "libsamplerate", although it is also known as Secret Rabbit Code (it takes the initials from Sample Rate Converter) [3]. The library has been created by Erik de Castro Lopo, and its last release was September 2004. The main feature of the converter is the ability to perform time varying conversions as well as arbitrary conversions.

Arbitrary conversion implies that any incoming stream can be resampled independently of its sampling rate. The resampler can upsample and downsample at a maximum factor of 256. This allows us to have many different applications providing different kinds of data, from high quality CD music (44.1 kHz) to normal telephony speech (8 kHz), all of them being converted to the same sample rate.

Time varying conversions are very useful when considering telephone applications running over the Internet network. A moving user that changes from network to network (each of them with their specific bandwidth and capacity) needs to be able to switch CODECs and sampling rate depending on the available link and network bandwidth. These changes in the parameters of the transmitted data need to be followed by a change in the resampling converter to fit the new incoming sampling frequency.

Apart from these two important features, the resampling application provides five different converters with specific characteristics. There are three different underlying technologies for the converters: band limited interpolation, zero order hold conversions, and linear conversions. All three band limited interpolation converters are based on Julius O. Smith's techniques. The specific characteristics of the filters are (extracted from [3]):

- BEST_QUALITY: this is a band limited interpolator derived from the mathematical sinc function and this is the highest quality sinc based converter, providing a worst case Signal-to-Noise Ratio (SNR) of 97 decibels (dB) at a bandwidth of 97%.

- MEDIUM_QUALITY: this is another band limited interpolator much like the previous one. It has an SNR of 97dB and a bandwidth of 90%. The speed of the conversion is much faster than the previous one.

- FASTEST: this is the fastest band limited interpolator and has an SNR of 97dB and a bandwidth of 80%.

- ZERO_ORDER_HOLD: a Zero Order Hold converter (interpolated value is equal to the last value). The quality is poor but the conversion speed is very fast.

- LINEAR: a linear converter. Again the quality is poor, but the conversion speed is very fast.

The selection of the converter based on the performance requirements of the system will be described in the next section of the report. Depending on the required speed and the processor resources available, one or another converter should be chosen.

The sampling library has three different APIs, two of them being of particular interest for this thesis. The simplest one converts a block of mono or stereo samples to the desired output sample rate. This API consists of one function, which takes the selected converter, the number of channels, and the data structure as parameters. The data structure is common to this simple interface and the other two more complex interfaces, so I will go a little more into detail. The structure is called SRC_DATA, and is defined as follows:

```
typedef struct
    {   float  *data_in, *data_out ;
        long   input_frames, output_frames ;
        long   input_frames_used, output_frames_gen ;
        int    end_of_input ;
        double src_ratio ;
    } SRC_DATA ;
```

The *data_in, data_out, input_frames, output_frames, and src_ratio* fields must be fill in by the user. The first two are pointers to the arrays containing the input and output data. *Input_frames* and *output_frames* represent the number of samples of data pointed by the two previous pointers. It has to be noted that in case of monophonic sound these values refer to the

length of the arrays, but for stereo sound these two numbers are equal to half of the length of the arrays. To conclude, *src_ratio* specifies the conversion ratio defined as the output sampling rate divided by the input sampling rate.

The function implemented by this simple API just takes one element of the SRC_DATA type, and performs the conversion leaving the data in the *data_out* array. This API is not useful for my application since it maintains no state of the converter from one call to the next one. A more complex API is needed for spatial sound purposes. This is the full API defined in the "libsamplerate" library, which allows time varying sample rate conversions on streaming data.

The full API consists of five different functions. These are:

- SRC_STATE* src_new (int converter_type, int channels, int *error) ;
- SRC_STATE* src_delete (SRC_STATE *state);
- int src_process (SRC_STATE *state, SRC_DATA *data) ;
- int src_reset (SRC_STATE *state) ;
- int src_set_ratio (SRC_STATE *state, double new_ratio) ;

First we have to create a sample converter object. We do this using the *src_new* function. Here we specify the type of converter, and the number of channels of the input. The complementary function is *src_delete.* Once the converter is created, we use the same SRC_DATA structure defined before to define the input and output parameters, and call the *src_process* function to make the sample rate conversion. We can change the ratio of the conversion between calls to the *src_process* function by calling the *src_set_ratio.* When we want to return to the original converter we can call the *src_reset* function.

These five functions are supported by two auxiliary functions that convert blocks of *shorts* into *floats* and vice versa. Since the input to the *src_process* has to be a block of *floats,* and "minisip" processes sound in *short int* samples these two functions allow us to feed the converter with the correct type of data, and then convert it again to match the "minisip" requirements.

To conclude, it is particularly significant that the library has no dependencies beyond what is provided by the standard C library, thus it should compile and work on any operating system.

15. Conclusions and Further Work regarding the tools and methods selected

Some important conclusions that will influence the creation of the spatial audio application can be extracted from this part of the report:

- the programming language for the initial files will be C; when the system proved to be stable and all the required features worked properly, a translation into C++ as well as the integration in the "minisip" code was performed

- no spatial audio libraries were used; in its place self written code was used, allowing resampling to be included as an integral process together with the spatialization

- optimization of the code was done using lookup tables, i.e., indirect memory referencing, instead of software emulated floating point multiplications when possible

- the "libsamplerate" library was used for the resampling processes; the full API interface provided by the library was used, since it is intended for streaming applications with continuous calls to the resampling functions.

Only one sampling library has been thoroughly analyzed in this thesis, since it is considered to be the best open source option available. Nevertheless, other options could be considered, particularly the resampling library proposed in [10] that has special optimization, and is intended for real-time applications. Mat Hans should be contacted in order to gain access to the source code or at least the API.

The spatial audio libraries should be reconsidered and studied in depth. The complexity of these libraries is high, but the specific features of OpenAL for spatial streamed audio make it worth considering. Furthermore, specific solutions for Windows users should be considered, since the port to this operating system is expected to be finished soon, and the majority of "minisip" users will likely be using it.

To sum up, alternative tools to the ones presented in this part of the report should be analyzed, and their performance compared to the solution selected here.

PART III : THE APPLICATION

Approach and final implementation

of the spatial audio application with

multi sampling rate support

16. Goal of the Implementation

Until now 3D sound has been mainly developed by the entertainment industry to provide enhanced audio experiences to their customers. The design that is shown in the following pages introduces the field of spatial audio in the arena of communications, specifically in a VoIP environment as offered by "minisip". Currently applications are designed for the user to hold one conversation and to perform one task at a time. As a result of this thesis, it will be possible to see the effects of speech signal processing on telephony, and devise new applications based on this enhanced listening experience, as well to introducing multitasking as an integrated component.

The way that most of the VoIP products on the market work right now, the information carried between users is highly dependent both on the network and the hardware used for the communication. This thesis focused on the specific hardware limitations imposed by the audio parts of the computers, PDAs, and other VoIP enabled devices. A generic implementation of an audio system can be seen in figure 8.

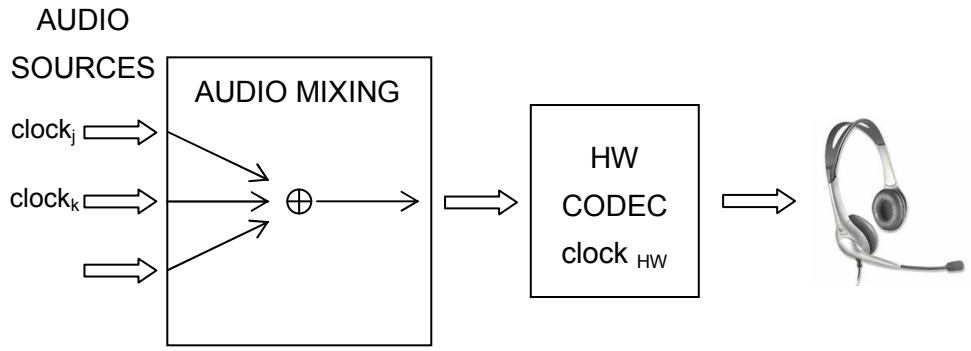


Figure 8 – General Audio Disposition for VoIP Applications

The previous figure shows us some of the main limitations of the system. Incoming sound sources might have different sampling rates, and these could be at the same time different from the hardware sampling rate. The first problem comes when mixing sources with different sampling rates. If we are to mix these sources into a single output stream, the resulting sampling rate needs to be properly defined, and such that the hardware can reproduce this combined stream at the appropriate rate. This is easily done if we have inputs with the same input rate as output.

If we consider sources with the same sampling rate, but different from hardware CODEC sampling rate. There are two different scenarios:

- if $clock_k \leq clock_{HW}$ the decoding of the data is potentially lossless. This is the case for "minisip" before this thesis. The iPAQ has a hardware CODEC running at 16 kHz sampling rate and the input encoded with a G.711 CODEC providing 8 kHz sampling rate. Then the sound has to be upsampled on reception to match the hardware requirements.

- in the other hand, if $clock_k > clock_{HW}$ the decoding is lossy, it cannot process the information at the same speed as it is provided. In this case, if we have an incoming source with a high sampling rate (CD audio at 44.1 kHz for example), we would have to downsample it to match the hardware requirements, thus losing quality.

The first goal of the implementation was to support high quality audio while exploiting the fact that low sample rate sound sources already have to be upsampled. For this purpose, the hardware CODECs of our communication devices were set to high quality, and all the incoming sources resampled to this high rate. With this solution we have no degradation of high quality incoming sources, but we also prepare the support for spatial audio (as shown on page 29, spatial audio requires a high sampling rate to have a good time resolution when performing the delay operations). Since all the sources are now resampled to one single sampling rate, this allows integrating them into a unique stream without degrading the properties of the individual components.

Once the initial goal of mixing is achieved and the data is properly prepared, it is time to perform spatialization operations on it. Spatialization introduces multitasking as an integrated feature for any communication device. The possibility of establishing SIP sessions with different types of audio applications while having their sound delivered all at the same time increases the user's capabilities. Once this new mode of communication is enabled, new applications can be designed. Finding the best uses of these new capabilities is of great importance. Recent studies have shown that women are better prepared for multitask than men [44]; thus women might be the focus for these new multitask applications. The first project for spatial audio based services is already been developed by students in the Communication System Design (CSD 2005) course [43], which is focused on enhancing videoconference services.

On the other hand, the increase in data communication has its drawbacks. Traffic to and from a spatial audio enabled user holding different audio streams at the same time has a big impact on the network utilization and the resources available. Further analysis on these implications, as well as finding the network limitations for multistream support are possible tasks for upcoming projects.

The whole implementation process can be seen as an iterative process made up of small steps (figure 9) that convert data from an incoming stream of data with a random sampling rate, to a stream of data with a fixed length, a specific sampling rate, and specific delay and amplitude modifications in its stereo channels. Having these fixed parameters makes it possible to have a number of sources integrated into a single output source; thus the resulting sound stream is independent of the incoming parameters which allows designing the entire system depending only on this last stream and not on the individual ones.

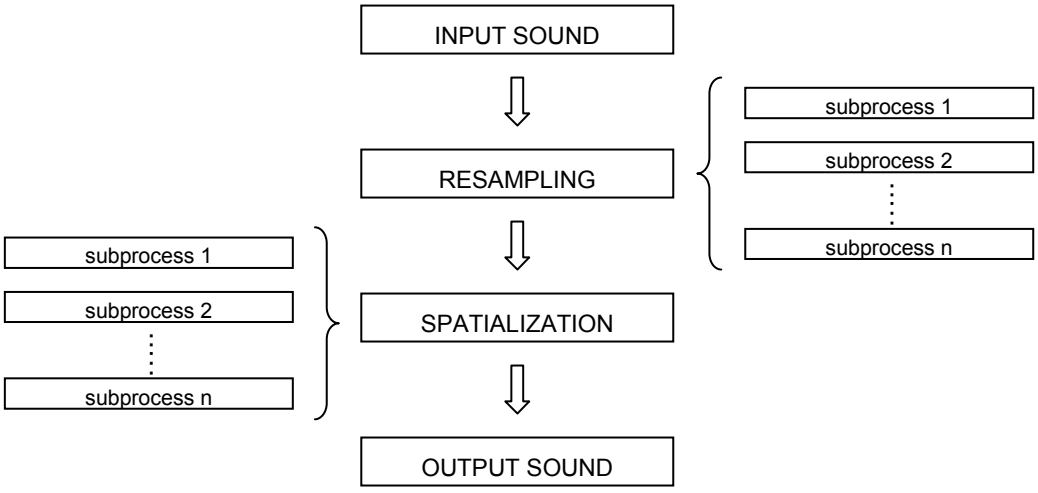


Figure 9 – Scheme of the Spatial Audio Implementation

The specific steps followed for the implementation are introduced in the following two sections, first with a C approach to the problem, and finally with the definitive implementation of the spatial audio module for “minisip”.

17. First Approach: the C example

In order to reuse some of the code used for the first part of the thesis the initial implementation of the spatial audio implementation was developed in C. The sound examples used for this part are not streaming audio data, but rather WAVE files converted to raw audio samples. All of them were 16 bit stereo sound files. The objective of the programming was to create a simple application that read these files and then build upon it the complex spatial operations. The code for the final C implementation can be seen in Appendix A, where only the code for the spatial audio has been shown (the complete code can be found at http://www.e.kth.se/~isp/exjobb/cfiles).

The first step to simulate streaming audio is to read the information from the file in blocks such as the ones that arrive to the audio device in a VoIP session. To make the simulation realistic and since the final target is “minisip”, I used the same block size it used. This means that blocks of 160 mono samples are extracted every 20 ms from an 8 kHz sampling rate CODEC, and then transformed into stereo by duplicating the information into two channels. To simplify this task I preprocessed the monaural files and converted them to stereo. After that transformation the block reading process was done in blocks of 320 samples.

Once the block of samples was fetched, the following step (as shown in figure 9) was resampling. As explained before the main goal of the implementation was to support high quality audio, and for this reason the sampling rate of the hardware CODEC was set to 44.1 kHz. The parameters needed for the resampling were the input sampling rate, the length in monaural samples of the input, the length in monaural samples of the output, and the ratio between the output and the input sampling rates. These parameters are related since the ratio between sampling rates is the same ratio as the one between output and input buffer lengths.

The resampling process is then broken into three sub-processes. The fact that the resampling routine processes floating point numbers while the audio is usually stored in 16 bit integers implies that a float-to-integer and an integer-to-float conversion must be wrapped around the resampling process itself. The conversion routines are already provided by the resampling library. As a result of the resampling the incoming 320 samples of 8 kHz sampled audio were converted into 1764 samples at 44.1 kHz sampling rate. It was then time for the spatialization.

My first approach to spatialization consisted of a direct operation on the incoming block of data turning it into a spatialized output block in a single operation. The determination of the

delayed channel and the exact number of delay samples as well as the scaling values for the channels are dependent on the play out position determined by the user, and obtained from lookup tables. The process itself consisted of reading the samples of the resampled input one by one; every time a sample was read it was placed into the output stream in the appropriate position by separating the left and the right channels by the exact number of samples (do not forget that stereo audio is stored in a channel alternating fashion, L, R, L, R, …). The problem with this kind of implementation is that the delayed channel starts to overflow when the delay causes the input samples to exceed the limit of the output buffer. To solve this limitation an additional buffer is used. This buffer saves the extra samples and keeps them for the next iteration. An example can be seen in figure 10.

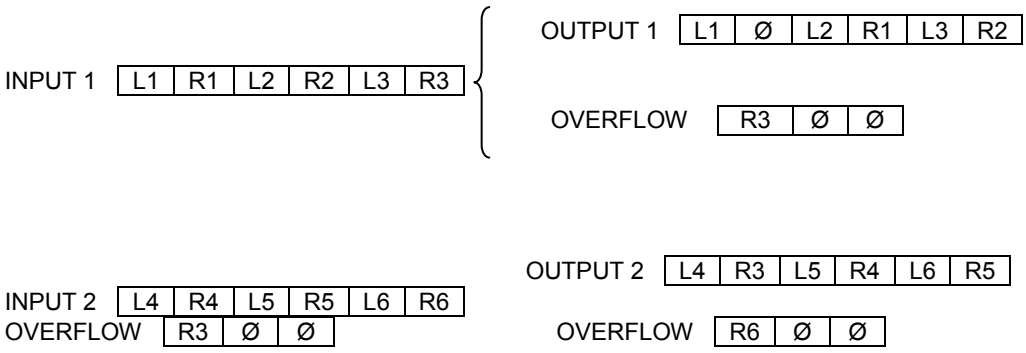


Figure 10 – Delay Control using Overflow Buffer

The figure shows an example where the right channel is delayed one sample from the left channel. Due to this delay the last sample of the right input channel ought to be out of the range of the output buffer. This sample is stored into the overflow buffer and processed with the next block of samples in the next iteration.

An alternative to this technique is the use of circular buffers. Each circular buffer contains the samples of one of the input stereo channels. There are two different pointers, one for reading and one for writing to the buffers. As shown in figure 11, the samples are first written into the circular buffers based on the state of the writing pointer from the previous iteration, and then the output buffer is extracted using the reading pointer at the place where it was left by the previous reading process.

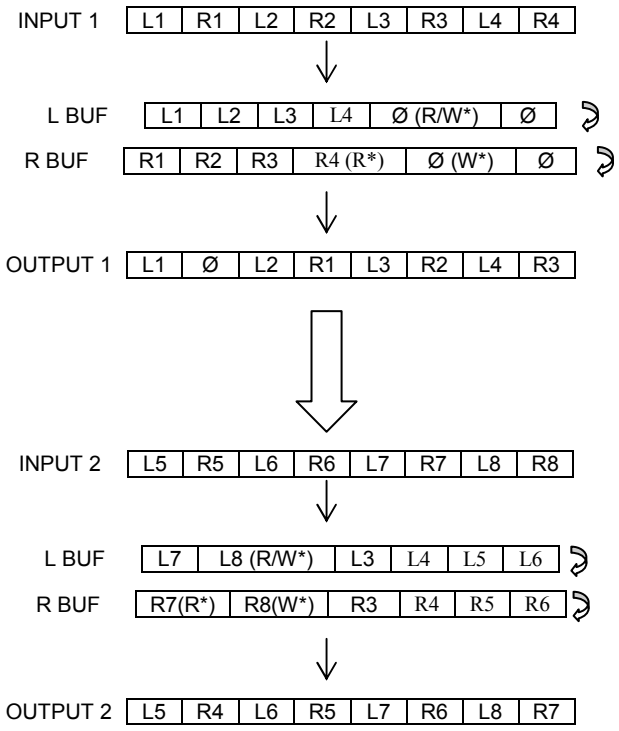


Figure 11 – Delay Control using Circular Buffers

The important details to observe in this figure are the different positions of the read (R*) and write (W*) pointers depending on the channel. Since the right channel has a delay of one sample, its pointers are also delayed one sample, so the writing pointer follows the same rhythm as the left channel, but the reading pointer is one sample behind.

In the C file in Appendix A the circular buffer method is used. However, here only one source is considered. If there were more than one source, then the output for each source should be added together before sending the sound to the hardware CODEC of the soundcard.

18. Final Version: the C++ API and Integration with "minisip"

The previous section introduced the basic principle of the resampling and spatialization routines that were used for the "minisip" implementation of spatial audio. The next step was to analyze the sound structure of "minisip" and find a way to integrate my module, whether introducing it as an external module or modifying existing code. The sound system of "minisip" before this thesis can be seen in figure 12. It follows the general description presented in figure 8, with three different classes involved.

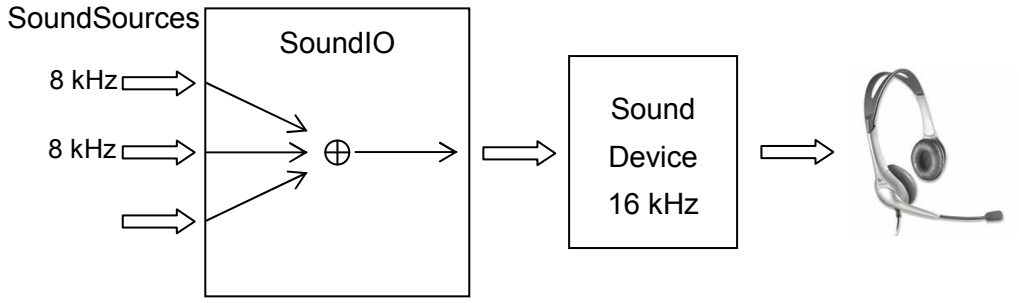


Figure 12 – "minisip" Sound System

The SoundSource class represents the audio information, for now only VoIP calls; SoundIO represents the audio mixing module, and although only its function as an input module was considered for this thesis, it also provides the structure for the outgoing audio; and finally the hardware CODEC is represented by the SoundDevice class. "minisip" allows defining different devices, but the one used right now is an OSSDevice [44].

Table 6 – Classes used for "minisip" Sound System

Classes used:	
SoundSource	Gets sound from the audio stream and stores it into a buffer, then provides this buffered data to SoundIO
SoundIO	Mixes the different streamed data contained in the buffers provided by the SoundSource(s)
SoundDevice	Low level interaction with the hardware

The integration of the final spatial audio module consisted in the creation a new class, SpAudio, declared as a friend of SoundIO (i.e., has access to its private variables), and modification of the SoundSource and SoundIO classes. Let us first introduce the new class,

SpAudio. SpAudio is created by SoundIO and performs the spatialization of the sound buffers that it is provided. It defines one method called `spatialize` that takes an input buffer, an output buffer, and a SoundSource as parameters. This method is exactly the same function that was presented in the previous section. It also defines a second method for assigning positions to the incoming sources as they register in the system. This method looks in a table and assigns a position to the new source depending on the number of sources in the system.

The proposed design provides automatic assignation of the positions. The system assigns the central position to the first incoming call, and then proceeds by separating the sound sources as much as possible as they enter the system. The possible configurations depending on the number of calls in the system are shown in figure 13. This automatic assignment of the position should be one of the first things to be considered when improving the system, since it should be possible to manually override the assignment, to assign positions depending on pre-established priorities, or to have more than one source in the same position.

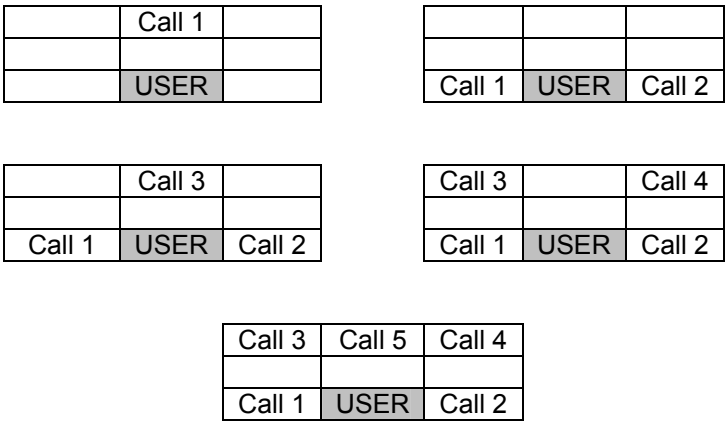


Figure 13 – Possible Configurations of the Spatial Grid depending on the number of calls in the system

The main modification made to the SoundSource class was to introduce resampling. The most important method introduced is called `resample` and it is basically an adaptation of the function shown in the previous section. The resampling depends on the sampling rate of the SoundSource, which is a private variable of the class. The majority of the methods and variables introduced in SoundSource have been declared as private, in view of the fact that ideally we want to have a number of sources with different types of sound data, not just phone calls, and each of them might need to implement its own method; thus, subclasses for each of the sound sources will also be something under consideration for further development of the system.

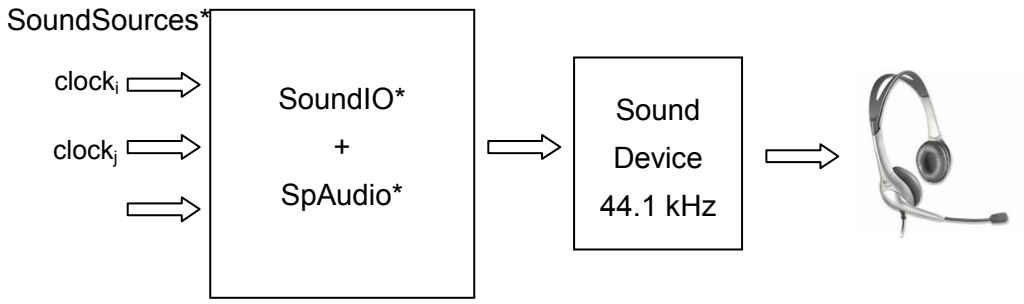


Figure 14 – "minisip" Sound System with Spatial Audio

The last modifications were made in the SoundIO class. Here, the `playerLoop` method was the major change. This loop reads one by one the buffers from the SoundSources and creates a single output buffer by adding their contribution. The `playerLoop` method has been modified so it reads each buffer, resamples it depending on the incoming sampling rate, spatializes it depending on the position that the spatial audio module has assigned it, and finally adds it to the output buffer that will be then sent to the sound device. It was in this point where we found some unexpected problems. Although the resampling and spatializing routines seemed to work well, the output sounded in a strange delayed and interrupted way. After setting some counters and with the help of a debugger we found out that the routine sending data to the hardware CODEC was taking more than 23 ms to do it, thus producing a delay in the communication since the information from the buffers was being read every 20 ms. We increased the size of the internal buffer that seemed to be causing the problem and everything worked properly.

Table 7 – Classes used for "minisip" Sound System after spatialization

Classes used:	
SoundSource*	Gets sound from the audio stream and stores it in a buffer, it resamples the sound in this buffer and sends the resampled information to SoundIO
SoundIO*	Mixes the spatialized streamed data contained in the buffers provided by the SoundSource class
SpAudio	Does the spatialization of the data when required by SoundIO. It also assigns position to sources depending on the current number of sources in the system.
SoundDevice	Low level interaction with the hardware

It is important to consider some details of the implementation:

- the converter for the resampling has to be carefully initialized, since processing depends on if the input audio is mono or stereo.

- moreover, tests showed that the best converter available produces a delay in communication (the processing time for this converter is around 15 ms). Converter number 2 is the one recommended for standard applications, whereas when listening to music or other entertainment audio converter number 1 is recommended. For the worse case scenario converter 4 provides GSM quality with incredibly fast performance.

- the proper definition of the of the writing and reading pointers is also very important, since they have to maintain their values between calls and they have to be properly initialized (to 0).

- automatic gain control should be added to ensure appropriate output levels at every moment. Depending on the number of sources interacting at the same time the amplitude of the output can vary over a very wide range. The simplest solution is to leave the gain control for the user, e.g., not introducing any amplitude control dependent on the number of active sources. With this solution the user adapt their conversation to the current sound that they receive, as they would do in a face-to-face conversation where one stops talking when another starts. A more complex solution consists in assigning different value attenuations depending on the virtual position and the number of active sources. User experiments should be performed to determine which of the two solutions is preferred.

19. Conclusions and Further Work on the Implementation

This part of the report has shown the details of implementing spatial audio functionality into an existing SIP User Agent. The implementation is supported by the theoretical results of the first part of the report. Example code in C has been also provided in order to familiarize the reader with spatialization, and to have a module that is independent of the final implementation, so it can be reused for other applications.

The following parts have been implemented:

- enabling of resampling as basic operation performed on incoming sources, resulting in a single high sampling rate for the output. This gives us the possibility of having high quality audio without degradation and allows us to integrate the spatial audio module.

- implementation of the spatial audio environment by modifying some of the existing classes, providing them with the necessary variables and methods.

- implementation of a spatializing method based on circular buffer caching of information, providing up to 5 different spatial locations.

The final result is a basic implementation of a spatial audio tool for streaming audio. Further work on the implementation could introduce graphic commands to run the system. A major need is to implement manual control of the spatializing process, i.e., the user can assign positions by simply pre-assign the sources a priority or by doing so *on the fly*. It is also important to perform some user tests and gather some results related to the automatic gain control, and decide whether not gain control should be done or should be left to human interaction, or if some kind of control depending on the number of sources and the current amplitude of the output should be done.

Additional further work on the spatial audio module includes integration with a speaker recognition system, the analysis of the possibility and need of having a greater number of spatial positions, the adaptation of the spatial sound to multiple videoconferences, and modifications of the module that can lead to the introduction of new services and enhanced use of multitasking.

PART IV: CONCLUSIONS & FURTHER WORK

20. Conclusions

This report has presented the design and implementation of a spatial audio module for VoIP communication using “minisip”. The whole process from the theoretical analysis of spatial audio conditions to the final implementation has been described in the previous sections.

The report gives an idea about how the human ears interact with the sound environment surrounding them, and introduces delay and intensity as the two main parameters when identifying a sound location. The possibility of digitally modifying these two cues for a non-spatialized monaural sound is the basis of the spatializing process.

Delay and scaling operations were done in this thesis by applying speech signal processing techniques, and simplifying the processes to obtain fast operation and minimum delay. The report has utilized lookup tables as an efficient way of replacing high computational cost floating point operations. The design and implementation of the spatial audio module were a trade-off between quality of audio and consumption of resources. All the system variables had to be adjusted to offer a stable service while having high quality audio.

During the development of the spatial audio module the need for a second design and implementation effort arose: since spatial audio could not be achieved with the low sampling rate of the incoming audio, resampling was needed. Resampling tools have been introduced and analyzed in this report, and have been integrated into the spatial audio experience. Having resampling integrated in the system allows high quality audio be reproduced without losing quality, as well as gives the appropriate background not just for spatial audio but also for text-to-speech and speech-to-text conversions [18].

Enabling resampling and spatial audio provides the basis for successful multitasking, with a proper integration of sound sources and an efficient distribution in the listening field of the user. Applications that can directly benefit from this module are multiconferencing and videoconferencing, as the spatial audio module makes the task of handling different audio sources much easier.

However, there is a downside to this scheme of multisource management. Being able to listen to multiple sources means that the traffic to and from such a user is fundamentally changed. It is no longer limited to a single stream or a single call. Some examples for these effects are: great increase of the downlink audio traffic - since you can be in multiple calls, and a change in

the interarrival distribution of packets - since even with silence suppression the probably of not having a packet in given time interval drops as the product of the probability distributions of the individual sources.

The final result is a spatial audio module for “minisp”, a SIP User Agent that is being developed by students at KTH. The module will hopefully provide the basis for further projects within this context.

21. Further Work

Spatial audio offers numerous possibilities for further development. The spatial audio module presented in this report can be taken as a starting point for this development, by implementing new applications based on it, as well as improving its performance and its user interface.

Further work considering the impact and the possibilities introduced by spatial audio should be considered. Some of these could be:

- study the possibility of plugging new CODECs into "minisip". The possibility of selecting a different CODEC depending on the actual conditions of the access network as well as being able to manually select the desired CODEC are some of the proposed investigations. This is also of major importance when talking about multiple audio streams, since the coexistence of multiple streams increases the amount of data received by the device. Additionally, all of the streams need not utilize the same CODEC; for example the sender might have only limited bandwidth via his current link, even though the receiver has a high bandwidth link and high quality audio from other sources.

- the effects on the traffic to and from the mobile user's device should carefully be examined. This analysis together with the spatialized audio system must enable the determination of the maximum number of streams manageable by a "minisip" user depending on the capacity of the network. This goal is directly related to the previous one, since the amount of traffic incoming and outgoing from the device can be varied by changing the CODEC used or by pre-caching contents. This later approach is being examined in a parallel project by Johan Sverin and Inmaculada Rangel Vacas [18]. As the SIP protocol, used for this VoIP application, allows the renegotiation and changing of CODEC during a conversation, the available bandwidth can be dynamically accommodated. Additionally, SIP allows multiple CODECs to be negotiated, thus traffic from multiple encodings might be sent to allow for different receivers to receive different quality, to allow for short-term changes in connectivity (for example, a vertical handoff for just a short period as a user moves between two buildings on a campus). Note that the use of resampling allows for easily mixing the audio from different rate CODECs that in this case are associated with the same source.

- performance analysis of the spatial audio service should be done, focused on the user experience and also the efficiency of the designed module. Users' opinions should be gathered and taken into account to enhance the module and provide a solution that better suites the requirements of the users.

- the implementation of the manual control of the spatializing process, as well as enabling SIP connections with different other audio sources different (e.g., radio, music, and others).

- study of the possibility of having spatial audio based on head related transfer functions. Implement this alternative and compare it to the use of simple spatial cues.

- study the creation of a spatial audio environment with a higher number of positions. Determine whether this is feasible, profitable, and manageable from the network, the user, and the device point of view.

- consider the introduction of a more efficient resampling algorithm such as the one presented in [10].

REFERENCES AND BIBLIOGRAPHY

[1] Blauert, J. *Spatial Hearing: The Psychophysics of Human Sound Localization,* The MIT Press, Cambridge, Massachusetts, 1997

[2] Bronkhorst, A.W. *The Cocktail Party Phenomenon: A Review of Research on Speech Intelligibility in Multiple-Talker Conditions,* Acustica 86, pp. 117-128, 2000

[3] de Castro Lopo, E. *Secret Rabbit Code Library*, v. 0.1.2, September 2004, http://www.mega-nerd.com/SRC

[4] Doelle, L.L. *Acoustic Requirements of Meeting Rooms,* Public Works Canada, Ottawa, Ontario, 1988

[5] Eliasson, E., and Billien, J. *minisip*, KTH, Stockholm, 2004, http://www.minisip.org/

[6] Grantham, D.W. *Hearing: Spatial Hearing and Related Phenomena,* Academic Press, Chapter 9, pp. 297-345, 1995

[7] Huang, X., Acero, A., and Hon, H. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development,* Prentice Hall, New Jersey, 2001

[8] Johnston, A., and Levin, O. *Session Initiation Protocol Call Control - Conferencing for User Agents,* IETF, Internet Draft, 2004

[9] Maguire Jr., G.Q. *2G1325/2G5564 Practical Voice Over IP (VoIP): SIP and Related Protocols,* January 2004, http://www.imit.kth.se/courses/2G1325/VoIP-2004.pdf

[10] Mar, C.T., Hans, M.C., Smith, M.T., Simunic, T., and Schafer, R.W. *A High-Quality, Energy Optimized, Real-Time Sampling Rate Conversion Library for the StrongArm Microprocessor*, Hewlett-Packard Labs, Technical Report, HPL-2002-159, 2002

[11] Mauer, F. *Push-2-Talk in VoIP Decentralized,* Term Project, KTH, Stockholm, 2004

[12] Mazzoni, D. *Libresample Library*, v. 0.1.3, January 2004, http://www-ccrma.stanford.edu/~jos/resample/Available_Software.html

[13] Moore, B.C.J. *An Introduction to the Psychology of Hearing,* San Diego, CA, Academic Press, 1997

[14] Park, S., Hillman, G., and Robles, R. *A Novel Structure for Real Time Digital Sample Rate Converters with Finite Precission Error Analysis*, Int. Conf. Acoustic Speech Signal Processing, vol. 5, pp. 3613-3616, 1991

[15] Perrot, D.R., and Saberi, K. *Minimun Audible Angle Thresholds for Sources Varying in both Elevation and Azimuth,* J. Acoust. Soc. Am. 87, pp. 1728-1731, 1990

[16] Plenge, G. *On the differences between localization and lateralization,* J. Acoust. Soc. Aa. 56, pp. 944-951, 1974

[17] Rajamani, K., Lai, Y., and Farrow, C.W. *An Efficient Algorithm for Sample Rate Conversion from CD to DAT*, IEEE Signal Speech Processing Letters, vol. 7, no.10, pp.

288-290, October, 2000

[18] Rangel Vacas, I., and Sverin, J. *Context Aware and Adaptative Mobile Audio,* Masther Thesis, Stockholm, 2005

[19] Schreier, M. *Audio Server for Virtual Reality Applications,* Brunel University, 2002

[20] Searle, C.L., et al. *Model for Auditory Localization,* J. Acoust. Soc. Am. 60, pp. 1164-1175, 1976

[21] Shaw, E.A.G., and Vaillancourt, M.M. *Transformation of Sound-pressure Level From the Free Field to the Eardrum Presented in Numerical Form,* J. Acoust. Soc. Am. 78, pp. 1120-1123, 1985

[22] Shilling, R.D., and Shinn-Cunningham, B.G. *Virtual Environments Handbook: Virtual Auditory Displays,* K.Stanney (ed), Lawrence Erlbaum, Associates, Inc., 2000

[23] Shinn-Cunningham, B.G., Kopco, N., and Santarelli, S.G. *Tori of Confusion: Binaural Localization Cues for Sources within Reach of a Listener,* J. Acoust. Soc. Am. 107, pp. 1627-1636, 2000

[24] Smith, J.O. *Sampling Rate Conversion Program,* v. 1.7, June 2002, http://www-ccrma.stanford.edu/~jos/resample/Available_Software.html

[25] Smith, J.O., and Gosset, P. *A flexible sampling-rate conversion method*, in Proc. IEEE Int. Conf. Acoustic Speech Signal Processing, vol. 2. pp. 19.4.1-19.4.4, 1984

[26] Telecommunication Systems Laboratory, Department of Information Technology and Microelectronics, KTH, Stockholm, http://vvv.it.kth.se/labs/ts

[27] Wightman, F.L., and Kistler, D.J. *Headphone Simulation of Free-field Listening. I: Stimulus Synthesis,* J. Acoust. Soc. Am. 85, pp. 858-867, 1989

[28] Wightman, F.L., and Kistler, D.J. *Headphone Simulation of Free-field Listening. II: Psychophysical Validation,* J. Acoust. Soc. Am. 85, pp. 868-878, 1989

[29] Wightman, F.L., and Kistler, D.J. *The Dominant Role of Low-frequency Interaural Time Differences,* J. Acoust. Soc. Am. 91, pp. 1648-1661, 1992

[30] Wightman, F.L., Arruda, M., Kistler, D.J., and Wenzel, E.M. *Localization Using Nonindividual Head-related Transfer Functions,* J. Acoust. Soc. Am. 94, pp. 111-123, 1993

[31] Wireless@KTH, KTH, Stockholm, http://www.wireless.kth.se/

[32] Yost, W.A. *Fundamentals of Hearing: An Introduction,* San Diego, CA, Academic Press, 1994

Web sites:

[33] http://interface.cipic.ucdavis.edu/index.htm, CIPIC Interface Laboratory (October, 2004)

[34] http://www.dform.com/inquiry/spataudio.html, The Ultimate Spatial Audio Index (October,

2004)

[35] http://home.earthlink.net/~shillingr/audio.htm, Auditory Displays and Immersive Sound Links (October, 2004)

[36] http://www.ietf.org/1id-abstracts.html, IETF Internet Drafts (September, 2004)

[37] http://www.tecnun.es/asignaturas/Informat1/AyudaInf/Index.htm, Handbooks for the most used programming languages (in Spanish) (December, 2004)

[38] http://www.micosoft.com/windows, Microsoft Windows Home Page (September, 2004)

[39] http://www.linux.org, The Linux Home Page (September, 2004)

[40] http://www.apple.com, Apple's Home Page (September, 2004)

[41] http://www.openal.org, Open Audio Library (October, 2004)

[42] http://www.vrlab.umu.se/research/openalpp/, OpenAL++, An Object Oriented API for Spatial Sound (October, 2004)

[43] http://csd.ssvl.kth.se/csd5/, Communication System Design 2005 course held in KTH, Kista Campus

[44] http://www.careerjournal.com/columnists/workfamily/20030321-workfamily.html, Career Journal, Various Articles related to Human Multitasking

APPENDIX A:

C Code for a Spatial Audio Implementation

```
/*
 * play a set of 16 bit linear stereo samples sampled at 8kHz,
 * in the desired space location,
 * first resampling them up at 44100 Hz, on the selected audio device
 *
 * option -c to choose the type of converter (see Secret Rabbit Code webpage)
 *
 * option -p to select the position of the source [1,5]
 *
 * it uses the full API of the Secret Rabbit Code for resampling
 * see  http://www.mega-nerd.com/SRC/api_full.html
 *
 * the file is split into 20 ms frames = 160 samples per frame and channel
 *
 * lookup tables used for the volume scaling
 *
 * compile with:
 * gcc  c_api_example.c -o api -lsamplerate
 */

#include <samplerate.h>
#include <sys/soundcard.h>

/* the following are needed for stat() */
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define LIM 65536
#define POS 5
#define SHIFTSIZE 36

/* variables for locating process */
int lchdelay[POS]={0,0,SHIFTSIZE,0,0};
int rchdelay[POS]={0,SHIFTSIZE,0,0,0};

float lchvol[POS]={1,0.8,0.6,1,0};
float rchvol[POS]={1,0.6,0.8,0,1};

short int lookupright[LIM][POS];
short int lookupleft[LIM][POS];

SRC_DATA src_data;
SRC_STATE *src_state ;
int channels=2;
int converter;
int position;
int error;

short int* data_in;
short int* data_out;
short int* lbuffer;
short int* rbuffer;
short int* short_output;
float* float_in;
float* float_out;
int pointer;

void createTable(float chvol[POS],short int lutable[LIM][POS] ) {
  int i,j;

  for (j=0;j<POS;j++){
    for(i=0;i<LIM;i++){
      lutable[i][j]=(short int)((float)(i-32768)*chvol[j]);
    }
  }
}
void initSpatial(){
  createTable(rchvol,lookupright);
  createTable(lchvol,lookupleft);
}

void resample (short *input, short *output, int isize, int osize, SRC_DATA src_data,
SRC_STATE *src_state){

  src_data.data_in=float_in;
  src_data.data_out=float_out;

  src_short_to_float_array (input, float_in, isize);

  if ((error = src_process(src_state,&src_data))){
```

```
    printf ("\nError : %s\n", src_strerror (error)) ;
    exit (1) ;
  }

  src_float_to_short_array(float_out,output,osize);

}

int spatialize (short *input, short *leftch, short *rightch, int position, int pointer,
short *outbuff){

  int i=0;
  static int j=0, k=0;

  for(i=0;i<1764;i++){
    if(i%2 == 0){
      leftch[(j+lchdelay[position])%950]=input[i];
      j=(j+1)%950;
    }

    else {
      rightch[(k+rchdelay[position])%950]=input[i];
      k=(k+1)%950;
    }
  }

  for(i=0; i<882;i++){
    outbuff[(2*i)]=lookupleft[leftch[pointer]+32768][position];
    outbuff[(2*i)+1]=lookupright[rightch[pointer]+32768][position];
    pointer=(pointer+1)%950;
  }
  return pointer;
}


int main(int argc, char *argv[])
{
while ((c = getopt(argc, argv, "d:hr:i:c:p:v")) != EOF) {
    switch (c) {
    case 'd':
      dev_flag++;
      dspname=optarg;
      break;

    case 'r':
      sscanf(optarg, "%d", &rate);
      rate_flag++;
      break;

    case 'c':
      sscanf(optarg, "%d", &converter);
      conv_flag++;
      break;

    case 'p':
      sscanf(optarg, "%d", &position);
      pos_flag++;
      break;

    case 'i':
      input_file_name=optarg;
      /* open file for reading in binary mode */
      if ((input_fileP=fopen(optarg,"rb")) == NULL) {
                              fprintf(stderr, "\nError in opening file: %s\n", optarg);
                              exit(1);
      }
      break;

    }
  }



  /* get info from the file */
  if ( stat(input_file_name, &file_stat_buf) == -1) {
    fprintf(stderr, "Unable to stat the input file\n");
    exit(1);
  }

  /* compute the number of (short int) samples in the input file */
  input_file_size = file_stat_buf.st_size;
```

```
  number_of_samples=input_file_size/sizeof(short int);


  /* creating src_state structure for recursive calling  */
  src_state=src_new(converter,channels,&error);

  /* creating src_data for resampler calls  */
  src_data.data_in = float_in;
  src_data.data_out= float_out;
  src_data.input_frames = 160;
  src_data.output_frames = 882;
  src_data.src_ratio = 5.5125;

  /* read sound in blocks of 320 samples= 20 ms * 8 kHz * 2 channels  */
  while ((counter+320)<number_of_samples){

    for (i=0; i <320; i++) {
      number_read = fread(&in,
                          1,
                          sizeof(short int),
                          input_fileP);
      data_in[i]=in;
    }

    resample(data_in,data_out,320,1764,src_data,src_state);
    pointer=spatialize(short_output,lbuffer,rbuffer,position,pointer,data_out);

    if (write(dspFD, data_out,sizeof(short int) * 882 *2) < 0) {
      fprintf(stderr, "Error encountered in writing output samples\n");
      exit(1);
    }
    counter=counter+320;
  }

  src_delete(src_state);
  fclose(input_fileP);
  close(dspFD);
}
```

IMIT/LCN 2005-01

www.kth.se