Movements in Binaural Space: Issues in HRTF Interpolation and Reverberation, with applications to Computer Music

Volume 1 of 2

BRIAN CARTY

PhD Dissertation

NUI Maynooth

Music Department

August 2010

Head of Department: Professor Fiona Palmer

Supervisor: Dr Victor Lazzarini

Table of Contents

Table of Figures

Acknowledgements

Writing these acknowledgements in the depths of my final submission preparations, I naturally feel some relief to reach this stage in the PhD process. However, it is with a sense of poignancy that I will submit this thesis, as it marks the official end of my student-mentor relationship with my supervisor. I feel genuinely privileged to have worked with Victor for the last number of years. I am constantly awed by his level of commitment and generosity. I am also becoming increasingly aware of the influence his professionalism, methodology and enthusiasm has had upon me. In spite of the deep and satisfying understanding of my topic my thesis work has afforded me, I consider the professional, formative, educational and personal relationship I have developed with Victor as by far the most beneficial aspect of the program.

Martha has been a constant source of joy throughout. The PhD process has brought with it moments (often elongated!) of self doubt, isolation and equally elation. These moments were shared patiently and lovingly.

My family home provides an unfaltering refuge of support and encouragement. I am, as always, extremely grateful and forever indebted to my parents and brother.

I also firmly believe in the importance of time away from the books/computer/compiler. Thanks to all who have shared interesting discussions, runs and pints over the last few years.

On a more practical note, I would also like to acknowledge and sincerely thank IRCSET and NUIM for scholarship support. Their financial assistance allowed me to focus completely on my studies. The opportunity to travel to several international conferences and share work with my peers was extremely beneficial. I would also like to commend IRCSET on what I believe to be a courageous and crucial program.

The music department in Maynooth has been extremely supportive. Staff members have consistently impressed and inspired me as both educators and colleagues. Having studied at both undergraduate and postgraduate level, and worked extensively in the department, I would like to commend Professors Gillen and Palmer and all staff on such an exemplary organisation.

Computer Science staff provided generous, enthusiastic and patient DSP support throughout. Thanks also to all who helped me get over the finish line (Mark, Martha, Matthieu, and Sul).

Relevant Publications by the Author

Book Chapters

Carty, B. Binaural Processing: A Sample Application, Boulanger, R. (ed.), The Audio Programming Book, MIT Press, to appear

Conference Papers

* Carty, B. and Lazzarini, V. Multibin: A Binaural Audition Tool, DAFx, 2010

* Carty, B. and Lazzarini, V. HRTFearly & HRTFreverb: Flexible Binaural Reverberation Processing, ICMC, 2010

Carty, B. Design of a Binaural Reverberator, Music & Audio Signal Processing: First Irish Workshop, 2010

* Carty, B. and Lazzarini, V. Binaural HRTF Based Spatialisation: New Approaches and Implementation, DAFx, 2009

* Carty, B. and Lazzarini, V. Frequency-Domain Interpolation of Empirical HRTF Data, AES, 126th Convention, 2009

* Lazzarini, V. and Carty, B. New Csound Opcodes for Binaural Processing, LAC, 2008

Carty, B. New Csound Opcodes for Binaural Processing, Sounds Electric, 2007

Journals

Carty, B. Multi-channel and Binaural Spatial Audio: An overview and Possibilities of a Unified System, Maynooth Musicology, 2, 2009

Carty, B. HRTFmove, HRTFstat, HRTFmove2: Using the new HRTF Opcodes, The Csound Journal, 2008

Carty, B. Artificial Simulation of Audio Spatialisation: Developing a Binaural System, Maynooth Musicology, 1, 2008

Software Development

HRTF Opcodes for Csound:

http://www.csounds.com/manual/html/hrtfmove.html
http://www.csounds.com/manual/html/hrtfmove2.html
http://www.csounds.com/manual/html/hrtfstat.html

Binaural Reverberation Opcodes to be submitted to Csound.

Python MultiBin Application to be made available online.

A number of other publications are being considered.

* included in Volume 2

Abstract

This thesis deals broadly with the topic of Binaural Audio. After reviewing the literature, a reappraisal of the minimum-phase plus linear delay model for HRTF representation and interpolation is offered. A rigorous analysis of threshold based phase unwrapping is also performed. The results and conclusions drawn from these analyses motivate the development of two novel methods for HRTF representation and interpolation. Empirical data is used directly in a Phase Truncation method. A Functional Model for phase is used in the second method based on the psychoacoustical nature of Interaural Time Differences. Both methods are validated; most significantly, both perform better than a minimum-phase method in subjective testing.

The accurate, artefact-free dynamic source processing afforded by the above methods is harnessed in a binaural reverberation model, based on an early reflection image model and Feedback Delay Network diffuse field, with accurate interaural coherence. In turn, these flexible environmental processing algorithms are used in the development of a multi-channel binaural application, which allows the audition of multi-channel setups in headphones. Both source and listener are dynamic in this paradigm. A GUI is offered for intuitive use of the application.

HRTF processing is thus re-evaluated and updated after a review of accepted practice. Novel solutions are presented and validated. Binaural reverberation is recognised as a crucial tool for convincing artificial spatialisation, and is developed on similar principles. Emphasis is placed on transparency of development practices, with the aim of wider dissemination and uptake of binaural technology.

Chapter 1. Introduction

1.1 Context

Audio spatialisation is a multi-faceted, swiftly-developing and high-potential research field. Essentially the discipline aspires to recreate accurate spatial listening environments, ideally with a fine degree of control. Even a brief review of the vast literature offers an insight into the numerous relevant aspects of the topic. When attempting to artificially recreate spatial environments, several approaches can be taken. A pragmatic, psychophysical approach may focus on subjective listening tests to examine the capabilities of the human auditory system. The discipline of Signal Processing is perhaps more concerned with the implementation of optimal processes for creation and delivery of spatial audio. Computer Science researchers may focus more on system and algorithm implementation. The hardware used for signal capture and reproduction also constitutes another potentially self-contained research field.

All of the above topics offer challenges and open research questions which merit abundant independent research. The approach taken here is more holistic, intending to proffer more universal solutions. The control offered by headphone-based audio and potential of binaural processing focuses the work, however, loudspeaker approaches are also considered in a more integrated approach than the literature generally offers. The specific challenges of headphone based artificial spatialisation thus swiftly come to the fore. Typically, headphone systems use binaural technology: exploiting Head Related Transfer Functions (HRTFs), which describe how the auditory system processes sound from particular locations. Imposing HRTFs on a non-spatialised source adds the spatial properties of the HRTF.

The intricacies and development challenges of binaural processing constitute the majority of this thesis. A wide-reaching literature review, considering the cross-disciplinary potential approaches above informs the process. Particular rigour is applied to areas of the literature that rely on assumptions and potentially fallible, but generally employed processes. Specifically, the assumption that the HRTF can be represented as a minimum-phase system and a linear delay, and threshold based phase unwrapping are reconsidered.

This reappraisal of the topic informs the development of a comprehensive suite of spatialisation software. Although mathematics, physics and signal processing inherently monopolise this process by necessity, the goal of harvesting the creative potential of binaural processing is constantly considered. Indeed, it is the creative use of HRTF processing that inspires the development of binaural reverberation tools; HRTF processed audio appears internalised and somewhat unnatural if not augmented by environmental processing.

This thesis thus attempts to offer a comprehensive account of binaural audio, readdressing several core attributes of the discipline. Specifically, novel approaches to HRTF representation and interpolation are offered. This work then informs the development of HRTF based environmental processing, which, in turn, is used in an application that unifies headphone and loudspeaker spatialisation. The challenge of artificial audio spatialisation is thus addressed in a complete manner, placing the specifics of HRTF processing in the context of the broader, cross-disciplinary literature.

This introductory chapter will briefly discuss the wider fields of relevance to the work. Significant (mainly didactic) references will be offered; more detailed references will be cited where more appropriate in later chapters. The chapter concludes with a discussion of the contribution of the thesis.

1.2. Background

1.2.1 Sound Waves

Sound is essentially a mechanical disturbance transferred through a medium. It travels as a longitudinal wave, transferring energy from molecule to molecule. It can thus be quantified in terms of velocity. In a typical scenario, the medium in question is air, but this is by no means exclusive. As energy is required to disturb the medium’s molecules, the intensity of the disturbance is reduced with distance. The disturbance will generally emanate three-dimensionally, thus the energy dissipates. This reduction is a frequency dependent process (high frequency will inherently loose energy more quickly).

An understanding of sound is often presented from a cumulative point of view. A repeating pattern of molecular disturbance will imply a periodic wavefront and lead to a pitched sound. The simplest form of periodic sound source is generated by a sinusoidal oscillation. More complex, ‘real world’ sounds can be thought of as sums of various sinusoids, at different frequencies and amplitudes and with different temporal evolutions. For more details, see [80 and 59].

1.2.2 Sound Perception

Psychoacoustics can be defined as the study of sound perception. As discussed, sound is an intrinsically physical phenomenon. The molecular displacements involved arrive at the eardrum, after being ‘processed’ (in the natural sense of the word) by the external ear. The various reflections occurring due to the complex shape of the pinna (see stylised schematic, figure 1.1) and enclosed nature of the auditory canal affect the incoming sound in a frequency dependent way. The eardrum transfers the air displacement to a mechanical vibration of the small bones in the middle ear, which amplify the vibrations. This energy creates waves in the incompressible fluid contained in the Cochlea (in the inner ear). The Basilar Membrane is displaced by these waves and reacts at different locations to different frequencies (see the straightened Cochlea in figure 1.2). Hair cells on the membrane trigger nerve firings in the Organ of Corti, which sends the signal to the brain for higher processing. The Basilar Membrane thus acts as a (logarithmic) frequency analyser. The resolution of this analysis is dictated by Critical Bands, which define the difference in frequency required for sinusoidal sources/components to be perceived as separate. Critical Bandwidth (which is again frequency dependent) can be modelled using Equivalent Rectangular Bandwidth (ERB) filters (ERB essentially relates filter centre frequency with effective critical bandwidth).

One of the most pertinent points of note with regard to psychoacoustics is the subjectivity of the area. Indeed, research in the field typically involves subjective testing to arrive at statistical norms. Further detail on psychoacoustics can be found in [80, 182, 40 (some interesting discussions of the topic), 143 (more experimental details) and 27 (details on perception of complex auditory scenes)]. Of particular interest in terms of this work is the perception of spatial audio, which will be discussed in the next section.

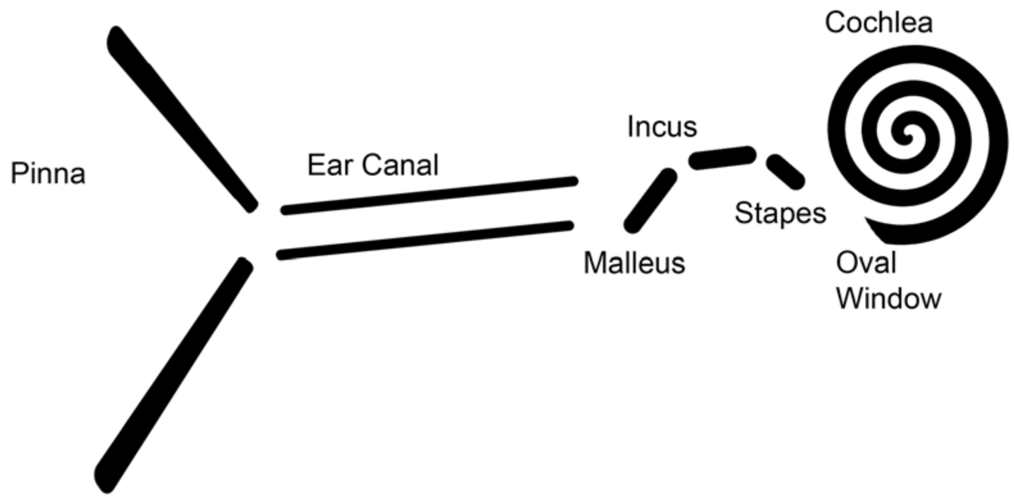


Figure 1.1: Ear schematic

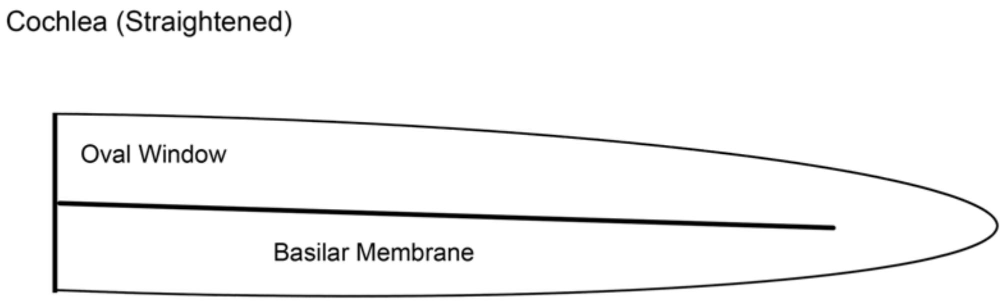


Figure 1.2: Cochlea (straightened)

1.2.3 Sound in Space

In order to recreate spatial auditory scenes, an understanding of the spatial hearing system is crucial. Environmental processing is also typical of common listening scenarios, so should also be addressed. This section discusses these topics, as well as offering an introduction to artificial spatialisation.

1.2.3.1 Spatial Hearing

Sound localisation refers to the ability of a subject to establish the position of a sound event in their sonic environment. The binaural nature of the auditory system provides the primary cues used to localise a source. If a sound source is located to the right of a listener, the right ear will receive the source signal slightly before, and at a slightly higher level than the left ear. These signal discrepancies are used to locate the source and are labelled Interaural Time and Intensity Difference respectively (ITD and IID). The auditory system performs extremely accurately in optimal conditions, with localisation blur of 1 degree reported [93] (more real world scenarios perform less accurately).

Generally, ITD provides salient localisation cues at lower frequencies and IID at high. As time differences are eventually understood as phase differences (IPDs), which are ambiguous at higher frequencies due to the periodic nature of phase, ITD breaks down above 1500 Hz. Intensity based differences are prone to diffraction effects, which are more notable at lower frequencies; IID is therefore a more reliable cue at higher frequencies.

Monaural information can also provide (secondary) localisation cues. Pinna filtering can aid in localisation in the median plane (where interaural cues will be minimal). Sources at the same angle from the subject's median plane originating from the front and back will imply very similar interaural differences. Extrapolating to three dimensions (considering elevation), several source locations will imply the same interaural cues in the so called 'cone of confusion'. Spectral differences can be helpful in this scenario.

In reverberant environments, the Precedence Effect, or more self-descriptively the Law of the First Arriving Wavefront states that the auditory system will localise according the first arriving wavefront in a reflective environment (or a scenario involving any swiftly successive correlated sounds). This spatial integration occurs up to 30 msecs. Early reflections in reverberant environments will thus not excessively degrade localisation ability. Sound localisation abilities and restrictions have consistently inspired and informed this work. More detail on the topic can be found in [80, 21 and 101].

1.2.3.2 Spatialisation

Sound spatialisation refers to the artificial placement of a sound source at a particular location within a subject's spatial environment. Solutions to and applications of sound spatialisation constitute the body of this work, and will thus be discussed in detail in later chapters.

Several approaches can be taken to sound spatialisation. Broadly, two general techniques can be identified: loudspeaker and headphone methods. Both are discussed in this work, although focus is placed on headphone based binaural algorithms. An introduction to binaural spatialisation is available in [14] where the headphone approach is advocated for control reasons.

1.2.3.3 Head Related Impulse Responses and Transfer Functions

Head Related Impulse Responses (HRIRs) describe how sound from a specific location is altered from source to tympanic membrane. When represented in the frequency domain (discussed below), HRIRs are known as HRTFs (and will be hereafter referred to as HRTFs unless specifically in reference to the time domain). Each point in a subject's spatial environment will be represented by a unique pair of HRTFs (1 each for the left and right ear). All localisation cues mentioned above will be encompassed in these HRTF pairs.

Generality of the HRTF is achieved as it is a response to a frequency rich/impulsive signal, so detail is available for frequencies of interest. The HRIR is obtained using binaural recording techniques to capture how the ear responds to this impulsive signal (the *impulse response* of the ear). A comprehensive review of binaural, HRTF-based spatialisation is given in chapter 2; for an introduction, see [14].

1.2.3.4 Environmental Acoustics

Of particular interest in the context of this work is how sound behaves in an enclosed space (for example a concert hall). A sound source in such an environment will arrive at listeners not only directly from the source, but also after reflecting off the various boundaries in the environment. This phenomenon is known as reverberation and remains a topic that merits vast amounts of research. These reflections will tend to be specular when the wavelength of the source sound is smaller than the boundary. In the opposing case, larger wavelengths will behave in a more diffuse manner. Typical sound sources of interest may have components which behave in a partly specular, partly diffuse manner.

Reflective surfaces will also absorb some of the source sound's energy, once again, in a frequency-dependent manner. Inherently, this reverb can be categorised in terms of the time it takes to dissipate: the reverb time, or T_{60}; the 60 here referring to a reduction in 60 dB. Reflections can trace recursive paths, leading to emphasis being afforded to particular frequencies. This modal response can characterise a listening space and can dominate at low frequencies.

1.2.4 Digital Audio

Digital encoding offers a flexible, efficient and accurate means of storing and processing audio. In this work, audio will be manipulated as Pulse-Code Modulated (PCM) digital signals. In PCM form, audio is stored as samples representing the amplitude of a signal at given points in time. Samples are taken at regularly-spaced intervals, defined by the sampling period, which is the inverse of the sampling rate. The sampling rate of a signal dictates its frequency resolution. Specifically, frequencies up to half the sampling rate (also called the *Nyquist Frequency*) can be represented accurately. The popularised sampling rate of 44.1 kHz thus fits well with the human optimal hearing range of 20 Hz – 20 kHz.

Audio or indeed any type of signal can be represented in several ways. Traditionally, audio is viewed, edited, processed and auditioned in the time domain (which represents how audio energy causes pressure changes in the medium over time, as above). However, the frequency domain can provide more useful insights into the properties of the signal in certain scenarios.

1.2.4.1 The DFT and the STFT

Frequency domain signals are sampled in frequency as opposed to time. The significance of this representation is highlighted by the frequency analysis basis of the auditory system. Individual sinusoidal components of a signal can be examined, and their magnitude and phase can be extracted in the frequency domain. The former quantifies the relative strength of the signal at each frequency in the analysis, the latter, the phase/starting point of the component relative to a full cycle.

The Fourier Transform and its inverse allow transparent transfer from one domain to another. When considering discretised digital audio, the Discrete Fourier Transform (DFT) is appropriate. The spectral representation offered by the DFT is inherently time invariant. The number of samples analysed in each transform is defined by the analysis window. Magnitude and phase values (derivable from the rectangular form output: the complex sinusoid at each bin frequency) essentially represent the parameters of the sinusoids required to recreate the analysed signal. The inverse transform combines these component frequencies to transparently return to the time domain. The interaction of the various relative frequencies, magnitude and phase of the components reintroduce temporal information. A large analysis window will lead to higher spectral resolution, but, by design, more temporal averaging.

The DFT outputs spectral details from –Nyquist Frequency to Nyquist Frequency. Therefore, an analysis of 128 samples will yield 128 complex numbers (essentially 256 values). As audio is a real signal, negative frequencies are complex conjugates of their positive counterparts. Optimisation can thus be achieved by only storing the positive values; an output of half the number of complex numbers, plus 1, as both 0 Hz and the Nyquist Frequency are required. However, both of these frequencies will be represented by purely real numbers. Therefore, they can both be stored in the memory required for one complex number. Thus the input and output buffers used in the transform can be the same size. The Fast Fourier Transform (FFT) further optimises the DFT.

The Short-Time Fourier Transform (STFT) offers additional development of Fourier theory. Dynamic signals are dealt with by windowing the input and output signal and processing the input in overlapping sections. The STFT can follow dynamic spectra and reduce smearing caused by rectangular windowing (essentially by employing non-rectangular windows such as inverted-raised cosine windows).

1.2.4.2 Digital Filters

HRTF processing can be seen as a filtering operation in binaural spatialisation. In Digital Signal Processing (DSP), filters are used to process signals, and are characterised by their respective filter equations. A generalised filter equation is offered in equation 1.1.

$$y(n) - \sum_{k=1}^{N} b_k y(n-k) = \sum_{k=0}^{M} a_k x(n-k) \quad (1.1)$$
,

where $y(n)$ represents output (and its *N* delays, where *N* is finite) and $x(n)$ input (and its *M* delays, where *M* is finite). A filter's transfer function describes its output, in terms of its input.

$$Y(z) = H(z)X(z), \quad (1.2)$$

where Y represents filter output, X input and $H(z)$ transfer function. $H(z)$ can also be shown as a ratio of polynomials. The numerator relates to input delays (feedforward); the coefficients and denominator, output delays (*z* is a complex variable).

$$H(z) = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 - \sum_{k=1}^{N} b_k z^{-k}} \quad (1.3)$$

Filters thus combine delayed versions of signals. Finite Impulse Response (FIR) filters delay copies of the input to the system. Infinite Impulse Response (IIR) filters have feedback elements, using delayed copies of the output of the system. Filters are elegantly described on the unit circle in the complex z plane. Here, z refers to a complex variable, which maps frequency onto the unit circle (the transfer function's domain is the z plane: $H(z)$). The top half of the unit circle represents frequencies f from 0 Hz to the Nyquist Frequency, $e^{j\omega}$ where $\omega = 2\pi f / sr$ and *sr* is the sampling rate (in Hz). The z plane is illustrated in figure 1.3, below.

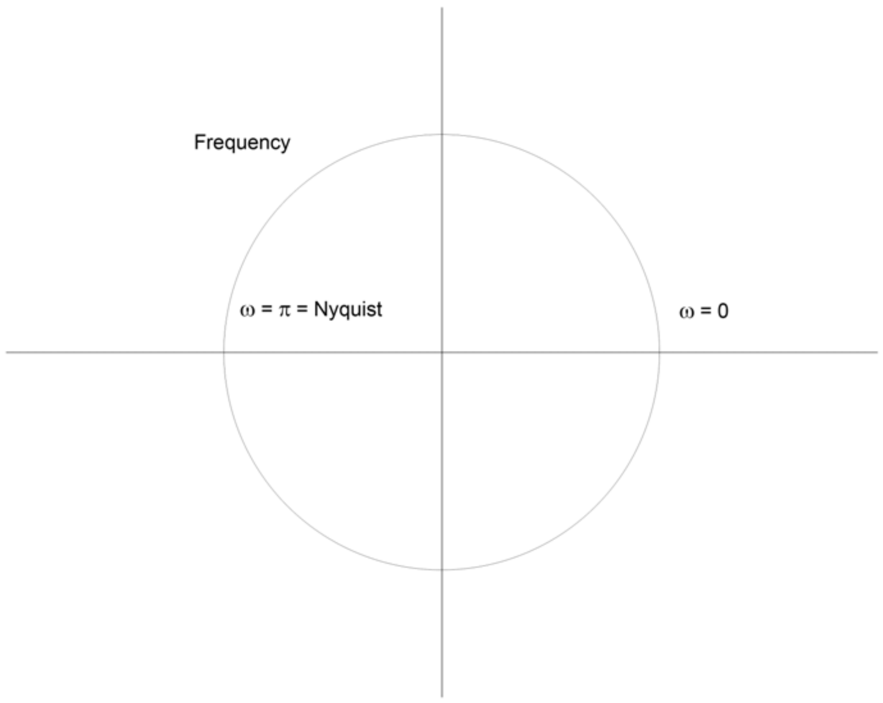


Figure 1.3: The z plane

As shown in equation 1.3, the transfer function of a filter can be put in terms of a ratio of two complex polynomials. Zeros of the numerator correspond to the transfer functions zeros and the zeros of the denominator correspond to the transfer function poles. These can be plotted as specific locations in the z plane. The magnitude response of a filter at a particular frequency can thus be inferred by the location of poles and zeros on the unit circle representation. Essentially, zeros are locations in the z plane which reduce the output magnitude of adjacent input frequencies (if a zero is on the unit circle, output will theoretically be 0), measured on the unit circle, $z = e^{j\omega}$. Poles, conversely, result in a boost to adjacent frequency inputs (and a theoretically infinite output to poles on the unit circle). Mathematically, zeros can be thought of as the roots of the numerator, and poles the roots of the denominator of the transfer function. Locations on the plane near to a zero will result in lower magnitude response than those further away. Conversely, magnitude response at locations near to a pole will be greater than those further away.

This raises an important concern. If poles lie outside the unit circle, the magnitude of the pole is greater than one, which can lead to stability issues with the system. A system must therefore have all poles within the unit circle to be stable. A number of comprehensive texts on the topic of digital audio signal processing are available, including [194, 144, 159 and 203].

1.2.4.3 Convolution and HRTFs

Convolution can be used to impose the characteristics of one signal upon another. Perhaps the most intuitive example of this lies in the domain of artificial reverberation; a room's impulse response can be imposed on an arbitrary signal using convolution. Similarly, a HRTF can be imposed upon an arbitrary signal. Convolution can be a costly process when performed in the time domain. The convolution of the signal $f(n)$ with the impulse $g(n)$ can be defined as:

$$\sum_{m=0}^{n} f(m)g(n-m) \tag{1.4}$$

Thus time domain convolution is essentially a *translate and scale* operation, which can become extremely computationally expensive for longer impulses [195 (further insight is offered in chapter 3 with respect to implementation)].

Time-domain convolution is equivalent to frequency-domain multiplication and vice versa [159]. Therefore, the convolution operation can take advantage of the FFT. A much more efficient process is thus implied, particularly as impulse lengths get longer. HRTF spatialisation can thus be understood as a spectral process; analyse the source signal and the HRIR, and alter the source in the same way the external auditory system does. The magnitudes and phases of the input signal are boosted/attenuated and delayed in accordance with the HRTF. Typically, an overlap-add convolution technique is used to segment the signal into appropriate buffers. The output of the convolution will be longer than the input signal (its length will be that of the input + that of the impulse - 1). Overlapping sections will be added to the next buffer output. If dynamic HRTF processing is desirable, the STFT may be required to remove any discontinuities as HRTF filters change from process to process.

1.3 Thesis Contribution

A comprehensive review of binaural spatialisation is offered in chapter 2, which concludes with a detailed critique of the assumption that HRTFs can be modelled as minimum-phase plus linear delay systems. Conclusions motivate a re-evaluation of traditional approaches to binaural spatialisation.

Chapter 3 introduces two novel approaches to dynamic HRTF spatialisation. Again, some commonly held assumptions in the literature are challenged in a detailed study of phase unwrapping. The conclusions drawn serve to formalise the suggested novel algorithms.

Implementation of these algorithms is discussed in chapter 4. A significant portion of this thesis is dedicated to the discussion and dissemination of the many nuances involved in efficient implementation of often complex signal processing techniques. Objective and subjective validation of the theory and implementation of the novel algorithms is also offered.

The potential of the novel algorithms is explored in chapter 5. Firstly, a review is offered of artificial reverberation, from a binaural perspective. A reverberation framework is then developed. A re-appraisal and update of classic and more modern techniques is offered as a flexible, efficient solution.

Application of the developed techniques is discussed in chapter 6, as development culminates in a software application which aims to integrate the (generally mutually exclusive) loudspeaker and headphone based approaches to sound spatialisation. The developed tool allows for the versatile and dynamic audition of any multichannel loudspeaker/multiple sound source setup in headphones.

Concluding comments and possibilities for further work are offered in chapter 7. A hard copy of more relevant publications and code is offered in a separate volume of appendices. Additional material is available on the accompanying CD-ROM.

1.4 Conclusion

This chapter introduced the context of the novel aspects of this thesis. Essentially, a reappraisal of HRTF based spatialisation will be offered, including a rigorous review of generally accepted practices. New methods for HRTF interpolation will be suggested and validated. Binaural environmental processing will also be considered. An integration of headphone and loudspeaker techniques will essentially practically apply the techniques developed.

A background to the most significant disciplines relevant to the thesis is also offered. Finally, contributions of the various sections of the thesis are listed.

Chapter 2. Binaural Audio: Literature Review

2.1 Introduction

HRTF based research has essentially mirrored the relatively recent growth in DSP development and research, spurred on by ever faster processing power and the burgeoning field. Physical investigations of the effects of outer ear anatomy drove initial HRTF insights [for example, 135], which were followed by early attempts to utilise the HRIR as a spatialisation tool [for example, 102]. The interpolation of HRTFs thus swiftly became a pertinent research topic.

Several different approaches to binaural research result in developments in the area, with two forerunners. The first involves studies in acoustics and psychoacoustics, which require accurate models of binaural hearing, and, in turn, provide the raw material necessary for binaural system development. The second is the more computational discipline of binaural system development, with various motivational applications, including aeronautics simulation systems, virtual reality, teleconferencing and gaming. Since the initial realisation of the potential of binaural systems, research has tended to trend towards ever more efficient ways of representing the HRTF, minimising processing and storage. Here, a more empirical approach is suggested after considering the literature. Results (see section 4.3) indicate the success of this approach.

A review of the vast literature on the topic follows, which aims to discuss in some detail the more prevalent approaches to HRTF processing and interpolation.

2.2 Literature Review

Perhaps an appropriate place to start in any literature review on the topic of binaural system development, is Begault's '3d Sound for Virtual Reality and Multimedia' [14]. This publication puts the problem into perspective, providing an eminently readable account of the challenges involved in the area. Begault raises an apt point at the offset, highlighting the often confusing or even apparently contradictory terminology used in this relatively young field of research (for example 'virtual reality', would perhaps be better served by the term 'virtual environment').

Another insightful point made here is that although vision is unquestionably dominant over hearing from a sensory point of view (a point often mentioned in the literature), audio possesses a unique ability to inspire imagination and enhance sensory experiences. It is the relative lack of sensory data that an audio-only presentation constitutes that allows it to engage higher-level cognitive processing. Audio is therefore deemed essential in multi-modal virtual environment processing. The book discusses the many design issues, and candidly describes the challenge for the system designer as a 'juggling act', referring to the desired balance between accuracy and processing costs.

2.2.1 Acquiring HRTFs

Using HRTFs in a binaural processing system assumes availability of an accurate HRTF dataset. Measuring the HRTFs of a human subject can be time consuming and difficult, with a high degree of accuracy required. Typically, a subject sits still in an anechoic environment. A signal is then played from a specific angle and the response captured by microphones close to the eardrum/at the entrance of the ear canal [7].

The signal used can be an impulse, or for better signal to noise ratio (SNR), a longer signal such as a maximum length sequence/Golay sequence or swept sine. The impulse response in these cases can be derived using de-convolution. The necessity for precise equipment, a patient, static subject and expensive anechoic environment highlight the impracticalities involved. Equalisation of all equipment must also be considered, adding another step in the process, increasing potential for impulse response contamination. This applies to both HRTF acquisition and reproduction in a potential spatialisation system (specifically headphone response). In a related matter, acoustic coupling with various headphone styles for reproduction are discussed in [191]. Cavities created by circumaural headphones, for example, can alter the frequency response at the eardrum.

In [105], the authors state that dataset measurements typically take 1.5 hours. Recently, in [18], an automated HRTF measurement system is suggested. A user moves his/her head in line with an LED lighting system. Golay codes are used as source signals, and the resulting impulses are stored as minimum-phase signals (interaural delay is calculated using energy onset detection). Again, the authors highlight the many issues involved with regard to apparatus interference. This measurement technique is used in the SLAB application [224].

Conveniently, a number of databases of pre-measured HRTFs have been made available by the various research teams involved. The MIT dataset is used here [142]. Measurements were made in an anechoic chamber in 1994, using maximum length techniques. The KEMAR (Knowles Electronics Mannequin for Acoustics Research) dummy was used [31]: a manikin designed using averaged human measurements (the principal source used was 4852 subject measurements). In [31], details are given on the care taken in design of such an instrument and the subtleties involved that can alter the ear response. The symmetry of the manikin allows for two different datasets to be captured in one measurement process by using different pinnae. Only one half of the dataset is required, as, for example, the response for the left ear at 90 degrees is equal to that of the right ear at 270 degrees. To remove the response of the measurement system and the ear canal, free field (with respect to a particular location) or diffuse field (with respect to an average over all directions) equalisation can be applied [93]. A diffuse field equalised dataset is made available (thus non-directional data is removed), truncated to 128 samples (removing the delay caused by the distance from the speaker and system latency).

The CIPIC database is also freely available [4]. In contrast, this database consists of measurements from individual subjects. 1250 measurements were made for each ear, with 27 anthropomorphic measurements for each subject. Attempts to correlate anthropomorphic measurements had mixed success. For example, very little correlation was found between head height and pinna height. From the point of view of HRTFs, ITD and head width correlate strongly. The LISTEN database also offers individual HRTF sets [124].

Another approach to HRTF representation is the more functional spherical head model. In [3], the authors set out to derive the most accurate radius for a spherical head model. The literature suggests 8.75 cm, but is variable. They use Kuhn’s findings [108] that a Woodworth Model (a spherical-head geometric model) [228] fits empirical high frequency ITD well, and that low frequency can be relatively increased. Their study is motivated by the fact that a spherical head is often used in binaural research as a simple model, so accuracy of head radius theoretically improves results.

In [5], the authors elaborate on the difficulty in empirically measuring HRTFs, pointing out that low-frequency response is difficult to capture, due to the windowing of the response and the physical size of the loudspeaker. Rectangular windowing of the response is a common practice, to remove reflections (even good anechoic rooms exhibit some low-frequency reflection, also there may be reflections from the apparatus) and reduce the length of the impulse response. Accurate low-frequency output requires a relatively large speaker, which can be awkward in a situation where a movable, flexible emitter is needed. In these scenarios, HRTF processing can sound unnaturally thin. The spherical-head model can provide good low-frequency response, but suffers from the omission of torso reflections (as well as the pinna, as discussed in [37]). Torso reflections are considered in the model presented in [5]; they cause a combing effect, which begins as low as 600 Hz for elevated sources. The authors do concede that the resulting HRTF can only be described as ‘primitive’, as it does not consider non-linearities in the head shape, and more significantly, omits the pinnae. Therefore high frequency detail is not addressed. An interesting solution is suggested, whereby a HAT (head and torso) model’s magnitude spectrum is used at low frequencies, and empirical HRTF magnitude for higher frequencies. The HAT phase spectrum is used throughout, as high frequency phase information is less significant.

A spherical head model is used in [55] to investigate range dependence of the HRTF, concluding that ITD does not change strongly over distance, but IID does. These changes in IID become significant for ranges less than 5 times the head radius. In [7], the authors again use a spherical head model in an investigation of the contralateral (further from source) HRTF (as opposed to the ipsilateral, which is nearer to the source). Interestingly, in analysing the contralateral HRTF, the authors conclude that it is a complex function which ‘frequently exhibits non minimum-phase’ traits (the significance of minimum-phase will be discussed later in this chapter). Ultimately, the study models the contralateral HRTF as the appropriate ipsilateral HRTF low-pass filtered and delayed by the appropriate amount (in accordance with ITD). A 5 degree error result (relatively good) is reported. Difficulties caused by the contralateral HRTF are also discussed in [37]: it has potential for low SNR due to lower signal level and possible measurement noise.

Although there clearly are difficulties involved in acquiring HRTFs, the empirical approach is generally favoured. The time, expense and potential pitfalls involved in making HRTF measurements imply the desirability of a generalised HRTF dataset. The MIT dataset, discussed above, is an attempt to provide such a solution.

2.2.2 Individualised Datasets

In [221], the use of non-individualised HRTFs was investigated. In reviewing the literature, the authors discuss studies which illustrate that sources spatialised with individualised HRTFs (in headphones) perform comparably to free-field sources (e.g. loudspeakers), with some minor degradation: a small increase in front/back reversals and a decrease in elevation accuracy. The subjective tests performed use the HRTFs of a ‘good’ listener/localiser (which raises an interesting paradigm: can listening through the ears of a good localiser teach and improve the localisation capabilities of a bad localiser? This is not as relevant an issue in the KEMAR, averaged set). Noise bursts are spatialised and responses are analysed. This often-cited study concludes that the more subtle elements of spatial hearing are more greatly affected by using non-individualised HRTFs, i.e. the finer spectral details which provide front/back and elevation cues.

Analysing this study a little more deeply helps to place auditory localisation in perspective. Elevation judgements caused particular difficulties. Four subjects (of the total 16) had difficulties with HRTF elevation judgements, but two of these also had difficulties with free-field elevation judgements. As well as highlighting the individual nature of localisation ability, this result also exposes the relative weakness of vertical-plane localisation.

Reversals were also discussed in detail in this study. Due to the cone of confusion, sources that are spatialised to the front of the listener are often perceived from the back (a common occurrence, partially due to the fact that the listener expects to see a source in front of them). Typically, reversals from front to back occur, but the opposite were occasionally observed in this study. Also, the previously neglected up/down reversals were also encountered here. When these reversals were accounted for, it was concluded that non-individualised HRTFs provide a useful tool for binaural processing. The study recognises that training, as well as further interaction, such as a multi-modal system, may improve results. Using a more generalised HRTF dataset, such as that derived from KEMAR may also help. As mentioned in [14], reversals are corrected for in most studies. The lack of a visual stimulus as the cause of reversals is also postulated in [221]. Also, the use of head movements to reduce reversals is highlighted.

A more recent study [19] which looked at the 3 main suggested improvements to binaural HRTF systems: head tracking, individualised HRTFs and reverberation processing found no significant difference between using individualised HRTFs and those of a dummy head. It is, however, important to highlight that this study used only speech as a test source (a wider band source may reveal difficulties with the non-individualised data; equally, depending on the desired application, speech sources may be the primary audio source, so non-individualised HRTFs may be ideal).

In [214], the authors conclude that using individualised HRTFs offers improved presence over non-individualised results. Presence is an important factor in virtual environments, providing a more convincing spatial scene. Efforts to provide individualised HRTFs without the need for measurements have also been documented. In the approach presented in [235], a photo of a subject’s external ear is analysed. An attempt is then made to match this data to the closest dataset available in the CIPIC database [4], which, as discussed above, includes anthropomorphic data. The user marks features on the photo (as well as providing a reference length) and a best fit approach is taken. Interestingly, due to potential (/common) lack of symmetry, each ear is processed separately. In this study, a HAT model is used for low frequencies, as above. The authors report that the HAT model works well, but individualisation is less successful.

In another approach [176], a 3D mesh of the subject’s ear is used. A ray-tracing algorithm is then employed to model HRTFs. This work, although preliminary, is promising for HRTF individualisation.

2.2.3 HRTF Data

Once acquired, HRTF datasets typically constitute the primary component of a binaural artificial spatialisation system. A large portion of HRTF based research, particularly in the DSP domain, has focused on analysing the data in order to reduce it from a perceptual, processing and storage point of view. The majority of implementations use a minimum-phase plus delay approach to HRTF representation. That is, the HRTF is broken into a minimum-phase system and an all-pass system. The all-pass system is assumed to have a linear phase response, and is thus interpreted as a pure (frequency independent) delay. Briefly, the main benefits of this decomposition are: phase spectra can be derived from magnitude spectra in minimum-phase systems; also, minimum-phase systems imply a compact filter. This assumption is discussed in detail later in this chapter, where a critique is also offered. The interpolation of HRTF data is a topic that has motivated an even greater body of research. Essentially, the main aim here is to accurately portray non-measured points (the dummy head/subject measurements mentioned above are discrete; all locations cannot be considered). The minimum-phase plus delay approach is prevalent in both HRTF data analysis and interpolation, to the point that studies that *do not* use the approach are highlighted, where significant, below.

2.2.3.1 Analysis and Representation

Approach	**Data Pre-processing Requirements**	**Relative Storage Requirements**
PCA [105]	Minimum-phase transform, analysis to find basis functions	Low
Spherical Harmonics [57]	Spherical harmonic decomposition, ITD processing	Low
SFRS [38]	Minimum-phase transform, representation preparation	Standard
Psychoacoustic Smoothing [29]	Complex filter derivation	Low
Current Work	None (discussed below)	Standard

Table 2.1: Various approaches to HRTF data analysis

In [105], a Principal Component Analysis (PCA) approach is taken. In the analysis, the authors found 5 basis functions which can account for 90% of the magnitude spectra (remembering the minimum-phase approach, as above) of the HRTFs of 10 measured subjects. The work draws an analogy to Fourier analysis when describing the method: PCA analyses the HRTF magnitude functions in the same way as the

Fourier Transform analyses a signal; PCA outputs a set of basis functions, the FT outputs a set of sinusoidal components. Each magnitude spectrum is theoretically reconstituted by combining the resulting basis functions with appropriate weights. The study also subjectively evaluated the approach. PCA-based HRTFs performed well when compared with empirical HRTFs, providing a reported 'adequate' approximation. Interestingly, the first basis function appeared to deal with elevation, while the other 4 higher frequency ones deal with front-back and horizontal-plane localisation, broadly.

A continuous, functional representation of HRTFs as spherical harmonics up to degree 17 is presented in [57]. The spherical-harmonic paradigm is also used to decompose the soundfield in *ambisonics*: the multichannel analysis, storage and reproduction solution. The goal here is a continuous measurement to avoid discretisation. Again, the process is analogous to the sinusoidal FT; in this case, the analysis produces harmonics on a sphere. The study reports good results, particularly when the analysis is performed in the frequency domain.

These decompositions of HRTFs into functional representations are reviewed and discussed in [120]. The advantages of decomposition to a functional representation are stated explicitly: compact HRTF representation and ease of interpolation of HRTFs. PCA, Independent Component Analysis (ICA) and spherical harmonics are discussed. Another benefit of the functional approach is that every source in a multiple source scenario can be processed with the same filters, with different gain factors, so computation costs are reduced.

A more representation-focused approach is offered in [38], from the larger scale [36]. The concept of Spatial Frequency Response Surfaces (SFSRs) is introduced. Essentially, this involves a colour plot for each frequency bin in the

HRTF analyses. Each plot uses colour height to illustrate the energy at different azimuths (*x* axis) and elevations (*y* axis). Various frequency 'hotspots' are discussed and explained physiologically.

In [30], HRTFs are reduced to critical-band filters, each defining a left and right magnitude and an interaural phase. In a similar study [29] successful smoothing of HRTFs to the resolution of a gammatone filter (which models the cochlea) is reported. Three subjects were asked to listen for differences in musical samples processed with empirical and smoothed HRTFs of varying filter order (further smoothing was implied by reducing the order of the band-pass filters used to emulate the cochlea). This reduction in filter complexity will be further discussed below, in the context of deriving appropriate directional filters for HRTF processing.

2.2.3.2 HRTF Interpolation

Although some studies focus solely on the representation of HRTF data, data analysis is often performed with HRTF interpolation in mind. Therefore HRTF data analysis, representation and interpolation are often intrinsically linked. Interpolation is required in two scenarios: the first involves statically spatialising virtual sources at non measured locations; the second involves moving sources through dynamic trajectories. In the first case, interpolation essentially constitutes improving the spatial resolution of the dataset. Consequences of omitting interpolation may not be severe in this application of HRTF processing. Provided the dataset is relatively dense with respect to the Minimum Audible Angle (discussed in more detail below), simply using the nearest measured HRTF may be sufficient.

The second scenario: dynamic sources moving around a listener's (virtual) spatial environment requires interpolation more urgently. Simply switching the HRTFs being used from one to another as a nearer measured location becomes available may cause an audible discontinuity. This is discussed in [107], which suggests various crossfade methods to avoid this perceptual discontinuity. Objectively, a windowed overlap-add process performed best (compared to square root, cosine and Fourier based fade in/out envelope shapes).

Reference	**Method**	**Comments**
Convolvotron: Early Approach [222]	HRIR mixing, time domain	Combing Effects
Convolvotron: updated [14]	Minimum-phase HRIR mixing, time domain	Improvement on above
[233]	Spherical head model, magnitude interpolation, frequency domain	n/a
SFSRs [38]	Inherent to representation	n/a
[231]	Emphasise spectral differences	Reduces reversals, improves source movement
IPTFs [65]	Ipsilateral and contralateral difference used	n/a
[1]	Invert source-listener: emit source from eardrum	n/a
[211]	Phase Vocoder based	Non-minimum-phase
[76]	Filter root alignment	n/a

Table 2.2: HRTF interpolation summary

Perhaps a natural initial attempt at HRTF interpolation is to simply mix adjacent HRIRs, an approach initially taken in the 'Convolvotron' [222], an early hardware implementation of HRIR based artificial spatialisation (for more detail, see [14]). However, the authors report unnatural combing effects for dynamic sources. Time-domain interpolation of this nature (using empirical HRIRs) is problematic, as adjacent HRIRs may have different inherent delays. Therefore, interpolation of an impulse with a pre-impulse delay (leading to smearing of the two impulses), or cancellation (a pressure trough corresponding with a pressure peak) may occur (illustrated in figure 2.1, below; also discussed and illustrated in [14]).

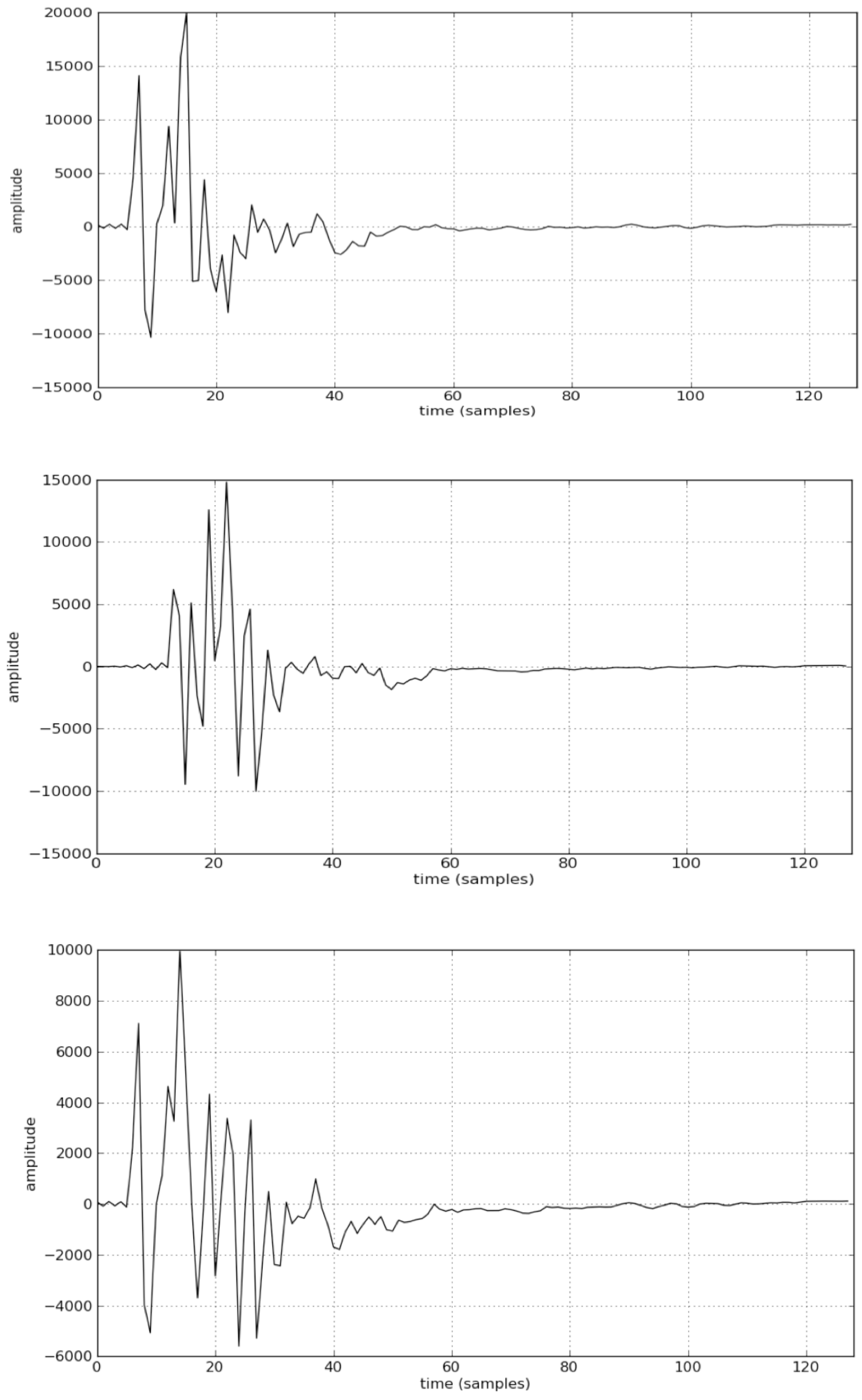


Figure 2.1: Three plots illustrating an extreme example of the problems with time-domain interpolation; a HRIR for 0 degree elevation, 0 degree angle (top) mixed with that at 0 degree elevation, 45 degree angle (middle). The result (bottom) illustrates pressure peak and trough cancellation and time smearing.

This difficulty was observed in the study, and a minimum-phase plus delay approach was suggested to avoid differences in arrival time (see also [14]). This minimum-phase based approach was then implemented in the next generation Convolvotron. Interestingly, in this study, a high perceptual tolerance to interpolation was reported.

Interpolating by combining adjacent impulses can be performed in the time or frequency domain. The frequency domain is more suitable, as it allows representation of data as magnitude and phase values, which correlate to intensity and time differences when considered binaurally. The frequency domain therefore represents a higher level of binaural processing than the pressure fluctuations of the time domain. This is confirmed in [77], which concludes that frequency-domain interpolation is superior. Objectively, if intermediate measurements in a HRTF dataset are removed and replaced with interpolated versions, success of the interpolation algorithm can be inferred by comparing the interpolated and empirical HRTFs. This study ([77]) did not use minimum-phase HRTFs, but did remove the initial inherent delay to avoid the time domain problems discussed above. Interpolation methods used were spherically-weighted four-point interpolation and spherical-spline interpolation. Polynomial-spline interpolation considers the whole dataset, as opposed to just the nearest measured values, so has the potential to be a more comprehensive and accurate approach. However, it is considerably more computationally costly. Spherical-spline interpolation gave best results. In [153], however, linear interpolation performed better than spline interpolation in a minimum-phase-based frequency-domain process when a relatively small number of HRTFs were used in the interpolation procedure.

In implementing frequency domain interpolation, phase values need particular attention. As phase is a periodic quantity, direct interpolation may result in error. This is illustrated in the figure 2.2, below; 2 phase values are shown, 10 degrees and 50 degrees. A phase value for a HRTF half way between these measured values will be assumed to be 30 degrees in a linear interpolation scenario. However, the 50 degree phase value may actually imply 410 degrees, i.e. 50 degrees plus 1 full cycle. It is not clear how this phase interpolation issue is dealt with in the above study [77].

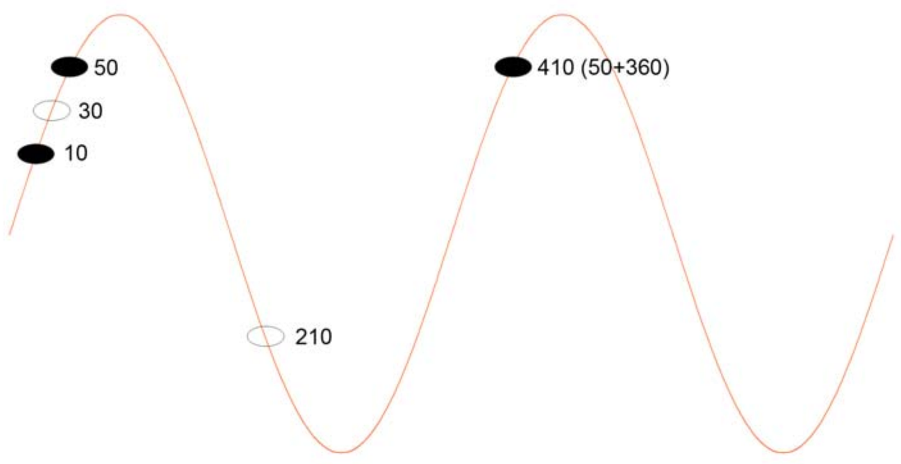


Figure 2.2: Phase interpolation

This flawed nature of phase interpolation is discussed in [233], which notes that a highly populated dataset can minimise this problem for low frequencies, in accordance with a Nyquist criterion (the same authors describe the HRTF interpolation problem as an ‘open research question’ in [234]). The more complete solution suggested in forthcoming chapters is motivated by a desire for accuracy across all frequencies. The apparently appropriate solution of phase unwrapping is also discussed in forthcoming implementation chapters, which conclude that phase unwrapping cannot be deemed an infallible approach.

Considering the arrival time in empirical HRIRs and interpolating in the time domain is examined in [130]. Interestingly, in this scenario, a linear interpolation algorithm performed better than spline and DFT-based (essentially over sampling using vectors of every *i*th HRIR sample) algorithms.

In a study of the often neglected median plane [154], objective tests (on minimum-phase, magnitude-spectrum interpolation) suggest spline interpolation is best for sparse datasets and linear for more densely sampled data.

Required spatial resolution of the empirical dataset is also an important consideration. In [140], linear, time domain minimum-phase interpolation suggests that the number of empirical measurements can be greatly reduced, thus reducing storage requirements. Different regions also appear to require different resolutions.

Audible differences in the magnitude spectra of HRTF filters were investigated in [78] (differences in ITDs were not included). Overall, a poor ability to recognise differences of 1 degree was reported, rising swiftly to excellent ability at 16 degree differences. Elevation differences were more sensitive. The smallest audible change in location is known as the Minimum Audible Angle (MAA) [143].

Also particularly relevant in the context of development of an artificial spatialisation system is ability to perceive changes in directional location of moving sources. In [74], the author concludes that ‘the auditory system is relatively insensitive to motion’. Experiments also illustrate high individual differences. The amount of source movement for a moving, as opposed to static, source to be perceived is known as the Minimum Audible Movement Angle (MAMA). Figures of 8.3 degrees at 90 degrees/sec and 21.2 degrees at 360 degrees/sec are given in [161] (suggesting that extremely fast motion can be afforded low priority in artificial-spatialisation tasks). Interestingly, this work [161] also highlights the apparent lack of studies on moving sources (albeit in 1977), perhaps due to practical limitations (literature on static localisation is described as ‘enormous’). The study also states that extrapolation of dynamic localisation from static results may not be valid. The area is further reviewed in [143].

Several other approaches to interpolation of HRTFs exist, for example, the SFSRs discussed above [38] inherently incorporate interpolation, as part of the process of deriving the spatial plots. In an interesting approach [231], front/back reversals are reduced and moving source perception improved by emphasising spectral differences (essentially suppressing less prominent spectral components). In [65] the authors build on the idea of processing the mono input to their spatialisation application with the ipsilateral HRTF for the ipsilateral output, then *this result* with the contralateral HRTF *divided by* the ipsilateral to arrive at the contralateral output. The Inter-Positional Transfer Function (IPTF) is developed from this technique. It essentially represents the ratio of a HRTF with its neighbour. When performing HRTF interpolation, these IPTFs are used in place of empirical measurements for nearest neighbouring HRTFs. The benefit of the method is the ability to represent the IPTF as a lower order function.

The typical source-receiver model is inverted in [1]. The analysis source is emitted from the eardrum, and recorded using a circular array. The resulting spatial frequency implies 5-degree resolution is necessary for derivation of a complete dataset. Possible aliasing at higher frequencies is implied by the Nyquist Theorem, solutions for which are suggested in the elaboration in [2]. A similar inversion is performed in [56], which looks at interpolation as a scattering process. Interpolated HRTFs (uniquely, in both angle *and* distance) can be derived by considering an appropriately spatially sampled scattering solution.

In an interesting non-minimum-phase approach [211], an overlap-add solution is suggested. Phase Vocoder processing is used for smooth movement (which conveniently allows for Doppler-Effect pitch processing). MAMA criteria are suggested, with no interpolation in between points (with 5 degree resolution). Objectively, analysis appears to show smooth spectral evolution in moving sources. It is this author's opinion that subjective tests could verify the method, from the point of view of omitting interpolation, particularly for slow moving sources at points in HRTF space where there are relatively large differences between adjacent measurements.

In [233], magnitude interpolation and a spherical head model for phase are used. The necessity of frequency domain processing is highlighted, from an ITD point of view, due to the sensitivity of the auditory system. Finally, in [76] an interpolation method based on the alignment of the filter roots is presented. Here, the roots of minimum-phase FIRs are aligned and interpolated in a process that guarantees a minimum-phase output.

2.2.4 Dynamic Source Processing

Practical considerations when implementing a functioning artificial spatialisation system include update rates and filter design. In [223], the authors state that a 60 Hz update rate is required with a maximum of 100 ms latency for smooth motion. Two possible approaches to dynamic-source behaviour are discussed: output cross fading and parameter cross fading (essentially updating/interpolating spatialisation parameters). Conversely, the DIVA system [187] reports that a 20 Hz update rate is sufficient (with sample by sample processing of delays and gains). The system discussed in [184], when performing at capacity has a 60 Hz refresh rate, 29 ms latency and accuracy of 1 degree resolution. The author reports a slight degradation from actual sources in this scenario (time-aligned time-domain linear interpolation was used). Further reduction in fidelity leads to further degradation (as expected). In an interesting, if apparently anomalous observation, ‘clearly audible’ discontinuities when lower update rates were used appeared to be ignored by participants. In [93], the update rate is divided into an interpolation period and a commutation period. The interpolation period may be dictated by hardware constraints, and is listed as typically 10ms to 40 ms. The commutation period used in the FIR implementation discussed is the length of a time-domain filter. Essentially, the commutation period updates spatialisation parameters, in a process that does not necessarily have to provide accurate static filters

When used in spatialisation tools, HRTFs are implemented as filters. Again, this section is intrinsically linked to preceding discussions on HRTF analysis, representation and interpolation. HRTF directional filters can be designed to be FIR or IIR. A discussion of each implementation is now presented.

2.2.5 HRTF Processing using FIR Filters

Essentially, an empirically measured HRIR can be thought of as an FIR filter. As discussed above, these HRIRs are typically truncated using a rectangular window, which may affect the impulse response. Implementation of FIR filters is thus relatively straightforward as convolution. As discussed, a minimum-phase transformation is often imposed on these FIR filters, allowing further order reduction. Interpolation of FIR filters can be performed in the time or frequency domain, using several algorithms of varying complexity, as above.

2.2.6 HRTF Processing using IIR Filters

IIR models of HRTFs essentially aim to model the HRTF as a recursive process. In so doing, theoretically significant aspects of the frequency response can be maintained, while processing costs can be reduced. Therefore, IIR implementations involve data analysis, representation and interpolation issues.

In [172], IIRs are highlighted as beneficial when low-order processing is required. The authors concede that optimised IIRs are simply not able to capture all detail of HRTFs. The difficulty with dynamic processing with IIRs is highlighted. As the filters have feedback terms, updating coefficients may cause discontinuities (there are also stability issues). This difficulty is also stated and discussed in [213]. Therefore, coefficients are updated at the sampling rate. Bearing this in mind, interpolation of IIRs is discussed in detail (in [172]). Filter coefficient interpolation is compared to pole/zero interpolation. The difficulty in the latter is the matching of poles/zeros: i.e. which pole from filter A corresponds to which pole from filter B (assuming they are of the same order)? A structured approach is suggested, whereby the z plane is divided into bands, each of which contain one pole and one zero, thus resolving this issue. The difficulty posed by the possibly non-matching number of real poles and zeros in IIR models to be interpolated is also discussed. Relatively low objective error measurements were achieved by the structured IIR model (20 poles/zeros), from both a model (14%) and interpolation (19%) point of view.

In [109], the authors address the challenge of finding a recursive filter of a desired order which best matches the frequency response of the empirical HRTF. An all-pole model (which lacks anti-resonances/zeros) based on linear prediction and a least-square (minimises errors between empirical and derived on a dB scale) model are suggested. Good results are achieved with low order IIRs (six poles and six zeros). No audible difference was reported for most locations.

In another IIR based study [183], the authors discuss Active Sensory Tuning (AST) as a potential tool for HRTF interpolation. Essentially, perceptual tests inform a gradient-search method for pole location between neighbouring HRTFs (basically following the slope of poles based on least-square error). A genetic algorithm (i.e. based on previous choices) aids the AST.

In [93], the authors provide a brief historical overview to IIR approximation of HRTFs, and use a least-square approximation. The difficulty with interpolation of recursive systems by coefficient update is again mentioned. The authors again suggest sample by sample processing, or a dual-filter solution, allowing recently replaced filter coefficients to decay properly. The possibility of a transient response at the output dictates this requirement, due to the previously mentioned difficulties of recursion with updated parameters. A frequency-warping technique is suggested to improve low-frequency resolution (essentially spectrally focusing on lower frequencies; the topic of warping is elaborated upon in [81]). In this comprehensive study, the interpolation of IIRs and FIRs is also discussed. Fourier Series/Spherical Harmonic decomposition is mentioned as an ‘ideal interpolation’, with a linear model suggested for real-time implementations. Highlighting the interpolation/commutation paradigm, in a sufficiently dense dataset, only commutation is needed. The substantial work concludes that IIR and FIR models have independent benefits, with FIR judged better for dynamic processing. This conclusion is echoed in [187], which deals with the overall topic of interactive virtual environments.

In another review of filter design [82], filter order required for adequate transparency is discussed. In a literature review, order 6 IIRs to order 512 FIRs are referenced. The differences in order are explained by non-uniformity in the experiments referenced (it is apparent from this author’s work that this is somewhat of an issue in the field). Windowing is again discussed, with a rectangular window proving more transparent than a Hamming window, despite spectral leakage caused by potential discontinuities arising from any truncation. The benefit of warping the frequency response in IIR design is also discussed here, specifically fitting the filter response to a psychoacoustic (as opposed to linear) scale (for example, using ERB). For 75% of subjects to notice no difference to a reference filter, order 40 FIR, 25 IIR and 20 warped IIR were deemed sufficient. The study agrees that an FIR approach is more suitable for dynamic sources. In [83], a further objective analysis by the research team suggests warped IIRs can be further reduced to order 16.

2.2.7 Discussion

Having considered the literature, it appears that some contrasting information exists regarding the finer detail of artificial binaural spatialisation. For example, the ideal filter order, refresh rates, most successful interpolation methods, necessity for individualisation and other considerations all appear to result in differing requirements in different scenarios. Definitive conclusions are difficult to make, due to the varying parameters involved in experiments, as commented in the literature. Some studies focus more on dynamic processing, others on accurate low-order static filters. In some cases, audio is combined with virtual visual displays, and some degree of error is acceptable in return for reduced processing demands. In others, high-fidelity audio is crucial. A strikingly common theme in many of the studies quoted is the relatively small subjective groups (for example, the subjective tests performed in the often cited [111] use only four subjects). It is the opinion of this author that larger subject groups would provide more reliable data (as is the case in other, more established statistically-based disciplines). It is with all of this in mind that a more generic approach is taken to implementation here, discussed in detail in the next chapter.

It is clear from the literature that many studies have focused on data reduction, the majority of which use the minimum-phase assumption. The current work reconsiders the minimum-phase assumption, motivated by initial preliminary listening tests which suggested an audible difference between minimum-phase and non-minimum-phase HRTFs. Another aim of this work is to investigate the possibility of more direct use of HRTF datasets, avoiding the (often complex) data preparation necessary in many of the above methods, thus rendering HRTF processing more accessible. A more detailed discussion and critique of the prevalent minimum-phase assumption follows.

2.3 Minimum-phase

Typically, a frequency domain signal exhibits unrelated magnitude and phase spectra [159]. However minimum-phase systems (a particular subset) do exhibit a direct relationship between their magnitude and phase spectra. Bearing in mind the difficulty in interpolating phase data due to the ambiguity involved (as discussed above), this is a particularly interesting characteristic. If phase can be derived from magnitude, phase interpolation becomes unnecessary, as interpolated phase can be obtained from the relatively straight-forward (and, more significantly, valid) interpolation of magnitude values.

From a DSP perspective, all of a minimum-phase system’s poles and zeros must lie within the unit circle. Intuitively, all poles must lie within the unit circle for system stability. If a system also has all zeros within the unit circle, it follows that its inverse is stable. Such systems are defined as minimum-phase systems.

Oppenheim and Schafer [159] observe that any rational system function can be broken into a minimum-phase and an all-pass system. An all-pass system can be defined as one which has a magnitude response that is absolutely constant with respect to frequency [203]. Therefore, the magnitude of the minimum-phase all-pass decomposition is represented solely by the minimum-phase system, and the phase is reconstituted by both the all-pass and minimum-phase representations.

The system in question can thus be defined as:

$$H(z) = H_{\min}(z) H_{ap}(z), \tag{2.1}$$

where $H_{\min}(z)$ is a minimum-phase system and $H_{ap}(z)$ is an all-pass system. The process of decomposition can be defined thus: any zeros outside the unit circle in the system will be removed, and inserted into the all-pass system. As the all-pass function, by definition, has a flat magnitude response, these zeros must be cancelled. Cancelling poles are thus inserted into the all-pass system, to cancel these zeros. These cancelling poles in the all-pass function will lie inside the unit circle, and will cancel the zeros outside. The all-pass function is thus completed, and is stable (all poles within unit circle). The overall system, however, now contains extra poles (the cancelling all-pass poles). These are in turn cancelled by adding zeros at the same location. These zeros will also therefore be inside the unit circle, so can be included in the minimum-phase system, to avoid a constant cycle of cancellation should they be included in the all-pass. These zeros are known as the conjugate-reciprocal zeros of the zeros that lie outside the unit circle in the original system [159].

In summary: $H_{\min}(z)$ contains the poles and zeros of $H(z)$ that lie within the unit circle on the z plane, as well as conjugate reciprocal zeros of the zeros of $H(z)$ that lie outside the unit circle, which are shifted to $H_{ap}(z)$. $H_{ap}(z)$ also contains cancelling poles to satisfy the all-pass definition of constant magnitude response.

2.3.1 Minimum-phase and HRTFs

Begault defines the HRTF as ‘the spectral filtering of a sound source before it reaches the ear drum that is caused primarily by the outer ear’ [14]. However, it is undesirable to use HRTFs that contain the auditory canal response of the dummy head in artificial spatialisation applications, as the listener, using headphones that transmit audio from the entrance of the ear canal, is then essentially listening through two auditory canals, that of the dummy head and their own. This is avoided in the MIT dataset used here through diffuse field equalisation, as discussed above.

Interestingly, the MIT dataset that contains the system and ear canal response (the ‘compact’ dataset), on casual observation of the author, often leads to more exaggerated artificial localisation, perhaps suggesting that listening through both auditory canals helps improve a sense of externalization by exaggeration.

In [135], the transfer function from the free soundfield to the external ear and the transfer function of the ear canal are treated separately. The authors decomposed their measured transfer functions into minimum-phase and all-pass functions in order to obtain a clear representation of phase without the 2π ambiguity. While doing this, they realised that the minimum-phase function appeared to contain almost all the detail of the phase spectrum. It is clear from their illustrations of a free field to auditory canal entrance transfer function that the all-pass phase does indeed approximate linearity. Regular large discontinuities in the phase spectra can be observed, representing jumps of 2π (the periodic nature of phase explicitly illustrated). The authors surmise that the all-pass phase approaches linearity for the free field to ear canal function in question. Some non-linearities are however evident. In the particular case under investigation, the phase spectrum of the all-pass component clearly deviates from linearity just after 5 kHz, around 10 kHz and approaching 15 kHz. The paper goes on to assert that the all-pass component of the full HRTF (including the ear canal response, as defined by Begault [14]) exhibits a ‘nearly linear’ phase response up to 10 kHz; the external ear approximates a minimum-phase system in this frequency range.

This approximate all-pass linearity provides the key to minimum-phase based HRTF binaural processing. If linearity is not assumed/valid, the resulting non-linear all-pass heavily detracts from the method. A linear-phase function can be implemented as a simple time delay. This time delay can be realised using a time-domain, frequency-independent delay line; quite a simple and efficient process to implement. The observation in [135] of approximate linear phase has become an instrumental factor in binaural HRTF based processing, and has been used in several studies of HRTFs, as above. However, in this thesis, this minimum-phase assumption is re-evaluated. Alternatives are developed which do not rely on the approximations involved. This essentially involves engaging more directly in the phase ambiguity problem.

The minimum-phase and (assumed linear) all-pass decomposition of HRTFs thus results in three data objects: The minimum-phase filter for the left and right ear, and the delay between them. The overall magnitude will be represented by that of the minimum-phase filter (whose magnitude is the same as that of the empirical HRTF); the overall phase will be constituted by the minimum-phase phase spectrum (derivable from the magnitude spectrum) plus a frequency independent, linear delay. Figure 2.3 shows an empirical HRTF, its minimum-phase version and their common magnitude plot.

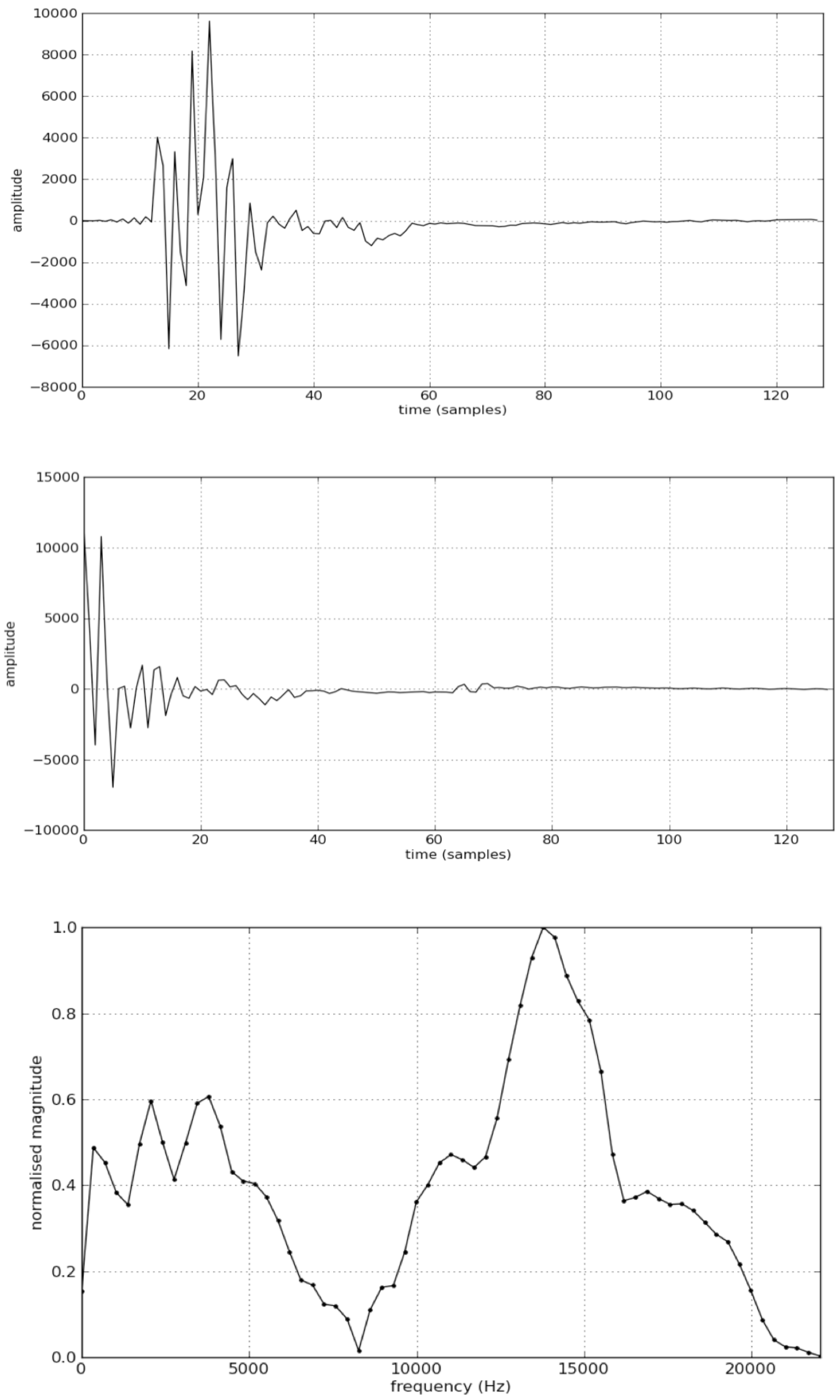


Figure 2.3: Three plots illustrating the HRTF for 0 degree elevation, 0 degree angle (top), its minimum-phase representation (middle) and their common magnitude spectrum (bottom); the solid line indicates original and dots indicate minimum-phase magnitude spectra.

When dealing with FIR filters, the minimum-phase approach is not only beneficial in that it allows the phase spectrum to be derived directly from the magnitude spectrum; it also constitutes the shortest realisable FIR filter containing all the relevant data before the function dissipates to zero energy. This can be seen clearly in the figure above.

The description of HRTFs as minimum-phase filters and delays above is validated theoretically in the work on decomposition of impulses in [135]. However, perhaps a more pertinent validity test from the point of view of a developer of artificial spatialisation tools involves psychophysical testing of a subject group. Kulkarni's and co-authors' seminal work in this area examining the sensitivity of human subjects to HRTF phase spectra [111, and also in the more compact: 110] is often cited. In the study, firstly (in preliminary objective tests), minimum-phase plus delay impulses were compared to the empirically measured HRTFs they were derived from. Waveform coherence was measured and a specific phase error test was performed. Essentially, for this test, the best fit straight line was derived from the residual phase when the minimum-phase was subtracted from the empirical phase. The error in the linear fit of this line is then a measure of the deviation of the minimum-phase plus delay model from the empirical.

Interesting results arise from both tests. Coherence values were high for the 2 data sets used. The four results (left and right for each data set) range from 75% to 97% of coherence indices above 0.9. Coherence values were systematically worse at lower elevations and extremes of the horizontal plane. It is suggested that this is due to the shadowing effect of the head and interactions with the torso making the all-pass delay non linear, a phenomenon discussed in [108]. This is supported by better performance at higher elevations, where there is less obstruction in the path to the contralateral ear. Phase error results enforce this assumption.

Several psychophysical tests were then performed. Typically, four processed sounds were presented to the listener, the second or third (randomly) processed with minimum-phase plus delay, the rest empirical. Subjects were asked to pick the odd signal.

In the first experiment, random noise bursts were used, with random locations in the data set. Subjects had a 50% chance of success, and performed at chance, suggesting minimum-phase plus delay is a valid model. It is worth noting that these are not localisation tests, but rather ‘odd one out’ or ‘spot the minimum-phase’ tests. Therefore monaural (information presented to one ear, with the other ear muted) as well as binaural conditions were tested. In the second test, a smaller subset of locations was tested. 0, 90, 180 and -90 degrees relative to the listener in the horizontal plane were chosen. Similar results were achieved, with the noteworthy exception of one subject picking up on the low coherence of the right ear of the data set (the data set used was the one that scored the lowest, by a significant margin, coherence score of 75%; the others were all above 90%).

The next experiment in Kulkarni *et al* used the same parameters as that immediately preceding it, with the exception that the noise burst was not randomised, but fixed for each test. Monaural results were similar to the random noise burst results, but binaural results show better than chance answers at 90 and -90 degrees for two out of four tested subjects. These subjects cited a positional cue as the reason for the improved differentiation ability (i.e. not a difference in the timbre of the source; as monaural and binaural experiments are used and localisation is not being tested, rather signal coherence, the specification of a location cue is significant). It is also worth noting that one of these subjects also performed better than chance in the random noise experiment.

The two subjects who appeared to be able to distinguish correctly between minimum-phase plus delay and empirically-measured HRTFs for fixed noise burst sources at a success rate greater than chance were asked to perform another test. The test was similar to the preceding one, but only looked at 90 and -90 degree locations. Sources were filtered for a low and high pass test signal. Results showed that low-pass results were at similar above chance levels to the previous test, but high-pass results were at chance. This clearly points to a low-frequency cue present at extremes of the horizontal plane, aiding the subject in distinguishing between minimum-phase plus delay and empirical impulses. These results are expected, bearing in mind the predominance of ITD at low frequencies, as per the duplex theory (magnitudes are the same in the empirical and minimum-phase impulses, so the ability to distinguish differences is based on timing information, which is more relevant at lower frequencies). In [225] psychoacoustic experiments (offsetting ITD and IID against each other) confirm this dominant role of low frequency ITD.

Kulkarni *et al* then take a closer look at the empirical and minimum-phase plus delay impulses at the areas of interest (the extremes of the horizontal plane) as well as 0 degrees and 180 degrees. The interaural phase for lower frequencies is graphed for both models. As expected, 0 and 180 degree minimum-phase plus delay models do not deviate significantly from empirical measurements, but 90 and -90 degrees do. These tests were performed on actual impulses, empirical and derived. A model of the auditory periphery is used to test processed sources, yielding similar results. Again, these results confirm that ITD plays an important role in localisation at low frequencies. The non-linearity of ITD is also confirmed, specifically, ITD is greater at lower frequencies, as in [108]. This perhaps suggests that modelling ITD as a linear delay is not adequate. Many of the finer non-linearities of the empirical ITD are modelled by the minimum-phase HRTF, but these experiments highlight the approximation involved in modelling the HRTF as a minimum-phase filter and linear delay.

Final experiments used to test the overall importance of phase in localisation provide a sense of perspective on the accuracy needed in phase representation. For a fixed source at the four primary locations previously used, users could not tell the difference between linear and empirical phase models. The linear model is based on a zero-phase HRTF (magnitudes as per empirical, phase zeroed) plus a delay extracted directly from the empirical HRTF. This suggests that the finer structures of phase are not overly important, as long as the overall delay is approximated in accordance with that of the empirical. A reversed-phase model, however, where empirical phase was simply reversed and added to a linear delay was successfully distinguished by subjects. Hence, although phase does not need to be exact, a phase model that strays greatly from the empirical is easily perceived. Phase disparity is greater between minimum-phase and empirical than linear phase and empirical for the models in Kulkarni *et al* (which implies it offers an attractive model for binaural processing; all the more so when the error, < 20 μs is within (generally, depending on source type) the reported just noticeable difference/jnd for ITD).

The study concludes that phase spectrum differences are not significant monaurally. Furthermore, binaurally, a relative insensitivity to interaural phase spectra was illustrated, with the exception of low frequency ITD, which needs to be accurate. The minimum-phase model was deemed ‘adequate’ for ‘most positions’.

Instances of minimum-phase invalidity are also discussed in [32]; specifically highlighting the contralateral HRTF and instances of strong torso reflection. In [185], the authors again investigated efficient FIR and IIR HRTF representations. Interestingly, as part of this study, 256 point minimum-phase HRTFs (relatively long) were tested against 256 point empirical HRTFs and were detectable in certain cases. In [191], the author highlights that although the assumption is convenient, the external ear is not strictly minimum-phase.

The issue with the discrepancy in empirical and HRTF plus delay filters lies in the non-linearity of the all-pass section. An extremely relevant issue in implementation of a minimum-phase based binaural spatialisation system is how to extract this linear delay/how to approximate the all-pass section as a linear delay. This will be discussed in the next section.

2.3.2 ITD Extraction

In [139], the authors further discuss the minimum-phase all-pass decomposition. They view the HRTF as a minimum-phase system, a delay *and* an all-pass system, and conclude that omission of this all-pass is audible in some cases.

Shortly afterwards, the same researchers published on the specifics of breaking the HRTF into minimum-phase and all-pass components [163]. Focusing on methods of delay extraction, their review of the literature mentions Jot's linear curve-fitting method [93], a cross-correlation maximum method, and a threshold detection method. Jot's linear curve fitting [93] suggests modelling the ITD over a low frequency range, instead of the whole range of the empirical impulses. A delay based on the empirical data from 1000 to 5000 Hz is used. The excess phase is examined, and a best fit linear curve is derived for this frequency band. The method is described as 'more robust' than that detailed in [111], and is employed in the commercial *spat~* spatialisation toolkit. The cross-correlation method (used in the study outlined in detail above [111]) analyses the left and right channel of the HRTF for similarity, with regard to a delay applied to the lagging signal. The maximum of the result yields an appropriate ITD estimation. Importantly, in [111], further accuracy is gained by performing the same procedure to the minimum-phase representations and subtracting this value, as they will be inherently included in the spatialisation process. Threshold methods essentially use analysis of signal pressure onsets.

Once again (in [163]), the potential difficulties with minimum-phase HRTFs, as well as the almost universal adoption of the technique are mentioned: 'although minimum-phase HRTFs with a pure delay as ITD are now widely applied, some may still be audibly different from measured HRTFs'. The paper highlights potential problematic locations, by focusing on the remaining all-pass component in the minimum-phase, linear-phase plus all-pass break down. When the interaural group delay at 0 Hz introduced into the binaural signal by the all-pass components is greater than 30 μs, omitting the all-pass is audible (as concluded in [139]). An extra delay is suggested in these situations. The authors conclude that the minimum-phase plus delay model can be improved by taking the Interaural Group Delay Difference (IGD) at 0Hz of the excess phase components (details of which can be found in [141]). Thus another method of extracting the delay is arrived at (adding another layer of complexity).

Delay extraction methods are also discussed in [32], which again mentions a threshold method, interaural cross correlation (updated to consider signal envelope), and fitting of excess phase. The cross-correlation method performed best in an experiment whereby subjects matched minimum-phase delays to empirical, individualised HRTFs. Relative success of the minimum-phase model was reported in the horizontal plane.

More recently, yet another approach is presented in [149]. The difference of arrival time is represented as the maximum of the correlation of the HRIR and its minimum-phase representation. This subtle update (based on the similarity of the HRIRs and their minimum-phase counterparts) illustrates that the research question is still open.

2.3.3 Implementing Variable Delays

From an implementation point of view, although a delay line can be thought of as a simpler process than a frequency dependent filter, the situation becomes more complex when fractional delays become appropriate. In a discrete digital system, delay length possibilities increment sample by sample. In [93], implementation needs of delay lines required for spatial processing are hypothesised. At a sampling rate of 50 kHz, a single sample delay will represent a time difference of 20 µs. In terms of ITD, this implies an angle of approximately 2.7 degrees. This appears adequate if average localisation blur is taken to be 3.6 degrees (as cited). However, as discussed in [21], the lower limit of ITD resolution may be as low as 1 degree. In [234], the authors highlight the need for frequency domain application of ITD, using the figure of 7 µs as the resolution of the ear (1/3 of a sample at 44.1 kHz).

More significantly, however, is the need for smooth variance in delay lines. In the case of dynamic source processing, delay lines truncated to sample by sample resolution will cause undesirable noise. For these reasons, delay line interpolation is required.

As an aside, it is assumed that minimum-phase based systems store and apply delays at a sampling-rate resolution, when, clearly, a higher resolution may be appropriate. This constitutes another motivation for the novel methods introduced in the next chapter, which both use frequency domain ITD.

In [93], an additional first order all-pass is suggested to add more time accuracy (which will offer reasonably accurate frequency response up to ¼ of the sampling rate). FIR Lagrange interpolation is discussed in [188] (considering not just the adjacent point, but also further points on each side of the nearest available measurement). The question of non-integral delays is thoroughly considered in [115]. FIR (e.g. with Lagrange interpolation) and IIR (e.g. all-pass) methods are discussed. In conclusion, this work recommends testing out a simple filter to start with, for example a Lagrange interpolated FIR of length 2 or 4 (order 1 or 3; even lengths give linear phase response). As will be discussed in the next chapter, Lagrange of length 2 is deemed sufficient here (essentially a linear interpolator). Worst case scenario low-pass effects are discussed in the work. However, the relative simplicity of the approach, the frequent need for numerous delay lines (e.g. the reverb implementation discussed later) and the acceptable response motivate its use.

To reiterate, interpolated delay lines typically attenuate high frequencies, and are therefore not ideal. However, informal listening tests performed in [38] suggest that these artefacts are not significant (the present author believes that artefacts are not severe but must be considered in certain scenarios).

2.4 Conclusion

In conclusion, a thorough literature review suggests that there are several open research questions in the field, and highlights the multi-faceted nature of HRTF based spatialisation. The minimum-phase plus delay approach is often assumed, however, there are clearly instances where it is not valid. Extraction and dynamic implementation of delay lines is also non-trivial in the implementation of minimum-phase based systems. With all of this in mind, the development of novel, non-minimum-phase techniques is proposed.

Chapter 3. New Methods for Artificial Spatialisation

3.1 Introduction

This chapter reviews current Computer Music tools for binaural, HRTF based spatialisation and gives theoretical detail of the novel approaches suggested (motivated by the preceding literature review). A detailed discussion of phase unwrapping, in the context of HRTF interpolation is also offered, concluding the chapter.

3.2 Current Computer Music Tools

One of the primary initial goals of this research was to provide a stable, flexible and efficient open source tool for HRTF based binaural spatialisation. The solutions available at the time this project was commenced will be discussed below (*hrtfer* for Csound and *earplug~* for Pure Data (PD)), as well as more recent additions to the open source repertoire. The more established approaches are summarised in table 3.1, below.

Approach	**Detail**	**Comments**
hrtfer	MIT dataset, truncates to nearest source	Lack of interpolation
earplug~	Time domain interpolation algorithm	Computationally costly
iem_bin_ambi	Virtual ambisonics	Static setup

Table 3.1: Summary of established Computer Music binaural spatialisation tools

3.2.1 *hrtfer* for Csound

Csound is an open source software tool used for audio based research and creative activities. The core of the system is based on opcodes: processing units with a specific function (e.g. signal generation/processing, for more see [33, 25]). One such opcode is *hrtfer*, developed by Eli Breder and David MacIntyre in 1996. The opcode uses the MIT HRTFs [142], and essentially convolves the input with the appropriate HRTFs. Accurate spatialisation is available at static locations which correspond exactly to HRTF measured points. However, at non-measured points, the nearest data will be used, resulting in potential angular imprecision. This lack of interpolation also causes a problem in the case of moving sources. The source jumps from measured point to measured point, often resulting in discontinuities in the output. Statically, the dataset is relatively dense at key locations (primarily the horizontal plane), but does not satisfy MAA requirements. The authors suggest a fade out of the old convolution result and a fade in of the new to minimise these ‘clicks’, which does reduce the severity of the noise. This feature is, however, disabled in the latest version of Csound, as it causes dropouts in the audio. When implemented by this author, the crossfades do reduce the discontinuities to a degree, depending on the frequency content/bandwidth of the source (narrow-band sources are still effected by the abrupt FIR filter switching; more noisy sources may mask the switch). In all sources, however, jumps in location may perceptually imply a staggered path, when a smooth trajectory is more desirable. This opcode could greatly benefit from an interpolation algorithm, as discussed in the previous chapter. Also, *hrtfer* uses the compact set of HRTFs, when perhaps the diffuse set is now more appropriate.

3.2.2 *earplug~* for PD

PD offers a more visual default editor than Csound, allowing users to edit a canvas/patch, using object boxes not unlike csound opcodes. One such object is *earplug~* [229], which does offer an interpolation algorithm. The four nearest measured points are used to find an interpolated value in between. A new HRTF is thus derived for each processing block (64 samples, as per PD's default). Previous interpolated HRIRs are stored and a similar interpolation is performed between the current and previous HRIR (over time in this case; essentially fading out the old and in the new, a computationally costly process). This interpolation is performed in the time domain (on empirical HRIRs), the issues with which have been discussed in the previous chapter. Also, as discussed in chapter 1, time domain convolution (of 128 point interpolated HRIRs) is considerably more computationally costly than convolution performed in the frequency domain.

3.2.3 Virtual Loudspeakers: *iem_bin_ambi*

In [148], a virtual loudspeaker approach is taken, implemented in PD. Essentially, static HRTFs are used to spatialise sources at loudspeaker positions. This paradigm will be further discussed in chapter 6. Briefly, in a static listener scenario, this approach removes the need for HRTF interpolation; source movement can be controlled by the multi-channel signal feeding the virtual loudspeakers. In this case, ambisonics is chosen as the multichannel algorithm. Inherently, any imperfections of the multichannel approach will be reflected in the binaural reproduction. The *iem_bin_ambi* objects realise this algorithm.

3.2.4 More Recent Approaches

The work on IIR HRTF filters discussed in the previous chapter is presented as a PD external *mobile*~ [172, 173] (under development). In [181], the virtual ambisonics approach (implemented as the *Girafe* system) is discussed, with a view to implementation in Super Collider: a dynamically typed, client/server based audio processing language.

Perhaps most interesting is the minimum-phase based *CW_binaural~*, presented at the PD convention in July 2009 [54]. The research paper presenting this work clearly highlights the issues previously raised with minimum-phase interpolation implementation. The object is designed using an object-oriented paradigm, to allow for updates; its authors underline the continuously developing nature of the research field, echoing the conclusions to the previous chapter. Currently, cross-correlation is used to estimate ITD, but desirability of support for various methods is highlighted. The authors also highlight the potential timbral distortion imposed by using simpler approaches to delay lines. More complex methods will reduce this distortion, but are more costly. In fact, in the benchmarking tests performed, CPU peak usage is the same for independent HRTF filtering (an FFT-based 128-point filter) as it is for implementation of an independent 3rd order delay; 3.12 peak CPU usage. A significant reduction is reported when using linear interpolation (1.56 peak usage, only slightly larger than with non fractional delays). In another interesting CPU usage test, 6.24 peak usage is reported in FFT based processing of the complete process and 20.2 in time-domain convolution processing. It is also important to consider real-time performance when implementing delays; higher order delay line interpolation algorithms typically require use of past *and* future samples, increasing latency. Using previous samples only makes the process causal.

It is also worth pointing out that commercial, proprietary solutions are not discussed here, as the source code/algorithms used are not available. However, perhaps the most commonly used commercial tool in Computer Music: *spat~*, for Max/MSP, is based on Jot’s research, which is abundantly referenced in this work.

In conclusion, several solutions exist in the Computer Music domain (judging by recent trends, it is hoped that activity in the area will continue to flourish). In light of the literature reviewed in the previous chapter, there is, however, a need for an approach that avoids the minimum-phase approximation, minimises any data preparation/processing/compression (primarily to allow for direct creative use by non-experts), allows for smooth movement of sources, is efficient, and is supported in a flexible and dynamic environment.

3.3 Novel Algorithms: Theoretical Discussion

Two new approaches to HRTF interpolation, spatialisation and dynamic processing are presented as a direct response to the preceding literature review. Their overall motivation and theoretical methodology and justification are presented in the rest of this chapter. In the following one, implementation specifics are discussed, from a low level coding point of view (a resource often omitted from the literature). Finally, objective and subjective tests validate the methods.

3.3.1 Motivation

This section refers directly back to chapter 2. With regard to HRTF individualisation, the MIT diffuse field equalised dataset is chosen as averaged non-individualised data [142]. Although the presented solutions are optimised for the angular and elevation resolution of this particular dataset, any other configuration is possible (and in fact generally simpler) with minor code updates (a command line solution example is offered using the LISTEN database [124] in the ‘Chapter4/listen’ folder on the accompanying CD-ROM, and is discussed below). The MIT solution is presented as it is deemed the most suitable averaged dataset; it is also appropriate for validating the concept and implementation of the new methods. Data is used directly. No pre-processing is required (with the exception of placing spectral versions of the HRIRs into two large datafiles, for efficiency of processing and ease of use). Therefore, no complex signal processing is required by a user who may wish to use a particular dataset. This is a core idea of the algorithms presented. As has been highlighted in the literature review in section 2.2, there are several potential ways to minimise/compress data, many of which perform well in subjective and objective tests. The approach taken here is to reduce data minimisation to truncation to 128 point audio files. It is hoped that this provides a more generic solution. It also removes potential for data loss, which is pertinent in any minimisation algorithm. The other core goal of these solutions is to avoid the minimum-phase approximation. Once again, minimum-phase has been shown to perform well but is by no means transparent. Therefore, empirical data is used directly (this point is also related to the empirical-data-use goal, as transforming a dataset into minimum-phase plus delay data is by no means trivial).

An immediate criticism of this approach is that it does not utilise potential processing and storage savings. The approaches below are highly optimised (see the discussion of source code in chapter 4). It is also hoped that the preceding discussion of direct empirical data use justifies the approach: a user does not have to employ the often complex (and often not fully documented) approaches to data minimisation, and furthermore does not have to be troubled about what the data minimisation may be omitting from the dataset. Equally, a dataset may be processed by a user before being employed (a personalised dataset with improved low-frequency resolution may be used if prepared in an appropriate way, for example).

One caveat to direct empirical use occurs in the Functional Phase Model, discussed below. This approach essentially extracts accurate low-frequency phase from the dataset. However, if this analysis is not desirable, results from the MIT data can be used, which represents an average low-frequency scaling factor.

From the point of view of interpolation, frequency-domain processing is employed throughout. The relevant insensitivity to phase information at high frequencies, high-frequency accuracy of a spherical head model, and need for accurate low-frequency phase (all discussed in the previous chapter) informs the processes. Phase interpolation is therefore focused on in the solutions presented. The significant difficulty of delay extraction is avoided by using empirical data, as opposed to minimum-phase. In an approach similar to that in [93], an update rate defined by the length of the HRIR filter is used (344 Hz at 44.1 kHz sampling rate). This ensures smooth movement for all but the most extreme source trajectories (which will typically breach the boundaries implied by MAMA specifications), and is deemed necessary to avoid artefacts (which may be present in systems with lower update rates). In the real-time implementations, the ‘interpolation period’ is defined by the host; Csound or the hardware being used. When required (for example in delay lines), sample-rate based update is enforced. FIR filters are chosen, as the empirical data can be understood directly in this form. Also, as discussed, FIR filters better lend themselves to dynamic processing.

3.3.2 Phase Truncation

The first novel approach epitomises the goal of direct empirical data use. Four-point linear magnitude interpolation is used (although this method cannot account for local spectral anomalies, it is deemed appropriate for real time applications [93]). The fidelity of the phase spectrum in interpolated HRTFs is dictated by the density of the data set. The basic approach of truncating moving sources to the nearest measured point is developed and focused on phase data here. The relative insensitivity to phase data (one of the three main conclusions in [111] is that ‘listeners are insensitive to the details of the interaural phase spectrum as long as the interaural time delay of the combined low-frequency part of the waveform is maintained.’) is exploited here. Also, accurate low-frequency ITD is maintained. Nearest empirical phase data to the required point is used.

When dynamic sources are processed, phase values ‘jump’ from empirical point to point. As discussed, changing a filter during a dynamic process can lead to discontinuities in the output. Continuously processing the input with old and new data and cross fading is deemed too computationally expensive. Therefore, brief parametric crossfades are introduced to remove any discontinuities in an efficient as possible manner. These crossfades are parametric in that the user can decide how long they are. The shortest available crossfade is one processing buffer (the length of a HRIR: 128 samples). This length is appropriate for noisy sources, which can mask any remaining discontinuity. However, more narrow-band sources may require a longer crossfade to render any ‘click’ inaudible. A default value of eight buffers is suggested (1024 samples: a very brief period of extra processing: old HRTF data processed and faded out, new faded in). Crossfades of 1024 samples allow 43 equally spaced changes of index per second at a sampling rate of 44.1 kHz (crossfades will begin to overlap after this point). Five degrees is the highest angle increment resolution in the data set. Forty-three 5 degree changes per second imply a speed of 215 degrees per second. This limitation falls comfortably within the previously discussed MAMA limits.

Smooth, artefact-free dynamic source trajectories are therefore achieved. A high update rate provides a robust solution to magnitude spectra. Phase spectra are dealt with in a psychoacoustically-based manner. Insensitivity is exploited, while the importance of accurate low-frequency fidelity is maintained (albeit within the constraints of the density of the dataset). Significantly, no data preparation, minimisation or compression is required; therefore, the user does not require knowledge of complex DSP. This direct use of empirical data and high correlation to empirical data makes the algorithm ideal for both expert and casual use. The algorithm is outlined in figure 3.1, below.

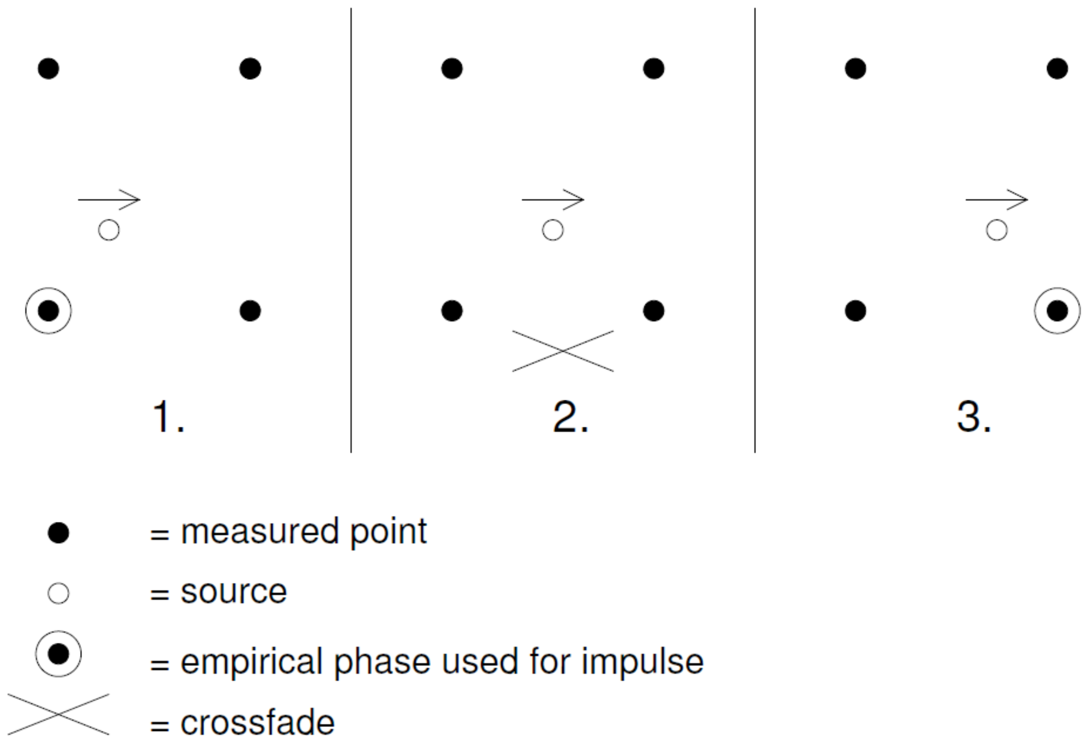


Figure 3.1: Phase truncation: source at 3 points in a trajectory from left to right. At point 1, bottom left phase is used. At point 2, a crossfade occurs. At point 3, bottom right phase is used.

3.3.3 Augmented Spherical Head Model

A second novel approach to HRTF interpolation for dynamic source spatialisation is presently suggested. The approach essentially combines an empirical (magnitude interpolation and derivation) and functional (phase derivation) model. This hybrid approach has its basis in psychoacoustics. Linear magnitude interpolation, as discussed above, performs adequately when endeavouring to interpolate HRTFs for use in real-time dynamic artificial spatialisation applications. Therefore, it is employed again here, as in the Phase Truncation Model. Once again, the novel aspect of the Functional Model lies in the treatment of phase.

From a functional point of view, the main challenge in phase derivation is to identify a model that accurately calculates IPDs, therefore implying correct ITDs. Perhaps a good starting point and one frequently suggested and used in applications is to approximate the head to a sphere, as discussed in the previous chapter. This obviously greatly simplifies the complex filtering of the non-uniformly shaped head, outer ear and torso, losing a lot of spectral detail. The perceptual distortion of the spatial image caused by this gross simplification of phase is related to the discussion on sensitivity to phase in the previous chapter. In general terms, it was concluded that low-frequency ITD, and therefore phase of the left and right ear, is spectrally the most relevant, as it provides the predominant phase cue.

It is also apparent from the above discussion that these finer details of phase may not be perceptually necessary, due to the auditory system's lack of sensitivity to phase spectra. The idea behind this method is to start with the spherical head model, and to investigate possible improvements, bearing in mind this insight into phase sensitivity.

The spherical head model can be used to calculate ITD mathematically. Considering the case of the horizontal plane, ITD can be initially estimated as the distance between the point of arrival at the nearer ear and that at the further ear. This distance can be defined as:

$$d \sin \vartheta , \qquad (3.1)$$

where *d* is head (/sphere) diameter and ϑ is the angle off lateral centre of the source. Simple laws of trigonometry can be used to illustrate the above formula. Figure 3.2 illustrates the calculation.

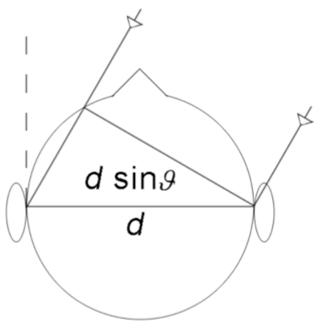


Figure 3.2: ITD calculation

This clearly underestimates ITD somewhat, as the sound source will travel around the head (by diffraction). This extra distance is given by the length of the arc:

$$r\vartheta \tag{3.2}$$

Time differences can be derived by dividing distances by speed. Thus dividing the above formulae by the speed of sound in air gives the ITD for a particular angle off lateral centre. Therefore, the combined formula for the horizontal plane [80] is:

$$\frac{r(\vartheta + \sin\vartheta)}{c}, \tag{3.3}$$

where *c* represents the speed of sound in air. This more accurate formula is illustrated in figure 3.3, below.

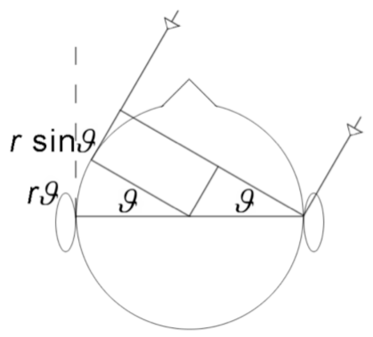


Figure 3.3: More accurate ITD calculation

Considering the third dimension, which essentially represents variable source elevation, a simple cosine factor can be used to reduce the ITD as the source gets nearer to directly above the listener (maintaining the spherical-head approximation). As HRTF datasets are typically measured at an equal distance from the listener at each location (hence only one measurement at 90 degree elevation in the MIT dataset: the source will be directly above the listener), the ITD will become smaller as the source moves above or below the horizontal plane.

$$\frac{r(\vartheta + \sin\vartheta)}{c}\cos\phi \tag{3.4}$$

where ϕ represents the elevation. This additional scaling factor is intuitive, bearing in mind the trigonometric identity $\cos(-A) = \cos(A)$, so elevation values of 90 or -90, directly above or below the listener, will give a scaling factor of 0. As the elevation angle gets closer to 0, or the horizontal plane, this scaling factor will approach 1. Minnaar *et al* describe this formula as the Extended Woodworth/Schlosberg Formula [141]; it is also referred to in [187]. This formula will be referred to as the Woodworth Model henceforth [228].

Other models for ITD have been suggested, as discussed in chapter 2. Kuhn, in a definitive study of ITD in the horizontal plane, proposes two functional models, for the low and high end of the audible spectrum [108]. These models imply that low-frequency ITDs are greater than their higher frequency counterparts by a ratio of 3:2. Of particular interest, this study also concludes that the Woodworth Model is valid for steady state high frequency ITDs and clicks, where ITD is essentially reduced to arrival times of portions of the source sound’s amplitude envelope. Kuhn’s results also imply that ITD is frequency independent below 500 Hz and above 3000 Hz.

Zotkin *et al* use the Woodworth Model for HRTF phase modelling and a magnitude interpolation algorithm [234, 233]. This system forms the basis of a virtual audio scene rendering tool. The software developed uses the CIPIC database, which contains HRTF data from 43 subjects, as well as detailed measurements of their ears.

A psychoacoustically-based reappraisal and improvement on this model is suggested here. As concluded above, low-frequency consistency of empirical and employed ITD is crucial to accurate modelling. Also, critical works in the area [108, 111] agree that higher-frequency ITD is not as significant; more specifically, a Woodworth-based ITD can account for steady state high frequency ITDs [108]. Physiologically, as discussed in chapter 1, IPD-based localisation breaks down above approximately 1500 Hz. Therefore a low-frequency, frequency-dependent scaling factor is introduced as a more complete solution, requiring minimal extra processing. Essentially, frequency-dependent ITD is extracted from the empirical HRTFs for the low-frequency band of interest. These new values are then used as frequency-dependent scaling factors in the synthesis of the phase spectrum for the desired HRTF.

The cross-correlation, onset detection and other more complex methods of extracting ITD are typically used to derive a frequency-dependent delay to add as the all-pass component in a minimum-phase plus delay HRTF synthesis system, as discussed in chapter 2. However, perhaps a more accurate approach in this case is to attempt to extract the explicit IPDs from the empirical data, and translate this phase information into time difference data.

Primarily, psychoacoustically-based parameters are imposed on the range of the spectrum to be scaled. 1500 Hz is therefore used as the upper boundary for scaling. It is considered redundant to improve computational phase accuracy when perceptual limitations will ultimately dictate. Physical IPD restrictions for sinusoidal sources can be further quantified by finding the maximum unambiguous frequency for a specific source location. At IPDs of 180 degrees and greater, the source location is uncertain.

As with phase interpolation, this uncertainty is a result of the periodic nature of phase. At higher frequencies and large angles off horizontal centre, IPDs start to inherently include one or more full phase cycles of 2π. This implies further potential for confusion, as a number of perceived source locations are possible. ITD uncertainty is illustrated in figure 3.4, below.

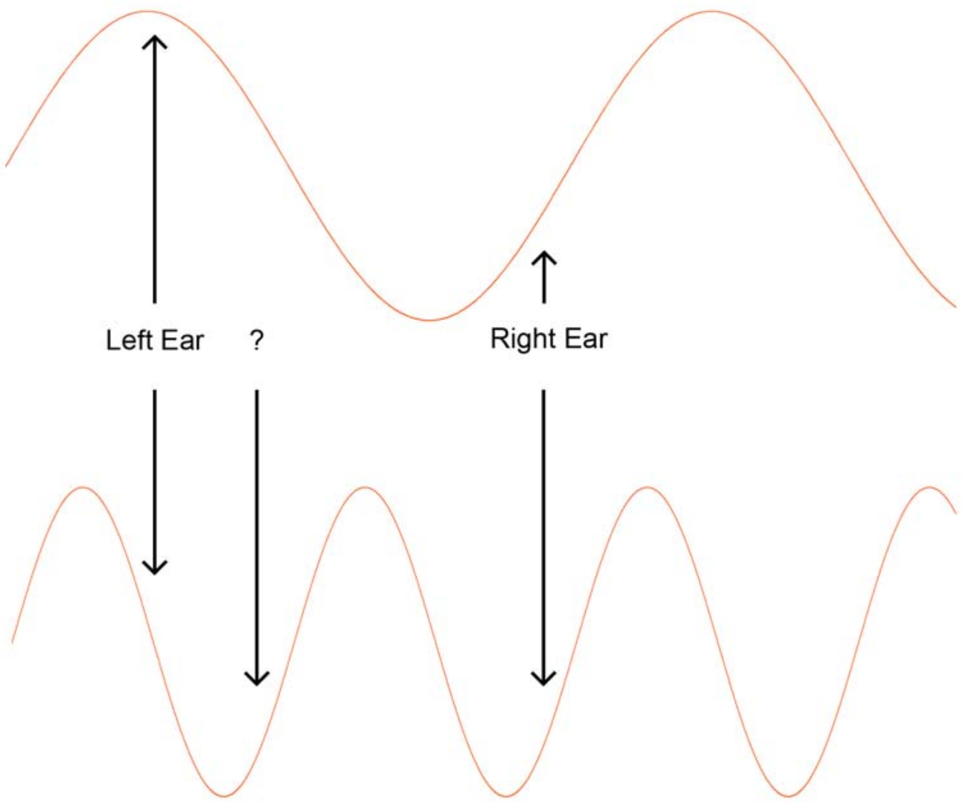


Figure 3.4: ITD uncertainty; a low frequency source (above) affords unambiguous IPD cues, whereas a high frequency source implies ambiguity due to any number of inherent multiples of 2π. Overall direction of arrival is assumed to be known here (i.e. onset is heard).

The maximum frequency for a specific source location can be calculated thus:

$$\frac{c}{2r(\vartheta + \sin\vartheta)(\cos\phi)}, \tag{3.5}$$

where *r* is the head radius (again assuming a spherical head), *c* is the speed of sound, ϑ is the angle and ϕ the elevation of the source. This essentially represents the frequency that corresponds to half the distance around the head to the contralateral ear. The formula is used to calculate the maximum frequency for a specific source location when endeavouring to extract unambiguous interaural phase differences. This brings up another important issue: radius of the head is not equal from all directions. The radius used here is derived from KEMAR's head *length*: 9.55cm.

Radius derived from head *breadth* is significantly smaller, highlighting the approximations involved in assuming the head as a sphere.

Using head *length* minimises the threshold value used, thus minimising error. Therefore, where appropriate, the 1500 Hz threshold is reduced, in accordance with psychoacoustical and physiological limitations. This reduction is only manifested towards the horizontal extreme of the hemisphere used.

Resolution for extremities in the horizontal plane will be as low as ca 700 Hz (for elevation 0, angle 90). Most values will, however, allow resolution to the threshold of 1500 Hz. Specifically, in higher bins at the FFT resolution employed here (for the 128 point FFT used, the relevant bins at 44.1 kHz sampling rate are 344.53125, 689.0625, 1033.59375 and 1378.125 Hz), 243/183 out of 342 possibilities for higher bins are obtainable (there are only 342 possibilities in the 710 file database due to the symmetry of the dataset used, and the lack of interaural phase difference at 0 and 180 degrees). It is important to note that the formula is based on the assumption that the head is a sphere, and will therefore introduce inaccuracies. However, this is not significant in this case, as the maximum value is just used as a threshold. Actual phase differences are calculated from empirical data.

3.3.4 Non-linear ITD

Calculating the unambiguous IPD threshold frequency is the first step in extracting psychoacoustically relevant IPD, to be used to augment the spherical-head model, aligning it more closely with the empirical data. The goal is to derive a non-linear ITD curve for sub 1500 Hz values for each of the 710 HRTFs in the database. Due to the sheer volume of files, a generic formula is desirable. Also, from a processing point of view, a generalised curve is desirable, to avoid an individual scaling curve for each file. As discussed, the curve is expected to be predominantly > 1 (with an expected approximate ratio of 3:2), as low-frequency ITDs are greater than higher frequency ITDs.

When endeavouring to derive this curve computationally, the previously mentioned resolution of π can be increased to 2π, as source location direction is known. Practically, impulses will always come from the right if the angle is less than 180 degrees (with the exception of 0 and 180 degrees, where there is no IPD in symmetric scenarios). This is always the case for the MIT dataset in question, as only the right hemisphere is being analysed in the symmetric dataset. Right phase will therefore always be expected to be larger/arriving first/leading. IPD can thus be defined as right phase minus left. If there is an anomaly in this calculation (if the right phase does not lead the left), the right phase can be augmented by 2π. A more rigorous approach, which is critiqued in detail below, is to unwrap the phase. This involves looking at the current and previous phase differences and correcting jumps of greater than π by adding/subtracting multiples of 2π. This method becomes more relevant for higher frequencies, where the potential for phase ambiguity is greater. It aims to provide an 'unwrapped' phase for the whole spectrum, as opposed to the psychoacoustically non-ambiguous frequencies (and more specifically limited to those under 1500 Hz, where applicable) that the simpler method addresses.

The code used for the extraction of the non-linear IPD will now be discussed (see *nonlinitd.cpp* in the Chapter3 folder of the accompanying CD-ROM). As this is a subset of the code discussed in the phase unwrapping discussion, below, only a brief overview is offered, to avoid repetition (processes used in the spatialisation tools are also not discussed, as they are dealt with in the more suitable context of the applications considered in the next chapter). Firstly, the unambiguous phase limitation and Woodworth ITD for the location in question are calculated:

```
maxonecycle = (c / (2 * (radianangle + sin(radianangle)) * 9.55 *
               cos(radianelev)));

woodworthitd = (radianangle + sin(radianangle)) * 8.8 *
                cos(radianelev) / c;
```

Each HRTF for the appropriate dataset is opened (3 sampling rates are offered in the real time implementations, so analysis needs to be done at three different sampling rates) and an FFT is performed on the data. The phase of each component is extracted.

```
/* 0Hz and nyq: real */
inl[0] = fftl[0];
inl[1] = fftl[irlength / 2];
inr[0] = fftr[0];
inr[1] = fftr[irlength / 2];

/* mag/phase format: polar, SQUARE(x) is (x)*(x) */
for(i = 2, k = 1; i < irlength; k++, i+=2)
{
      inl[i] = sqrt(SQUARE(fftl[k]) + SQUAREfftl[irlength - k]));
      inl[i+1] = atan2(fftl[irlength-k],fftl[k]);
      inr[i] = sqrt(SQUARE(fftr[k]) + SQUARE(fftr[irlength - k]));
      inr[i+1] = atan2(fftr[irlength-k],fftr[k]);
}
```

To reiterate, processes and practices such as the FFT algorithm, interface and data processes will be discussed in the context of the main applications. An unwrapped phase difference is then calculated. From this value, a real ITD for the bin in question is calculated, and a scaling factor derived. The values are printed to a text file, which allows analysis of each HRTF individually. The curve shape of each HRTF can be retrieved; patterns can be followed etc, as discussed below. The text file is organised thus: angle, elevation, maximum frequency for unambiguous interaural phase difference, Woodworth ITD, then bin by bin phase information (left, right phase, phase difference, derived ITD and scaling factor) up to 1500 Hz, or the maximum realisable frequency without phase ambiguity for each HRTF.

The frequency bins in question are then focused on. Each bin is checked for explicitly (as higher resolution processing is possible by changing the impulse size in *defs.h*). Four frequency bins are relevant here. The value of the bin is incremented by the scale factor for the HRTF in question, and the number of values for that bin is incremented (not all of the HRTFs will provide vales for the two higher bins, due to the restrictions discussed above; the higher bins within the 1500 Hz range may be higher in frequency than the unambiguous limit). Each value is printed before the next file is processed.

```
for(i = 2; i < irlength; i+=2)
{
      freq = (i / 2) * sroverN;

      phasel = inl[i + 1];
      phaser = inr[i + 1];

      phasedif = phaser - phasel;

      while(fabs(phasedif - phasedifold) > pi)
      {
            if(phasedif > phasedifold)
                  phasedif -= twopi;
            else
                  phasedif += twopi;
      }

      realitd = phasedif / (twopi * freq);

      if(woodworthitd)
            scale = realitd / woodworthitd;
      else
            scale = 1.0;

      if(freq < maxonecycle && freq <= 1500)
            fprintf(fdata,"bin: %f\tphl: %f  phr: %f\t\tphdif: %f
                   realitd: %f  scalefact: %f\n", freq, phasel,
                   phaser, phasedif, realitd, scale);

      /* check exact freqs for low/high res... */
      /* 44.1k: 344.531250, 689.062500, 1033.593750, 1378.125000 */
      /* 48k, 96k: 375, 750, 1125, 1500 */
      if(freq == 344.531250 && freq < maxonecycle)
      {
            bin1 += scale;
            bin1no++;
      }
      if(freq == 689.062500 && freq < maxonecycle)
      {
            bin2 += scale;
            bin2no++;
      }
      if(freq == 1033.593750 && freq < maxonecycle)
      {
            bin3 += scale;
            bin3no++;
```

```
        }
        if(freq == 1378.125000 && freq < maxonecycle)
        {
              bin4 += scale;
              bin4no++;
        }

        phasedifold = phasedif;
}
```

Finally, an average scaling factor is arrived at by dividing the total scaling factor for a bin by the number of values accumulated for that bin.

```
nonlinitd[0] = bin1 / bin1no;
nonlinitd[1] = bin2 / bin2no;
nonlinitd[2] = bin3 / bin3no;
nonlinitd[3] = bin4 / bin4no;
```

As with all code, care is taken to delete dynamically allocated memory and close files. In summary, ITDs are derived from empirical IPDs and compared to Woodworth ITDs. Scaling factors are then calculated. The average of all scaling factors for each bin of the low-frequency spectra of the HRTFs is thus stored. Figure 3.5, below illustrates these scaling factors. As processing at 48 kHz and 96 kHz are offered, the appropriate scaling factors are also calculated for these sampling rates. By updating the *defs.h* file, the bin frequencies, the data folder name and the name of the text files, the non-linear curve can be derived for these sampling rates also.

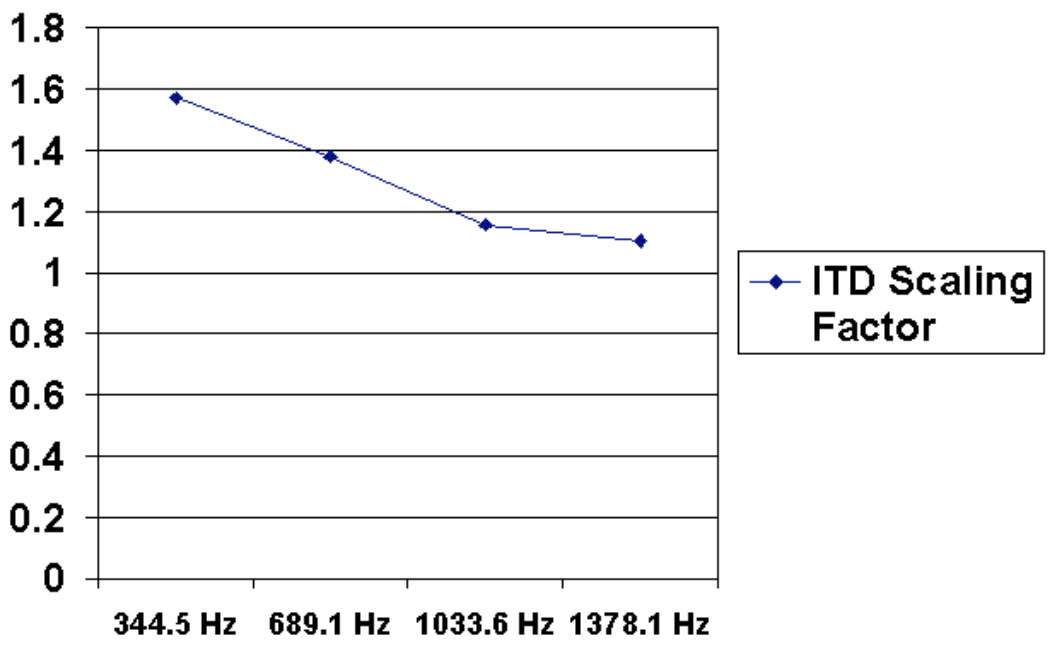


Figure 3.5: Non-linear scaling factors for ITD

This figure illustrates the bins of interest for a 44.1 kHz sampling rate. The dataset can be analysed by looking through the text file printed as part of the program. A higher resolution FFT analysis (essentially achieved by zero padding the impulse responses to 1024 samples) gives a deeper insight into the dataset (see *nonlinitdhighres.txt*). As expected, the scaling factor curve is predominantly > 1, but is not as uniform as a simple 3:2 ratio. Some anomalies are evident. For example, the curve occasionally falls below a ratio of 1:1, with a particularly noticeable dip for low sources behind the listener. Also, for frontal sources, a higher ratio for lower frequencies is also apparently appropriate.

However, the curve does show a general trend, so an averaged model is used across location. Therefore, an accurate, frequency-dependent ITD has been derived based on the literature review presented. The values derived from this Extended Woodworth/Schlosberg Non-linearly Low-frequency Scaled (hence forth Functional) Model are then used in the re-synthesis of the phase spectrum. Using the simple, steady Woodworth model for high frequency, ambiguous, less perceptually relevant phase and a much more accurate phase derived from the empirical data for the lower frequency region model provides a psychoacoustically derived fit of the actual behaviour of ITD.

3.3.5 Implementing a Working Spatialisation Tool using the Non-linear ITD Curve

Having derived a non-linear low-frequency curve, issues involved with implementing these scaling factors, both theoretically and practically will now be discussed.

3.3.5.1 Applying Phase

The values derived from the Functional Model are then used directly in the re-synthesis of the phase spectrum of the required HRTF in the spatialisation application. Magnitude values are interpolated as before, phase values are derived from the Functional Model. ITD is transformed to IPD by simply multiplying each spectral bin frequency by 2π times the ITD (from the opcode implementation discussed in the next chapter):

```
phasel = TWOPI_F * freq * -(itd/2);
phaser = TWOPI_F * freq * (itd/2);
```

The formula used essentially multiplies 2π by the bin frequency to give the amount of rotations around the unit circle per second for the frequency in question. The ITD value is the amount of seconds delay. Therefore, a phase value for each frequency is derived. Timing information is thus introduced into the signal.

Importantly, the leading ear is given a positive orientation, and multiplied by half the ITD value. The lagging ear/ear further from the source is given a negative orientation, and multiplied by minus half the ITD. This apparently unintuitive operation is discussed below.

It is perhaps helpful to view the problem from both an ITD and IPD point of view. ITD is, in these circumstances, a vectorial quantity; it has direction. ITD will be positive on the nearer side to the source, and negative on the farther. From a phase point of view, the leading ear will always have a larger phase, as above. Positive phase goes to the nearer ear, as the source is arriving from that side.

For example, a source from right: the right ear will be given a positive phase, the left, negative. Phase difference will then imply the correct ITD, as it does in the scenario discussed above whereby IPD is extracted as opposed to imposed. Each ear essentially follows its own phase function, as illustrated in figure 3.6, below. This breakdown of phase is also utilised by Zotkin [234, 233].

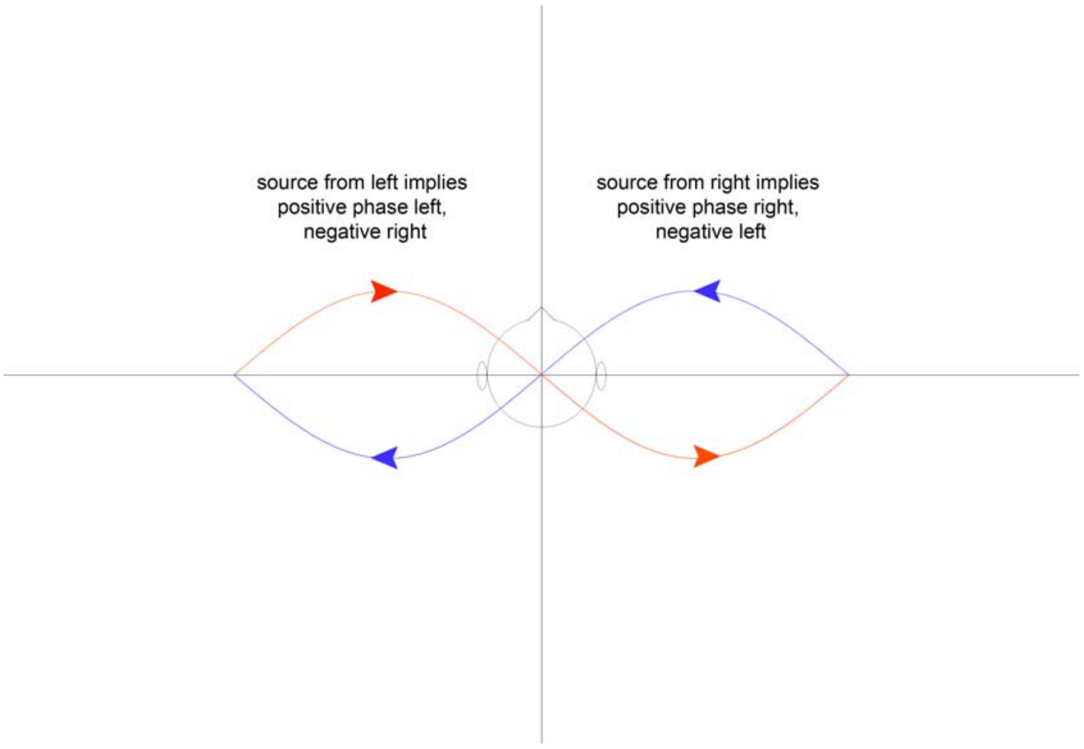


Figure 3.6, IPD orientation

There is an *x* axis switch depending on direction of source arrival. This switch can alternatively be thought of as right to left implying positive to negative, and left to right implying positive to next positive cycle (π to 3π), which is equivalent to wrapping back to negative.

3.3.5.2 Impulse Shifting

Imposing phase values in this way will mean that the zero-centred/zero-phase impulse will wrap to the end of the impulse for the nearer ear (the positive phase essentially implying an earlier onset, which wraps to the end of the impulse; the negative phase is delayed, as it gets to 0 phase later). Moving back into the time domain, it now appears that the nearer ear impulse happens after the further ear, as the nearer impulse has wrapped around to the end of the impulse. See the below figure for an example.

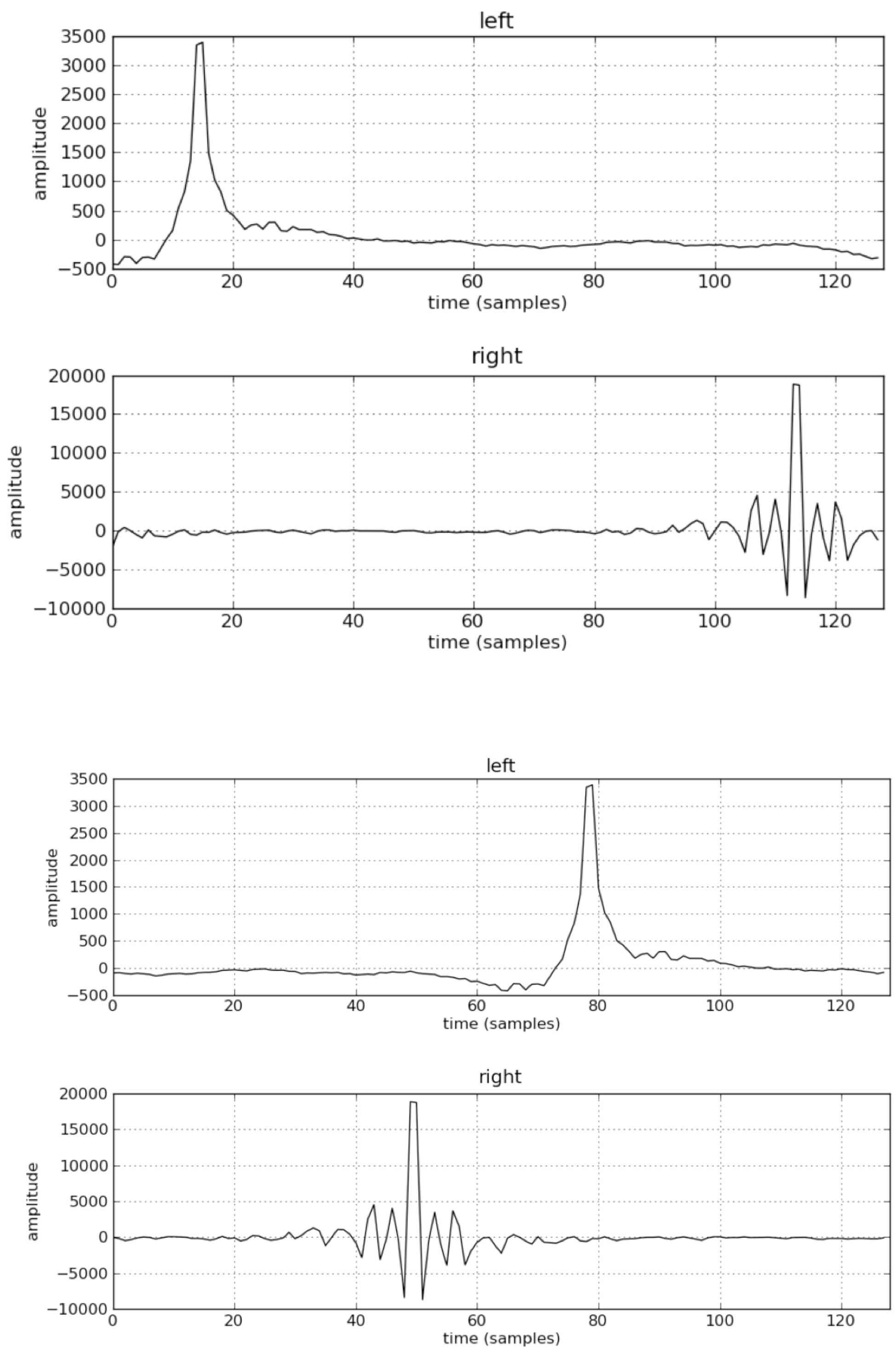


Figure 3.7: A non-shifted (above) and shifted (below) Functional Phase based Stereo HRIR, for a source at 0 degree elevation, 90 degree angle.

This is clearly an unnatural result. Although IPD will be correct, even a casual observation of the impulse illustrates the error in the order of the sound reaching the respective ears. For this reason, the impulse is shifted in time, by half the size of the buffer. This shift ensures a causal filter, and is also performed in the linear phase model in [111]. Essentially, this process adds the correct phase spectra to the zero-phase, magnitude-only impulse and moves it to be centred around the mid point of the filter. The result is a time-accurate and phase-accurate filter. Interestingly, adding this time alignment provides much better localisation. This highlights the importance in correct onset time as well as phase spectra for localisation.

In the figures above, both HRTFs represent the HRTF for a source to the right of the listener (0 degree elevation, 90 degree angle). The right ear should intuitively receive the signal first. As the functionally derived phase wraps around the zero time point, this is not the case, as shown in the first figure. If the impulse is shifted, to be centred around the centre tap of the filter, the situation is rectified. Interaural phase and onset time are now both correct.

An STFT process is required for dynamic sources, as phase is no longer derived to match magnitude (minimum-phase) or static (Phase Truncation). Spatialisation in this scenario is more successful without the impulse shift. As the STFT is used here, the process cannot strictly be defined as convolution. A more accurate description is perhaps an STFT-based filtering process. Magnitudes and phases are imposed on the input sound, but the full convolution output is not saved, as the output is the same size as the input/impulse buffer. The magnitude spectrum is, however, filtered by the impulse and the phase spectrum is also processed to mimic the delays inherent in the phase spectra of the derived impulse. Due to the processing departures from traditional convolution employed here, audible high frequency noise will appear if there are abrupt peaks in an impulse. Usually, impulses start at the beginning of the file, temporally, so these peaks will be windowed in the output. Shifting the impulse is therefore not desirable in STFT implementation (and indeed, introduces noise due to non windowed, centred impulse peaks). Spatial characteristics are however emphasised by repetition.

3.3.5.3 A Step towards Individualisation

The Functional Model uses the Woodworth formula as a basis for initial ITD calculation. As this formula includes a radius parameter, the user can enter an appropriate radius for *their* head. This provides an element of individualisation. However, as the MIT dataset is used, HRTF data will imply listening through KEMAR's ears, with KEMAR's head and torso altering auditory events. Also, as discussed in the next chapter, there appears to be an optimal radius for low-frequency accuracy with regard to comparison to the empirical data, which may not necessarily be a good fit of an arbitrary user's HRTFs. This radius based individualisation is also recognised in [32].

3.4 Phase Unwrapping Issues

The difficulty with phase interpolation has been discussed above. Phase unwrapping was also mentioned, in the context of deriving a non-linear ITD. Unwrapped HRTF phase is often utilised in the literature (see section 3.4.4). However, this task is not trivial, particularly when dealing with complex signals such as HRTFs.

Phase is a periodic quantity. Output of a Fourier Transform can be transformed into polar form, as discussed above, to represent a particular bin's magnitude and phase values. Phase, being a periodic quantity, is represented within a $-\pi$ to π range. Frequently, the spectral component being measured by the bin in question represents the reported phase value, +/- a multiple of full cycles/2π. This implies that the component in question has the phase value reported by the FFT, with some extra full periods of its sinusoidal cycle.

To visualise the phase evolution of, for example, a harmonic of a note from a particular tone, the phase of the partial can be extracted at various equally-spaced points in the tones lifetime (using STFT analysis, for example). This phase can then be viewed over time. Inconsistencies in the phase can be removed by 'unwrapping' it. To achieve a smooth phase plot, jumps in phase are replaced by incorporating the appropriate 2π factor. In the example below, the phase inconsistency can clearly be seen. Adding 2π to the phase output clearly resolves the issue.

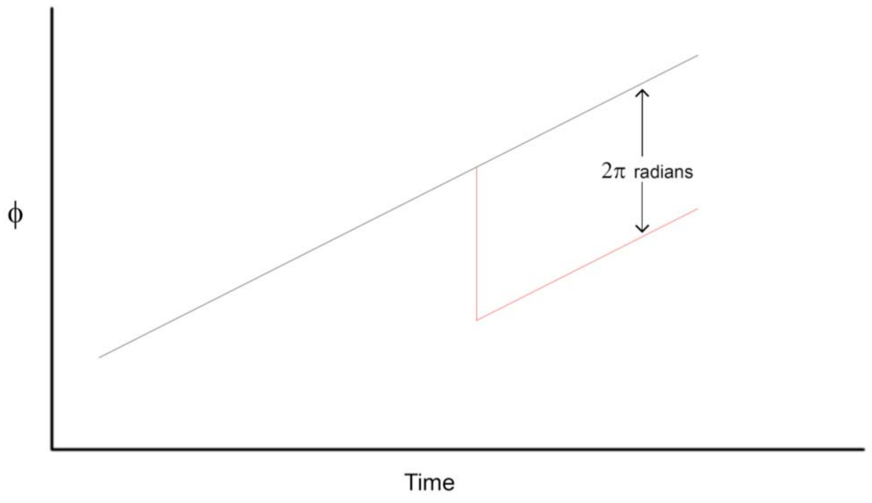


Figure 3.8: Phase unwrapping: the apparent jump in phase is corrected by adding 2π

The partial's phase evolution in time can thus be clearly seen. This is perhaps an intuitive illustration, as the sinusoidal component has clearly mapped out 1 full angular cycle before the 'jump'.

A less intuitive, though equally valid scenario is the vertical unwrapping of phase. If an overall approximately linear delay to all partials in a particular FFT frame is expected, phase can be similarly unwrapped; in this scenario frequency to frequency. A smooth phase plot can thus be obtained, this time illustrating phase over frequency range. For example, if all partials are delayed by a specific time interval, a linear plot would be expected, as illustrated in [14]. The non-linearities in unwrapped HRTF phase plots can be explained by the various non-linearities involved in the system: pinna shape, the role of the head, etc. This implies that some frequencies are delayed for slightly longer/shorter time intervals than others and is to be expected. Also, the expected time delays implied by the phase differences are expected to broadly agree with the data published in [108].

3.4.1 Phase Unwrapping Algorithms

Phase unwrapping algorithms are summarised well in a recent paper by Karam and Oppenheim [96]. One standard approach involves assuming that the differences between consecutive phase values do not go over a particular threshold. This threshold is typically chosen to be π (half a full cycle). If a phase difference of larger than the threshold is detected, multiples of 2π are subtracted/added appropriately, to bring the result into an acceptable/expected range. The algorithm is discussed in more detail below. As mentioned in [96], this threshold method fails if the phase varies rapidly. An upsampling in frequency is suggested to improve accuracy [196]. Once again, the threshold of this upsampling varies from signal to signal, and becomes impractical when implementing efficient analysis/processing on large datasets.

Another common method mentioned in [96] is to use integration. The unwrapped phase is attained using the derivative of the unwrapped phase, which is represented by the imaginary part of the ratio of the derivative of the DFT and the DFT. Once again, this method is problematic, this time with respect to the integration step size. An adaptive decrease of the step size has been suggested.

Unwrapped phase changes in proportion to the proximity of poles or zeros to the unit circle. Another method attempts to consider the zeros close to the unit circle separately. These zeros are located using polynomial factoring. More successful composite algorithms, using either threshold detection or integration with this polynomial factorisation method are reported.

Karam and Oppenheim [96] go on to show that threshold detection and integration based methods both have approximately an 85% success rate in tests with synthetic signals with randomly chosen zeros. Integration methods are significantly slower. Polynomial factoring performs correctly less than 50% of the time, but the composite methods, although significantly slower than the (Matlab implemented) threshold method, perform at a greater than 99% success rate. The paper's conclusion is insightful, as it quantifies phase unwrapping as an unsolved problem. The relatively recent nature of this paper (more recent than the publications that mention phase unwrapping without elaboration, as discussed below) and this conclusion illustrate the importance of considering phase unwrapping more completely.

3.4.2 Phase Unwrapping and the MIT Dataset

It is with the above insights in mind that a thorough investigation of the phase of the MIT HRTF dataset is embarked upon. The code in *phaseunwrap.cpp* (in the 'Chapter3' folder) will now be discussed (again, common details are discussed in the context of the larger scale applications, in the next chapter). Firstly, variables and FFT plans are setup. For each file in the HRTF dataset, the maximum frequency for phase unambiguity is calculated, based on a spherical head with a radius of 9.55 cm, as above. This detail is printed to a text file, and is only required for reference and clarity.

A Woodworth formula based ITD is then calculated and printed to the text file. As a time value is required, radians are used. Each stereo HRTF is read into a left/right buffer (padded in the case of a high resolution process), and transformed to the frequency domain. For simplicity, a magnitude, phase version of the spectral data is preferred. Phase is then unwrapped using the threshold method. This method is used as it is implemented in Matlab [129], which is a commercial tool frequently used for audio digital signal processing analysis and research [128].

Unwrapping is commenced at the first non-zero bin. Phase at 0 Hz will be either 0 or π/-π. A positive real value implies phase of zero, negative π/-π. This can perhaps be visualised clearly as a sinusoidal wave, which is purely real, so must start its evolution at 0 or 180 degrees: a zero crossing. 180 degrees/π and -180 degrees/-π are ambiguous in this scenario. If 0 Hz is included in the unwrapping, this ambiguity can cause difficulties and errors in IPD calculation. A test of the data illustrates that beginning unwrapping at the first non-zero bin of a 128-point FFT gives acceptable IPD values for that bin for all HRTFs in the dataset: slightly above the expected Woodworth calculation [108]. This test is performed initially by ensuring that all IPDs implied by the first non-zero left and right bin give a positive IPD. Later on in the code, all bins below a 1500 Hz limit are tested against the expected Woodworth ITD. This test validates the non-linear scaling factor extraction discussion above, showing that the values for the first bin are as expected (the test is performed to validate the unwrap method for low frequencies). Values are within an acceptable range of ratio variation when compared to the expected ITD (they are all expected to be higher, but also expected to vary due to the complexity of the HRTFs: a range of .6 – 2.7 times the Woodworth ITD is deemed acceptable).

Following this, phase is unwrapped. Unwrapping the phase of the left and right HRTF independently is perhaps more robust than unwrapping the phase difference; both are tested (and give slightly different, but equally imperfect results).

The double-precision floating point absolute value of the phase value in question and the value for the previous phase bin is calculated, and tested against π. If there is a jump of greater than this threshold, a factor of 2π is added/subtracted until the phase is within the desired range. If the current phase is greater than the previous, 2π is subtracted, if less than, it is added. This is performed for the left and right phase, and also the difference between the wrapped phase values. Phase difference of the left and right unwrapped phase is then calculated. In dealing with this hemisphere of data in the symmetrical dataset, right phase will lead left. Therefore, phase difference can be calculated by subtracting the left phase from the right. The code snippet below unwraps the phase of the left channel.

```
while(fabs(phasel - previousphasel) > pi)
{
      if(phasel > previousphasel)
      {
            phasel -= twopi;
            fprintf(fdata,"-");
      }
      else
      {
            phasel += twopi;
            fprintf(fdata,"+");
      }
}
```

The text file (convenient for analysing the data) is laid out as follows (see figure 3.9 for an example): each HRTF file is labelled, the psychoacoustically unambiguous ITD frequency is listed (for reference), and then the Woodworth ITD is listed for the location being analysed. Each bin, from the second to the one preceding the Nyquist Frequency is then listed (the Nyquist Frequency is deemed not relevant, as it is purely real). The left phase is listed, empirically followed by unwrapped. Additions/subtractions of 2π are noted using +/- sign. The right phase is treated similarly. The difference of these unwrapped phases and the ITD this difference implies are listed next. Finally, the unwrapped phase difference is listed, followed by the implied ITD. Although perhaps ungainly, this is a practical and clear way to view the data.

```
el: -40 az: 6

maximum freq for accuracy to one cycle with (spherical) head radius 0.0955m:
11235.938148
wwitd: 0.000041

1  344.53          pl:-0.3518    newpl:-0.3518  pr: -0.1192   newpr: -0.119208  pdif: 0.2326
realitd: 0.000107            pdifunwrap:0.2326         realitd: 0.000107
```

Figure 3.9: Phase unwrapping output file sample entry

3.4.3 Problems with Phase Unwrapping

This novel insight into the phase unwrapping model using comparison of the unwrapped phase differences with expected ITD illustrates how the typically used unwrap method is simply unreliable across the frequency spectrum of all HRTFs in this dataset, and raises concerns about using it generally.

An example of the failure of unwrapping the phase difference, and unwrapping left and right phase and noting the difference will now be given. If looking at the approach of unwrapping the phase difference, a clear example of the problem can be seen at elevation -40, angle 6 (see figure 3.10). At bin 30, a negative ITD is implied. This is in error, as it suggests a source in the hemisphere to the left of the listener, not the hemisphere being analysed. The phase difference between bin 30 and bin 29 is not greater than π. Therefore, no unwrapping occurs. However, a phase difference of greater than π is clearly needed to arrive at an acceptable ITD. If a positive multiple of 2π is included here, the ITD is within the correct range, and the result is the same as that of the independently unwrapped left and right phase. Note that this multiple may be needed sooner, to avoid the low value in bin 29. Running the analysis at high resolution illustrates that an appropriate phase jump is clearly omitted, as negative ITD again occurs (high resolution can simply be achieved by changing the impulse length to 1024; the code will automatically zero pad).

```
29  9991.41      pl:2.6516  -----  newpl:-28.7644  pr: -2.6939  ----  newpr: -27.826606  pdif:
0.9377  realitd: 0.000015          +pdifunwrap:0.9377       realitd: 0.000015
30  10335.94     pl:1.6947  -----  newpl:-29.7212  pr: 0.3410  ----   newpr: -24.791752  pdif:
4.9295  realitd: 0.000076          pdifunwrap:-1.3537       realitd: -0.000021
```

Figure 3.10: Example 1 of phase unwrapping issues

As an example of the case of error in calculation of left and right unwrapped phase independently, at elevation 10, angle 10, at bin 26, the absolute difference of the left phase value to the previous is approximately 2.36, but should be 3.93 to achieve an acceptable ITD (i.e. another 2π needs to be subtracted). Interestingly, unwrapping the phase difference resolves this issue. Also, a higher resolution analysis resolves the issue in this case.

```
25  8613.28      pl:-2.0286  ----  newpl:-27.1614  pr: 2.4578  ----   newpr: -22.674929  pdif:
4.4864  realitd: 0.000083          pdifunwrap:4.4864        realitd: 0.000083
26  8957.81      pl:0.3327  ----  newpl:-24.8001  pr: -0.0118  ----   newpr: -25.144526  pdif: -
0.3445  realitd: -0.000006         +pdifunwrap:5.9387       realitd: 0.000106
```

Figure 3.11: Example 2 of phase unwrapping issues

Simply ensuring that the phase accumulates from the previous is also not a valid solution due to the non-linearities in the dataset. For example, at elevation 0, angle 90, bin 47. The right phase is slightly greater (phase accumulates with a negative orientation here) than the previous, which appears to be the correct and expected behaviour. Another example at the same elevation can be found at angle 120, bin 13, again the right phase. Again, the high resolution text file offers further insight.

The problem is essentially that at places there should be jumps of greater than the π threshold due to the non-linearities in the HRTF data. Manually going through the data to correct it with respect to the expected ITD is impractical. The unwrapping method is simply not reliable for the whole HRTF spectrum. The threshold value of π is chosen, as if a smaller value is chosen, correcting a value between the threshold and π will result in a bigger jump. A value greater than π as a threshold may allow jumps that are in error. It is also worth noting that negative ITD should be considered an extreme error (due to it being counterintuitive: the source travelling a further distance should never arrive first); more subtle mistakes, caused by the same problems, are not highlighted in this analysis.

Higher resolution processing also highlights another failing of phase unwrapping. It can be clearly seen that more 2π multiples have been unwrapped as the process reaches the last illustrated bin (for example at elevation -40, angle 90). Therefore, 128 point processing has not considered enough multiples of 2π due to its lower resolution.

Looking through the file, it is clear that this is a problem for higher frequencies, as phase varies to a greater extent. This is perhaps intuitive when the relative unambiguity of lower frequencies is considered. It is important to note that all phase values in the dataset behave as expected, with respect to the Woodworth ITD and the expected scale factor in the frequency range of interest for the non-linear IPD extraction.

3.4.4 Phase Unwrapping Experimental Insights

This insight into phase unwrapping is particularly pertinent in the domain of HRTF processing. Recognising limitations and potential problems is necessary when looking closely at HRTF phase. It is hoped that this novel approach of comparing unwrapped phase to expected ITD highlights some of the issues with previous work, as well as the care necessary when dealing with complex phase spectra. It is also hoped that this insight strengthens the Functional Model suggested here, as it works within the constraints of the potential inaccuracies of phase unwrapping.

As a coda to this section, a brief discussion of phase unwrapping from the point of view of the literature is offered. Typically, Matlab [128] (threshold method) is mentioned or the method of unwrapping is not discussed. In [216], the authors use IIDs to make IPDs unambiguous in a source localisation application. Results are matched to a HRTF database to get source location, building on previous work [217; the overall work is published as 215]. Furthermore, a parametric HRTF lookup model is introduced. Frequency-dependent scaling factors are discussed in what can be considered an inverse application to how they are used in the present study: localisation as opposed to spatialisation. In a recent work based on Viste’s model [146], the authors implement this inversion. Introduction of the scaling factor is a considerable development; unwrapping of phase is, however, not elaborated upon. It is perhaps more appropriate to focus on the more important low-frequency part of the spectrum, confident that unwrapping is reliable in this band. Unwrapping is also mentioned in the previously discussed [30]. Finally, in [205], linear interpolation of phase is discussed, which implies unwrapping; direct interpolation of phase leads to problems with ambiguities.

In summary, it has been shown that phase unwrapping, upon close inspection, is not an infallible method. Therefore, care and consideration must be taken before employing the method, particularly with complex signals such as HRTFs.

3.5 Conclusion

In conclusion, novel methods of HRTF modelling and interpolation are offered. Phase Truncation uses empirical data directly, exploiting relative insensitivity to phase spectra. The Functional Model uses a psychoacoustically motivated model of phase, extracting an accurate low-frequency ITD. The methods aim to avoid data transformation, compression and processing. This chapter also offers a re-appraisal of phase unwrapping as a frequently-used technique in the domain of HRTF processing, concluding that careful consideration is required before employing the method.

Chapter 4. Algorithm Implementation and Validation

4.1. Introduction

This chapter offers an insight into the implementation of the introduced algorithms as both offline and real time processing tools (Csound opcodes). The detail and nuances of implementation of algorithms developed in the DSP research field is an often-neglected topic. The goal of this chapter is thus to explicitly discuss how to create an efficient, user friendly application using the algorithms discussed in the previous chapter. Considerable effort has been dedicated to this development; it constitutes a significant portion of this work. The dissemination of this work as open source, accessible tools with the option of real-time processing was always a priority; indeed, the relative lack of such tools served as a motivation for this study. A command-line version of the Phase Truncation algorithm is offered; similar implementations for the other algorithms naturally extend from this and are not discussed. Implementation of all algorithms as Csound opcodes, with the possibility of real-time processing is then discussed. Finally, objective and subjective tests which confirm the success of the algorithms are discussed.

4.2. Command-line Implementation

The following discussion of a command line implementation of the Phase Truncation algorithm is based upon [34]. In [34], the code is presented in a more didactic and extended fashion, the intention being to illustrate how to build up a relatively large scale signal processing application using procedural programming methods. As the example used in [34] essentially offers an efficient command-line version of the Phase Truncation algorithm, the same code will be discussed here. In this discussion, the code is similarly deconstructed, but less detail is afforded to more trivial aspects. For all command-line-based examples in this work, certain aspects of file in/output, coding style etc are dealt with in a similar manner. For this reason, these features of the code are discussed only here, and only briefly. Note that all command-line processing uses double floating-point precision. The C++ language is used.

4.2.1 Data Preparation

The MIT dataset is used here. Other datasets can be easily accommodated, such as the LISTEN database [124]. In the 'Chapter4/listen' folder on the accompanying CD-ROM, one of the several available (human subject) LISTEN datasets available is prepared using *datapreparationLISTEN.cpp*. The command-line program below is implemented in *LISTENmover.cpp*. Note that the database is less densely sampled, and is not generalised. In the case of the MIT dataset, storage can be optimised as the complex, diffuse dataset is symmetrical. The processing code must therefore reflect this. A source at 270/-90 degrees on the listener's horizontal plane can be accurately represented by a source at 90 degrees, with the left and right HRTF data switched. Only measurements from 0 – 180 degrees need to be stored in this scenario, significantly reducing memory requirements. A more real world scenario, involving an individualised set of HRTFs, as opposed to a dummy head, requires a full dataset to be stored, as perfect symmetry is unlikely.

datapreparation.cpp constitutes a command line program which prepares the HRIR data for processing (see Appendix 1 in Volume 2 for all code relevant to the command line implementation; all code appendices are also available on the accompanying CD-ROM). Essentially, the code reads each HRIR file (as mentioned above, a HRIR for each measured location is provided), transforms it into the frequency domain and prepares two large files containing all the left and right HRTF data respectively. Storing, opening, accessing and closing one large file is deemed more efficient than dealing with each file separately. Preparing the data in this way allows the opportunity to perform the transform from the time to frequency domain at this offline preparation stage. As the main program, and, more significantly, the real time implementations use frequency-domain processing, performing the transformation at this stage greatly reduces online/real time processing tasks.

As mentioned in [34], certain aspects of code are more easily understood, particularly by readers familiar with the C language, by simply observing the code. For example, variable declaration can become more apparent by observing the variable in use later in the program.

Briefly, a custom header file is used for both the data preparation and main command line application. *defs.h* pre-empts the code, including *stdio.h* for in/output, *stdlib.h* for utility functions, *math.h* for mathematical functions, *string.h* for string manipulation, *sndfile.h* [123] for soundfile processing and *fftw3.h* [62] for Fourier Transform processing:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sndfile.h>
#include <fftw3.h>
```

The length of a mono HRIR is defined as `irlength`, `irpadlength` represents a padded HRIR (overlap add convolution is used, to be discussed below; `overlapsize` is also related). In the command line program, source trajectories are defined using a breakpoint text file. The maximum number of breakpoints (`maxbrkpts`) and a breakpoint function (`bkpt`) are thus also declared.

```
/* correct length for mono impulse */
#define irlength 128
/* padded impulse */
#define irpadlength 256
/* convolution overlap */
```

```
#define overlapsize 127
/* maxumum points in a trajectory */
#define maxbrkpts 101

void bkpt(int *pers, double *els, double *angs, int *noofpoints,
          int maxpts);
```

datapreparation.cpp includes defs.h. Files are iterated through using an elevation and angle variable, which are initialised to minimum values, as per the dataset (see above). Various processing buffers, output file pointers and sound file variables are declared. Also of significance in the declarations section is the FFT setup:

```
/* setup variables */
/* min elev, angle, increment, iterators */
int el = -40, az = 0, inc, i, j, k;
/* input from HRTF file */
double input[2 * irlength];
/* separate input into left and right */
double inl[irlength], inr[irlength], fftl[irlength], fftr[irlength];
/* file pointers */
FILE *foutl, *foutr;
/* fft plans */
fftw_plan forwardl, forwardr;
/* strings for filename */
char filename[14];
char hrtffile[22];
/* file in pointer */
SNDFILE *finhrtf;
/* file info */
SF_INFO *psfinfohrtf;
/* memory for file info */
psfinfohrtf = new SF_INFO;
```

FFTW is used throughout the command line examples. A detailed discussion of the inner workings of FFTW is beyond the scope of this work. The main processing paradigm is to break down the requested task into smaller sub tasks which can be optimised based on the architecture being used. The online manual [64] is an excellent resource for usage of the algorithm. The web page also points to more information for the interested reader [66]. Initial command line implementations utilised a complex FFT by Peterson [162]; it provided flexible and immediately useful code. FFTW2 was also used in early versions (FFTW3 involves a re-design of the API, so is incompatible). However, it was decided, bearing in mind the performance results of FFTW3 [63] to update to FFTW3 processing. These results suggest that FFTW is ‘typically superior to other publicly available FFT software, and is even competitive with vendor-tuned codes’. It is also extremely flexible, and is not specifically designed for a particular architecture. Another particularly relevant feature of FFTW, from the point of view of audio processing, is that it offers ‘fast transforms of purely real input’. In keeping with the ethos of this work, it is also free software.

FFTW3 transforms require transforms to be defined. These transforms will later be implemented. The forward FFTW3 plans (which examine the current processing machine and decide upon optimal implementation) used to transform the left and right HRIRs into HRTFs are thus defined:

```
/* setup fft plans (see fftw documentation) */
forwardl = fftw_plan_r2r_1d(irlength, inl, fftl, FFTW_R2HC,
                            FFTW_ESTIMATE);
forwardr = fftw_plan_r2r_1d(irlength, inr, fftr, FFTW_R2HC,
                            FFTW_ESTIMATE);
```

Multiple plans with the same arguments and different in/output arrays share data to speed up plan generation. This approach is also taken in the main program. The FFTW3 API and interface are intuitive and flexible. As a real FFT is appropriate here (see chapter 1), a real plan is setup, using the following function from the API:

```
fftw_plan_r2r_1d(int n, double *in, double *out, fftw_r2r_kind kind,
                 unsigned flags);
```

The function name dictates the type of plan; here `r2r` refers to real to real, and its dimension, here one-dimensional. The first three arguments are trivial: FFT size, input and output buffer. The `r2r` plan exploits the redundancy involved in performing the DFT on a real audio signal: the negative frequencies are complex conjugates of the positive ones (the DFT returns the sampled spectrum from –Nyquist Frequency to +Nyquist Frequency, as discussed in chapter 1). Also, the values of 0 Hz and the Nyquist Frequency will always be real, so discarding the imaginary part of this complex number allows for a spectral output buffer equal in size to the input time-domain buffer. As discussed below, care must be taken to treat these values correctly when processing. The `kind` argument takes one of a number of predefined kinds of transform. `FFTW_R2HC` is appropriate here: real audio data to 'half complex' spectral data. The spectral output buffer contains the transformed data, which is stored in the format:

$r_0, r_1, r_2, ..., r_{n/2}, i_{(n+1)/2-1}, ..., i_2, i_1$

r_k is the real part of the kth output, and i_k is the imaginary part. In this case, frequency bin k has its real part at `out[k]` and its imaginary part at `out[irlength-k]`. Finally, `flags` completes the parameter list. It is possible to attempt to prepare the best plan for the architecture being used, by running several FFTs (clearly time consuming but finds the best FFT for the system in question: increasing initialization time, decreasing run time). `FFTW_ESTIMATE` does not do this, but prepares a reasonable plan, and is used here as this is not a real time application (although using real FFT processing, pre-processing data and other optimisations make it very efficient). Plans can then be executed at will by the programmer; input data can be updated by simply updating the input array. This division of labour between planning and execution characterizes FFTW3.

Also of interest in the declaration stage of the data preparation program is soundfile declaration. *libsndfile* is used for sound file processing. Again, *libsndfile* is an open source sound file processing library, written in C. It allows flexible processing of audio files, with an intuitive API. Relevant functions will be discussed as they arise in the code. At declaration (see above), a soundfile pointer is setup, as well as a pointer to the structure that will contain data on the soundfile. Memory is then allocated to this structure.

Each of the 368 HRIR files is then processed. The program assumes the HRIR files are all in a folder called ‘diffuse’ in the working directory. The nomenclature of the MIT dataset is intuitive and clear, but implies some inelegant code from an iteration point of view. As discussed above, the amount of measurements per elevation varies, and the occasion arises whereby the incrementation labelling is not constant (-40 and 40 degree elevation alternate between 6 and 7, with the exception of every 7th value, which repeats a 6!). Opening the appropriate file is dealt with using appropriately formatted strings (concatenating an updated file name to the appropriate folder name), updated using elevation and angle incrementation.

```
/* prep for file open string */
strcpy(hrtffile,"diffuse/");

/* prep file names */
if(az < 10)
      sprintf(filename, "H%de00%da.wav", el, az);
else if(az >= 10 && az < 100)
      sprintf(filename, "H%de0%da.wav", el, az);
else if(az >= 100)
      sprintf(filename, "H%de%da.wav", el, az);

/* sort out incrementation based on elev */
if(el == -40)
{
      if(inc != 6 || j % 7 == 0)
            inc = 6;
      else inc = 7;
}
else if(el == -30 || el == 30)
      inc = 6;
else if(el == -20 || el == -10 || el == 0 || el == 10 || el == 20)
      inc = 5;
else if(el == 40)
{
      if(inc != 6 || (j - 276) % 7 == 0)
            inc = 6;
      else inc = 7;
}
else if(el == 50)
      inc = 8;
else if(el == 60)
      inc = 10;
else if(el == 70)
      inc = 15;
else if(el == 80)
      inc = 30;
else if(el == 90)
```

```
      inc = 0;

/* put together for full name */
strcat(hrtffile, filename);
```

The *libsndfile* file open function returns a null pointer if it fails to open the requested file. The conditional statement below checks for this, returning an error if the file cannot be opened. This construct is used throughout. The function also indicates how the file should be opened (read/write) and the structure used to store data on the file (previously allocated). The data is read in frames, as opposed to samples, which implies two samples per frame for stereo, etc.

```
/* open appropriate file */
if(!(finhrtf = sf_open(hrtffile, SFM_READ, psfinfohrtf)))
{
      printf("error opening file\n");
      exit(1);
}
```

Each stereo interleaved file can then be read and simply separated into mono left and right channels. In the same loop, the buffers are scaled down a little, to avoid any potential distortion on output (this is more of an issue with the minimum-phase and Functional Models, as the data transformations involved can introduce increased pressure peaks). These buffers can be transformed to the frequency domain using the FFTW plans, by calling `fftw_execute`.

```
/* read in file */
sf_readf_double(finhrtf, input, irlength);
/* close file */
sf_close(finhrtf);

/* put (double: -1.0 to +1.0) input into seperate left and right
buffers, scale a little */
for(i = 0; i < irlength; i++)
{
      inl[i] = input[2 * i] * .65;
      inr[i] = input[(2 * i) + 1] * .65;
}

/* fft */
fftw_execute(forwardl);
fftw_execute(forwardr);
```

Another optimisation is then performed. As discussed, the Phase Truncation algorithm will use polar spectral information: magnitude and phase, as opposed to rectangular real and imaginary. The other algorithms (minimum-phase and Functional) will use magnitude values directly, and derive phase, so can use the same data files. For convenience of processing, the real valued 0 Hz and Nyquist Frequency values are stored in the first and second point of the buffer, then the magnitude and phase at each bin, sequentially. This involves a re-ordering of the FFTW layout, grouping magnitude and phase in pairs, for ease of processing. It is important to note that 0 Hz and the Nyquist Frequency are stored as real values, thus maintaining their polarity, which dictates their phase (as discussed in the previous chapter). Spectral data is written, in binary form, to the 'datal.raw' and 'datar.raw' files.

```
/* 0Hz and nyq */
inl[0] = fftl[0];
inl[1] = fftl[irlength / 2];
inr[0] = fftr[0];
inr[1] = fftr[irlength / 2];

/* mag/phase: polar */
for(i = 2, k = 1; i < irlength; k++, i += 2)
{
      inl[i] = sqrt(SQUARE(fftl[k]) + SQUARE(fftl[irlength - k]));
      inl[i+1] = atan2(fftl[irlength-k],fftl[k]);
      inr[i] = sqrt(SQUARE(fftr[k]) + SQUARE(fftr[irlength - k]));
      inr[i+1] = atan2(fftr[irlength-k],fftr[k]);
}
```

This data is written to a large left and right optimised spectral file. Incrementation of angle and elevation (if appropriate) is then performed.

```
/* incrementation */
az = az + inc;

if(j == 28 || j == 59 || j == 96 || j == 133 || j == 170 || j == 207
   || j == 244 || j == 275 || j == 304 || j == 327 || j == 346
   || j == 359 || j == 366)
{
      /* change elevation,reset variables */
      el = el + 10;
      az = 0;
      inc = 0;
}
```

To conclude, dynamically allocated memory is freed, files are closed and FFTW plans destroyed:

```
/* clear memory, close files */
delete psfinfohrtf;
fclose(foutl);
fclose(foutr);
fftw_destroy_plan(forwardl);
fftw_destroy_plan(forwardr);
```

This program can be compiled thus:

```
g++ datapreparation.cpp -o dataprep -I/usr/local/include
      -L/usr/local/lib -lsndfile -lfftw3
```

This assumes that libsndfile and FFTW3 are installed in the default locations. Thus double precision HRTF datafiles are prepared. For the Csound opcodes, floating point precision is used. The double precision data is simply cast to float and written to a float file. Thus the Csound datafiles are more compact; floating point precision is deemed adequate.

4.2.2 Main Program

The main program, *binauralmover.cpp*, uses this prepared data. Essentially, its task is to use the HRTF data to artificially recreate user-defined source trajectories. Smooth, artefact-free dynamic source behaviour is desirable, thus the Phase Truncation algorithm is employed to interpolate HRTFs at run time. Four-point linear magnitude interpolation is employed. The difficulties with phase interpolation are addressed using the Phase Truncation algorithm. The *defs.h* file used by the preparation program is also used by the main program. As discussion of the finer detail of the code can become involved, figure 4.1, below illustrates an overview of the Phase Truncation implementation, as well as the Functional Phase Model and Minimum Phase approaches. It can therefore be referred to as required, and also serves to compare the overall approaches to phase processing employed.

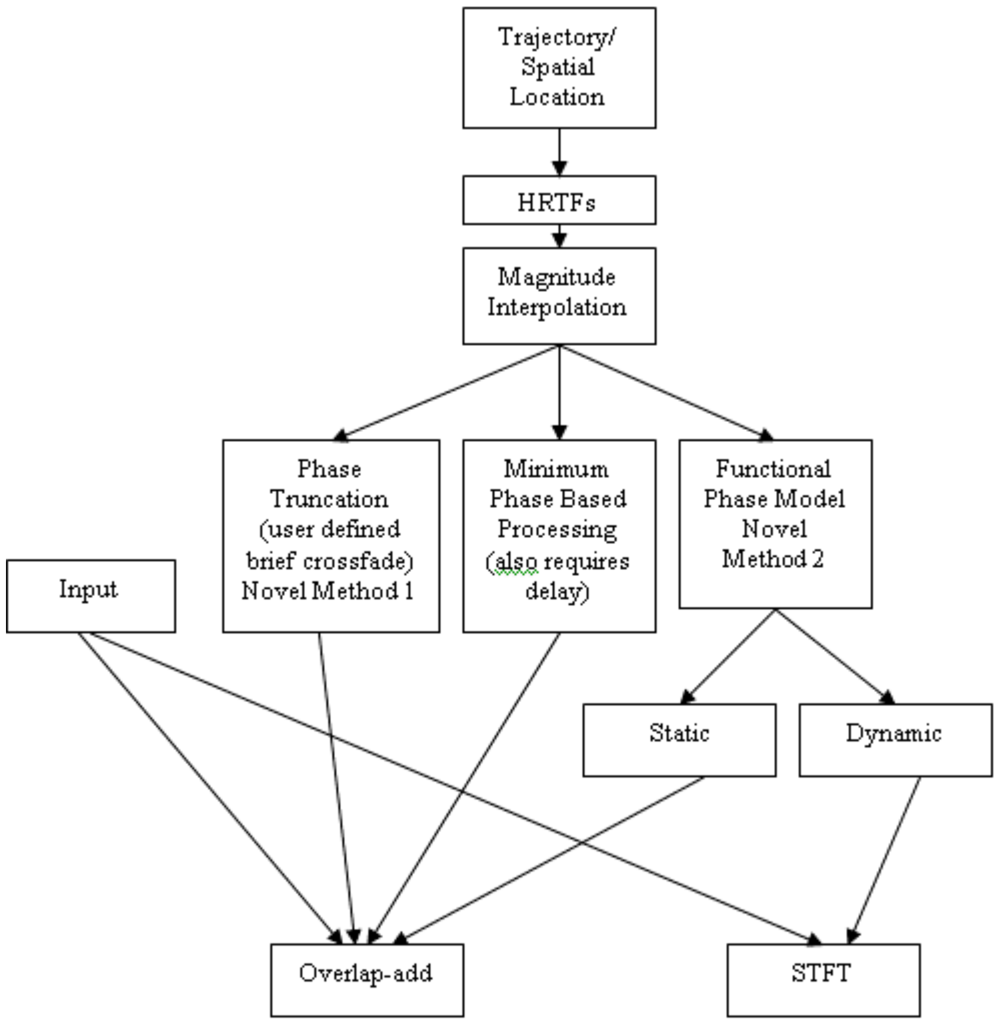

Trajectory/ Spatial Location
HRTFs
Magnitude Interpolation
Phase Truncation (user defined brief crossfade) Novel Method 1
Minimum Phase Based Processing (also requires delay)
Functional Phase Model Novel Method 2
Input
Static
Dynamic
Overlap-add
STFT

Figure 4.1: An overview of the main HRTF processing involved in the algorithms discussed; reading a source location and processing using HRTFs accordingly.

In the declarations section, the number of measurements per elevation is stored in an array which is accessed by the interpolation algorithm:

```
int elevationarray[14] = {56, 60, 72, 72, 72, 72, 72, 60, 56, 45,
                          36, 24, 12, 1};
```

Arrays of pointers to double are used to store and access the HRTF files:

```
/* arrays to store addresses of where all left and right hrtfs are
   stored: arrays of pointers to double. */
double *hrtfarrayl[14][37], *hrtfarrayr[14][37];
```

Dynamic trajectories, crossfades, interpolation, convolution, previous data used for fading out when applicable, soundfile in/output and FFT transformations all require variable declarations [34]. More FFTW plans are needed here than in the data preparation program. In this case, inverse transforms are required to perform inverse FFTs (type `FFTW_HC2R`). Also, transforms of different sizes are needed, as both zero-padded and non-zero-padded buffers need to be processed.

```
/* fftw plans */
fftw_plan invhrtfl, invhrtfr, forhrtflpad, forhrtfrpad, forin;
fftw_plan invoutl, invoutr, invoutlold, invoutrold;

invhrtfl = fftw_plan_r2r_1d(irlength, hrtflinterp, hrtfltd,
                            FFTW_HC2R, FFTW_ESTIMATE);
invhrtfr = fftw_plan_r2r_1d(irlength, hrtfrinterp, hrtfrtd,
                            FFTW_HC2R, FFTW_ESTIMATE);
forhrtflpad = fftw_plan_r2r_1d(irpadlength, hrtflpadtd,
                               hrtflpadspec, FFTW_R2HC,
                               FFTW_ESTIMATE);
forhrtfrpad = fftw_plan_r2r_1d(irpadlength, hrtfrpadtd,
                               hrtfrpadspec, FFTW_R2HC,
                               FFTW_ESTIMATE);
forin = fftw_plan_r2r_1d(irpadlength, inbuf, inspec, FFTW_R2HC,
                         FFTW_ESTIMATE);
invoutl = fftw_plan_r2r_1d(irpadlength, outlspec, outl, FFTW_HC2R,
                           FFTW_ESTIMATE);
invoutr = fftw_plan_r2r_1d(irpadlength, outrspec, outr, FFTW_HC2R,
                           FFTW_ESTIMATE);
invoutlold = fftw_plan_r2r_1d(irpadlength, outlspecold, outlold,
                              FFTW_HC2R, FFTW_ESTIMATE);
invoutrold = fftw_plan_r2r_1d(irpadlength, outrspecold, outrold,
                              FFTW_HC2R, FFTW_ESTIMATE);
```

User input is required to decide the size of the crossfades. As discussed above, one buffer of 128 samples (which can be thought of as the processing *control rate*) may be sufficient for noisy sources, which can hide any possible discontinuity which may not be completely removed by the fade. More narrow-band sources are less forgiving, and typically require a longer fade. Values of 1–24 processing buffers are enforced. An appropriate generic value is 8.

```
/* setup crossfades: over user defined number of convolution cycles
*/
printf("enter number of processing buffers for fades (>1),8 is good
       for musical source,less for noisy sources:\n");
scanf("%d",&fade);
if(fade <= 0)
{
      printf("fade number must be positive, exiting\n");
      exit(1);
}
if(fade > 24)
      fade = 24;
fadebuffer = fade * irlength;
```

The user is also prompted for the name of a mono input file, which will be spatialised. In this case, datafiles are limited to a sampling rate of 44.1 kHz (the real-time solutions offer more flexibility; extension to consider other sampling rates is trivial), so input file sampling rate should match.

HRTF datafiles are then read (from the working directory) and stored. An efficient data structure is used to store and access the HRTFs. Each elevation (14 values) is iterated through. On each iteration, the appropriate number of angle values/HRTFs are iterated through (remembering the symmetric nature of the dataset). Memory is allocated for each HRTF, and filled with the appropriate data (from the left/right datafiles). A three-dimensional structure is thus derived; elevation increments on the *x* axis, angle increments on the *y* axis, and spectral bins on the *z* axis. A two-dimensional array of pointers is possibly more intuitive from a visualisation point of view. Each array point represents an elevation and angle. The pointers store the first memory location of the buffer storing the appropriate HRTF. Thus data access is efficient and intuitive.

```
/* store files */
for(i = 0; i < 14; i++)
      for(j = 0; j < elevationarray[i] / 2 + 1; j++)
      {
            /* hrtfarray[i][j] = &hrtfarray[i][j][0] */
            hrtfarrayl[i][j] = new double [irlength];
            hrtfarrayr[i][j] = new double [irlength];
            fread(hrtfarrayl[i][j],sizeof(double), irlength,
            hrtfleft);
            fread(hrtfarrayr[i][j],sizeof(double), irlength,
            hrtfright);
      }
```

Next, *libsndfile* is used to setup the data structure storing the details on the output file, which will be stereo, have a sampling rate of 44.1 kHz and the same format (bit rate and file type) as the input file.

```
/* initialise the SF_INFO structure (need to do this before opening
   file!), same as input but stereo */
psfinfoout->samplerate = psfinfoin->samplerate;
psfinfoout->channels = 2;
```

```
psfinfoout->format = psfinfoin->format;
```

Dynamic source trajectories are read using a breakpoint file, the parsing of which is dealt with by an external function, defined in *binauralmoverfunctions.cpp*.

```
/* function to read, check and store trajectory */
bkpt(percentages, elevs, angles, &countbkp, maxbrkpts);
```

Pointers to buffers to store percentage, elevation and angle values, an integer to keep track of the number of breakpoints and a maximum breakpoint control value constitute the arguments.

The function itself prompts the user to enter the name of a text file that defines the trajectory. It then reads through the file, storing each percentage, elevation and angle value. Elevations are truncated to the legal range (data is only available from -40 degrees, presumably due to the floor in the measurement room, and up to 90 degrees: directly above the listener), percentage values must accumulate and be between 0 and 100. The function is completed when a percentage value of 100 or the end of the file is recognised. The last percentage value should also be 100. The number of points (passed as an address to allow the function to update) is incremented on each iteration.

```
void bkpt(int *pers, double *els, double *angs, int *noofpoints, int
maxpts)
{
      /* file details */
      FILE *finbkp;
      char bkpfilename[100];
      int i;

      printf("enter breakpoint file (integer value
             percentages),include.txt extension (<100
      characters):\n");
      scanf("%s",bkpfilename);

      if(!(finbkp = fopen(bkpfilename,"r")))
      {
            printf("error opening breakpoint file, exiting\n");
            exit(1);
      }

      for(i = 0; i < maxpts; i++)
      {
            /* read input from file */
            if(!feof(finbkp))
            {
                  fscanf(finbkp,"%d",&pers[i]);
                  fscanf(finbkp,"%lf",&els[i]);
                  if(els[i] > 90.0)
                        els[i] = 90.0;
                  if(els[i] < -40.0)
                        els[i] = -40.0;
                  fscanf(finbkp,"%lf",&angs[i]);

                  /* do checks */
                  /* legal % values ? */
                  if(pers[i] > 100 || pers[i] < 0)
                  {
                        printf("error, breakpoint file must run from
                               0 to 100, exiting\n");
                        exit(1);
                  }
                  /* percentage accumulation */
                  if(i > 0 && pers[i] <= pers[i - 1])
                  {
                        printf("error, percentage values must
                               accumulate...%d is not > %d,
                        exiting\n",pers[i],pers[i-1]);
                        exit(1);
                  }

                  /* end at 100% */
                  if(pers[i] == 100)
                  break;

            *noofpoints = *noofpoints + 1;
            }
            else
            break;
      }

      /* check last value is 100 */
      if(pers[*noofpoints] != 100)
      {
            printf("error, percentage values must conclude with 100,
                   not %d, exiting",pers[i]);
            exit(1);
      }

      /* close file */
      fclose(finbkp);
}
```

The following therefore implies a trajectory from 0 to 90 degrees:

```
0 0 0
100 0 90
```

More complete input checking is desirable. However, this is essentially a test-bed/didactic [34] program; the real time programs are released, so are intended to be more robust.

Breakpoints are iterated through in the main loop of the main program. A loop within this loop runs to the appropriate percentage of the overall length of the convolved output (input: data available from input file + impulse: HRIR: 128 samples - 1). Each breakpoint percentage is considered here. A nested do, while loop processes the input at the control rate (`irlength`).

```
/* main loop */
for(x = 0; x < countbkp; x++)
{
      start = sum;
      /* run to full length of convolved output */
      sum = (int)((psfinfoin->frames + irlength - 1) * percentages[x
                  + 1] / 100.0);

      do
      {
      …
      }
      while (k < sum);
}
```

Buffer by buffer processing can now be discussed. Of immediate interest is the calculation of the angle and elevation values. Simple linear interpolation is used from point to point in the breakpoint file. It is important to consider the breakpoints section by section here.

```
/* change elev and angle according to bkpt file */
elev = elevs[x] + (elevs[x + 1] - elevs[x]) * (double)(k - start) /
      (sum - start);
angle = angles[x] + (angles[x + 1] - angles[x]) * (double)(k –
        start) / (sum - start);
```

Elevations and angles are read using an indexing system. From the interpolated values, the nearest measured indices are calculated. For example, an elevation of 5 returns the value 4.5. The lower relevant elevation data is index 4, the higher is index 5. The fractional value returned by the above code is used to determine the relative amount of each elevation index to use in the interpolation process (only the high value is needed by the interpolation formula used, see below). For example, for an elevation of 7.5 degrees index 4 and 5 are again used (0 degree and 10 degree measurements respectively). 25% of the lower elevation and 75% of the higher elevation will be used.

```
/* two nearest elev indices */
/* to avoid recalculating  */
elevindexstore = (elev - minelev) / elevincrement;
elevindexlow = (int)elevindexstore;

if(elevindexlow < 13)
      elevindexhigh = elevindexlow + 1;
else
      elevindexhigh = elevindexlow;  /* highest index reached */

/* get percentage value for interpolation */
elevindexhighper = elevindexstore - elevindexlow;
```

Angle values are treated similarly to elevation values, if a little more flexibly. For example, a desired location of 270 degrees can be inputted literally, or by using -90 or indeed any value that results in a value of 270 when calculated modulus 360.

```
while(angle < 0.0)
      angle += 360.0;
while(angle >= 360.0)
      angle -= 360.0;
```

A subtle revision to the indexing system follows, this time used to check for a cross fade. The *nearest* index is required this time, to check if the source trajectory has moved on to a nearer measured point. The nearest angle index is calculated modulus the number of values at the appropriate elevation, which allows processing through 0 degrees.

```
/* as above,lookup index, used to check for crossfade */
elevindex = (int)(elevindexstore + 0.5);

angleindex = (int)(angle / (360.0 / elevationarray[elevindex]) +
                   0.5);
angleindex = angleindex % elevationarray[elevindex];
```

This preempts the main crossfade check. If either of the nearest indices are not equal to their previous values, a crossfade is initiated (provided the application has performed at least one processing control period). A warning is printed if already in a crossfade period. Due to the brief nature of crossfades, this is unlikely, but may occur in complex trajectories (where angle and elevation values are changing rapidly) or very swift trajectories. To avoid overlapping crossfades, the user may reduce the size of the crossfades to a level that is still tolerable with regard to noise, or slightly change the trajectory (bearing in mind limitations of the auditory system regarding swiftly moving sources). However, overlapping crossfades may not be audible, as a new crossfade is started in this scenario. If the previous crossfade is far enough into its evolution, the switch to the new crossfade may be inaudible. Initialising a crossfade involves storing the old HRTF data for fade out and setting/resetting crossfade variables. If any of the indices change, the nearest available phase value is updated (this *will* occur on the first run, as old index values are intentionally initialised to illegal values). Angle and elevation indices are used to read this HRTF data, whose phase spectrum will be used in the synthesis of the interpolated HRTF. Pointers are used to access the correctly indexed HRTF arrays. Hemispheric data is treated appropriately; the left and right channel are switched if required. It is at this point that reading files and ultimately understanding angles and elevations using indexing becomes clear.

```
/* crossfade happens if index changes:nearest measurement changes */
if (oldelevindex != elevindex || oldangleindex != angleindex)
{
      if(k > 0)
      {
            /* warning on overlapping fades */
            if(cross)
            {
                  printf("\nwarning: fades are overlapping: this
                         could lead to noise: reduce fade size or
                         change trajectory");
                  cross = 0;
            }
            /* reset l */
            l = 0;
            crossfade = 1;
            for(i = 0; i < irpadlength; i++)
            {
                  hrtflpadspecold[i] = hrtflpadspec[i];
                  hrtfrpadspecold[i] = hrtfrpadspec[i];
            }
      }

      if(angleindex > elevationarray[elevindex] / 2)
      {
            hrtfpl = hrtfarrayl[elevindex][elevationarray[elevindex]
                     - angleindex];
            hrtfpr = hrtfarrayr[elevindex][elevationarray[elevindex]
                     - angleindex];
            for(i = 0; i < irlength; i++)
            {
                  currentphasel[i]=hrtfpr[i];
                  currentphaser[i]=hrtfpl[i];
            }
      }
      else
      {
            hrtfpl = hrtfarrayl[elevindex][angleindex];
            hrtfpr = hrtfarrayr[elevindex][angleindex];
            for(i = 0; i < irlength; i++)
            {
                  currentphasel[i]=hrtfpl[i];
                  currentphaser[i]=hrtfpr[i];
            }
      }
}
```

The next part of the code uses similar constructs to the above. The four nearest empirical HRTFs are calculated and read, to be used in the magnitude interpolation operation, two for the low elevation in question, two for the high. Similarly to the elevation values, relative weightings are calculated. This weight calculation and the reading of the first of the four points are illustrated below (the other three use identical constructs).

```
/* avoid recalculation */
angleindexlowstore = angle / (360.0 / elevationarray[elevindexlow]);
angleindexhighstore = angle / (360.0 /
                      elevationarray[elevindexhigh]);

/* 4 closest indices, 2 low and 2 high */
angleindex1 = (int)angleindexlowstore;

angleindex2 = angleindex1 + 1;
angleindex2 = angleindex2 % elevationarray[elevindexlow];

angleindex3 = (int)angleindexhighstore;

angleindex4 = angleindex3 + 1;
angleindex4 = angleindex4 % elevationarray[elevindexhigh];

/* angle percentages for interp */
angleindex2per = angleindexlowstore - angleindex1;
angleindex4per = angleindexhighstore - angleindex3;

/* read 4 nearest HRTFs  */
/* switch l and r */
if(angleindex1 > elevationarray[elevindexlow] / 2)
{
      hrtfpl = hrtfarrayl[elevindexlow][elevationarray[elevindexlow]
               - angleindex1];
      hrtfpr = hrtfarrayr[elevindexlow][elevationarray[elevindexlow]
               - angleindex1];
      for(i = 0; i < irlength; i++)
      {
            lowl1[i] = hrtfpr[i];
            lowr1[i] = hrtfpl[i];
      }
}
else
{
      hrtfpl = hrtfarrayl[elevindexlow][angleindex1];
      hrtfpr = hrtfarrayr[elevindexlow][angleindex1];
      for(i = 0; i < irlength; i++)
      {
            lowl1[i] = hrtfpl[i];
            lowr1[i] = hrtfpr[i];
      }
}
```

Magnitude intepolation is then performed. Phase values are imposed in accordance with the nearest empirical HRTF's phase spectrum to the derived location. The resulting HRTF is stored in rectangular form, in the ordering format required by FFTW for inverse transform.

At this point, the insight into real FFT polar processing offered in the previous chapter becomes relevant from an implementation point of view. As mentioned before, the values for 0Hz and the Nyquist Frequency are stored in the spectral data files as purely real values. The magnitude values can be simply derived from the purely real values by taking the floating-point absolute value. As the imaginary part of the values will always be 0, the floating point absolute value will always give an accurate magnitude value. However, the phase value must be considered. The polarity of the real value will dictate its phase. A positive real value implies a phase of 0, negative a phase of π/- π, as per the inverse tangent function. Therefore, interpolated magnitudes are calculated using the floating absolute value of the real 0 Hz and Nyquist Frequency values. Phase of these values are enforced by observing the nearest measured phase spectrum polarity for these values. If the nearest phase values imply a phase of π/-π (due to a negative real value), a negative polarity is imposed upon the rectangular result (as above, the polar form used for interpolation processing is transformed back to rectangular for FFTW processing).

The magnitude interpolation process essentially interpolates the low and high elevation values (which may have different parameters due to the non-unform number of empirical points per elevation). The results are then interpolated.

Briefly, the linear interpolation formula works by simply adding the first value to the difference between the second and first value multiplied by the proportion of the second value required: a + (b - a) * proportion of b. It was decided that linear interpolation is both appropriate and sufficient here, particularly when considering the real time implementations as the main focus of this work (from an efficiency point of view), as mentioned in [93].

After dealing with 0Hz and the Nyquist Frequency, storing real values with appropriate polarity, organised in the FFTW format, the rest of the data is dealt with. The code below essentially interpolates magnitudes, applies phase and transforms the result back to rectangular form, storing it in FFTW format.

```
/* magnitude interpolation */
/* 0hz and Nyq real values */
/* organised in format of fftw */
magllow = fabs(lowl1[0]) + (fabs(lowl2[0]) - fabs(lowl1[0])) *
          angleindex2per;
maglhigh = fabs(highl1[0]) + (fabs(highl2[0]) - fabs(highl1[0])) *
           angleindex4per;
magrlow = fabs(lowr1[0]) + (fabs(lowr2[0]) - fabs(lowr1[0])) *
          angleindex2per;
magrhigh = fabs(highr1[0]) + (fabs(highr2[0]) - fabs(highr1[0])) *
           angleindex4per;
magl = magllow + (maglhigh - magllow) * elevindexhighper;
magr = magrlow + (magrhigh - magrlow) * elevindexhighper;
if(currentphasel[0] < 0.0)
      hrtflinterp[0] = -magl;
else
      hrtflinterp[0] = magl;
if(currentphaser[0] < 0.0)
      hrtfrinterp[0] = -magr;
else
      hrtfrinterp[0] = magr;

magllow = fabs(lowl1[1]) + (fabs(lowl2[1]) - fabs(lowl1[1])) *
          angleindex2per;
maglhigh = fabs(highl1[1]) + (fabs(highl2[1]) - fabs(highl1[1])) *
```

```
           angleindex4per;
magrlow = fabs(lowr1[1]) + (fabs(lowr2[1]) - fabs(lowr1[1])) *
          angleindex2per;
magrhigh = fabs(highr1[1]) + (fabs(highr2[1]) - fabs(highr1[1])) *
           angleindex4per;
magl = magllow + (maglhigh - magllow) * elevindexhighper;
magr = magrlow + (magrhigh - magrlow) * elevindexhighper;
if(currentphasel[1] < 0.0)
      hrtflinterp[irlength/2] = -magl;
else
      hrtflinterp[irlength/2] = magl;
if(currentphaser[1] < 0.0)
      hrtfrinterp[irlength/2] = -magr;
else
      hrtfrinterp[irlength/2] = magr;

/* other values are complex, in fftw format */
for(i = 2, j=1; i < irlength; j++, i+=2)
{
      /* interpolate high and low magnitudes */
      magllow = lowl1[i] + (lowl2[i] - lowl1[i]) * angleindex2per;
      maglhigh = highl1[i] + (highl2[i] - highl1[i]) *
                 angleindex4per;

      magrlow = lowr1[i] + (lowr2[i] - lowr1[i]) * angleindex2per;
      magrhigh = highr1[i] + (highr2[i] - highr1[i]) *
                 angleindex4per;

      /* interpolate high and low results,use current phase */
      magl = magllow +  (maglhigh - magllow) * elevindexhighper;
      phasel = currentphasel[i + 1];

      /* polar to rectangular, organised in fftw order */
      hrtflinterp[j] = magl * cos(phasel);
      hrtflinterp[irlength - j] = magl * sin(phasel);

      magr = magrlow + (magrhigh - magrlow) * elevindexhighper;
      phaser = currentphaser[i + 1];

      hrtfrinterp[j] = magr * cos(phaser);
      hrtfrinterp[irlength - j] = magr * sin(phaser);
}
```

Interpolated HRTFs are transformed back to the time domain using a half complex to real transform (FFTW_HC2R), zero padded, and transformed back to the frequency domain. Zero padding is necessary for the overlap-add convolution process (to avoid truncating the output). Overlap data is stored: the previous processed output overlaps with the current. In the case of crossfades, the overlap will be the previous processed output for the first of multi-buffer crossfades, and the previous processed old data for the rest.

```
fftw_execute(invhrtfl);
fftw_execute(invhrtfr);

/* scale and pad */
for(i = 0; i < irlength; i++)
{
      hrtflpadtd[i] = (hrtfltd[i] / irlength);
      hrtfrpadtd[i] = (hrtfrtd[i] / irlength);
}

for(i = irlength; i < irpadlength; i++)
{
      hrtflpadtd[i] = 0.0;
      hrtfrpadtd[i] = 0.0;
}

/* execute fft on padded hrtfs */
fftw_execute(forhrtflpad);
fftw_execute(forhrtfrpad);

/* look after overlap add */
for(i = 0; i < overlapsize ; i++)
{
      overlapl[i] = outl[i+irlength];
      overlapr[i] = outr[i+irlength];
      if(crossfade)
      {
            overlaplold[i] = outl[i+irlength];
            overlaprold[i] = outr[i+irlength];
      }
      /* overlap will be previous fading out signal */
      if(cross)
      {
            overlaplold[i] = outlold[i+irlength];
            overlaprold[i] = outrold[i+irlength];
      }
}
```

Input is read, zero-padded and transformed to the spectral domain before the convolution process occurs. Convolution in the frequency domain is performed by mutiplying spectra. Complex multiplication is required for all but 0 Hz and the Nyquist Frequency, which are real numbers. FFTW requires scaling to be performed, as the FFT result will include a factor of N (transform size) that needs to be compensated.

```
/* read input */
count = sf_readf_double(fin, inbuf, irlength);

/* zero pad */
/* fills last one with zeros from count */
for(i = (int)count; i < irpadlength; i++)
      inbuf[i] = 0.0;

/* fft input */
fftw_execute(forin);
```

```
/* convolution: spectral multiplication */
/* 0hz and Nyq */
outlspec[0] = inspec[0] * hrtflpadspec[0];
outrspec[0] = inspec[0] * hrtfrpadspec[0];
outlspec[irpadlength/2] = inspec[irpadlength/2] *
                          hrtflpadspec[irpadlength/2];
outrspec[irpadlength/2] = inspec[irpadlength/2] *
                          hrtfrpadspec[irpadlength/2];

/* complex multiplication according to fftw layout */
/* (a + i b)(c + i d) */
/* = (a c - b d) + i(a d + b c) */
for(i = 2, j = 1; i < irpadlength; j++, i+=2)
{
      /* real */
      outlspec[j] = inspec[j] * hrtflpadspec[j] - inspec[irpadlength
                    - j] * hrtflpadspec[irpadlength - j];
      outrspec[j] = inspec[j] * hrtfrpadspec[j] - inspec[irpadlength
                    - j] * hrtfrpadspec[irpadlength - j];
      /* imaginary */
      outlspec[irpadlength - j] = inspec[j] *
                                  hrtflpadspec[irpadlength - j] +
                                  inspec[irpadlength - j] *
                                  hrtflpadspec[j];
      outrspec[irpadlength - j] = inspec[j] *
                                  hrtfrpadspec[irpadlength - j] +
                                  inspec[irpadlength - j] *
                                  hrtfrpadspec[j];
}

fftw_execute(invoutl);
fftw_execute(invoutr);

/* scaled, as fftw is a sum */
for(i = 0; i < irpadlength; i++)
{
      outl[i] = outl[i] / irpadlength;
      outr[i] = outr[i] / irpadlength;
}
```

Crossfades are dealt with next. The crossfade or cross flags trigger the crossfade convolution process. Essentially, old HRTF data, stored when a crossfade is required, is convolved with the input. The cross variable ensures crossfade convolution is performed for the correct crossfade length (for more detail, see [34]). Values for the next index check are also stored in this section of code.

```
/* setup for fades */
if(crossfade || cross)
{
      crossout = 1;

      /* convolution */
      /* 0hz and Nyq */
      outlspecold[0] = inspec[0] * hrtflpadspecold[0];
```

```
        outrspecold[0] = inspec[0] * hrtfrpadspecold[0];
        outlspecold[irpadlength/2] = inspec[irpadlength/2] *
                                     hrtflpadspecold[irpadlength/2];
        outrspecold[irpadlength/2] = inspec[irpadlength/2] *
                                     hrtfrpadspecold[irpadlength/2];

        /* complex multiplication */
        for(i = 2, j = 1; i < irpadlength; j++, i+=2)
        {
              /* real */
              outlspecold[j] = inspec[j] * hrtflpadspecold[j] –
                               inspec[irpadlength - j] *
                               hrtflpadspecold[irpadlength - j];
              outrspecold[j] = inspec[j] * hrtfrpadspecold[j] –
                               inspec[irpadlength - j] *
                               hrtfrpadspecold[irpadlength - j];
              /* imaginary */
              outlspecold[irpadlength - j] = inspec[j] *
                                      hrtflpadspecold[irpadlength - j]
                                      + inspec[irpadlength - j] *
                                      hrtflpadspecold[j];
              outrspecold[irpadlength - j] = inspec[j] *
                                      hrtfrpadspecold[irpadlength - j]
                                      + inspec[irpadlength - j] *
                                      hrtfrpadspecold[j];
        }

        /* ifft, back to time domain */
        fftw_execute(invoutlold);
        fftw_execute(invoutrold);

        /* scaling */
        for(i = 0; i < irpadlength; i++)
        {
              outlold[i] = outlold[i] / irpadlength;
              outrold[i] = outrold[i] / irpadlength;
        }

        cross++;
        cross = cross % fade;
}

/* for next check */
oldelevindex = elevindex;
oldangleindex = angleindex;
```

Finally, output is prepared and written. In the case of a crossfade, the input processed with the old HRTF data is faded out, the new faded in. An audio-rate linear fade is used. In the vast majority of processing control periods, however, the output buffer is simply filled with the processed data. The main control rate variable is iterated and the output is written. A check is performed to ensure the output file size is correct (input + impulse – 1 samples).

```
if(crossout)
```

```
      for(i = 0; i < irlength; i++)
      {
            lrout[2 * i] = (outlold[i] + (i < overlapsize ?
                  overlaplold[i] : 0.0)) *
                  (1.0 - (double)l / fadebuffer) +
                  (outl[i] + (i < overlapsize ? overlapl[i] : 0.0))
                  * (double)l / fadebuffer;
            lrout[(2 * i) + 1] = (outrold[i] + (i < overlapsize ?
                  overlaprold[i] : 0.0)) *
                  (1.0 - (double)l / fadebuffer) +
                  (outr[i] + (i < overlapsize ? overlapr[i] : 0.0))
                  * (double)l / fadebuffer;
            l++;
      }
else
      for(i = 0; i < irlength; i++)
      {
            lrout[2 * i] = outl[i] + (i < overlapsize ? overlapl[i]
                           : 0.0);
            lrout[(2 * i) + 1] = outr[i] + (i < overlapsize ?
                                 overlapr[i] : 0.0);
      }

/* do every irlength samples! */
k += irlength;

/* if on last run, only write output length mod irlength frames */
if(k > psfinfoin->frames + irlength - 1)
      sf_writef_double(fout, lrout, (psfinfoin->frames + irlength –
                       1) % irlength);
else
      sf_writef_double(fout, lrout, irlength);
```

The only remaining points of interest in the main program are the deallocation of the two-dimensional array of pointers, and the destruction of the FFTW plans:

```
for(i = 0; i < 14; i++)
      for(j = 0; j < elevationarray[i] / 2 + 1; j++)
      {
            delete[] hrtfarrayl[i][j];
            delete[] hrtfarrayr[i][j];
      }

fftw_destroy_plan(invhrtfl);
…
```

The code can be compiled, assuming default locations for FFTW3 and *libsndfile*, with the command:

```
g++ binauralmover.cpp binauralmoverfunctions.cpp –o mover
      –I/usr/local/include –L/usr/local/lib –lsndfile –lfftw3
```

As discussed, when run, the program will prompt for the number of buffers for each crossfade and the file names for the input mono audio and the text file describing the trajectory. A sample trajectory file is included as *move.txt*. A brief noise burst, *noise.wav* and musical sample *sample.wav* are included as source audio. The narrowband guitar riff performs well with the suggested eight crossfade buffers. The noisy source can better mask any discontinuities caused by phase updates, so one crossfade buffer is sufficient, and, in fact leads to a more continuous result.

4.2.3 Real-time Implementation

A command line implementation of the Functional Model follows from the above. The novel algorithms have also been implemented as Csound opcodes, to allow for real-time processing (using Csound's FFT). A discussion of these opcodes inherently involves an implementation of the Functional Model, as well as a minimum-phase implementation. Therefore, it is perhaps more appropriate to present these algorithms as real-time implementations here, after discussing the more pertinent command-line points above. Many of the coding constructs follow from the command-line discussion and the code is again thoroughly commented; repetition in this discussion is minimised accordingly.

The opcodes have been a part of Csound since version 5.08 (February 2008). However, a recent complete code update was completed, with various improvements of code clarity, algorithms and optimisation. It is this code which will be discussed (see *hrtfopcodes.c*, Appendix 2). It is also important to recognise that offline processing is also possible with Csound, in the form of file output.

The file defines three opcodes: *hrtfmove*, *hrtfmove2* and *hrtfstat* [46, 47, 48]. *hrtfmove* offers Phase Truncation based processing, as above; as well as minimum-phase based processing. *hrtfmove2* uses STFT processing and the Functional Model. STFT processing is necessary to avoid artefacts as the phase changes control period by control period in a dynamic trajectory, as discussed in chapter 3. However, static sources do not suffer from these artefacts, as the phase is not changing. *hrtfstat* exploits this fact, reverting back to overlap-add processing, improving efficiency. Opcode usage is discussed elsewhere [35].

A number of definitions/declarations are required for the opcodes, including the appropriate non-linear ITD scaling value arrays and extracted minimum-phase delay values (both discussed at length above). Note that HRTF data (magnitude, phase, delays and scaling factors) is stored as `float` (*datafilesfloat.cpp* writes these lower precision files).

It is also necessary to check that the byte order of the file is correct for the architecture being used. If a big endian architecture is being used, a byte swap function is called. This function was added by the Csound core developers after the issue arose on first release. Detail of developing plug-in opcodes is beyond the scope of this discussion, and is dealt with elsewhere [121, 44]. A standard approach to opcode development is adopted accordingly: a constructor/opcode initialisation function and a processing function are declared, as well as a structure containing the internal variables required (dataspace) for an instance of the opcode.

4.2.3.1 `hrtfmove`

hrtfmove's outputs are simply the stereo processed audio. Inputs are the unspatialised input mono audio, a control (k) rate angle and elevation parameter, an initialisation (i) time file name for the left and right HRTF data file, the mode of operation (minimum-phase or Phase Truncation), defaulting to 0 (Phase Truncation), the number of buffers for crossfades and the sampling rate. The latter three arguments are optional.

```
aleft,aright hrtfmove asrc, kaz, kel, ifilel, ifiler [, imode = 0,
                      ifade = 8, sr = 44100]
```

Memory allocation is dealt with using AUXCH variables in Csound; memory is allocated dynamically in the initialisation function. Significant variables will be discussed as they arise. *hrtfmove*'s initialisation function declares some variables local to the function/method (these will be lost after the method is called, however, variables in the data structure will be maintained), such as Csound's file pointer format, used to open and store the HRTF files. Optional inputs are set to local variables, to avoid unnecessary repeated reference of the structure (note that MYFLT refers to single or double-precision floating point precision, depending on the Csound install).

```
/* left and right data files: spectral mag, phase format. */
MEMFIL *fpl = NULL,*fpr = NULL;
int i;
char filel[MAXNAME],filer[MAXNAME];

int mode = (int)*p->omode;
int fade = (int)*p->ofade;
MYFLT sr = *p->osr;
```

Input values are read (and checked); the sampling rate dictates the default impulse length, padded impulse length and overlap size. Note that three typical sampling rates are allowed, 44.1 kHz, 48 kHz and 96 kHz. Each sampling rate requires a new dataset, so it is hoped that offering these three options strikes a balance between flexibility and data size/clarity.

Data files are opened, using Csound's file open function, which includes the endian check, and the dataspace is populated with appropriate values. Pointers in the dataspace are set to point to the first memory location of the HRTF data files.

```
/* flag for process type: default phase trunc */
if(mode == 1)
{
	p->minphase = 1;
	p->phasetrunc = 0;
}
else
{
	p->phasetrunc = 1;
	p->minphase = 0;
}
```

```
/* fade length: default 8, max 24, min 1 */
if(fade < 1 || fade > 24)
      fade = 8;
p->fade = fade;

/* sr, defualt 44100 */
if(sr != 44100 && sr != 48000 && sr != 96000)
      sr = 44100;
p->sr = sr;

if (UNLIKELY(csound->esr != sr))
      csound->Message(csound, Str("\n\nWARNING!!:\nOrchestra SR not
                      compatible with HRTF processing SR of:
                      %.0f\n\n"), sr);

/* setup as per sr */
if(sr == 44100 || sr == 48000)
{
      irlength = 128;
      irlengthpad = 256;
      overlapsize = (irlength - 1);
}
else if(sr == 96000)
{
      irlength = 256;
      irlengthpad = 512;
      overlapsize = (irlength - 1);
}

/* copy in string name */
strcpy(filel, (char*) p->ifilel);
strcpy(filer, (char*) p->ifiler);

/* reading files, with byte swap */
if (UNLIKELY((fpl = csound->ldmemfile2withCB(csound, filel,
             CSFTYPE_FLOATS_BINARY, swap4bytes)) == NULL))
      return
            csound->InitError(csound, Str("\n\n\nCannot load left
                                          data file, exiting\n\n"));

if (UNLIKELY((fpr = csound->ldmemfile2withCB(csound, filer,
             CSFTYPE_FLOATS_BINARY, swap4bytes)) == NULL))
      return
            csound->InitError(csound, Str("\n\n\nCannot load right
                                          data file, exiting\n\n"));

p->irlength = irlength;
p->irlengthpad = irlengthpad;
p->overlapsize = overlapsize;

/* the amount of buffers to fade over. */
p->fadebuffer = (int)fade*irlength;

/* file handles */
p->fpbeginl = (float *) fpl->beginp;
p->fpbeginr = (float *) fpr->beginp;
```

Memory is then allocated. If the memory does not exist, or it exists and is less than the appropriate size, memory is allocated dynamically. Delay buffers for minimum-phase processing are allocated according to a safe maximum delay time. All memory is zeroed; one example of this is given below:

```
if (!p->insig.auxp || p->insig.size < irlength * sizeof(MYFLT))
csound->AuxAlloc(csound, irlength*sizeof(MYFLT), &p->insig);
…
memset(p->insig.auxp, 0, irlength * sizeof(MYFLT));
```

The appropriate minimum-phase window is defined (use of the window in the minimum-phase process is discussed below).

```
win = (MYFLT *)p->win.auxp;

/* min phase win defined for irlength point impulse! */
win[0] = FL(1.0);
for(i = 1; i < (irlength / 2); i++)
      win[i] = FL(2.0);
win[(irlength / 2)] = FL(1.0);
for(i = ((irlength / 2) + 1); i < irlength; i++)
      win[i] = FL(0.0);
```

An interesting update to the command line process is also initiated here.

Interpolation processing is only necessary if the source moves. As optimisation is crucial for the real time process, this redundancy is exploited. Variables are setup and used to check if the source has moved since the last processing period. They are initialised to values out of the legal range, to ensure processing occurs on the first control period:

```
/* setup values used to check if src has moved, illegal values to
   start with to ensure first read */
p->anglev = -1;
p->elevv = -41;
```

The processing function starts by referencing the dataspace, again to avoid unnecessary multiple referencing. Local variables are also declared (whose values do not need to be maintained from call to call). The method is called every `ksmps`. Therefore, processing at audio rate is looped. Input and output buffers are filled/read at this rate.

```
n = csound->ksmps;

for(j = 0; j < n; j++)
{
      /* ins and outs */
      insig[counter] = in[j];

      outsigl[j] = outl[counter];
      outsigr[j] = outr[counter];

      counter++;
      …
```

A slightly different construct is used to ensure a fade does not happen on the first run, as processing is real-time in this scenario, as opposed to offline.

```
if(phasetrunc)
{
      /* used to ensure fade does not happen on first run */
      if(initialfade < (irlength + 2))
            initialfade++;
}
```

The majority of processing occurs at the internal control rate, essentially based on the length of a HRIR:

```
if(counter == irlength)
```

Crossfade flags are reset; angle and elevation values are treated similarly to the command-line code. Processing only proceeds if the angle or elevation value has changed since the last internal control rate. The bulk of the costly processing is included in this conditional statement, so, for static sources, efficiency is improved.

```
/* only update if location changes! */
if(angle != p->anglev || elev != p->elevv)
{
      /* two nearest elev indices to avoid recalculating */
      elevindexstore = (elev - minelev) / elevincrement;
      elevindexlow = (int)elevindexstore;
      …
```

Elevation and angle indexing is performed as with the command-line program. If Phase Truncation programming is being performed, crossfade initialisation is also dealt with similarly. In the real time implementation, the data files are accessed more simply. They are loaded into independent memory directly, and accessed using a skip variable, which is incremented according to the location of the appropriate HRTF. For example:

```
/* store point for current phase as trajectory comes closer to a new
   index */
skip = 0;
/* store current phase */
if(angleindex > elevationarray[elevindex] / 2)
{
      for(i = 0; i < elevindex; i++)
            skip +=((int)(elevationarray[i] / 2) + 1) * irlength;
      for (i = 0; i < (elevationarray[elevindex] - angleindex); i++)
            skip += irlength;
      for(i = 0; i < irlength; i++)
      {
            currentphasel[i] = fpindexr[skip + i];
            currentphaser[i] = fpindexl[skip + i];
      }
}
else
{
      for(i = 0; i < elevindex; i++)
            skip +=((int)(elevationarray[i] / 2) + 1) * irlength;
      for (i = 0; i < angleindex; i++)
            skip += irlength;
      for(i = 0; i < irlength; i++)
      {
            currentphasel[i] = fpindexl[skip+i];
            currentphaser[i] = fpindexr[skip+i];
      }
}
```

The current phase and four nearest buffers are filled accordingly. Interpolation is also a similar process to the command line.

The log magnitude is required for minimum-phase, so magnitude values are stored accordingly. `log(0)` is avoided, as it is undefined:

```
logmagl[i] = LOG(magl == FL(0.0) ? FL(0.00000001) : magl);
```

Deriving the minimum-phase from the log magnitude of the interpolated HRTF follows a real cepstrum method. In [81], the method is described explicitly. The window function is defined as:

$$win(n) = \begin{cases} 1. \; n = 1 \, or N/2+1 \\ 2. \; n = 2,...,N/2 \\ 0. \; n = N/2+2,...,N \end{cases} \tag{4.1}$$

The IFFT (using Csound's internal inverse FFT) of the log magnitude of the HRTF is windowed. The (complex) exponential of the spectral result, transformed back into the time domain constitutes the minimum-phase response. As discussed above, this minimum-phase preparation process is costly. Jot also mentions the possibility of direct phase interpolation, but this involves storing a different dataset, as well as offline preparation of this dataset [93].

```
if(minphase)
{
      /* ifft!...see Oppehneim and Schafer for min phase
         process...based on real cepstrum method */
      csound->InverseRealFFT(csound, logmagl, irlength);
      csound->InverseRealFFT(csound, logmagr, irlength);

      /* window, note no need to scale on csound iffts... */
      for(i = 0; i < irlength; i++)
      {
            xhatwinl[i] = logmagl[i] * win[i];
            xhatwinr[i] = logmagr[i] * win[i];
      }

      /* fft */
      csound->RealFFT(csound, xhatwinl, irlength);
      csound->RealFFT(csound, xhatwinr, irlength);

      /* exponential of result */
      /* 0 hz and nyq purely real... */
      expxhatwinl[0] = EXP(xhatwinl[0]);
      expxhatwinl[1] = EXP(xhatwinl[1]);
      expxhatwinr[0] = EXP(xhatwinr[0]);
      expxhatwinr[1] = EXP(xhatwinr[1]);

      /* exponential of real, cos/sin of imag */
      for(i = 2; i < irlength; i += 2)
      {
            expxhatwinl[i] = EXP(xhatwinl[i]) *
                             COS(xhatwinl[i + 1]);
            expxhatwinl[i+1] = EXP(xhatwinl[i]) *
                               SIN(xhatwinl[i + 1]);
            expxhatwinr[i] = EXP(xhatwinr[i]) *
                             COS(xhatwinr[i + 1]);
            expxhatwinr[i+1] = EXP(xhatwinr[i]) *
                               SIN(xhatwinr[i + 1]);
      }

      /* ifft for output buffers */
      csound->InverseRealFFT(csound, expxhatwinl, irlength);
      csound->InverseRealFFT(csound, expxhatwinr, irlength);

      /* output */
      for(i= 0; i < irlength; i++)
      {
            hrtflpad[i] = expxhatwinl[i];
            hrtfrpad[i] = expxhatwinr[i];
      }
}
```

The minimum-phase/Phase Truncation based zero-padded HRTF can then be used in the convolution process.

Another step involved in the minimum-phase process is adding the frequency independent delay, in place of the all-pass system. For dynamic source trajectories, this involves interpolating the appropriate delay time, in a similar, if simpler manner to HRTF reading and interpolation. Note that delays were calculated using the method from [111], as it is frequently referenced as the study that validates the minimum-phase model. All of the above processing only occurs if the angle or elevation change, highlighting the redundancy of static sources.

```
…
delayfloat = delaylow + ((delayhigh - delaylow) * elevindexhighper);
```

Overlap-add convolution can then be performed to generate the output. Overlap data is dealt with similarly to the command-line program (considering crossfades in the case of Phase Truncation). Csound provides a function for complex multiplication of real FFT buffers. Crossfade convolution and output are also dealt with similarly. Minimum-phase output involves a variable delay line. The delay is imposed on the left or right signal, depending on which hemisphere the source lies in. A variable delay construct which allows zero delay is used (by writing before reading). Feedback is not feasible on such a construct, but is not required here.

```
if(angle > FL(180.0))
{
      vdtr =  delayfloat * sr;
      vdtl = FL(0.0);
}
else
{
      vdtr = FL(0.0);
      vdtl = delayfloat * sr;
}

/* delay right */
```

```
if(vdtr > mdtr)
      vdtr = FL(mdtr);
for(i = 0; i < irlength; i++)
{
      rpr = ptr - vdtr;
      rpr = (rpr >= 0 ? (rpr < mdtr ? rpr : rpr - mdtr) : rpr +
             mdtr);
      posr = (int) rpr;
      fracr = rpr - posr;
      delmemr[ptr] = outr[i];
      outvdr =  delmemr[posr] + fracr*(delmemr[(posr + 1 < mdtr ?
                posr + 1 : 0)] - delmemr[posr]);
      outr[i] = outvdr;
      ptr = (ptr != mdtr - 1 ? ptr + 1 : 0);
}

/* delay left */
if(vdtl > mdtl)
      vdtl = FL(mdtl);
for(i = 0; i < irlength; i++)
{
      rpl = ptl - vdtl;
      rpl = (rpl >= 0 ? (rpl < mdtl ? rpl : rpl - mdtl) : rpl +
             mdtl);
      posl = (int) rpl;
      fracl = rpl - (int) posl;
      delmeml[ptl] = outl[i];
      outvdl =  delmeml[posl] + fracl*(delmeml[(posl + 1 < mdtl ?
                posl + 1 : 0)] - delmeml[posl]);
      outl[i] = outvdl;
      ptl = (ptl != mdtl - 1 ? ptl + 1 : 0);
}

p->ptl = ptl;
p->ptr = ptr;
```

4.2.3.2 hrtfstat

hrtfstat is defined next. Outputs are left and right spatialised digital audio, as before. Inputs are angle and elevation values, this time at i time, as the opcode is meant for static source processing, as well as the mono input audio and file names. Optional inputs are a value for the head radius used in the formula for the Functional Model, and sampling rate.

```
aleft, aright hrtfstat ain, iang, iel, ifilel, ifiler [,iradius =
                       8.8, isr = 44100]
```

The initialisation function uses many of the same constructs as *hrtfmove*. Head radius defaults to 8.8 cm (see below for reasoning). As source location and interpolation only needs to be performed once, when the opcode is initialised, it can be performed in the initialisation function.

IPD is applied using the Functional Model, as discussed above, in the theoretical discussion of the method. A radian angle is required for the formula, so the angle in degrees is transformed (it is also brought into the relevant hemisphere, if appropriate). The Woodworth formula is used as an initial spherical-head ITD estimate. This implementation of the formula requires the angle to be in the first quadrant (relative to the listener's front centre, anti-clockwise). The Woodworth based ITD is given by the `itdww` variable.

```
/* woodworth process */
/* ITD formula, check which ear is relevant to calculate angle from
*/
if(angle > FL(180.))
      radianangle = (angle - FL(180.)) * FL(PI / 180.);
else
      radianangle = angle * FL(PI / 180.);
/* degrees to radians */
radianelev = elev * FL(PI / 180.);

/* get in correct range for formula */
if(radianangle > FL(PI / 2.0))
      radianangle = FL(PI) - radianangle;

/* woodworth formula for itd */
itdww = (radianangle + sinf(radianangle)) * r * cosf(radianelev) /
        FL(c);
```

Magnitude interpolation is performed as before. Phase for 0 Hz and the Nyquist Frequency is set to 0, by making their magnitudes positive (this is in line with the Functional derivation of phase, as opposed to empirical). The functional interaural phase spectrum is then applied. First, the appropriate frequency is derived from the point in the iteration, the sampling rate and the size of the FFT buffer (the latter 2 constituting the `sroverN` variable). For the appropriate sampling rate, the scaling factor array is used to scale the ITD for the appropriate low-frequency bins. A final scaling factor of 1.0 ensures that non-scaled ITD values maintain the Woodworth formula value. As discussed above, ITD is transformed into phase by transforming time differences to phase differences. The ITD is halved; the leading ear gets a positively oriented phase, the lagging negative. With magnitude and phase derived, it is possible to return to rectangular form and the time domain.

```
freq = (i / 2) * p->sroverN;

/* non linear itd...last value in array = 1.0, so back to itdww */
if(p->sr == 96000)
{
      if ((i / 2) < 6)
            itd = itdww * nonlinitd96k[(i / 2) - 1];
}
if(p->sr == 48000)
{
      if ((i / 2) < 6)
            itd = itdww * nonlinitd48k[(i / 2) - 1];
}
if(p->sr == 44100)
{
      if((i / 2) < 6)
            itd = itdww * nonlinitd[(i / 2) - 1];
}

if(angle > FL(180.))
{
      phasel = TWOPI_F * freq * (itd / 2);
      phaser = TWOPI_F * freq * -(itd / 2);
}
else
{
      phasel = TWOPI_F * freq * -(itd / 2);
      phaser = TWOPI_F * freq * (itd / 2);
}

/* polar to rectangular */
hrtflfloat[i] = magl * COS(phasel);
hrtflfloat[i+1] = magl * SIN(phasel);

hrtfrfloat[i] = magr * COS(phaser);
hrtfrfloat[i+1] = magr * SIN(phaser);
```

As discussed theoretically above, the buffers are shifted to ensure correct onset as well as interaural phase, using shift buffers. The impulse is centred around the centre tap of the filter, using the `shift` variable. Zero padding and spectral transformation can then be performed.

```
for (i = 0; i < irlength; i++)
{
      /* scale and pad buffers with zeros to fftbuff */
      leftshiftbuffer[i] = hrtflfloat[i];
      rightshiftbuffer[i] = hrtfrfloat[i];
}
```

```
/* shift for causality...impulse as is is centred around zero time
   lag...then phase added. */
/* this step centres impulse around centre tap of filter (then phase
   moves it for correct itd...) */
shift = irlength / 2;

for(i = 0; i < irlength; i++)
{
      hrtflpad[i] = leftshiftbuffer[shift];
      hrtfrpad[i] = rightshiftbuffer[shift];

      shift++;
      shift = shift % irlength;
}

/* zero pad impulse */
for(i = irlength; i < irlengthpad; i++)
{
      hrtflpad[i] = FL(0.0);
      hrtfrpad[i] = FL(0.0);
}

/* back to freq domain */
csound->RealFFT(csound, hrtflpad, irlengthpad);
csound->RealFFT(csound, hrtfrpad, irlengthpad);
```

The process function is straightforward for the static implementation of the Functional Model. It uses the `hrtfl/rpad` buffers, which contain the interpolated HRTFs, derived in the initialisation function, as the impulse in the convolution operation. No crossfades need to be considered, so the convolution operation is standard.

4.2.3.3 `hrtfmove2`

The dynamic version of the Functional Model, *hrtfmove2*, interpolates the impulse in the same way, but reverts to updating interpolation in the perform method, similarly to *hrtfmove*. Input arguments are again audio in, k-rate angle and elevation, i-time file names, i time STFT overlap, and head radius and sampling rate, as with the static Functional implementation.

```
aleft, aright hrtfmove2 ain, kang, kel, ifilel, ifiler
                        [, ioverlap = 4, iradius = 8.8, isr =
                        44100]
```

The design of the STFT process used is based on the Sound Object Library [197], with some updates for this specific application. As overlap-add convolution is not performed here, no padded impulse or overlap-add variables are required. STFT overlap is limited to 2, 4, 8 or 16 and defaults to 4. The STFT output, as discussed in chapter 1, is made up of the sum of a number of overlapping windowed outputs, so a number of processing buffers are required. This is done by essentially using large 1-dimensional buffers to represent two-dimensional constructs. A Hanning window is used in the STFT [144].

The processing function performs interpolation, as before, but this time in a different overall structure, optimised for flexible STFT processing. In the main audio-rate loop (the `ksmps` loop), the signal is distributed into the input buffer. The `overlapskipin` buffer is initially filled with values that represent the incrementation implied by the `overlap` variable (the `hopsize` is defined by the impulse divided by the `overlap`). Essentially, the buffer keeps track of the iterations through each of the input buffers used in the process. The first buffer's input starts at 0, the second at the hopsize, the third at twice the hopsize etc. In this way, the staggered input required is achieved. A similar buffer is used to keep track of the overlapping output.

```
for(i = 0; i < overlap; i++)
{
      /* so, for example in overlap 4: will be 0, 32, 64, 96 if ir =
         128 */
      overlapskipin[i] = p->hopsize * i;
      overlapskipout[i] = p->hopsize * i;
}
```

The input buffer, which essentially represents a two-dimensional array (implemented as one large buffer) of staggered input audio is filled (and windowed) accordingly.

```
/* distribute the signal and apply the window */
/* according to a time pointer (kept by overlapskip[n]) */
for(i = 0; i < overlap; i++)
{
      inbuf[(i * irlength) + overlapskipin[i]] = in[j] *
                                   win[overlapskipin[i]];
      overlapskipin[i]++;
}
```

The appropriate input buffer is read using the variable `t`, which decrements every time a buffer is processed. This variable ensures that the appropriate buffer is read when needed. Once read, the index of an input buffer is zeroed, so it is overwritten.

```
/* t used to read inbuf...*/
t--;
if(t < 0)
      t = overlap - 1;

/* insert insig for complex real, im fft */
for(i = 0; i < irlength; i++)
      complexinsig[i] = inbuf[(t * irlength) + i];

/* zero the current input sigframe time pointer */
overlapskipin[t] = 0;
```

Once processed with the appropriate interpolated HRTF, output is placed in a similar buffer to the large input array; again representing a series of 2-D buffers.

```
for(i = 0; i < irlength; i++)
{
      outbufl[(t * irlength) + i] = outspecl[i] / (overlap * FL(0.5)
                                    * (sr / FL(44100.0)));
      outbufr[(t * irlength) + i] = outspecr[i] / (overlap * FL(0.5)
                                    * (sr / FL(44100.0)));
}
```

Final output is dealt with in a similar way to input. Each outputted sample is the sum of each appropriately indexed buffer. Once again, the variable `t` deals with appropriate output buffer index resetting.

```
/* output = sum of all relevant outputs: eg if overlap = 4 and
   counter = 0, */
/* outsigl[j] = outbufl[0] + outbufl[128 + 96] + outbufl[256 + 64] +
   outbufl[384 + 32]; */
/*         * * * * [ ]         + */
/*           * * * [*]         + */
/*             * * [*] *       + */
/*               * [*] * *     = */
/* stft! */

outsuml = outsumr = FL(0.0);

for(i = 0; i < (int)overlap; i++)
{
      outsuml += outbufl[(i * irlength) + overlapskipout[i]] *
                 win[overlapskipout[i]];
      outsumr += outbufr[(i * irlength) + overlapskipout[i]] *
```

```
                win[overlapskipout[i]];
      overlapskipout[i]++;
}

if(counter == hopsize)
{
      /* zero output incrementation... */
      /* last buffer will have gone from 96 to 127...then 2nd last
         will have gone from 64 to 127... */
      overlapskipout[t] = 0;
      counter = 0;
}

outsigl[j] = outsuml;
outsigr[j] = outsumr;
```

Finally, the `OENTRY` structure is filled with the arguments: opcode size, processing rates (i and a rate here), out types, in types, i-rate function (initialisation), k-rate function (not applicable here, so `NULL`) and a-rate function (perform). Out types are 2 stereo audio streams for all three opcodes (‘`aa`’). Inputs vary for each, all take audio input as their first type, the dynamic opcodes (*hrtfmove* and *hrtfmove2*) take k-rate angle and elevation values next, the static opcode (*hrtfstat*) takes i rates here. String values for the HRTF file names follow. Optional arguments complete the input type lists. ‘o’ defaults to 0, and is i rate [44]. The `LINKAGE` macro in the code below deals with opcode registration. The plugin opcodes are thus completed.

```
/* see csound manual (extending csound) for details of below */
static OENTRY localops[] =
{
  { "hrtfmove", sizeof(hrtfmove),5, "aa", "akkSSooo",
    (SUBR)hrtfmove_init, NULL, (SUBR)hrtfmove_process },
  { "hrtfstat", sizeof(hrtfstat),5, "aa", "aiiSSoo",
    (SUBR)hrtfstat_init, NULL, (SUBR)hrtfstat_process },
  { "hrtfmove2",  sizeof(hrtfmove2),5, "aa", "akkSSooo",
    (SUBR)hrtfmove2_init, NULL, (SUBR)hrtfmove2_process }
};

LINKAGE
```

4.2.3.4 Opcode Optimisation

The discussion above represents a recent update to the real time implementation of the HRTF algorithms; as part of this update, complex FFT processing was replaced by real FFT processing for efficiency reasons. Issues like the previous discussion of

0 Hz and Nyquist Frequency values make this a non-trivial update. Code was also completely reviewed, resulting in efficiency and clarity updates (such as only updating spatialisation parameters when sources move, as above).

Moving a source from 0 degrees to 90 degrees with Phase Truncation based *hrtfmove*, using a complex FFT model (old code) averages a CPU time of .22 seconds for 2 seconds of processing, over 10 iterations (Intel Core 2 CPU, T7400 @ 2.16 GHz, Windows XP). The real FFT update improves this figure to .19 seconds. Interestingly, *hrtfer*, with no interpolation averages .235 seconds (it is assumed that code optimisations are responsible for improvements here). The minimum-phase version of *hrtfmove* averages .21 seconds. All processing for these tests was done at a sampling rate of 44.1 kHz. For static sources using *hrtfmove*, the optimisation applied reduces the process (with the same parameters with the exception of the trajectory) to an average of .16 seconds. These results are summarised in table 4.1, below.

Algorithm	**Average Time Taken**
hrtfmove (complex FFT)	.22
hrtfmove (real FFT)	.19
hrtfmove (minimum phase)	.21
hrtfer	.235

Table 4.1: Comparison of average time taken by various algorithms to process 2 seconds of dynamic trajectories.

In summary, Phase Truncation takes less computation time than a similarly setup minimum-phase process. Also, significant improvements can be derived from real FFT processing (as opposed to compex, a 14% reduction; also, less storage is required).

4.3. Algorithm Testing

In order to validate the theory and implementation of the algorithms developed, both objective and subjective tests were performed. The primary goal of smooth, artefact-free source motion was constantly considered in the inception and development of testing methods.

If objectively/numerically testing a HRTF interpolation algorithm such as a minimum-phase plus delay model, typically, as in [111] the derived HRTF can be compared numerically to the empirical measurement for a particular location. A degree of error can be obtained from this comparison. In the case of Phase Truncation, however, the empirical and algorithmically derived HRTF will be identical at empirically measured points (Phase Truncation aims to provide empirical measurements where available and a perceptually accurate HRTF spectrum elsewhere by interpolating magnitudes and truncating to the nearest measured phase).

A suggested approach to this difficulty is presented in [77]. A subset of the measured data can be taken. For example, if a HRTF is available at 5, 10 and 15 degrees, the 10 degree measurement can be estimated using the interpolation technique, then compared to the empirical measurement. In the case of Phase Truncation, this will result in a significantly less accurate result than in a real usage scenario which employs all measured points. Part of the strength of the algorithm is based on the typically densely measured datasets currently available. In a dataset measured at 5 degree increments, the phase spectrum will never be more than 2.5 degrees in spatial error. However, in the above experimental procedure this error will be 5 degrees. Also, interpolated magnitude values will deviate more than typical usage for the same reasons. Furthermore, the primary goal of this work is to provide smoothly moving sources, with accurate, artefact-free transitions from one empirically measured point to another. The update of open source HRTF processing, binaural reverb and multichannel binaural applications that the HRTF algorithms were designed for highlight this goal. As mentioned in [211], high-quality virtual reality and multimedia applications require a smooth perception of sound source movement. It is with the above considerations in mind that a perceptual test is considered more appropriate to validate the method. The Phase Truncation method is thus tested by listeners for any artefacts introduced when it is used to dynamically spatialise source trajectories.

The Functional Model can be tested numerically as above, as it is similar to the minimum-phase approach, in that it creates a whole new dataset. For example, the modelled point at 5 degrees azimuth and 0 degree elevation can be tested against the empirical measurement at that point, unlike in the Phase Truncation case where the empirical and interpolated points constitute identical filters. The Functional Model can also be incorporated into the perceptual movement tests.

4.3.1 Objective Testing

Several approaches to objective testing were considered. The gammatone filter discussed in [192] was initially considered as a suitable tool. Essentially, the spectrum is bandpass filtered in a manner that imitates the cochlea. The implementation investigated uses four second-order filters per band. Originally, the intention was to investigate a frequency dependent ITD band by band. However, the short HRIR signals did not integrate well with the model (in relation to the time response of the gammatone filters). Some issues were also encountered with extracting the delay of the filtered signals.

Ultimately, a direct, clinical approach was taken. The goal of the test was to investigate the low-frequency temporal accuracy of the Functional Model, compared to the minimum-phase model. Therefore, a dataset of each was prepared, low-pass filtered and the ITD extracted using the maximum of the cross correlation of the left and right HRIR.

Once again, C++ code was used to perform these tests. The file *defs.h* in the 'Chapter4/testing' folder on the accompanying CD-ROM (which includes all relevant files) includes the relevant headers and defines default values. A delay and low-pass filter function are also declared. These functions are explicated in *functions.cpp*. The delay function is simple, and uses a number of samples as its input delay time:

```
double delay(double *sig, int dt, double *del, int *p, int vecsize,
             double sr)
{
      double out;
      for(int i=0; i < vecsize; i++)
      {
            out = del[*p];    /* read current val */
            del[*p] = sig[i]; /* write in val */
            sig[i] = out;     /* write o/p to buffer... */
            *p = (*p != dt - 1 ? *p + 1 : 0);   /* increment, if at
                                                   end, go to start! */
      }
    return *sig;
}
```

The low-pass is a simple first order recursive filter, and will be discussed in the context of the reverberation opcodes discussed in the next chapter.

The diffuse dataset is prepared in the file *diffusescaled.cpp*. The process essentially goes through each file, scales it down (all datasets are scaled by an empirically determined factor of .65, as the nature of the minimum-phase and functional-phase models lead to clipping if no scaling is applied) and stores it conveniently. A large file containing all impulses is also prepared, for observation purposes.

The functional dataset is prepared in the file *wwloop.cpp*. A little more consideration is required here. FFTW is again used for the Fourier transforms necessary, as discussed above. Each file is transformed to polar form (0 Hz and the Nyquist Frequency are given phases of 0, as a fully synthetic phase spectrum is favoured). Functional phases are derived, as in the opcode described above. However, for convenience, FFTW format is maintained in this scenario. Therefore, the format of magnitudes, followed by phases is employed:

```
/* complex */
for(i = 1; i < (irlength / 2); i++)
{
      /* mags */
      polarl[i] = sqrt(fftl[i] * fftl[i] + fftl[irlength - i] *
                  fftl[irlength - i]);
      polarr[i] = sqrt(fftr[i] * fftr[i] + fftr[irlength - i] *
                  fftr[irlength - i]);

      /* phases */
      freq = i * sroverN;

      /* recalculate, with reset on last iteration */
      if(i < 6)
      {
            itd = (radianangle + sin(radianangle)) * r *
            cos(radianelev) / c;
            itd = itd * nonlinitd[i - 1];
      }

      polarl[irlength - i] = twopi * freq * -(itd / 2);
      polarr[irlength - i] = twopi * freq * (itd / 2);
}
```

Maintaining FFTW’s format makes returning to rectangular form and the time domain convenient. The impulses are then shifted in time, for causality and correct orientation, as above.

The minimum-phase loop is a little more complex again, as in *minphaseloop.cpp*. In preparation for minimum-phase processing, the appropriate window is prepared. As above, the conjugate of the right impulse is derived and the cross spectrum calculated:

```
/* get conjugate of right response */
for(i = (irlength / 2 + 1); i < irlength; i++)
      crosscorr[i] *= -1;
```

```
/* cross spectrum, 0 Hz & Nyquist */
crossfft[0] = crosscorl[0] * crosscorr[0];
crossfft[irlength / 2] = crosscorl[irlength / 2] *
crosscorr[irlength / 2];

/* (x + yi)(u + vi) = (xu – yv) + (xv + yu)i complex multiplication
*/
for(i = 1; i < (irlength / 2); i++)
{
      crossfft[i] = crosscorl[i] * crosscorr[i] - crosscorl[irlength
                    - i] * crosscorr[irlength - i];
      crossfft[irlength - i] = crosscorl[i] * crosscorr[irlength –
                               i] + crosscorl[irlength - i] *
                               crosscorr[i];
}
```

The maximum of this cross correlation is, once again, understood as the interaural delay:

```
/* ifft, scale */
fftw_execute(invcross);

for(i = 0; i < irlength; i++)
      cross[i] = cross[i] / irlength;

/* get max sample value */
maxvalue = cross[0];
maxsampleval = 0;
for(i = 0; i < irlength; i++)
{
      if(fabs(cross[i]) > fabs(maxvalue))
      {
            maxvalue = cross[i];
            maxsampleval = i;
      }
}
```

Log magnitude values are then calculated. Again, the FFTW format is maintained. Also, to reiterate, it is important not to allow 0 log-magnitude values, as the log of 0 is undefined:

```
/* get log magnitudes, with zero phases for ifft */
/* 0 Hz & Nyq: positive mags */
magl = fabs(fftl[0]);
magr = fabs(fftr[0]);
logmagl[0] = log(magl == 0.0 ? 0.00000001 : magl);
logmagr[0] = log(magr == 0.0 ? 0.00000001 : magr);
magl = fabs(fftl[irlength / 2]);
magr = fabs(fftr[irlength / 2]);
logmagl[irlength / 2] = log(magl == 0.0 ? 0.00000001 : magl);
logmagr[irlength / 2] = log(magr == 0.0 ? 0.00000001 : magr);

for(i = 1; i < (irlength / 2); i++)
{
      magl = sqrt(fftl[i] * fftl[i] + fftl[irlength - i] *
      fftl[irlength - i]);
      magr = sqrt(fftr[i] * fftr[i] + fftr[irlength - i] *
      fftr[irlength - i]);

      logmagl[i]= log(magl == 0.0 ? 0.00000001 : magl);
      logmagr[i]= log(magr == 0.0 ? 0.00000001 : magr);
}
```

Phase values are left at 0 at this point. The time-domain version of the log magnitudes is windowed, transformed back into the frequency domain and the exponential is taken (care must be taken to correctly calculate the complex exponential):

```
/* exponential of complex result */
/* 0 hz and nyq purely real... */
expxhatwinl[0] = exp(fftl[0]);
expxhatwinr[0] = exp(fftr[0]);
expxhatwinl[irlength / 2] = exp(fftl[irlength / 2]);
expxhatwinr[irlength / 2] = exp(fftr[irlength / 2]);

for(i = 1; i < (irlength / 2); i++)
{
        expxhatwinl[i] = exp(fftl[i]) * cos(fftl[irlength - i]);
        expxhatwinl[irlength - i] = exp(fftl[i]) * sin(fftl[irlength -
        i]);
        expxhatwinr[i] = exp(fftr[i]) * cos(fftr[irlength - i]);
        expxhatwinr[irlength - i] = exp(fftr[i]) * sin(fftr[irlength -
        i]);
}
```

The process of extracting the interaural delay is then performed on the minimum-phase buffers, to extract the interaural delay between them. This value is then subtracted from the empirical extracted delay, before the appropriate data is written to file. A text file is written to the same folder as the HRIRs, detailing the empirical, minimum-phase and overall delay for each measured location.

As the data files have now been prepared, objective testing can commence. The file *hrtftestingloop.cpp* contains the code. The file for each location of the empirical data, Functional and minimum-phase models is opened and stored. A low-pass filter is then applied, with a cutoff at 1000 Hz (bearing in mind the 1500 Hz threshold of ITD and the filter response, as below in figure 4.2). This is a first order filter, with a smooth rolloff. Therefore higher frequencies will pass through the filter, but will be attenuated. This will work in the favour of the minimum-phase data, as the functional data is linearly approximated in the higher-frequency ranges:

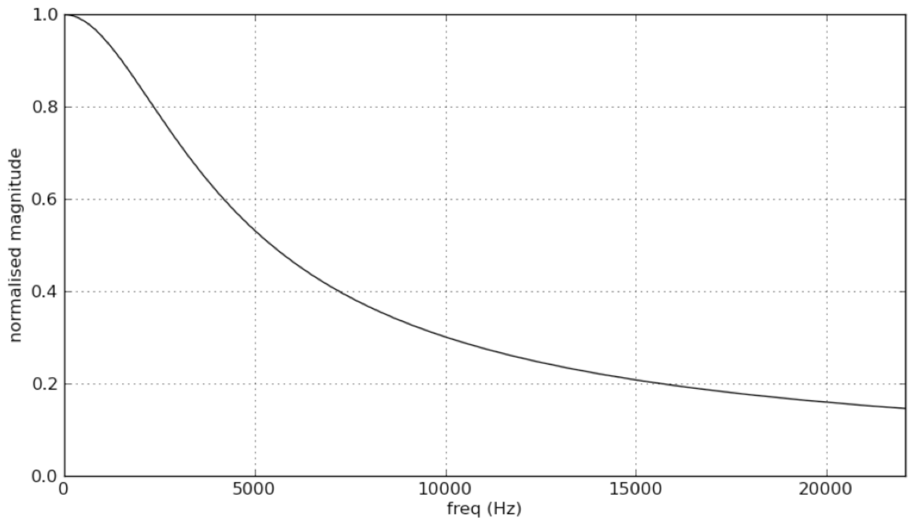


Figure 4.2: Low-pass filter used in objective test response

```
/* filtering */
/* LOW PASS ALL */
/* initialise internals to zero.... */
flemp = fremp = flempscal = frempscal = flmp = frmp = flww = frww =
0.0;
lowpass(inlemp, 1000.0, &flemp, irlength);
lowpass(inremp, 1000.0, &fremp, irlength);
lowpass(inlempscal, 1000.0, &flempscal, irlength);
lowpass(inrempscal, 1000.0, &frempscal, irlength);
lowpass(inlmp, 1000.0, &flmp, irlength);
lowpass(inrmp, 1000.0, &frmp, irlength);
lowpass(inlww, 1000.0, &flww, irlength);
lowpass(inrww, 1000.0, &frww, irlength);
```

The low frequency delay for each file is then extracted, as above. As an additional confirmation of the ITD extraction algorithm, both the empirical and scaled empirical dataset are processed.

Details are output to a text file, which lists each location, the extracted delays from each file at that location, and keeps track of the minimum-phase and Functional Model deviation from the empirical data.

Finally, the file *hrtftestingloop4x.cpp* contains the code to process up-sampled HRIRs. Essentially the same as *hrtftestingloop.cpp*, care must be taken to update buffer sizes and sampling rate based processes. Up-sampling of the empirical data, followed by derivation of the minimum-phase and Functional datasets was considered, as was higher resolution FFT analysis. However, as 44.1 kHz and 128-point impulses are assumed to be the most common processing parameters, datasets prepared at 44.1 kHz were up-sampled by a factor of four directly. These high-sampling-rate files provide more accuracy (no extra data can be extracted in the upsampling, but the interpolation in the process leads to more accurate delay extraction).

4.3.1.1 Results

As expected, creating the Functional dataset using different non-linear scaling factors, derived from different radii, gives different results. An optimal radius for minimal deviation from the empirical data appears to be approximately 8.8 cm. The dataset begins to stray further from the empirical data when radii move away from this value, getting either smaller or larger. The filter also has an effect on results. If all data files are high pass filtered, the minimum-phase data is more accurate, due to the spherical-head approximation. If the low-pass filter cutoff is set lower, the Functional Model becomes more accurate.

For a head radius of 8.8 cm and the filter set at 1000 Hz, at 44.1 kHz sampling rate, the minimum-phase Model deviates from the empirical data by a total of 251 samples, whereas the Functional Model only deviates by 211. This is for the entire dataset. For the most relevant horizontal plane, the minimum-phase deviation is 45 samples, the Functional Model 39.

The high resolution results confirm this result. For the same parameters, the minimum-phase deviation is 1076 samples, the Functional Model 827 (see figure

4.3, below). For the horizontal plane, the values are 183 and 156 samples respectively (figure 4.4). The higher error is expected as the sampling rate is higher.

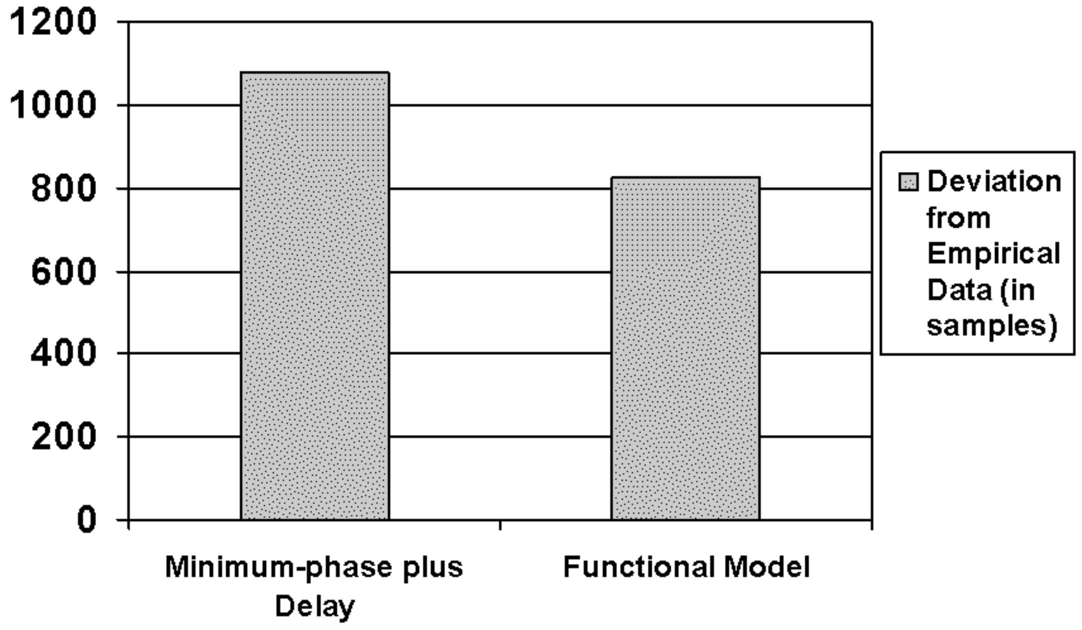


Figure 4.3: Overall high-resolution objective test results: the low frequency ITD extracted from the minimum phase model deviates further from the empirical data than that of the functional model

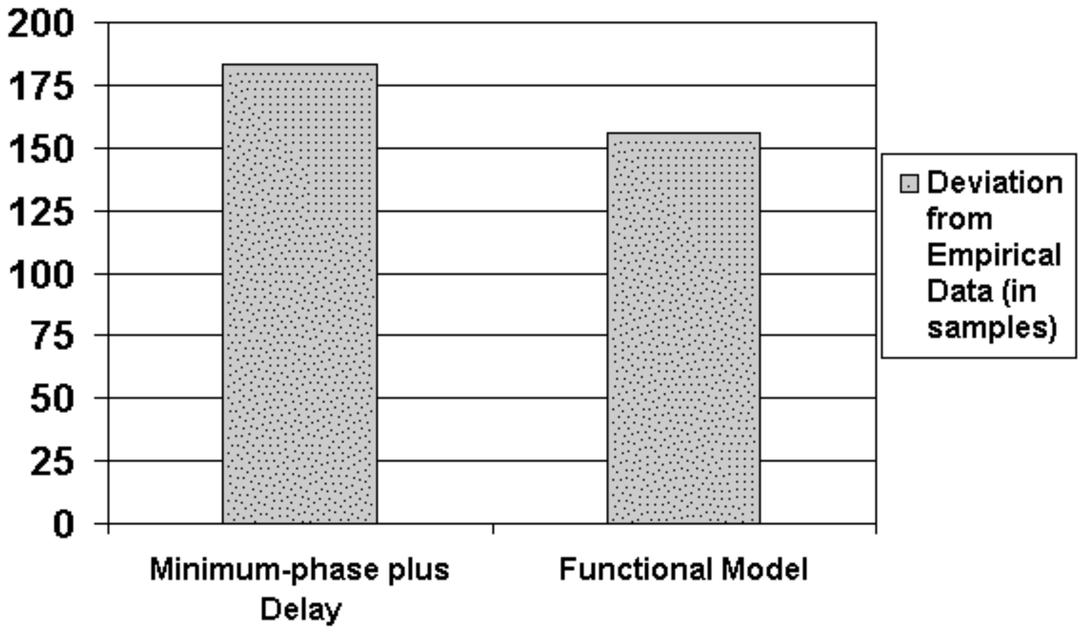


Figure 4.4: Horizontal plane high-resolution objective test results

This frequency specific objective test, developed specifically for the scenario under analysis illustrates clearly that the Functional Model more accurately represents the empirical data from a low-frequency ITD point of view (it also gives an insight into the optimal radius for processing: approximately 8.8 cm). This is the case both for the whole dataset and the most significant horizontal plane. Deviations from agreement with the empirical data are expected in the Functional Model, as the scaling involved is averaged to improve generality and efficiency. Also the filter used is quite forgiving to the minimum-phase approach, as it lets through some higher frequencies which exhibit a much less accurate ITD in the Functional Model, by design. The results for the minimum-phase data not only confirm the success of the Functional Model, but highlight the problems with assuming the all-pass component of the minimum-phase all-pass decomposition is linear. Once again, it should be noted that the delay extraction method used in [111] is used. As discussed, other methods of delay extraction exist (it is also important to realise that the method relies on similarity of the left and right impulses).

4.3.2 Subjective Tests

In the perceptual experiments presented, minimum-phase is presented as the generally accepted basis for achieving source movement by HRTF interpolation. Phase Truncation and the Functional Model are also presented, as the novel methods under investigation.

As discussed previously, some inaccuracies in the minimum-phase representation are mentioned by [111], specifically for contralateral areas and low elevations. It is important to note here that the experiments conducted in Kulkarni *et al* constituted a much more detailed study (preference tests vary in complexity from the extremely detailed to more casual, simple analysis [61]). Subjects were exposed to many more stimuli, over a longer period of time. Also, static sources were used in the comparison of minimum-phase plus delay and empirical measurements. Subjects had several hours of listening experience before the experiment was run, so could be considered experienced. Another important aspect of the experiments in Kulkarni *et al* is the dataset used. Datasets can vary considerably, as highlighted above. This is due to the fact that the hearing mechanism is not physiologically uniform from person to person. So, the minimum-phase representation of the dataset used in the perceptual experiments in Kulkarni *et al* (measured from a human subject) may be relatively more or less accurate per subject than that of the MIT KEMAR dataset employed here. Furthermore, only 50% (two of four) of the experienced listeners in the experiment demonstrated a distinct ability to identify the minimum-phase stimuli when asked to distinguish between minimum-phase and empirical. As mentioned before, this was specifically for the low-frequency content of the stimuli (hence the low-frequency scaling in the Functional Model).

In summary, although minimum-phase has been shown to be accurate for most locations, there are clearly some specific areas in the listener's spatial environment and certain conditions whereby a minimum-phase plus delay approximation of empirical HRTFs is not accurate, as per Kulkarni's and co-authors' seminal paper. However, it is difficult to reproduce these inaccuracies, particularly in brief experiments with non- experienced users. It is therefore anticipated that results of the listening tests *may not* show significant benefits of Phase Truncation or the Functional Model over minimum-phase plus delay. However, numerical tests do show that the Functional Model does provide a more accurate low-frequency phase spectrum when the complete dataset is considered. At the outset, it is expected that, as the current experiments are performed on non-experienced listeners, that a similar, high quality result for all methods will be observed.

Subjective tests were performed using Csound's FLTK opcodes [45]. It was found that a suitable interface could be designed using the flexible nature of the opcodes (the interface is illustrated in figure 4.5). A similar interface to that found in [125, 230 or 86] was designed. The test was designed based on ITU-R standards [85, 86]. Interestingly some problems with biasing in generally employed audio preference tests have recently been highlighted [232]. These include affective biases (based on subject mood, expectations, affective language in test), score mapping (problems with scoring scales) and user interface based biases.

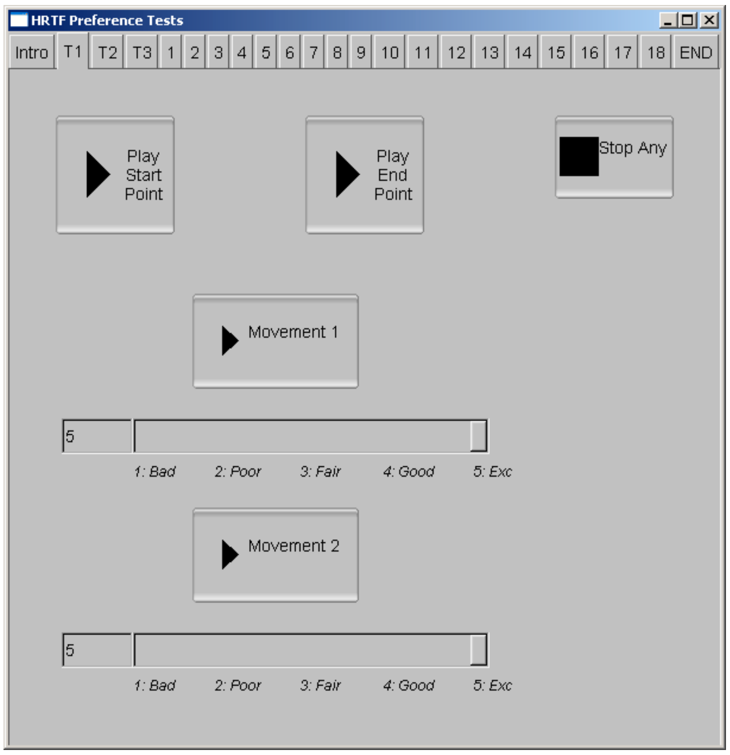


Figure 4.5: Subjective testing interface

A common subjective test of interpolation algorithms is the 4I-2AFC method (used for example in [111]): four interval two alternative forced choice method. Four samples are presented to a subject. The first and last are the same, and one of the second or third is different. Subjects are asked to tell which one. Chance performance then implies no perceptual difference between the samples. Again, due to the nature of the novel algorithms and the necessity for source movement in the final application, a moving source test was developed.

The designed test is similar to the A/B/Ref test defined in GuineaPig [84] (a preference testing software tool; used in [125]) as ‘three samples are played. Samples A and B are graded against the reference.’ Due to the restriction of not having a true reference signal, the source in question processed with static start and end point empirical HRTFs constitute the reference. In this case, the samples to be graded are the same sample moved from start to end point. Obtaining an accurate reference would involve recording each source moving along each trajectory in the same environment where the HRTFs were originally recorded. Practically, this would involve re-recording all HRTF measurements and source movements in the finely controlled circumstances mentioned above.

Consequently, due to the above considerations and limitations, a subjective test based on ITU standards was developed. The minimum-phase, Phase Truncation, Functional Model and original Csound *hrtfer* output are the four algorithms in question. Listeners were asked to rate the samples according to a five-point quality-grading scale [85]. The inclusion of the *hrtfer* output can be seen as an anchor condition, as per the MUSHRA method [86]; also, one of the goals of the current work is to further develop *hrtfer*. Anchors are test signals which are perceptually impaired purposefully to provide a base level (here, *hrtfer* is included as an anchor as it does not perform interpolation). Typically MUSHRA tests include the reference signal, with several samples to be compared, but again, adaptations must be made here due to the lack of a reference signal. Note, in this case, the anchor is not used in each test due to the nature of the designed test. The known issues with the *hrtfer* algorithm for moving sources, as discussed previously make it a suitable anchor candidate. Note that a diffuse dataset was prepared for *hrtfer*, as it uses a non-diffuse field dataset. This ensures that all algorithms use the same dataset. The test will now be described. User instructions read as follows:

Please listen to the sounds at the top of each tab (by clicking the buttons)
They represent a source sound located at a start and end point.
Then please listen to each of the movement sounds below (the sources are repeated in the movement).
These movement sounds aim to smoothly move the source sound from start to the end point.
You will then be asked to judge each movement sound.
PLEASE RATE EACH ON THE FOLLOWING CRITERIA:
You are judging movement sounds on smooth, artefact free movement.

5: Excellent: no distortion or noise
4: Good: some slight distortion or noise
3: Fair: distortion or noise audible
2: Poor: distortion or noise begins to affect listening experience
1: Bad: distortion or noise severely affects listening experience

eg Score of 5 represents smooth movement that sounds like the sound is convincingly moved from start to end point, without any alterations/clicks/noise added to the sound.
Some slight filtering/movement not completely smooth; perhaps the sound is a little altered scores 4.
Some slight clicks/jumps as the sound moves: 3
Clicks/jumps as the sound moves through its trajectory: 2
More severe clicks/jumps: 1

You may listen to each sound any number of times, 3 is recommended, you may need more for early tests to familiarise yourself with the task
Please click 'done' on the last tab when done...

NOTE: You are not judging how convincing the spatial location of the sound is, just the effect the movement has on the source sound, if any;
Some of the start/end points may not be easy to locate in headphones, but you should still be able to rate the movement.
Three training trials (T1-T3) are presented before the trials (1-18) begin...

A thorough, if verbose approach was purposefully chosen after initial trials. It was found that total immersion in the project can lead to inadvertent omission of an important instruction! Verbal instructions were also given both before and during the training phase regarding GUI use, sample content, etc.

The subject of the test: smooth source movement is defined, and the ITU-R quality-based scale defined, with each step explicitly described. Note that non-individualised HRTFs are being used here, which can lead to front-back confusion and localisation inaccuracies (as discussed). Therefore, spatial location is not being assessed in this test. Examples of impairments are also given. Users are permitted to repeat playback of reference and sample files, as desired. Also, users can stop samples if required, and cannot play more than one sample at a time. Participant training is important to the trial [86]; three sample tests are presented. These aim to familiarise subjects with the sound samples, task and interface. Movement was purposefully limited in speed to allow the subject to listen for changes in the output. Source sounds were also repeated to allow for user familiarity (all sources are introduced in the training phase, but preliminary trials suggested that source familiarity for long sources, as well as swift source movement were problematic). Start, mid and end points were considered to aid listeners in discerning audible changes. Mid points were, however deemed unnecessary by listeners in early trials.

Although not typically thought of as a tool for GUI preference test development, Csound proved to be a flexible and efficient environment. The source is included as *preftests.csd* in the 'Chapter4/preftests' folder on the accompanying CD-ROM (which also contains the results and source audio files). The FLTK opcodes are used to setup the GUI and then interact with the playback opcodes. A simple playback instrument (using `loscil`) and stop instrument provide the audio processes. An output text file stores the results. Using the outputs of the FLTK sliders at the time of completion, simple data analysis is also performed and printed to the text file. To avoid real-time processing overhead, all samples are prepared and stored in tables. The FLTK buttons trigger instances of the instruments.

Speech, a noisy signal and a narrow band musical signal were used as samples to cover various temporal and spectral scenarios [98]. Simple (movement in one plane only) and complex source trajectories were tested and spread evenly among the algorithms. Nine subjects were tested, all of whom had some experience of working with audio. The source samples were prepared in such a way to allow for maximum potential for analysis (nine results for each algorithm, three of each algorithm for each source, 18 simple, 18 complex movements, etc). All files were normalised, Phase Truncation uses fades of eight buffers for the vocal and musical sources, and 1 for the noisy source, the Functional Model uses an STFT overlap of 4.

4.3.2.1 Results

Each result file is included in the ‘results’ folder. From a statistical verification point of view, an assessment of normality was performed on the data (the main statistical reference used: [160]). Data in the Functional Model results group was not normally distributed. Parametric tests typically assume that the sample is normally distributed. Non-parametric tests make fewer assumptions (and are therefore appropriate for non-normal data and small subject groups) [232]. Therefore, non-parametric tests were used when investigating significance. The non-parametric equivalent to a one-way repeated measures ANOVA is a Friedman Test. Results of this test show a significant difference between groups. Comparing the ranks for the algorithms illustrates that there is a steady increase in scores in each group with the Functional Model ranking the highest, just above Phase Truncation, followed by minimum-phase and the anchor condition. Descriptive statistics are illustrated in table 4.2, ranks in table 4.3. Further statistical analysis was deemed unnecessary bearing in mind the relatively small sample size.

Method	**Mean**	**Standard Deviation**	**N**
hrtfer	2.77742744	.608413002	9
Minimum-phase Model	4.26723333	.345453380	9
Phase Truncation	4.63926089	.487147836	9
Functional Model	4.71220867	.409740078	9

Table 4.2: Subjective testing descriptive statistics

Method	**Mean Rank**
hrtfer	1.00
Minimum-phase Model	2.33
Phase Truncation	3.17
Functional Model	3.50

Table 4.3: Subjective testing ranks

The above analysis validates the novel algorithms. The means are illustrated visually in figure 4.6. Clearly, Phase Truncation and the Functional Phase Model perform close to a rating of ‘excellent’. Minimum-phase is closer to a rating of ‘good’, indicating that subjects reported some artefacts in minimum-phase based spatialisation. As discussed above, further insight is available due to the considered preparation of the test. Of particular interest here are the mean values for the noise source (figure 4.7). The minimum-phase model’s discrepancies are highlighted here; the noisy source is less forgiving than the narrow-band sources. Admittedly, improvements to the minimum-phase model may be possible, using different techniques for delay extraction or more complex delay line interpolation. Having said this, one of the core aims of the novel algorithms is to reduce these complexities/uncertainties.

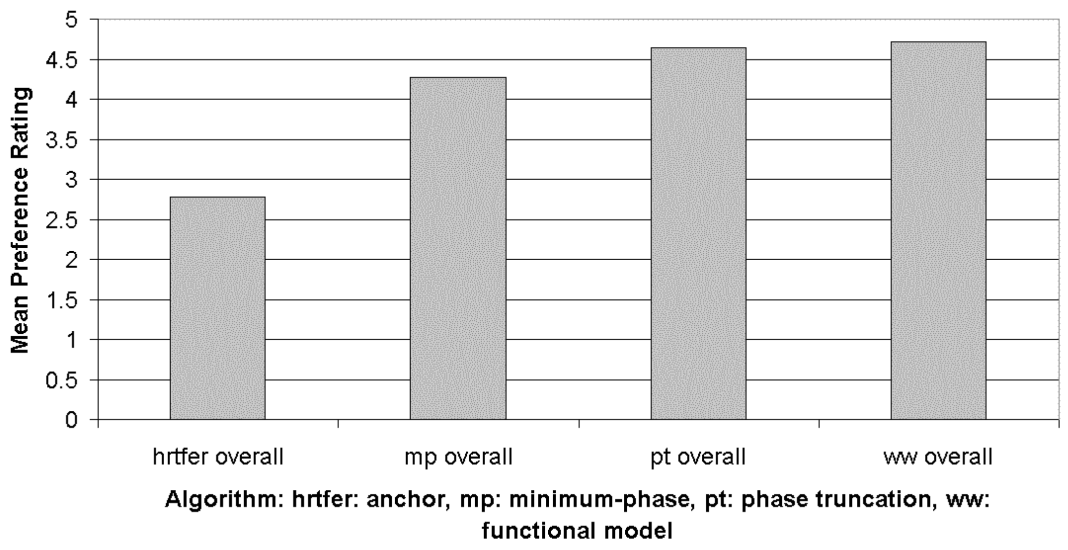


Figure 4.6: Overall preference test means

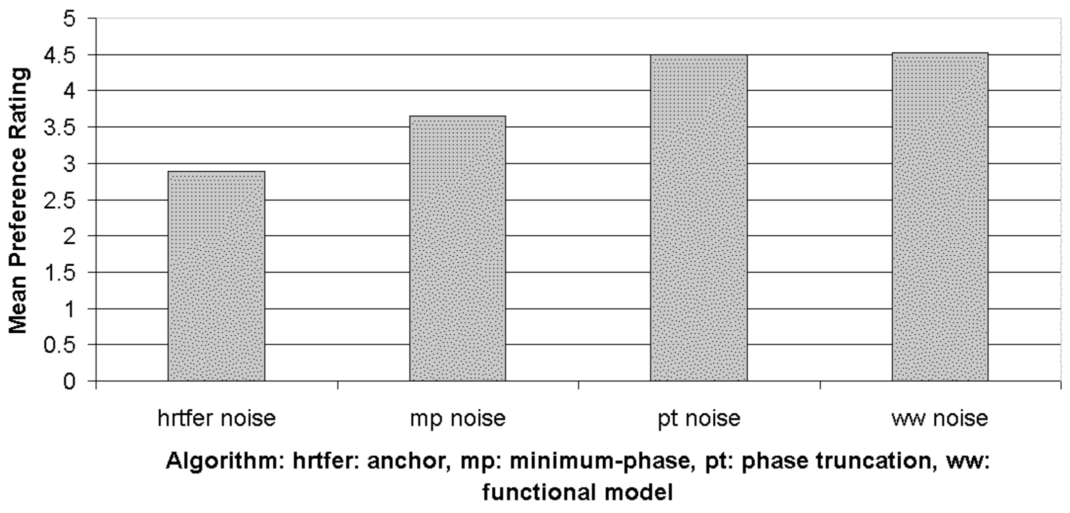


Figure 4.7: Preference test means: noise source

In conclusion, objective and subjective tests indicate that the novel algorithms perform better than a similarly prepared minimum-phase model. Efficient, user-friendly implementations of the algorithms are offered, complemented by what is hoped to be transparent development details. Both novel methods perform excellently; the Phase Truncation Model is more efficient as it uses overlap-add convolution, so should be perhaps preferred in a real-time scenario (see Chapter 5; a discussion of binaural reverberation, which illustrates a direct application of the algorithm).

4.4 Conclusions

In conclusion, this chapter has offered a detailed insight into implementation of the novel algorithms introduced in chapter 3. As with any detailed development project, an abundance of subtle challenges are faced. It is hoped that solutions to these challenges are presented in a more transparent way than the often minimally detailed literature.

Both algorithms offer an empirical alternative to a minimum-phase approach. The Phase Truncation algorithm is perhaps more suited to scenarios involving denser datasets and higher efficiency demand. The Functional Phase Model is perhaps more appropriate in a sparser dataset, where the functional phase can be used to model a more continuous spectrum (it is less efficient, however, as it uses the STFT). Both objective and subjective tests clearly highlight the success of the novel algorithms, which both perform better than a minimum-phase implementation. The goals of the work are thus met and surpassed.

Chapter 5. Binaural Reverberation

5.1 Introduction

A creative or functional use of the HRTF opcodes reveals the anechoic nature of their capture. In typical natural listening situations, environmental processing plays a significant role in the sonic experience. Therefore, it is important to investigate the application of reverberation to binaural audio. In an interesting and pertinent study, Begault [17] investigates the perceptual effects of adding synthetic spatial reverberation (based on [189]) to HRTF spatialised audio. It appears that adding synthetic reverberation to binaural signals greatly improves source externalisation, which perhaps quantifies the anecdotal difficulties reported by this author in using the HRTF opcodes creatively. However, localisation ability does suffer as a result of the addition of artificial reverberation [17]. Perhaps intuitively, an increased sense of distance was also achieved by adding artificial reverberation. The non-individualised nature of the HRTFs used does slightly challenge the results (also, perhaps a more advanced reverberation model may be more appropriate to draw decisive conclusions).

The precedence effect suggests that early reflections arriving within 5 - 40 ms (depending on source type) should not, in typical situations, have an effect on localisation of a direct source [143]. However, in [174], the authors show that the precedence effect does not eliminate all influence of room reflections. This perhaps goes towards explaining the reported reduced localisation ability. As the artificial reverberation under development here aims to simulate real-world scenarios, this reduction in localisation accuracy is perhaps desirable, as it is more appropriate to real-world experiences.

In [233], the authors highlight the need for reverberation processing in virtual audio spaces to improve externalisation. It is concluded in [180] that even simple reverb models aid externalisation; a study is cited in [16] which reports 2% externalisation being improved to 90% by adding reverberation. In [19], this requirement is also discussed; experiments suggested that addition of reverberation also actually improves azimuth localisation abilities (but not elevation). The influence of reflections on distance perception is clearly illustrated in [152].

This author’s initial goal, with regard to reverberation, was to simply examine the literature with the intention of augmenting the HRTF interpolation models with an appropriate and acceptable artificial reverberation, with the focus remaining on the direct source. However, it became clear that the HRTF algorithms could form the distinguishing feature in a much more integrated solution. A more general, user-friendly, parametric and complete solution to binaural processing was thus arrived at. With the focus remaining on advancing tools for Computer Music, usability and efficiency were prioritised and constantly considered when striving for accurate artificial binaural reverberation. The tools arrived at essentially update, re-contextualise, develop and improve existing classic algorithms.

5.2 Literature Review

A vast amount of literature is available on the topic of artificial reverberation, as it has many practical, academic and commercial applications. A discussion of a selection of publications with particular relevance to the context of this study is presented.

5.2.1 Historical Perspective

The discussion below of development of artificial reverberation is summarised in table 5.1.

Reference	**Method**
Schroeder [189], 1962	Comb and all-pass filters
Moorer [145], 1979	Feedback based, development of low-pass sections
Allen and Berkley [6], 1979	Image model
Stautner and Puckette [202], 1982	Feedback delay networks; introducing matrices
Kendall and Martens [102], 1984	Spatial Reverberator
Jot *et al* [various, see text]	Extensive system development, FDN based
Savioja [186], 2000	Wave-based methods

Table 5.1: Historical summary of artificial reverberation

Schroeder's seminal work of 1962 [189] discusses the need for control of frequency response and reflection density in artificial reverberation. He discusses the (frequency-response-based) suitability of comb and all-pass filters for the task, arriving at the conclusion that a mix of both is perhaps most suitable: a number of parallel comb filters feeding into a number of all-pass filters in series. The paper also discusses how longer low-frequency reverb times are inherently appropriate and even offers a solution to multi channel output. Of particular relevance to the current work is Schroeder's assertion that 'there are about 15 large response peaks in every 100 cps interval for a room with 1 sec reverberation'. This is generalised in [193] to:

$$M \geq 0.15 t_{60} f_s \tag{5.1}$$

This formula indicates that the total of the delay lines should be greater than .15 of the reverb time times the sampling frequency. As pointed out in [90], modal density is frequency dependent and is proportional to the square of frequency:

$$Density(f) = 4\pi V \frac{f^2}{c^3}, \tag{5.2}$$

where $V =$ volume, $f =$ frequency and $c =$ sound velocity. More recently [97], this modal density has been further investigated and updated using more complex (and computationally expensive) modal filters.

Schroeder's model was later extended to become the ubiquitous *freeverb*. Moorer [145] gives an historic insight into artificial reverberation before discussing in more detail the inherently low-pass nature of rooms, offering some detail on high-frequency air absorption with distance (this is more significant than it first appears, as sound will travel a significant distance when, for example, higher-order reflections in a reasonably sized room are considered). Further development of the low-pass element of the feedback reverberator and the importance of early reflections are also discussed.

Around the same time, Allen and Berkley [6] suggested using an image model to generate room impulse responses. A robust and pervasive algorithm, the model works by using virtual images of actual source sounds in virtual rooms adjacent to the actual room in question. The model, perhaps best understood visually (see figure 5.1, below), considers a source S, and listener L. Virtual sources V are mirror images of S in ever further virtual adjacent rooms. The visualisation offered here shows a virtual source in the room directly to the right of the actual room. The virtual source here represents a reflection off the right-hand wall. The virtual source in the virtual room to the right of that represents the reflection first off the left wall, then the right (two reflections: second order). Each virtual source thereby represents a reflection pattern. Each can be dealt with separately from a point of view of surface filter genealogy, distance travelled and spatial origin. The arrows in the image show actual trajectories modelled in the actual room, and extended virtual trajectories in the virtual rooms.

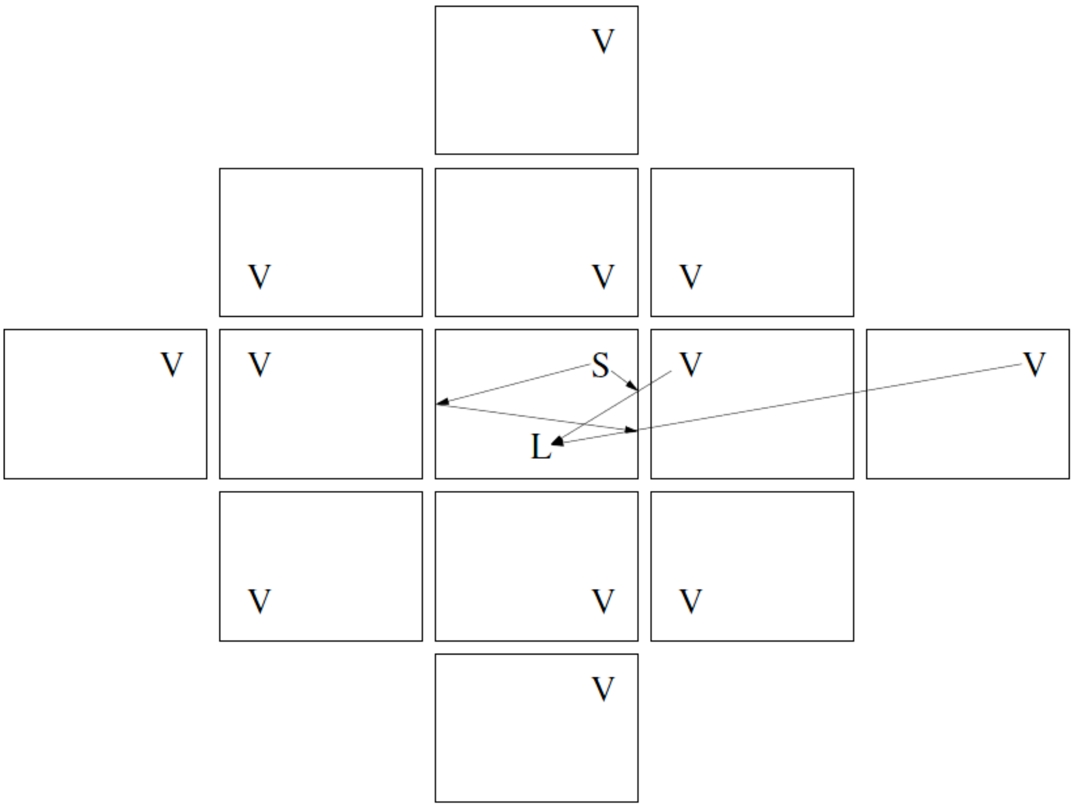


Figure 5.1: The Image Model

Originally designed to consider only shoebox shaped rooms, the model was soon extended to arbitrary polyhedra [23]. An implementation of this extension is discussed in [22], which again breaks artificial reverberation into early reflections (using the image model) and a later reverberant tail. The system is noteworthy in that it is optimised for concert hall modelling. The source and listener are assumed to be positioned centrally, and multi-channel output is suggested.

Ray tracing is another well established (for example it is discussed practically in [106], from 1967, in relation to concert hall acoustics) geometric model, but is criticised by Borish [23]. Briefly, the model assumes the source emits sound particles in all directions, which eventually reach an area around the listener after a series of (usually) specular reflections [23, 14]. An interesting frequency dependent implementation of ray tracing, considering diffusion is presented by Kuttruff [113]. Borish points out that ray tracing can omit reflections in error as rays which are terminated when they reach the listener may actually reach the listener again if continued. The finite number of rays emitted is another issue. The image model, due to its distance-based control, is more suitable for early reflections (both models will produce identical results if processed infinitely [23]).

Shortly after Allen and Berkley's work, Stautner and Puckette [202] introduced the idea [178] of the Feedback Delay Network (FDN). The unit reverberators introduced by Schroeder [189] were reconsidered using matrix mathematics. A number of comb filters were used, all of which fed into a multiplication matrix which dispersed all of the signals (this is discussed in more detail in the implementation section below). Each delay line then essentially feeds into every other delay lines feedback path. Jot, more recently, further investigated FDNs [90, 88, 89, 91 and 93], investigating analysis and synthesis of a particular impulse response, as well as parametric scenarios (again, discussed further below).

In 1990, Kendall *et al* [104] used PCA-based HRTFs and an image model for early reverberation. They discuss, in a practical nature, several perceptual and design idiosyncrasies involved in developing their spatial reverberator (first introduced in 1984 [102]), such as consideration of head rotation and HRIR measurement issues.

In another paper discussing this research [103], which visualises the process very well, the later reverb is discussed. The delay lines used for the first and second-order image model virtual sources feed into a recursive delay unit, thus modelling higher-order rooms. 'In between' rooms are dealt with by cross feeding the delays. Again, the possibility of multiple spatialised outputs is possible.

The 'Ball within the Box' (BABO) [179] paradigm, published in 1995, offers a more general approach, with its roots based in physical modelling, aiming to act as a general physical model of a resonating system. In a manner similar to that eventually adopted here, it uses an image model, and adds diffusion using an FDN. As it is physically based, delay lines used in this FDN are based on room/resonator size [178]. BABO is a complex and accomplished model, however, it does not focus on binaural reverb, which is the main issue dealt with in this thesis.

More recently (2008), Murphy *et al,* after Savioja [186] discuss recent updates to their Renderair system [147]. The system is based on a digital waveguide mesh, an extension of a digital waveguide (which is traditionally used to model the behaviour of a wave pattern on, for example, a string). The authors illustrate how a three-dimensional digital waveguide mesh is prohibitive due to processing power and memory requirements. The time required to compute a .8 second impulse using a 3-D mesh is 14 hours 18 minutes (for a relatively small music practice room). A hybrid approach of separating early and late reverb and using a 3-D mesh for the more significant early reverb and a 2-D mesh for the later reverberant field greatly reduces this computation time. This physical modelling approach offers a high level of accuracy, inherently considering, for example, diffraction. The model performs well in preliminary tests.

These waveguide techniques are summarised in [187, 186]. The waveguide mesh discussed above can be classified as a wave-based model, as it aims to solve the wave equation, using Finite Difference Time Domain methods. Finite Element Modelling and Boundary Element Modelling are other methods which can be considered as wave based. They both attempt to solve the wave equation numerically. The image model and ray tracing discussed above constitute another subsection of computational room modelling, labelled as ‘ray-based’. Scale modelling can also be a useful tool in room acoustics.

Convolution with a measured Binaural Room Impulse Response (BRIR) is another option. The method can be computationally costly, but is optimisable [69]. It also requires an interpolation algorithm to allow for dynamic source/listener behaviour. The difficulties involved are discussed in [205], where the problem is broken down by taking measured impulse responses, truncating early reflections, interpolating and modelling the late reverberant tail.

From an implementation point of view, a dynamically updated FIR for direct sound and early reflections is suggested in [233]. Care must be taken in this scenario to avoid audible inconsistencies as filters change. A dynamic time warping interpolation of early reflections is presented in [99]. As discussed below, an approach based on the novel HRTF interpolation algorithms is presented here.

To conclude this historical perspective, multi-channel reverberation algorithms should briefly be considered, as multi-channel domestic setups become more pervasive. In [190], constant power panning is used to spatialise a source, and spatially sampled FIRs reconstitute early reflections, with incoherent late reverberation for each channel generated recursively. More recently, the image model was used to derive multi-channel impulses from one or two room impulse responses in undetermined scenarios [112]. Related to the topic of multi-channel room impulse responses, Spatial Impulse Response Rendering (SIRR) [168] and the resulting Directional Audio Coding (DirAC) [164] move towards an analysis-synthesis method for analysing the spatial information from an input and outputting to an appropriate multi channel configuration (discussed further in chapter 6).

Commercially, many options for artificial reverb exists [20], with industry favouring Lexicon hardware and software [122], Eventide hardware [58] and Waves IR Convolution [220]. In [20], the authors objectively test six reverberation tools.

They mention that many corporate designs are proprietary, and highlight the large price range (the tested tools range from freeware to €2,000 outboard equipment). Interestingly, they conclude that ideal reverb design is an open question, and that the objective measurements they suggest (including interaural difference and interaural cross correlation) can only imply trends in quality.

5.2.2 A Focused Approach

It is clear from the above discussion that artificial reverberation using HRTFs is not a new area of research. Indeed, Kendall and Martens were working on their ‘spatial reverberator’ in the early 1980s [104], described by Begault as ‘perhaps the first implementation of the image model technique with HRTF filtering’ [14]. A frequently recurring feature of this and the other systems mentioned above is the decomposition of the impulse into early reflections and later reverb. This tradition is honoured here, and immediately raises the question of discriminating between early reflections and later reverberation.

Criteria for early reflections are reviewed in [133]. Options include: fixed values (50/80 ms being judged as ‘early’), reflection order based (widely used, fourth order suggested in literature [150]), mean free path based (the mean distance of a sound ray between two reflections in a room), reflection density (Schroeder suggests 1000 echoes per second [189], Griesinger up to 10000 [75]) and room volume (a simpler measure). Subjective experiments are presented comparing measured BRIRs with synthesised ones. These synthesised BRIRs are created by adding later impulses to early reflections with varying early reflection lengths. Although results are not discussed, reducing early reflections to 20 ms clearly has an adverse effect. In a later work by the same authors, 40 ms is suggested as an appropriate truncation point [134]. Interestingly, this work also confirms the diffuseness of the late tail, as several late tail locations and binaural head orientations were used with no noticeable degradation when interchanged.

Murphy and Stuart [204] use a statistical tool to decide when the early reflections end. The late diffuse field is inherently more normally distributed than the early reflections, which illustrate higher kurtosis. The focus of this study is more on measured, static impulses. The dynamic model presented in this thesis addresses moving sources and, through its integrated design removes the requirement for a crossfade from early to late.

The approach taken to early reflection duration in this work is mean free path and order based, but, more importantly, is parametric: the user is provided with suggestions, but can ultimately control the delay on the reverberant tail. Two opcodes are developed, `hrtfearly`, for early reflections, and `hrtfreverb`, for the later reverberant tail. The opcodes are discussed in detail below from an implementation point of view. Flexibility and usability were core design considerations. Therefore, `hrtfearly` can operate with either a small number of simple inputs for immediate use, or a more complete set of parameters for more expert environment modelling. `hrtfreverb` does not have as many parameters, but does require crucial and sensitive low and high frequency reverb-time arguments. However, `hrtfearly` outputs suggested values for these outputs based on the room geometry used, thus again allowing immediate use. This simple modular approach, inspired by the signal flow design of Csound allows for multiple sources (`hrtfearly`s) feeding into one model of the late reverb (`hrtfreverb`) of the room in question. This completely flexible, real-time, parametric paradigm is maintained throughout. A more physical approach to the control interface is taken than the perceptual approach discussed in [102] or [93].

It was decided not to add an extra low-pass filter for air absorption [see 145, 186], as this can be controlled using the high-frequency response of the surfaces (which will be considered numerous times, based on the number of related reflections) [91]. Alternatively, as Csound is the chosen implementation medium, a simple overall low-pass filter can be inserted into the general signal flow or indeed on the source before it is processed to simulate direct source distance.

It is with all of the above in mind that an early reflection/later reverb is developed. More specifically, the well established geometric image model is used as the starting point for the early reflection processing [6], and an efficient, similarly well established and common recursive model constitutes the basis of the later diffuse tail [91]. In developing a complete model, various updates and improvements are suggested.

5.3 Algorithm Design and Implementation

Considering the literature review and focused approach discussed above, algorithm design and implementation will now be discussed.

5.3.1 Early Reflections

As Smith [193] identifies, early reflections should be spatialised. Kendall and Martens discuss this in a paper giving background to their ‘spatial reverberator’, visually highlighting spatiotemporal changes as source/listener changes location [104]. They therefore spatialise first and second-order reflections. Begault has researched how accurate early reflections need to be, perceptually, from the point of view of artificial reverberation [15], investigating (amplitude) threshold levels for early reflections. In similar, more recent work by Jensen and Welti [87], masking levels in BRIRs were investigated with the intention of simplifying the necessary reverberant filters (replacing masked reflections with a simpler signal) required to model the space in question. A more parametric, dynamic solution is offered in the approach taken here.

In [93], Jot *et al* suggest a simplified model for early reflections. Initially, a stereophonic model is discussed: using only time and intensity differences for early reflections, omitting spectral cues. Using temporal integration as a justification, an averaged spectral filter to consider binaural cues and surface filtering is then suggested. ‘Preliminary’ psychoacoustic testing on these models using individualised HRTFs suggests that this average filter performs well compared to individually spatialised early reflections, with the exception of a significantly delayed lateral reflection. Furthermore, Jot *et al* discuss a diffuse filter applied to the late reflections in their model: the spectrum of the diffuse-field HRTFs multiplied by a Gaussian noise, the spectrum of which is characterised by the room. Using this diffuse filter alone results in some reported perceptual success. This scenario is updated here.

In the model presented here, greater accuracy in early reflections was decided upon for three reasons:

1 The potential unreliability of the precedence effect, discussed above.

2 The ongoing discussion in previous chapters of the overall paradigm of minimisation of data preparation (Jot’s averaged HRTF filter needs to be updated for dynamic sources; also, as discussed in chapter 2, Jot uses minimum-phase processing).

3 The availability of ever increasing processing power.

Early reflections are processed to the same degree of accuracy as the direct source. To the user who may find this to be too costly in processing power/an overestimation of the abilities of the auditory system (despite the above discussion), processing just the direct source is possible. In fact, the algorithm presented is very flexible and parametric; any order of image reflections from 0-4 is possible. The later reverberant field (discussed below) uses improved versions of Jot's FDN.

Therefore, a traditional, simplified model of spatialised direct source plus diffuse reverberant field is available (albeit with improvements discussed below), as well as a more accurate model of spatialised early reflections. Once again, this more accurate model fits with the overall goal of omitting data preparation from the process, allowing immediate, user friendly application and direct processing of empirical HRTFs. This reflection specific, individual approach to early reflections is similar to that employed in [186 and 187], which present an advanced auralisation system, DIVA. Again, the main difference here is the use of empirical HRTFs. Also, the late reverberant field is updated here (as below).

The image model is employed here; its implementation is discussed below. In criticism of the model, it is generally designed to only consider specular reflections. As discussed in chapter 1, sound waves only behave in a specular nature (angle of incidence equal to angle of reflection) when the wavelength of the sound source is smaller than the reflecting surface. In the opposing case, sound waves behave in a diffuse manner: sound reflects in all directions.

Also, high-order image model use can cause a comb filtering effect (as virtual sources are mirrored to equally spaced locations). This is avoided here as an independent model is used for late reverberation processing. A possible improvement to this problem by way of randomisation of virtual source locations was recently proposed [24].

5.3.2 Later Diffuse Field

Following the historical discussion above, the late diffuse field is based on Jot’s FDN. Essentially, the circuit splits an input signal, feeding separate copies to a number of delay lines. The delay lines feed into a matrix, which increases the density of the output, using cross fertilisation. A low-pass filter in each of the delay loops/comb filters allows for frequency-dependent reverberation times. However, reducing the high-frequency reverb time (relative to that of low frequencies; the typical scenario) also reduces high frequency energy, which is not ideal. Therefore, a compensation filter is used to boost these (higher) frequencies after the delay line outputs have been summed.

Jot discusses both reproduction of a measured room impulse and a parametric scenario [90, 88, 89, 91 and 93]. From an analysis-synthesis point of view [88, and the more exhaustive 90], an STFT is performed on a measured impulse. Then, the Energy Decay Relief (EDR) is calculated using a time reversed integration of this analysis:

$$EDR_h(t,f) = \int_t^{\infty} \rho_h(\tau, f)\,d\tau\,, \tag{5.3}$$

where *h(t)* is a time domain signal, transformed to ρ using the STFT.

So, starting at the end of the impulse, the remaining energy at any frequency can be calculated by moving slightly back in time. The EDR follows from the Energy Decay Curve (EDC), which is typically not frequency dependent. Normalising this EDR with the reverberation time constitutes incorporating the tone correction filter. Therefore, the normalised EDR at time 0 represents the initial (frequency-dependent) energy, as opposed to the total energy. Use of the resulting FDN as a more parametric artificial reverberator is discussed as an application of the analysis-synthesis method, with a realisation offered in IRCAM's *Spat* software [198]: a commercial product emerging from the research led by Jot. A comprehensive system, it again uses an early reflection (with an intermediate 'clustered' reflection option)/late reverb approach, minimum-phase HRTFs and several output options. More detail on the parametric scenario is given in [91], with equations for first-order frequency dependent reverb time and tone correction filters suggested (as implemented below).

Very recently, interaural coherence (a frequency dependent function) was added to Jot's FDN [137]. The relevance of coherence is discussed in a previous paper [60] which concludes that interaural coherence aids localisation in complex listening scenarios. Essentially, the authors developed a model of source localisation which only considers ITD and ILD when interaural coherence is above a threshold (in critical bands). This model accounts for documented localisation ability in complex listening environments, so strongly suggests the significant role of interaural coherence. Correct interaural coherence in binaural impulses was implemented for measured impulses in [137], in a manner similar to the analysis-synthesis work done by Jot [90, 88]. Here, this addition is taken further; it is considered in a parametric scenario. More significantly, dynamic sources/room impulses are considered. Interestingly, in a related work, Menzer and Faller have also worked on extracting a BRIR from an ambisonic response, by extracting direction for early reflections and processing with HRTFs [138]. Results in [137] show interaural coherence agreement between measured and synthesised impulse response. Therefore, the general technique is used in this work.

In criticism of this model, the Schroeder frequency is not considered, as conceded by Jot [90]. Also, perceptual research suggests three band FDN absorbent filters [91]. Bearing in mind the potentially high number of delay lines involved (see below), and the desire for a relatively simplified user interface, first-order filters are maintained (a 12th order minimum-phase IIR is used for the tonal filter in [88], second-order are discussed elsewhere [93]).

5.3.3 `hrtfearly`: Early Reflections Implementation

As with the HRTF algorithms, a considerable amount of effort was put into optimal implementation of the reverb algorithms. Command line C-based prototypes were developed for algorithm testing and verification. An accumulative process was employed, starting with a simple Schroeder model, progressing gradually to a full binaural two stage image model/FDN reverb. As with the HRTF algorithms discussed in previous chapters, real-time implementation within a well-supported, flexible framework is desirable. Working implementations of the algorithms in the form of tested, user friendly and immediate software solutions is a priority.

The implementation will now be discussed in some detail, as it is deemed an integral part of this research. The Csound implementations will be discussed, as they represent the majority of the non-trivial aspects of the command line solutions, and offer real-time processing. Aspects covered in previous chapters and trivial code details will not be discussed.

The early reflections code *hrtfearlies.c*, included in appendix 3 and in the ‘Chapter5’ folder on the accompanying CD-ROM essentially embeds the Phase Truncation algorithm within an optimised dynamic image model, including reflection filters. A broad overview of the early reflections algorithm is given in figure 5.2, below.

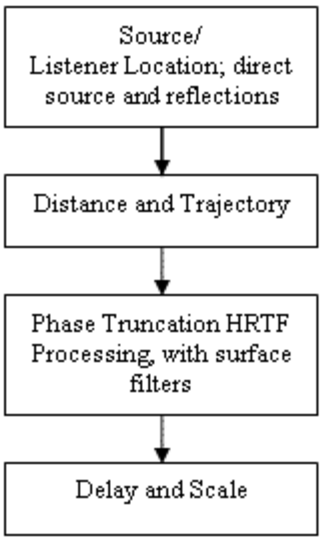

Source/
Listener Location; direct source and reflections
Distance and Trajectory
Phase Truncation HRTF Processing, with surface filters
Delay and Scale

Figure 5.2: Overall early reflection process

As before, an endian based byte switch is defined, as well as the required HRTF dataset constants. The low-pass filter used to apply the coarse surface response of the user-defined environment is a simple first order IIR, equivalent to that implemented in the tone opcode [49]. A gentle low-pass response is achieved by using the following definitions for *a* and *c* and the filter equation [53]:

$$y(n) = ax(n) - cy(n-1) \tag{5.4}$$

where $a = 1 + c$

and $c = \sqrt{(2 - \cos(2\pi \times freq/sr))^2 - 1} - 2 + \cos(2\pi \times freq/sr)$

The filter function arguments start with a pointer to MYFLT, in this case, a processing buffer. High and low response variables are next, followed by delay memory, a processing vector size and sampling rate. The function essentially derives appropriate filter coefficients from its response. Note that the filter is assumed to always be low-pass, in keeping with typical room surfaces and the inherent nature of high frequency energy.

```
MYFLT filter(MYFLT* sig, MYFLT highcoeff, MYFLT lowcoeff,
             MYFLT *del, int vecsize, MYFLT sr)
```

Variables are setup to avoid recalculation and simplify the equations. The high and low coefficients, passed to the function, are absorption coefficients, so the response of the filter at high and low frequencies is calculated by subtracting the coefficients from 1. As the low response is assumed to be greater than the high, it is used as a scaling factor. A low frequency and Nyquist Frequency response are thus arrived at. The Nyquist Frequency response must be less than or equal to $1/\sqrt{2}$ (implying a low-pass response which will have a cutoff/-3dB level at the Nyquist Frequency).

```
MYFLT costh, coef;
int i;

/* setup filter */
MYFLT T = FL(1.0) / sr;
MYFLT twopioversr = FL(2.0 * PI * T);
MYFLT freq;
MYFLT check;
MYFLT scale, nyqresponse, irttwo, highresponse, lowresponse, cosw,
      a, b, c, x, y;

irttwo = FL(1.0 / sqrt(2.0));

/* simple filter deals with difference in low and high */
highresponse = FL(1.0) - highcoeff;
lowresponse = FL(1.0) - lowcoeff;
/* scale factor: walls assumed to be low pass */
scale = lowresponse;
nyqresponse = highresponse + lowcoeff;
/* should always be lowpass! */
if(nyqresponse > irttwo)
      nyqresponse = irttwo;
```

The filter response at cutoff 5000 Hz is illustrated in figure 5.3, below.

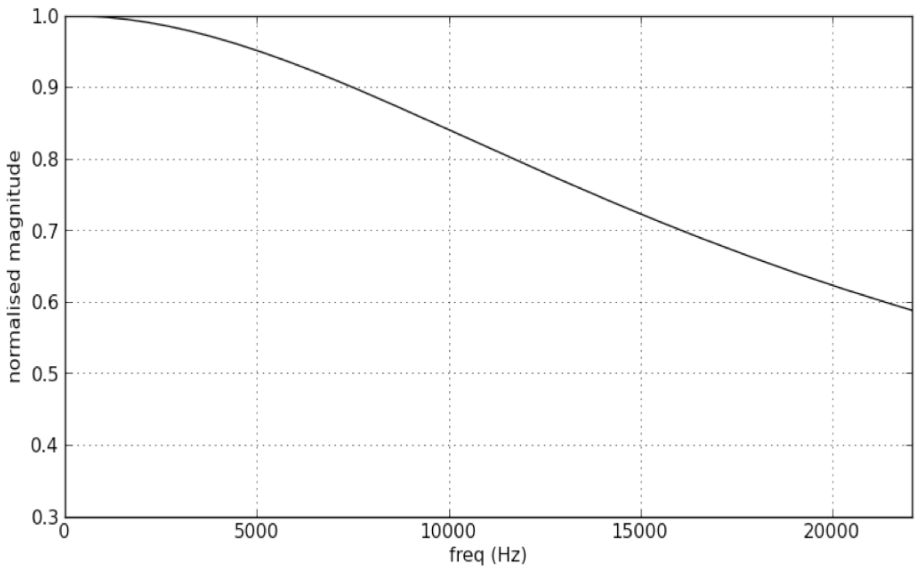


Figure 5.3: Low-pass response

The cutoff frequency (`freq`) of the filter is then obtained, using the response at the Nyquist Frequency. The magnitude response of the filter is used in this calculation, which is outlined below.

If the filter is understood as

$$sig[i] \times (1 + c) - del \times c \,, \tag{5.5}$$

where *del* is the previous output, *sig*[*i*] is the input and *c* is as described above. The magnitude response of this filter can be derived:

$$\left|H(\omega)\right| = \frac{1 + c}{\sqrt{1 + c^2 + 2c\cos(\omega)}} \tag{5.6}$$

The magnitude response at the Nyquist Frequency (*N*) can then be calculated

($\cos(\omega = {}^{2\pi freq}\!/_{SamplingRate})$ will be -1 in this case, as $freq = {}^{SR}\!/_{2}$):

$$N = \frac{1 + c}{\sqrt{1 + c^2 - 2}}$$
$$\Rightarrow N\sqrt{1 + c^2 - 2c} = 1 + c$$
$$\Rightarrow N^2(1 + c^2 - 2c) = 1 + 2c + c^2$$
$$\Rightarrow N^2 + N^2c^2 - N^2 2c - 1 - 2c - c^2 = 0$$
$$\Rightarrow (N^2 - 1)c^2 - 2(N^2 + 1)c + N^2 - 1 = 0$$

Recalling that the value of *N* is known (assuming limits of stability), by design, and that *c* is the unknown, the quadratic formula can be used to solve this quadratic equation for $c = x$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{5.7}$$

In this case,

$a = N^2 - 1$, $b = -2(N^2 + 1)$ and $c = N^2 - 1$.

The discriminant of the quadratic formula determines the nature of the solutions/roots. In this case,

$b^2 = 4(N^4 + 2N^2 + 1)$, and
$4ac = 4(N^4 - 2N^2 + 1)$
$\Rightarrow b^2 - 4ac = 16N^2$

As the discriminant is greater than 0 (assuming a non-zero response at the Nyquist Frequency), two distinct real roots are implied.

Substituting in to the formula, the roots are:

$$x = \frac{2N^2 + 2 \pm 4N}{2(N^2 - 1)}$$
$$= \frac{N^2 \pm 2N + 1}{N^2 - 1}$$

$$= \frac{(N \pm 1)^2}{N^2 - 1}$$

$$= \frac{(N+1)(N+1)}{(N+1)(N-1)} \text{ and } \frac{(N-1)(N-1)}{(N+1)(N-1)}$$

$$= \frac{(N+1)}{(N-1)} \text{ and } \frac{(N-1)}{(N+1)} \tag{5.8}$$

In this scenario, either of these roots will give the same result for the cutoff frequency of the filter, when substituted into the filter equations. This is proven below. In the code, the positive square root of the discriminant is chosen.

```
/* calculate cutoff, according to nyqresponse */
/* w = twopioversr * f (= sr / (MYFLT)2.0) (w = pi in the case of
   nyq...2pi/sr * sr/2) */
/* cosw = (MYFLT)cos(w);... = -1 in case of nyq */
cosw = FL(-1.0);

a = c = FL(SQUARE(nyqresponse) - FL(1.0));
b = (FL(2.0) * cosw * FL(SQUARE(nyqresponse))) - FL(2.0);

/* '+' and '-' sqrt in quadratic equation give equal results in this
   scenario: working backwards to find cutoff freq of simple tone
   filter! */
x = (-b + FL(sqrt(SQUARE(b) - FL(4.0) * a * c))) / (FL(2.0) * a);
```

Once the filter value of *c* has been calculated, further ‘reverse engineering’ is required to arrive at the actual cutoff frequency. As per the low-pass equation (5.4),

$$c = \sqrt{(2-\cos(2\pi \times freq/sr))^2 - 1} - 2 + \cos(2\pi \times freq/sr) \tag{5.9}$$

Therefore, setting

$$y = 2 - \cos(2\pi \times freq/sr), \tag{5.10}$$

$$c = \sqrt{y^2 - 1} - y \tag{5.11}$$

So, in solving for *y*,

$$\Rightarrow c + y = \sqrt{y^2 + 1}$$
$$\Rightarrow c^2 + 2cy + y^2 = y^2 + 1$$
$$\Rightarrow c^2 + 2cy + 1 = 0$$
$$\Rightarrow c^2 + 1 = -2cy$$
$$\Rightarrow \frac{c^2+1}{-2c} = y$$
$$\Rightarrow y = \frac{-c^2-1}{2c}, \tag{5.12}$$

or, in code:

```
y = (-FL(SQUARE(x)) - FL(1.0)) / (FL(2.0) * x);
```

Having arrived at this result, it is possible to prove that both roots of the quadratic equation (which represent the coefficient *c* in the magnitude response) will give the same result when substituted into the filter equation. The variable *c* in the above equation is substituted for each of the roots of the quadratic calculated above in the left and right columns below.

$$y = \frac{-(\frac{N+1}{N-1})^2 - 1}{2(\frac{N+1}{N-1})}$$

$$= \frac{\frac{-(N^2+2N+1)-(N^2-2N+1)}{(N-1)^2}}{2(\frac{N+1}{N-1})}$$

$$= \frac{\frac{-N^2-2N-1-N^2+2N-1}{(N-1)^2}}{2(\frac{N+1}{N-1})}$$

$$= \frac{-2N^2-2}{(N-1)^2} \div \frac{2(N+1)}{N-1}$$

$$= \frac{-2(N^2+1)}{(N-1)^2} \times \frac{N-1}{2(N+1)}$$

$$= \frac{-(N^2+1)}{(N-1)(N+1)}$$

$$= \frac{-(N^2+1)}{(N^2-1)}$$

$$y = \frac{-(\frac{N-1}{N+1})^2 - 1}{2(\frac{N-1}{N+1})}$$

$$= \frac{\frac{-(N^2-2N+1)-(N^2+2N+1)}{(N+1)^2}}{2(\frac{N-1}{N+1})}$$

$$= \frac{\frac{-N^2+2N-1-N^2-2N-1}{(N+1)^2}}{2(\frac{N-1}{N+1})}$$

$$= \frac{-2N^2-2}{(N+1)^2} \div \frac{2(N-1)}{N+1}$$

$$= \frac{-2(N^2+1)}{(N+1)^2} \times \frac{N+1}{2(N-1)}$$

$$= \frac{-(N^2+1)}{(N+1)(N-1)}$$

$$= \frac{-(N^2+1)}{(N^2-1)}$$

The two solutions are equal in this scenario, due to the filter equation. Therefore, only one root needs to be calculated, as above. Having set

$$y = 2 - \cos(2\pi \times freq/sr),$$

the inverse cosine of 2-*y* divided by 2π/SR gives the cutoff frequency:

```
freq = FL(acos(check));
freq /= twopioversr;
```

The filters coefficients can be calculated from the cutoff and used in the derivation of the filtered output: the input is multiplied by *a*, and *c* times the previous output is subtracted. The scaling, as discussed above, completes the process:

```
/* filter */
costh = FL(2.0) - FL(cos(freq * twopioversr));
coef = FL((sqrt(costh * costh - 1.0) - costh));

for(i = 0; i < vecsize; i++)
{
      /* filter */
      sig[i] = (sig[i] * (1 + coef) - *del * coef);
      /* scale */
      sig[i] *= scale;
      /* store */
      *del = sig[i];
}
```

The band-pass filter used to provide the more detailed surface response [175] has been implemented after the `eqfil` opcode [43]. The code has been rewritten for use here and essentially implements the appropriate second-order filter equations, processing vectors of audio in a similar manner to the low-pass function, above:

```
/* band pass for surface detail, from csound eqfil */
MYFLT band(MYFLT* sig, MYFLT cfreq, MYFLT bw, MYFLT g, MYFLT *del,
           int vecsize, MYFLT sr)
{
      MYFLT T = FL(1.0) / sr;
      MYFLT pioversr = FL(PI) * T;
      MYFLT a = FL(cos(cfreq * pioversr * 2.0));
      MYFLT b = FL(tan(bw * pioversr));
      MYFLT c = (FL(1.0) - b) / (FL(1.0) + b);
      MYFLT w, y;
      int i;

      for(i = 0; i < vecsize; i++)
      {
            w = sig[i] + a * (FL(1.0) + c) * del[0] - c * del[1];
            y = w * c - a * (FL(1.0) + c) * del[0] + del[1];
            sig[i] = FL(0.5) * (y + sig[i] + g * (sig[i] - y));
            del[1] = del[0];
            del[0] = w;
      }

      return *sig;
}
```

The broadly object-oriented design of a constructor and processing function, with a dataspace is once again followed here. As before, significant variables from the dataspace/internal variables in the defined structure will be discussed where relevant. Of particular note here is the requirement for multiple Phase Truncation processing variables, which are stored in dynamically allocated memory (elevation/angle indices, etc). Required inputs to the opcode are the unspatialised audio, the source and listener geometric location, HRTF data files and a default room. These requirements are purposefully minimised for immediate and convenient use. Optional arguments provide much more detailed processing: the number of Phase Truncation fade buffers, sampling rate, order of image model processing, inclusion of reflections from three dimensions, as opposed to solely the horizontal plane, a head-rotation value (k-rate), then room size and surface low and high-frequency absorption coefficients and band pass parameters for the walls, floor and ceiling.

The initialisation function declares local variables before dealing with defaults for optional parameters. Phase Truncation fades default to eight processing buffers, as before. By default, image sources in two dimensions are processed to first order. Phase Truncation based setup of buffer sizes and data file reading is performed. Three ‘preset’ rooms are available, number 1 being the default. Wall, floor and ceiling coefficients are setup if a default room is chosen:

```
if(defroom)
{
      p->wallcoefhigh = FL(.3);
      p->wallcoeflow = FL(.1);
      p->wallg1 = FL(.75);
      p->wallg2 = FL(.95);
      p->wallg3 = FL(.9);
      p->floorcoefhigh = FL(.6);
      p->floorcoeflow = FL(.1);
      p->floorg1 = FL(.95);
      p->floorg2 = FL(.6);
      p->floorg3 = FL(.35);
      p->ceilingcoefhigh = FL(.2);
      p->ceilingcoeflow = FL(.1);
      p->ceilingg1 = FL(1.0);
```

```
        p->ceilingg2 = FL(1.0);
        p->ceilingg3 = FL(1.0);
}
```

Values listed in [80] are loosely followed, to imply plasterboard walls, painted plaster ceilings and carpet floors, with some perceptual tweaking. A value of 0 entered into the preset room parameter implies that optional parameters will be used. High and low absorption coefficients and band-pass gains are then checked and set. Absorption coefficients must fall between 0 and 1, and band-pass gains between 0 and 10:

```
else
{
        p->wallcoefhigh = (*p->owlh > FL(0.0) && *p->owlh < FL(1.0)) ?
                           *p->owlh : FL(.3);
        p->wallcoeflow = (*p->owll > FL(0.0) && *p->owll < FL(1.0)) ?
                          *p->owll : FL(.1);
        p->wallg1 = (*p->owlg1 > FL(0.0) && *p->owlg1 < FL(10.0)) ?
                     *p->owlg1 : FL(.75);
        p->wallg2 = (*p->owlg2 > FL(0.0) && *p->owlg2 < FL(10.0)) ?
                     *p->owlg2 : FL(.95);
        p->wallg3 = (*p->owlg3 > FL(0.0) && *p->owlg3 < FL(10.0)) ?
                     *p->owlg3 : FL(.9);
        p->floorcoefhigh = (*p->oflh > FL(0.0) && *p->oflh < FL(1.0))
                            ? *p->oflh : FL(.6);
        p->floorcoeflow = (*p->ofll > FL(0.0) && *p->ofll < FL(1.0)) ?
                           *p->ofll : FL(.1);
        p->floorg1 = (*p->oflg1 > FL(0.0) && *p->oflg1 < FL(10.0)) ?
                      *p->oflg1 : FL(.95);
        p->floorg2 = (*p->oflg2 > FL(0.0) && *p->oflg2 < FL(10.0)) ?
                      *p->oflg2 : FL(.6);
        p->floorg3 = (*p->oflg3 > FL(0.0) && *p->oflg3 < FL(10.0)) ?
                      *p->oflg3 : FL(.35);
        p->ceilingcoefhigh = (*p->oclh > FL(0.0) && *p->oclh <
                              FL(1.0)) ? *p->oclh : FL(.2);
        p->ceilingcoeflow = (*p->ocll > FL(0.0) && *p->ocll < FL(1.0))
                             ? *p->ocll : FL(.1);
        p->ceilingg1 = (*p->oclg1 > FL(0.0) && *p->oclg1 < FL(10.0)) ?
                        *p->oclg1 : FL(1.);
        p->ceilingg2 = (*p->oclg2 > FL(0.0) && *p->oclg2 < FL(10.0)) ?
                        *p->oclg2 : FL(1.);
        p->ceilingg3 = (*p->oclg3 > FL(0.0) && *p->oclg3 < FL(10.0)) ?
                        *p->oclg3 : FL(1.);
}
```

Room preset 1 is a medium sized room, room 2 is small and room 3 is large:

```
/* medium room*/
if(defroom == 1)
{
        rmx = 10;
        rmy = 10;
        rmz = 3;
}
/* small*/
else if(defroom == 2)
{
      rmx = 4;
      rmy = 4;
      rmz = 3;
}
/* large*/
else if(defroom == 3)
{
      rmx = 20;
      rmy = 25;
      rmz = 7;
}
```

Optionally, parameters for the room size are checked (minimum room dimensions are $2 \times 2 \times 2$) and read:

```
/* read values if they exist, use medium if not valid (must be at
   least a 2*2*2 room!*/
else
{
      rmx = *p->ormx >= FL(2.0) ? *p->ormx : 10;
      rmy = *p->ormy >= FL(2.0) ? *p->ormy : 10;
      rmz = *p->ormz >= FL(2.0) ? *p->ormz : 3;
}
```

The number of sources (the actual source and the image sources) is then calculated. The value of the `impulses` integer starts at 1, as order 0 implies processing once: the direct source. In the two-dimensional case, the number of sources can be calculated by iterating up to the order number, and adding four times each iteration (see figure 5.1). Order 1 will have $1+4$, order 2, $1+4+8$, etc. In the three-dimensional case, images including ceiling/floor reflections need to be considered. The `temp` variable, initialised to 2, facilitates this calculation. The number of two-dimensional sources is calculated and increased by the appropriate number of three-dimensional images. Essentially, twice the preceding number of threads is added to `temp`, which starts out at 2 (one source for the topmost image, one bottom). For example, order three: the first iteration of the loop, `impulses` = 5, `temp` = 2 + 2(5) = 12 (essentially calculating order 2 three-dimensional sources), the second, `impulses` = 13, `temp` increases to 38, the third, `impulses` = 25. At this stage, the `temp` calculation is already completed, so the final addition happens, `impulses` = 63 and the loop is complete.

The number of sources is passed back to the dataspace, to be used in the performance function.

```
/* how many sources? */
if(threed)
{
      for(i = 1; i <= order; i++)
      {
            impulses += (4 * i);
            if(i <= (order - 1))
                  /* sources = 2d impulses for order, plus 2 * each
                     preceding no of impulses eg order 2: 2d = 1 + 4
                     + 8 = 13, 3d + 2*5 + 2 = 25 */
                  temp += 2*impulses;
            else
                  impulses = impulses + temp;
      }
}
else
{
      for(i = 1; i <= order; i++)
            /* there will be 4 * order additional impulses for each
               order */
            impulses += (4*i);
}
p->impulses = impulses;
```

Next, memory is dynamically allocated and zeroed, in a similar fashion to the HRTF opcodes. A number of sources are processed, depending on the order chosen by the user. Memory is allocated accordingly. The next task involves reverb time calculation. As a rectangular room is assumed, opposite surfaces will have the same surface area, so only three surface area calculations/variables are required.

```
wallS1 = rmy * rmz;
wallS2 = rmx * rmz;
cfS = rmx * rmy;
```

The Norris-Eyring reverb time formula is then calculated for the defined room [80]:

$$T_{60} = \frac{-0.161V}{\sum_{i=1}^{surf\,aces} S_i \ln(1-\alpha_i(f))}, \qquad (5.13)$$

where $\alpha_i(f)$ is the absorbtion coefficient of the surface at frequency f, S_i is its surface area and V the rooms volume. The denominator for low and high frequency is calculated for each surface, following from the code below.

```
Salphalow = wallS1 * FL(log(1.0 - p->wallcoeflow)) * FL(2.0);
```

A low and high frequency reverb time are thus arrived at. Informal listening tests suggested that a Q factor of .2666667 (a 4 octave bandwidth) provides appropriate bandpass characteristics. The -3dB cutoff frequencies are listed in the code (calculated using the formulas listed: 62.5 Hz – 1,000 Hz, cf 250 Hz, 250 Hz – 4,000 Hz, cf 1,000 Hz, 1,000 Hz – 16,000 Hz, cf 4,000 Hz) imply a wide filter, with smoother responses, the subtlety of which suits this application.

As delay is used to simulate source distance, a maximum delay is required to setup delay lines. The hypotenuse rule is used to calculate the maximum possible path in the room, which is extended to 3 dimensions if necessary:

```
maxdist = FL(sqrt(SQUARE(rmx) + SQUARE(rmy)));
if(threed)
      maxdist = FL(sqrt(SQUARE(maxdist)+SQUARE(rmz)));
maxdist = maxdist * (order + 1);
```

As mentioned in the code, a per-order calculation could potentially reduce memory requirements, but would introduce further complexity, and is deemed unnecessary when low-order processing is expected and imposed. The mean free path, used to calculate a suggested delay for the later reverberant tail is then calculated:

```
meanfreepath = FL(4.0) * vol / (surfacearea * p->c);
```

Delay line memory can then be allocated. As with the real-time HRTF opcodes, processing is optimised by minimising unnecessary calculations for static sources. Current locations are checked against previous ones to check if a source has moved, which implies the necessity to interpolate HRTFs etc. Therefore, the first values used in the check are set to be illegal, to ensure the first pass (initialisation) processes the source.

The processing function, as mentioned above, essentially nests HRTF Phase Truncation processes in a structure setup in the initialisation function. The first task completed by the processing function is to test for legal source/listener locations. These values are restricted to being inside the room:

```
if(srcx > (rmx - FL(.1)))
      srcx = rmx - FL(.1);
if(srcx < FL(.1))
      srcx = FL(.1);
```

The source and listener are expected to have a minimum physical size, so cannot lie on the boundaries (the limitation imposed is to be within 10cm of each boundary). Next, a k-rate section of the code calculates distances, delays and amplitudes for each source (the real one and all images). This processing only occurs if the source or listener has moved since the last processing pass:

```
if(srcx != p->srcxk || srcy != p->srcyk || srcz != p->srczk ||
   lstnrx != p->lstnrxk || lstnry != p->lstnryk ||
   lstnrz != p->lstnrzk)
```

Check values for this optimisation are first stored for the next pass:

```
p->srcxk = srcx;
```

Each source is processed in a nested loop operation. Convenient formulae for image model calculation are given in [131]. Basically, the i^{th} virtual source can be found using:

$$x_i = (-1)^i x_s + \left[i + \frac{1-(-1)^i}{2} \right] x_r , \tag{5.14}$$

where *i* is the image number, x_s is the source location and x_r the room dimension, all in the *x* plane. Subtracting the listener location gives the distance from (virtual) source to listener in the *x* plane. Negative values of *i* imply sources lying on the negative *x* axis. For example, the second-order image to the left of the actual room: *i* = -2, results in a location of:

$$x_i = (-1)^{-2} x_s + \left[-2 + \frac{1-(-1)^{-2}}{2} \right] x_r$$
$$x_i = x_s - 2x_r \qquad (5.15)$$

Positive values, conversely, represent positive *x* axis values:

$$x_i = (-1)^{2} x_s + \left[2 + \frac{1-(-1)^{2}}{2} \right] x_r$$
$$x_i = x_s + 2x_r \qquad (5.16)$$

Thus the geometry system is setup and calculated in each plane (including the z plane if three-dimensional processing is being considered). Nested loops start at the virtual point furthest from the real room on the negative plane, and move to that on the positive plane:

```
for(xc = -order; xc <= order; xc++)
```

The formula is interpreted for real-time processing thus:

```
formxpow = (int)pow(-1.0, xc);
formx = (xc + (1 - formxpow)/2) * rmx;
tempsrcx[M] = formxpow * srcx + formx;
```

Unnecessary duplication of calculation is thus avoided. The same author (as [131]) also suggests improving the speed of calculation of full impulses using sorted lookup tables [132], however, the low-order, real-time binaural nature of this implementation of the image model does not require this optimisation. The distance of each source is simply calculated using Pythagoras' Theoroem, and transformed into time by dividing by speed. Minimum distance is set to .45 m, as HRTF processing within this range is not accurate, as the near-field HRTF changes with distance (see below). Amplitude factors for each reflection are then calculated, avoiding distortion. A digital delay time is calculated for each image.

Audio-rate processing uses a variable delay line (which allows for 0 delay by writing to the delay line before reading) for each image, each contributing to the overall output for each sample. Care is taken to read the allocated buffers of memory at the correct locations, and to apply the correct amplitude to each reflection:

```
for(M = 0; M < impulses; M++)
{
      /* a rate vdel:*/
      rp = delp[M] - vdt[M];
      rp = (rp >= 0 ? (rp < maxdelsamps ? rp : rp - maxdelsamps) :
      rp + maxdelsamps);
      frac = rp - (int)rp;
      /* shift into correct part of buffer*/
      pos = (int)rp + skipdel[M];
      /* write to l and r del lines*/
      dell[delp[M] + skipdel[M]] = predell[counter + M * irlength] *
                                   amp[M];
      delr[delp[M] + skipdel[M]] = predelr[counter + M * irlength] *
                                   amp[M];
      /* read, at variable interpolated speed*/
      outltot += dell[pos] + frac*(dell[(pos + 1 < (maxdelsamps +
                                   skipdel[M]) ? pos + 1 :
                                   skipdel[M])] - dell[pos]);
      outrtot += delr[pos] + frac*(delr[(pos + 1 < (maxdelsamps +
                                   skipdel[M]) ? pos + 1 :
                                   skipdel[M])] - delr[pos]);
      delp[M] = (delp[M] != maxdelsamps - 1 ? delp[M] + 1 : 0);

      outsigl[j] = outltot;
      outsigr[j] = outrtot;
}
```

As with the HRTF opcodes, the rate at which the interpolation is performed is dictated by the length of the impulse responses used: 128 samples. Each reflection is once again considered, as above. Interpolation only occurs if the source has moved since the last HRTF buffer size process (an independent check to the k-rate check above).

At this point, the issue of near-field HRTFs is considered. In [56], the authors state that neglecting HRTF range dependence is ‘invalid for nearby sources’. As mentioned in chapter 2, within five times head radius (approximated to .45 m here, as above) is defined as significantly range dependent in [55]. A check for near-field sources is therefore made; near-field processing of HRTFs is not performed. The

HRTF used thus stays the same within this range. Relative *x* and *y* axis source to listener variables are calculated and the inverse tangent is used to calculate the source angle. A degree value, relative to polar North is calculated with a clockwise orientation:

```
/* - to invert anticlockwise to clockwise*/
angle = FL(-(atan2(tempy, tempx)) * 180.0 / PI);
/* add 90 to go from y axis (front)*/
angle = angle + 90;
```

The nature of the C++ `atan2` function dictates that only the case where both x and y variables are 0 needs to be checked for. If this occurs, the source is assumed to be in front of the listener.

Elevation calculation is a little more involved. A triangle is created whose apexes consist of the source, the listener, and a point directly above/below the source and level with the listener. The cosine rule can then be used to calculate the elevation of the source (the length of each line thus needs to be calculated):

```
/* cosine rule */
coselev = FL((SQUARE(bc) + SQUARE(ab) - SQUARE(ac)) /
             (2.0 * ab * bc));
elev = FL(acos(coselev)* 180.0 / PI);
```

If the source and listener are at the same x, y point, the source and listener are at the same location, or directly above/below each other:

```
/* source at listener*/
if(ac == FL(0.0))
      elev = FL(0.0);
      /* source above listener*/
else
      elev = FL(90.0);
```

If the z coefficient of the source is less than that of the listener, the angle is made negative, indicating a source below the listener. Elevation values are then checked, and index values are calculated as per the HRTF opcodes. Head rotation is considered in the angle calculation, by subtracting the arriving value, for example, an angle of 0 degrees with a rotation of 90 implies an angle of -90 degrees. The phase truncation process continues, with care being taken to read the correct values for the image in question, for example the old indices, 'cross' flag, etc. Dynamically allocated buffers must also be dealt with in a considered manner, using the `M` variable, for example `currentphasel/r` etc.

Surface reflections are considered next. Once again, convenient formulae are provided in [131]. The coefficient of each wall needs to be raised to the power of the number of reflections off that wall. The latter part of this calculation is formulated thus:

$$\left|.5i - .25 + .25(-1)^i\right| \tag{5.17}$$

for the wall (along the *x* axis) closest to the origin and

$$\left|.5i + .25 - .25(-1)^i\right| \tag{5.18}$$

for the wall opposite it, where *i* is the order of the virtual source in question. For example, order 2 implies one reflection off the wall closest to the origin and one off the opposite wall. Similarly, y and z axes wall reflections can be calculated. In code:

```
wallreflections = (int)abs((int)(xc * .5 - .25 +
                           (.25 * pow(-1.0, xc))));
```

Filters are applied directly to the HRIR, using the equations discussed above. Essentially, a series of filter processes occurs iteratively, for each reflection. In minimising the already lengthy list of expert parameters, each wall is assumed to have the same filter parameters in two-dimensional processing, so all wall filters can be considered as the same process. The left and right HRIR are low-pass filtered, then each band of the band-pass filter is processed (with the fixed parameters discussed above).

```
for(i = 0; i < wallreflections; i++)
{
      delsinglel = delsingler = FL(0.0);
      filter(hrtflinterp, p->wallcoefhigh, p->wallcoeflow,
             &delsinglel, irlength, sr);
      filter(hrtfrinterp, p->wallcoefhigh, p->wallcoeflow,
             &delsingler, irlength, sr);
      deldoublel[0] = deldoublel[1] = deldoubler[0] = deldoubler[1]
                   = 0.0;
      band(hrtflinterp, FL(250.0), FL(250.0) / p->q, p->wallg1,
           deldoublel, irlength, sr);
      band(hrtfrinterp, FL(250.0), FL(250.0) / p->q, p->wallg1,
           deldoubler, irlength, sr);
      deldoublel[0] = deldoublel[1] = deldoubler[0] = deldoubler[1]
                   = 0.0;
      band(hrtflinterp, FL(1000.0), FL(1000.0) / p->q, p->wallg2,
           deldoublel, irlength, sr);
      band(hrtfrinterp, FL(1000.0), FL(1000.0) / p->q, p->wallg2,
           deldoubler, irlength, sr);
      deldoublel[0] = deldoublel[1] = deldoubler[0] = deldoubler[1]
                   = 0.0;
      band(hrtflinterp, FL(4000.0), FL(4000.0) / p->q, p->wallg3,
           deldoublel, irlength, sr);
      band(hrtfrinterp, FL(4000.0), FL(4000.0) / p->q, p->wallg3,
           deldoubler, irlength, sr);
}
```

Note that delay memory can be reused in this scenario, as the filtering process is not performed on a continuous signal. The memory is zeroed on each pass, and a 128-sample HRIR is fully processed each time. The three-dimensional processing addition for floor and ceiling reflections illustrates how surfaces can be considered individually. HRIRs are zero padded, transformed to the frequency domain, and stored. The overlap-add convolution output process is then dealt with, as in the HRTF opcodes, bearing in mind the multi-image processing. This prepares the pre-delay output buffers, to be used in the audio-rate section of the processing function. It is hoped that the above discussion illustrates the original aspects of this real-time implementation of a binaural image model, as well as the many non-trivial programming design and implementation issues that are typically not discussed in the literature. The full code is presented in Appendix 3, with comments which highlight issues not discussed above.

5.3.4 `hrtfreverb`: Diffuse Field Implementation

The reverberant tail code (see appendix 3, and the accompanying CD-ROM) for the opcode `hrtfreverb` will now be discussed. The first new construct in this code is the declaration of the matrices used in the FDN process. Householder matrices are defined (size: 6×6, 12×12 and 24×24). A list of prime numbers, used as delay line lengths are defined next (primes are used to avoid common factors/emphasis in delay lines). The inputs and outputs of this opcode are a little simpler than that of `hrtfearly`. Output values are the stereo processed signal, and an i-rate variable which offers a suggested delay for the late tail. Inputs are the mono, non-spatialised input, a low and high reverb time, HRTF data file names/locations, and three optional parameters. These consist of sampling rate, mean free path and processing order. The reverb times and mean free path can be derived from an instance of `hrtfearly`, and the same order of processing can be used. This illustrates how the opcodes are intrinsically linked. Having said this, as discussed above, `hrtfreverb` is also designed to function as a stand-alone opcode.

The dataspace of `hrtfreverb` is less similar to that of the HRTF opcodes than that of `hrtfearly`. Delay-line iterators and dynamic memory pointers predominate. The filters, discussed above, used to achieve correct interaural coherence and tonal correction also require memory.

The initialisation function/constructor is again substantial, primarily due the setting up of the aforementioned filters. Another design issue is highlighted here. As presented, the opcodes can use the MIT HRTF data [142] at three sampling rates. This implies three sets of datafiles (two files for each set: left and right data). Adding the filter coefficients to these datafiles was considered. Ultimately, however, a simple dataset format was decided upon. Should the need arise, the code can be modified to accept other datasets, for example the LISTEN [124] database, as illustrated in the HRTF processing command-line code in ‘Chapter4/listen’. To conform to the broad infrastructure of the HRTF suite of opcodes, and benefit from the optimisations offered, any dataset must be prepared in a manner that is compatible with their design. Maintaining a simple structure here is thus desirable: a left and right HRTF datafile ordered and stored in polar format. Adding complex filter data to this file format complicates the process. Therefore, the opcode itself, which will be the sole user of the filter information, performs this calculation.

As before, variable declaration will be discussed in the context of non-trivial use, and is liberally commented in the code. The mean free path with respect to time (an optional input, which defaults to 0) is set to that of a medium room if it is not set, if it is less than or equal to the mean free path of the smallest allowed room in `hrtfearly` ($2 \times 2 \times 2$) or if it is greater than 1 (which implies a mean free path of 344 metres, implying a very larger space):

```
if(meanfp <= 0.003876 || meanfp > 1)
      meanfp = FL(0.0109);
```

Processing order defaults to 1 (as a *p* type [44]) and must be between 0 and 5. Reverb times must be positive and non-zero. Processing buffer sizes are setup, HRTF data files are opened and filter memory is allocated dynamically, as before. Some of this dynamically-allocated memory will be filled in the initialisation function, so does not need to be zeroed. Delay iterators are zeroed before delay line lengths are decided upon. A maximal reverb time is arrived at from the opcode inputs:

```
delaytime = rt60low > rt60high ? rt60low : rt60high;
```

Individual delay line lengths are then calculated, and form a key part of the FDN, as discussed above. Firstly, Schroeder’s criterion of 0.15 modes per Hz is considered:

```
delaytime /= 7;
```

Overall delay time should be greater than .15 (c1/7) of reverb time. The mean free path is an intuitively appropriate average delay, for each delay line in the FDN [193]. Therefore, the appropriate number of delay lines is decided upon by dividing the appropriate delay time by each available number of delay lines (6, 12 or 24), subtracting the mean free path, taking the absolute value, and using the lowest result:

```
/* which no. of delay lines implies ave delay nearest to mfp(which
   is an appropriate ave)? */
Msix = abs((int)(delaytime / 6) - meanfpsamps);
Mtwelve = abs((int)(delaytime / 12) - meanfpsamps);
Mtwentyfour = abs((int)(delaytime / 24) - meanfpsamps);
M = Mtwelve < Mtwentyfour ? (Msix < Mtwelve ? 6 : 12) : 24;
```

The delay time is then divided by the number of delay lines, to find an appropriate average delay. If the new delay time value is less than the mean free path with regard to time, it can be increased to the mean free path, thereby increasing the modes/Hz and having a more desirable average delay. A check is made for the maximum value in the array of primes (delay line length distribution is discussed below). A minimum value check also occurs, independently for each number of delay lines. Essentially, these checks are needed as a base delay is decided upon and chosen by proximity in the primes array. Other delays are chosen to be above and below this value. The checks ensure that this delay line length allocation does not go out of range. The base delay figure is chosen using the following code:

```
/* choose appropriate base delay times */
for(i = 0; i < 212; i++)
{
      if(M == 6)
            test = (i > 6 ? i : 6) - 6;
      else if(M == 12)
            test = (i > 15 ? i : 15) - 15;
      else
            test = (i > 16 ? i : 16) - 16;

      if(primes[i] > delaytimeint || primes[test] >
         meanfpordersamps)
      {
            basedelay = i - 1;
            if(primes[test] > meanfpordersamps)
```

```
                printf("\nfdn delay > earlies del..., fixed!");
            *p->idel = FL(meanfpordersamps - primes[test - 1]) / sr;
            break;
        }
}
```

This code loops through 212 prime values in the primes array. If the current value in the array is greater than the chosen delay time (as above), the base delay is set to the previous prime. Therefore, the maximum value chosen is value 210 (last array iteration: `i` = 211). As the array contains 229 values, array point 228 contains the last (0 is included as an array index). Therefore, up to 18 greater primes are available. If the appropriate base delay is found, the program breaks out of the loop. If the shortest of the prime values used as delay line lengths is greater than the delay implied by the mean free path and the order of processing (which is output by the opcode as an i-rate value), real-time processing (bearing in mind the inherent delay of one convolution buffer) is not possible, as the early reflections need to be delayed in this scenario to achieve a suitable delay on the later reverberation (if the suggested mean free path/order based delay is desirable). A test variable is used in this check to determine the shortest delay. If 6 delay lines are used, the shortest will be the base delay–6, as values above and below the base delay are used. Similarly, 12 delay lines imply a subtraction of 15 and 24 a subtraction of 16. This variable needs to stay positive.

The suggested output delay is also offered at this stage. The inherent delay in the FDN is subtracted from the delay implied using the mean free path and order inputs, to give the appropriate delay for the late reverberant tail. Just before this construct in the code, maximum and minimum values for the base delay are set. A maximum value of 10112 samples is set and minimum values are set to ensure there are enough primes before the base value to allow for the prime selection process, which chooses primes above and below the base value. The more delay lines used, the greater the spread of delay line lengths. Therefore, a higher minimum value exists for 12 and 24 delay line scenarios.

The array of primes is chosen somewhat arbitrarily but does possess a broad structure. Two primes in every hundred are chosen, with four in every hundred below 400 to allow for short reverb times. Also, approaching the value chosen for the high limit, independent reflections are audible due to the limited number of delay lines. Interestingly, this can result in an appealing compositional effect, but an unnatural reverberation. Typically, the extremely large rooms implied by such long delay lines are very unrealistic (the longer delays imply distances of approximately 80 metres in typical scenarios). The checks and limitations involved offer flexible processing, beyond the necessities of the reverberation algorithm and into a more creative realm, while also ensuring stability. Delay lines are then allocated:

```
/* fill delay data, note this data can be filled locally */
delaysp[0] = primes[basedelay];
delaysp[1] = primes[basedelay + 3];
delaysp[2] = primes[basedelay - 3];
delaysp[3] = primes[basedelay + 6];
delaysp[4] = primes[basedelay - 6];
delaysp[5] = primes[basedelay + 9];
if(M ==12 || M==24)
{
      delaysp[6] = primes[basedelay - 9];
      delaysp[7] = primes[basedelay + 12];
      delaysp[8] = primes[basedelay - 12];
      delaysp[9] = primes[basedelay + 15];
      delaysp[10] = primes[basedelay - 15];
      delaysp[11] = primes[basedelay + 18];
}
if(M ==24)
{
      /* fill in gaps... */
      delaysp[12] = primes[basedelay + 1];
      delaysp[13] = primes[basedelay - 1];
      delaysp[14] = primes[basedelay + 4];
      delaysp[15] = primes[basedelay - 4];
      delaysp[16] = primes[basedelay + 7];
      delaysp[17] = primes[basedelay - 7];
      delaysp[18] = primes[basedelay + 10];
      delaysp[19] = primes[basedelay - 10];
      delaysp[20] = primes[basedelay + 13];
      delaysp[21] = primes[basedelay - 13];
      delaysp[22] = primes[basedelay + 16];
      delaysp[23] = primes[basedelay - 16];
}
```

Values in the vicinity of the base delay are chosen to maintain the desired average delay. Also, the prime numbers minimise undesirable combing effects of delay lines accumulating (although a physical approach is also possible [179], this more generic and typical approach is favoured here). Delay lines are then setup and zeroed, as before:

```
if (!p->del1.auxp || p->del1.size < delaysp[0] * sizeof(MYFLT))
      csound->AuxAlloc(csound, delaysp[0] * sizeof(MYFLT),
      &p->del1);
```

Interaural coherence filters are then setup. This involves iterating through each HRTF file and extracting the appropriate information. As the dataset is symmetrical in this case (as discussed in previous chapters), measurements are doubled, as, for example, the HRTF for 0 degree elevation, 90 degrees angle is the same as that for 0 degree elevation, 270 degree angle with the channels interchanged [142]. This is, however, in error for measurements that are on the median plane, which should only be included once. As discussed above, the methodology from [137] is followed to obtain the interaural coherence. Firstly, the power spectrum is obtained:

$$\frac{1}{N}\sum_{i=1}^{N}\left|L_i(\omega)\right|^2+\left|R_i(\omega)\right|^2, \tag{5.19}$$

where there are N HRTFs in the dataset. The magnitude of the complex numbers representing each frequency bin is already stored in the datafile format used. The symmetry of the dataset is dealt with by considering the left and right buffers of each HRTF thus:

```
powerp[j] = powerp[j] + (MYFLT)SQUARE(bufflp[j]) +
            (MYFLT)SQUARE(buffrp[j]);
```

Locations on the median plane are omitted here using the following check (which appears inelegant but is necessary due to the nature of the data measurement):

```
if(i == 0 || i == 28 || i == 29 || i == 59 || i == 60 ||
   i == 96  || i == 97 || i == 133 || i == 134 || i == 170 ||
   i == 171 || i == 207 || i == 208 || i == 244  || i == 245 ||
   i == 275 || i == 276 || i == 304  || i == 305 || i == 328 ||
   i == 346 || i == 347 || i == 359 || i == 360 || i == 366 ||
   i == 367)
      skipdouble = 1;
else
      skipdouble = 0;
```

The `skipdouble` variable is used to flag iterations whereby the symmetrical doubling can be skipped. In this case, only the following code is necessary:

```
powerp[j] = powerp[j] + (MYFLT)SQUARE(bufflp[j]);
```

Instead of considering the left and right power, only the left is considered. Thereby, the full dataset is considered correctly. Each HRTF is considered once, in the symmetrical HRTF case by using both the left and right (which essentially represents the left channel in the opposite HRTF; the opposing hemisphere) functions. An average power spectrum is arrived at by performing the division by the total number of HRTFs in the full dataset. 0 Hz and the Nyquist Frequency are considered independently in a similar manner. The interaural coherence is calculated as:

$$\phi(\omega) = \frac{\left|\sum_{i=1}^{N} L_i(\omega) R_i^*(\omega)\right|}{\sqrt{\sum_{i=1}^{N} |L_i(\omega)|^2 \sum_{i=1}^{N} |R_i(\omega)|^2}} \tag{5.20}$$

The numerator is calculated first and requires conversion back to rectangular form as the complex conjugate and complex multiplication is required. 0 Hz and the Nyquist Frequency are dealt with in a straightforward manner, being purely real:

```
nump[0] = nump[0] + (bufflp[0] * buffrp[0]) + (buffrp[0] *
          bufflp[0]);
nump[1] = nump[1] + (bufflp[1] * buffrp[1]) + (buffrp[1] *
          bufflp[1]);
```

By multiplying the left data by the conjugate of the right, and vice versa, opposing hemispheres are both considered. The `skipdouble` check is once again utilised to resolve the symmetry issues: only the left data multiplied by the conjugate of the right is necessary in the median plane (as left data = right data in these cases). The multiplication is then achieved thus:

```
for(j = 2; j < irlength; j += 2)
{
      rel = bufflp[j] * (MYFLT)cos(bufflp[j + 1]);
      iml = bufflp[j] * (MYFLT)sin(bufflp[j + 1]);
      rer = buffrp[j] * (MYFLT)cos(buffrp[j + 1]);
      imr = buffrp[j] * (MYFLT)sin(buffrp[j + 1]);
      if(skipdouble)
      {
            nump[j] = nump[j] + ((rel * rer) + (iml * imr));
            nump[j + 1] = nump[j + 1] + ((rel * -imr) +
                          (iml * rer));
      }
      else
      {
            nump[j] = nump[j] + ((rel * rer) + (iml * imr)) +
                      ((rer * rel) + (imr * iml));
            nump[j + 1] = nump[j + 1] + ((rel * -imr) + (iml * rer))
                          + ((rer * -iml) + (imr * rel));
      }
}
```

Magnitudes are derived to complete the numerator calculation. Calculating the denominator is trivial in the case of a symmetrical dataset, as the left and right power is equal:

```
for(i = 0; i < irlength; i++)
      denomp[i] = powerp[i];
```

Finally, interaural coherence filters can be obtained using the following formulae:

$$u(\omega) = \sqrt{\frac{1+\phi(\omega)}{2}} \tag{5.21}$$

$$v(\omega) = \sqrt{\frac{1-\phi(\omega)}{2}}\quad , \tag{5.22}$$

implemented as code:

```
coherup[0] = FL(sqrt((1.0 + cohermagsp[0]) / 2.0));
coherup[1] = FL(sqrt((1.0 + cohermagsp[1]) / 2.0));
cohervp[0] = FL(sqrt((1.0 - cohermagsp[0]) / 2.0));
cohervp[1] = FL(sqrt((1.0 - cohermagsp[1]) / 2.0));

for(i = 2; i < irlength; i += 2)
{
      coherup[i] = FL(sqrt((1.0 + cohermagsp[i]) / 2.0));
      cohervp[i] = FL(sqrt((1.0 - cohermagsp[i]) / 2.0));
      coherup[i + 1] = FL(0.0);
```

```
        cohervp[i + 1] = FL(0.0);
}
```

Inverse Fourier transforms (without the need to go from rectangular to polar, as zero phase values imply equality) give the FIR filter coefficients. Note that the filters are shifted for causality; as zero phase has been applied, the filter wraps around the zero time point in the time domain. Therefore, a shift (half the filter length/typically 64 taps) is applied:

```
filtoutp[i] = HRTFavep[(i + (irlength / 2)) % irlength] * irlength;
```

Filters for power, left and right coherence are then zero padded and Fast Fourier Transformed for overlap-add convolution. These filters are kept at the HRTF resolution of 128 samples at 44.1 kHz (using overlap-add convolution with zero padding to 256). This design decision was made to maintain consistent binaural accuracy levels. Simpler, less costly IIRs may be appropriate in low complexity datasets, but future possible developments are considered regarding other datasets.

Efficient IIR filters are, however, used to model the room response [91]. Local copies of relevant dataspace members are assigned, as before, to avoid unnecessary referencing. Each delay line path essentially acts as a comb filter, with a frequency dependent gain factor. The gain of a comb filter can be described as:

$$g = 0.001^{\frac{\tau}{T_{60}}}, \tag{5.23}$$

where τ is the delay time and T_{60} the reverb time [53]. The reverb time is, in this scenario, frequency dependent. Therefore, so is the gain. The filter for each delay line can thus be described as:

$$\left|H(e^{j\omega T})\right| = 10^{\frac{-3\tau}{T_{60}(\omega)}}$$

$$\Rightarrow \log_{10}\left|H(e^{j\omega T})\right| = \frac{-3\tau}{T_{60}(\omega)}$$

$$\Rightarrow 20\log_{10}\left|H(e^{j\omega T})\right| = -60\frac{\tau}{T_{60}(\omega)} \text{ (as in [193])}$$

Jot [91] uses first order IIR filters to model this response:

$$H_i(z) = g_i \frac{1-a_i}{1-a_i z^{-1}}, \tag{5.24}$$

where $g_i = 10^{\frac{-3M_iT}{T_{60}(0Hz)}}$, M_iT is the delay line length as a function of time, and

$$a_i = \frac{\ln(10)}{4}\log_{10}(g_i)(1-\frac{1}{\alpha^2})$$
$$(\alpha = \frac{T_{60}(Nyquist)}{T_{60}(0Hz)})$$

Equations are implemented from [193], after [91]. Note that the formulas are valid for ‘not too small values of α and not too long delays’ [91]: i.e. realistic room parameters (checks are made for stability of output, as opposed to valid formulae, as the user may wish to employ extreme parameters for compositional use).

The value of α is simply calculated as:

```
alpha = rt60high / rt60low;
```

The values for low and high reverberation times have already been checked, as above. For each delay line, the constant value part of a_i is calculated as:

```
aconst = FL((log(10.0) / 4.0) * (1.0 - (1.0 / SQUARE(alpha))));
```

Actual values of a_i and g_i are then calculated using a loop:

```
for(i = 0; i < M; i++)
{
      exp = FL((-3.0 * delaysp[i] * T) / rt60low);
      gip[i] = FL(pow(10.0, exp));
      aip[i] =  exp * aconst;
      …
```

Stability is ensured by avoiding values of a_i of 1 or greater, as the filter equation involves multiplying the previous output by a_i. The filter can be inverted. This implies the extraordinary scenario of an overall high-pass room, which is counterintuitive when the transfer of high-frequency energy is concerned (remembering that `hrtfreverb` may be acting independently of `hrtfearly`, by design). Stability is similarly ensured in this scenario. If the filter becomes unstable for any of the delay lines, the `clipcheck` flag is set, and a linear response is imposed (the `do, while` loop is restarted):

```
if(aip[i] > .99 || aip[i] < -.99)
{
      printf("\nwarning, approaching instability, fixed with a flat
              late reverb!");
      clipcheck = 1;
      if(aip[i] > .99)
            rt60high = rt60low;
      else
            rt60low = rt60high;
      break;
}
```

Note that the stability of g_i is ensured as 10 is raised to a negative power, implying a value below 1 and above 0 for positive delay length and reverb time (both intuitively so). The tonal correction filter, used to compensate for the reduction of energy with reduction of delay time in the (typically) higher frequencies is also setup in the initialisation function. As discussed above, this is a first order FIR filter [193, 91]:

$$\frac{1-bz^{-1}}{1-b} \qquad (5.25)$$

The coefficient b is calculated as:

$\frac{1-\alpha}{1+\alpha}$, with α as before.

In code:

```
p->b = FL((1.0 - alpha) / (1.0 + alpha));
```

To complete the complex initialisation function, dataspace variables are zeroed/initiated as appropriate.

The processing function uses the preparation performed in the initialisation function to process the audio. Local copies of buffers/variables are declared, for example, the delay lines:

```
del1p = (MYFLT *)p->del1.auxp;
del2p = (MYFLT *)p->del2.auxp;
del3p = (MYFLT *)p->del3.auxp;
del4p = (MYFLT *)p->del4.auxp;
del5p = (MYFLT *)p->del5.auxp;
del6p = (MYFLT *)p->del6.auxp;

if(M==12 || M==24)
{
      del1tp = (MYFLT *)p->del1t.auxp;
      del2tp = (MYFLT *)p->del2t.auxp;
      del3tp = (MYFLT *)p->del3t.auxp;
      del4tp = (MYFLT *)p->del4t.auxp;
      del5tp = (MYFLT *)p->del5t.auxp;
      del6tp = (MYFLT *)p->del6t.auxp;
}
if(M==24)
{
      del1tfp = (MYFLT *)p->del1tf.auxp;
      del2tfp = (MYFLT *)p->del2tf.auxp;
      del3tfp = (MYFLT *)p->del3tf.auxp;
      del4tfp = (MYFLT *)p->del4tf.auxp;
      del5tfp = (MYFLT *)p->del5tf.auxp;
      del6tfp = (MYFLT *)p->del6tf.auxp;
      del7tfp = (MYFLT *)p->del7tf.auxp;
      del8tfp = (MYFLT *)p->del8tf.auxp;
      del9tfp = (MYFLT *)p->del9tf.auxp;
      del10tfp = (MYFLT *)p->del10tf.auxp;
      del11tfp = (MYFLT *)p->del11tf.auxp;
      del12tfp = (MYFLT *)p->del12tf.auxp;
}
```

The processing loop runs, as before, to the length of a control period. As per figure 5.4, two uncorrelated outputs are taken from the FDN. This is done, following [137], by ensuring that the vectors *c* and *d* are perpendicular. Every second value is simply made negative in each of the three even number of delay line cases (6, 12 or 24). The cross product of the vectors clearly illustrates this perpendicularity. As also mentioned in [137], keeping both scaling vectors to the same absolute value/magnitude is also inherently advisable to maintain some parity in the levels of both output channels. Hence the non-zero vectors. In previous incarnations of the FDN [193, 90, 137], each signal was scaled before being input into the delay line.

Initially, this signal flow was employed here also. However, in investigating efficiency, this scaling was moved to the output of the FDN, which reduces 6/12/24 divisions to 2. The factor of division is equal to the number of delay lines, and is essentially used to avoid distortion resulting from the combination of all matrix outputs. Figure 5.4, below, illustrates the overall process. The input is processed by the early reflections opcode, and added to the delayed, scaled output of the FDN. This gain and delay processing is assumed to be performed externally (a suggested delay is offered by `hrtfreverb`). The input signal is passed through the matrix. A left and right output are then derived using the processes discussed.

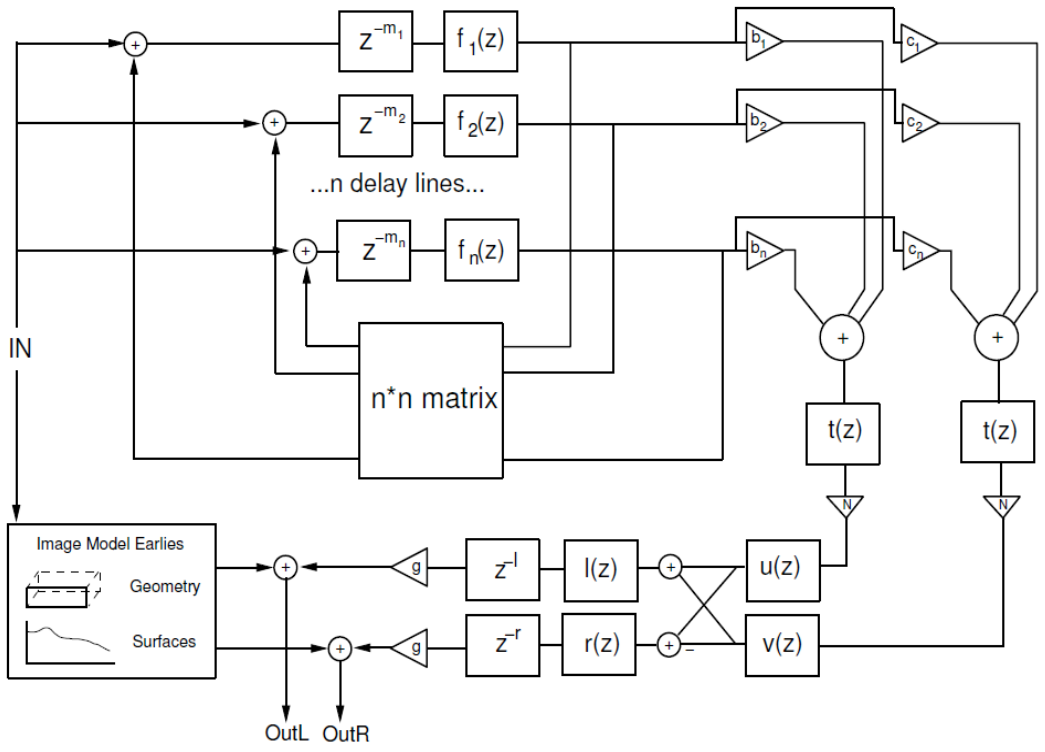


Figure 5.4: Schematic of overall binaural reverberation process: the input is sent to the FDN as well as the early model; the FDN input gets split and sent to each delay line, which also contains a low-pass filter. Uncorrelated outputs are then tone corrected and processed with coherence and binaural filters, delayed and scaled.

To complete this section of the code, the FIR tone correction filter is implemented.

The filter's transfer function:

$$\frac{1-be^{-j\omega}}{1-b} \tag{5.26}$$

can be rewritten as the filter's equation thus:

$$y(n) = \frac{e^{j\omega n} - be^{j\omega[n-1]}}{1-b}, \tag{5.27}$$

or

$$y(n) = \frac{1}{1-b}x(n) - \frac{b}{1-b}x(n-1) \tag{5.28}$$

In code:

```
/* dot product of l and r = 0 for uncorrelated */
tonall = (del1p[u] - del2p[v] + del3p[w] - del4p[x] + del5p[y] –
          del6p[z]);
if(M==12 || M==24)
      tonall += (del1tp[ut] - del2tp[vt] + del3tp[wt] - del4tp[xt] +
                 del5tp[yt] - del6tp[zt]);
if(M==24)
      tonall += (del1tfp[utf1] - del2tfp[vtf1] + del3tfp[wtf1] –
                 del4tfp[xtf1] + del5tfp[ytf1] - del6tfp[ztf1] +
                 del7tfp[utf2] - del8tfp[vtf2] + del9tfp[wtf2] -
                 del10tfp[xtf2] + del11tfp[ytf2] - del12tfp[ztf2]);

matrixlup[counter] = FL(((1.0 / (1.0 - b)) * tonall) - ((b /
                     (1.0 – b)) * inoldl));
matrixlup[counter] /= M;
inoldl = tonall;

tonalr = (del1p[u] + del2p[v] + del3p[w] + del4p[x] + del5p[y] +
          del6p[z]);
if(M==12 || M==24)
      tonalr += (del1tp[ut] + del2tp[vt] + del3tp[wt] + del4tp[xt] +
                 del5tp[yt] + del6tp[zt]);
if(M==24)
      tonalr += (del1tfp[utf1] - del2tfp[vtf1] + del3tfp[wtf1] –
                 del4tfp[xtf1] + del5tfp[ytf1] - del6tfp[ztf1] +
                 del7tfp[utf2] - del8tfp[vtf2] + del9tfp[wtf2] -
                 del10tfp[xtf2] + del11tfp[ytf2] - del12tfp[ztf2]);

matrixrvp[counter] = FL(((1.0 / (1.0 - b)) * tonalr) - ((b /
                     (1.0 – b)) * inoldr));
matrixrvp[counter] /= M;
inoldr = tonalr;
```

Inputs to the FDN are taken from the delay lines. Each delay line is passed through the appropriate low-pass filter, the coefficients of which were calculated in the initialisation function. As above, the filter equation can be arrived at from its transfer function:

$$H(e^{j\omega}) = g_i \frac{1-a_i}{1-a_i e^{-j\omega}}, \tag{5.29}$$

or

$$y(n) = g_i(1 - a_i)x(n) + a_i y(n-1) \quad (5.30)$$

In code:

```
for(j = 0; j < M; j++)
{
	inmatlpp[j] = (gip[j] * (1 - aip[j]) * inmatp[j]) + (aip[j] *
			dellpp[j]);
	dellpp[j] = inmatlpp[j];
}
```

Matrix multiplication is the next required process. Several matrices have been suggested, as discussed practically in [193, 89]. A householder matrix:

$$A_N = I_N - \frac{2}{N}\underline{u}_N \underline{u}_N{}^T \text{ [193]}, \quad (5.31)$$

where $\underline{u}_N{}^T = [1, 1, \ldots, 1]$ and I_N is the identity matrix, was found to perform well, so is chosen here (realised in the 6×6, 12×12 and 24×24 matrices at the start of the `hrtfreverb` opcode). Stability issues are reported in [204] not only with the embedded Householder, but also with a non-embedded Householder (with an ad hoc fix suggested; no issues were found with non-embedded Householder matrices in this work). Non-embedded matrices are thus used, which unfortunately reduces efficiency. Therefore non-power-of-two sizes can be used (if the matrix order is power-of-two size, a scaled unitary matrix can be used, avoiding a scaling factor for each operation). Each item in the output array is arrived at by multiplying the appropriate line in the appropriate matrix by each of the items in the input audio vector. Thereby, every delay line input is incorporated into each output, thus diffusing the input signal:

```
for(j = 0; j < M; j++)
{
	outmatp[j] = FL(0.0);
	for(k = 0; k < M; k++)
	{
		if(M == 24)
			outmatp[j] += (matrix24[j * M + k] * inmatlpp[k]);
		else if(M == 12)
```

```
                        outmatp[j] += (matrix12[j * M + k] * inmatlpp[k]);
                else
                        outmatp[j] += (matrix6[j * M + k] * inmatlpp[k]);
        }
}
```

To conclude the delay process, delay lines are filled with the input, delay line iterators progress and are checked. Audio-rate output is created in this way. Once again, an internal control rate of the HRTF filter size is also constantly processing. If an internal counter reaches this value, the FDN outputs are zero padded, Fast Fourier Transformed and convolved with the left and right interaural coherence filters. The final step in the application of the interaural coherence is then applied, as per figure 5.4, above.

```
for(j = 0; j < irlength; j++)
{
        hrtflp[j] = matrixlup[j] + matrixrvp[j];
        hrtfrp[j] = matrixlup[j] - matrixrvp[j];
}
```

The power filter is applied in another convolution, as the final step in the late reverberation process. The suggested delay of the late reverb onset and a (possibly related) user-defined scaling factor can easily be applied using existing Csound functionality. To complete the opcode, dataspace variables are updated for the next control rate.

5.4 Conclusion

It is hoped that the many subtle (and indeed not so subtle, for example filter stability) nuances involved in coding a functioning, open source, reliable implementation of existing artificial reverberation algorithms, with the addition of several updates, are somewhat demystified in the above chapter. It is hoped that the occasionally ostensibly verbose discussion of implementation topics so often omitted from the literature is thus justified.

In summary, a review of artificial reverberation, in the context of binaural processing was offered. A re-appraisal of classic methods and integration of more recent developments led to the development of two opcodes, `hrtfearly` for high resolution, flexible early reflection processing and `hrtfreverb`, which provides an efficient, stable recursive diffuse field, with accurate interaural coherence (built on a dynamic FDN). The opcodes are designed to be user friendly, while also offering detailed control if required. They also integrate with each other well, as `hrtfearly`’s outputs can be used to inform the processing in `hrtfreverb`.

Chapter 6. Applications

6.1 Introduction

An application of the HRTF and binaural environmental processing tools will be discussed in this chapter. The tool, MultiBin, is essentially an auralisation tool. A user can place a source in a room of their own design; both source and listener are dynamic in this real time scenario. Although designed to be generic, the primary application of MultiBin is the audition of multi-channel audio algorithms. Essentially, each loudspeaker in a desired multi-channel setup is modelled binaurally. The multi-channel signal is then sent to the application for dynamic audition. The chapter thus commences with a brief overview of multi-channel processing (reduced in scope from previous literature reviews), followed by discussion of MultiBin, from an implementation and usage point of view.

6.2 Historical Context of Multi-channel Audio

In [127], an historic overview of spatialisation techniques, tools and trends is offered. The earliest system discussed used multiple telephone transmitters and receivers in 1881. The stereo techniques of Blumlein followed in the 1930s, followed by Disney's *Fantasia* multi-channel development. The *Musique Concrète* and *Elektronische Musik* schools of the 1950s further developed spatialisation as a composition tool. Computer based research then began to inform spatialisation and environmental processing systems, the point at which the literature review offered here, and in chapter 2 essentially begins. A brief overview of multi-channel spatialisation algorithms is offered below, in the context of a multi-channel binaural tool.

6.3 Stereo

Although stereophony strictly refers to any system that delivers spatial sound, ‘stereo’ playback has come to represent two-channel audio reproduction [182]. Typically, amplitude differences between the left and right loudspeaker give rise to the perception of phantom sources between them. In [166], Pulkki demonstrates how amplitude panning laws work: the ipsilateral loudspeaker signal is combined with the delayed contralateral signal for each ear. Relative amplitudes imply different phases when summed, which in turn imply ITD. The study uses a binaural model to illustrate that amplitude panning performs well, particularly at lower frequencies. A binaural model is used for testing [167].

It is generally accepted that an equilateral triangle, whose apexes consist of the two loudspeakers and listener constitutes the ideal reproduction scenario [182]. Wider angles (greater than 60 degrees) give rise to less stable phantom sources [127]; in quadraphonic systems, the loudspeakers subtend 90 degrees, which is thus problematic. Various methods exist for stereo source capture and artificial spatialisation. Ultimately, the technique is limited to phantom sources between the loudspeakers (with the exception of more complex approaches to enhancing the spatial scene, such as transaural processing [68]) and a small sweet spot [127] (restricted by the Precedence Effect [41]).

6.4 Vector Base Amplitude Panning (VBAP)

VBAP stems from constant gain amplitude panning, essentially extending stereo to the full horizontal plane or full three-dimensional processing. It uses the nearest loudspeakers (two for horizontal, three for full three dimensions) to the desired source location for reproduction [165]. It offers a flexible, efficient solution.

6.5 5.1

5.1 is a front-centric multi-channel system [182], often used with visual presentations. It thus suffers from wide angles between front and surround loudspeakers. It also suffers from sweet spot issues [200, 127]. An interesting criticism is presented in [200]. In cinema reproduction, a listener sitting toward the rear of the room may actually be behind one of the surround channels (they are typically duplicated in large reproduction rooms). Therefore, any intended spatial image is destroyed.

6.6 Ambisonics

Ambisonics offers a holistic approach; it considers capture, storage and flexible reproduction of spatial audio [71, 70]. In [127], the theory is explained. Ambisonic signals are stored in B-format, a flexible format which stores the velocity component of the signal in various directions, as well as the overall pressure component. Simple trigonometric formulae are used to represent the location of a source in spherical geometry. For a source at angle A and elevation B (counter-clockwise), coordinates are calculated thus (for first order encoding):

$$x = \cos A \cos B \tag{6.1}$$

$$y = \sin A \cos B \tag{6.2}$$

$$z = \sin B \tag{6.3}$$

B-format signals for the x, y and z directions, as well as the overall pressure component can then be calculated:

$$\begin{aligned}X &= input \times \cos A \cos B \\ Y &= input \times \sin A \cos B \\ Z &= input \times \sin B \\ W &= input \times .707\end{aligned}$$

The scaling factor on *W* is used to ensure a more even distribution. It can be made more general by considering the x, y and z position of the source [126]. A point at angle 0, elevation 0 will be treated thus:

$$\begin{aligned} X &= input \times \cos 0 \cos 0 = input \\ Y &= input \times \sin 0 \cos 0 = 0 \\ Z &= input \times \sin 0 = 0 \\ W &= input \times .707 \end{aligned}$$

Furthermore, a point at angle 45 degrees (this is actually 315 degrees in the appropriate counter clockwise scenario), 0 degree elevation implies:

$$\begin{aligned} X &= input \times .707 \\ Y &= input \times .707 \\ Z &= 0 \\ W &= input \times .707 \end{aligned}$$

The spatial distribution of the source is thus stored in the B-format file. Alternatively, a soundfield microphone is used to capture this information.

From a decoding point of view, each loudspeaker is sent a combination of the B-format components. Loudspeakers essentially spatially sample spherical harmonics. Psychoacoustic phenomena are considered in decoding, with phase information used in the low-frequency range and amplitude at higher frequencies [151]. Therefore, filtering is required. The signal sent to an arbitrary loudspeaker can be calculated using its relative location. For example, to decode to a square:

Loudspeaker at angle 45 degrees (front left: counter clockwise), elevation 0:

$$Signal = W + .707(X + Y)$$

Loudspeaker at angle 315/-45 degrees (front right), elevation 0:

$$Signal = W + .707(X - Y)$$

Loudspeaker at 135 degrees (back left), elevation 0:

$$Signal = W + .707(-X + Y)$$

Finally, loudspeaker at 225 degrees (back right), elevation 0:

$Signal = W + .707(-X - Y)$

By increasing the number of B-format channels, higher-order Ambisonics can be realised (second order uses 9 sources, third order 16). This improves spatialisation accuracy and allows for a larger sweet spot. Due to its nature as a sound field recreation tool, typically all loudspeakers work together in the reproduction of Ambisonics. This increases any potential problems caused by the Precedence Effect. Also, as signal phases may imply cancellations, introducing a listener to the sound field can cause challenges [79]. Near-field compensation of loudspeaker bass response is also required (all sources are assumed infinite, so finite distance loudspeakers require compensation [151]).

6.7 Wave Field Synthesis (WFS)

WFS can be thought of at its most elemental as sound field reconstruction using a ‘wall of loudspeakers’. More accurately, it uses Huygen’s Principle to recreate a propagating wave using secondary sources placed along the wavefront [41, 201]. WFS enhances spatial sound; a listener can walk towards a source, which gets naturally louder as they do. There are no ‘sweet spot’ problems [206]. This is particularly promising when the results of tests performed in [98 and 13] are considered; neither Ambisonics nor VBAP can be relied upon for consistent spatial localisation in the context of a distributed audience in a reverberant environment.

In [41], the potential of WFS is explicitly defined in the context of other multi-channel setups: ‘WFS, on the other hand, aims at reproducing the true *physical attributes* of a given sound field *over an extended area* of the listening room.’ Arrays of loudspeakers are used in reproduction (for example, 192 loudspeakers are used in the horizontal plane cinema example in [200]).

Three types of source can be represented:

1 Virtual point sources outside the array. Crucially, this source will appear to be in the same location for all listener positions.

2 Plane waves: infinite point sources (not a ‘real world’ scenario; in [41] the analogy is drawn to the sun appearing to follow passengers when travelling in a car, its angular direction not changing).

3 Virtual sources in front of speakers. Here, the wavefront converges onto a fixed position. Although an exciting prospect, this technique is inaccurate between loudspeaker array and target position.

In [201], the relationship and similar limitations of near-field corrected higher order Ambisonics and WFS are noted.

From a recording point of view, several options exist, including close micing and Virtual Panning Spots, using stereophonic techniques to represent extended spatial sources [41]. Live recording of the sound field with microphones in loudspeaker reproduction positions can provide a high fidelity reproduction (the reproduction room should be anechoic). This is discussed from an acoustic consultancy point of view in [52]. MPEG4 coding of sources using an object-oriented paradigm is discussed in [41]; each source is stored separately.

Of particular interest are the challenges involved in implementing environmental effects. In [41], the effect of the room is added using eight virtual speakers setup around the array (i.e. sources created on the array). Early reflections can be processed using impulse responses, geometric models or perceptual models (as discussed in the previous chapter). In [12], implementation of room effects in the software tool WONDER is discussed. A dry source requires attenuation and delay of a signal to be spatialised for each loudspeaker. However, addition of room effects can quickly intensify processing requirements. Essentially, *each* point in the virtual soundfield has a unique impulse response for *each* speaker, potentially leading to massive amounts of data. As with HRTFs, discrete sampling is performed. Only closest impulses are buffered in the reproduction system discussed for efficiency. Discontinuities caused by impulse shifts are minimised using crossfades. Other possibilities include separating early reflections (using a model or short FIRs) and late reverberation. Problems with the reproduction room are highlighted in [226]. For example, the evolution of a focused source's early reflections will not originate from the source, but from the combination of loudspeakers creating the source.

Despite the apparent potential of the technique, there are difficulties with WFS implementation, as discussed in [206 and 226]. These include spatial aliasing: spatial and spectral errors due to discretisation of the array. Above the alias frequency, the time difference between two successive loudspeaker signals interferes. In [226], OPSI is presented: Optimised Phantom Source Imaging in WFS. It aims to avoid aliasing by using WFS under the alias frequency, and stereophonic sources above.

Limitations of array dimensions can also have an effect (a finite array is used; the theory is based on an infinite array). Diffraction waves originate from the edges of the array: these appear as echoes. This effect can be reduced using a tapering window to decrease the weight on loudspeakers towards the edge of the array. This reduces the diffraction effects but also the listening area.

Reflections of the listening room also pose a problem (this issue is addressed using inverse filtering in [67]). Horizontal-only arrays (often employed for practical reasons) also imply limitations, as they result in no height information; also all room impulse reflections will be put into the horizontal plane.

An interesting application of WFS is discussed in [136 and 207]; essentially, a headphone signal is generated using a BRIR. A circular WFS array is then used to send transaurally-processed [68] point sources to ear locations. This somewhat inverts the multi-channel binaural paradigm discussed later in this chapter. The benefits of the approach are the removal of the necessity for dynamic crosstalk cancellation filters (which essentially constitutes a HRTF interpolation process).

From an implementation point of view, development of the WFS application WONDER is documented in [8, 10, 9, 12, and 11] and available from [227].

6.8 The Multi-channel Binaural Paradigm

Using binaural technology to represent multi-channel audio is a relatively intuitive concept. Binaural techniques can be used to spatialise a sound source to the location of a loudspeaker. If an appropriate interpolation algorithm is available, the listener, or even the loudspeaker can change relative location. Furthermore, if accurate binaural reverberation processing is available, a particular listening environment can be virtualised, with source distance, motion (including Doppler Effect [144]) and environmental processing. This is a promising prospect; the environmental processing of an ideal mixing room and optimal user position can be recreated, for example.

In [148], the concept is applied to Ambisonics. Benefits are highlighted: only static HRTF processing is required, the number of HRTF filters required is constant, and independent of the number of sound sources. Also, Ambisonic vector rotations can be used to process head rotations; the Ambisonic scene, as opposed to the binaural scene is rotated in a more efficient process. In a room model based on an image model of early reflections, Ambisonic order is reduced for less important reflections as part of an optimisation process. As discussed in chapter 3, implementation of the relevant algorithms can be found in [148]. This approach is also discussed in [14], in the context of the BAP 1000 system, which uses a virtual stereo control room and 9 HRTF sets appropriate for an advertised ‘90-95%’ of the population.

More recently, attempts have been made to optimise the paradigm. In [95], Jot suggests that the high level of control available in the binaural domain should be used to essentially improve multi-channel formats. Accordingly, binaural B-format is discussed (the technique is introduced in [94]). Essentially, minimum-phase HRTFs are broken into B-format signals (an analysis of the complete HRTF dataset). Each source in the reproduction is processed with a delay and encoded to B-format. All sources can then be mixed and filtered with B-format HRTFs. Left and right outputs can then be summed. Therefore, a source at a particular location is encoded using Ambisonic formulae. The idea is furthered in [94], using discrete-panning functions.

An analysis-synthesis approach is presented in [73]. A spatial scene is analysed and re-spatialised binaurally. A vector for each frequency bin of a multi-channel source is used to derive locations for that bin. This analysis informs a HRTF based STFT resynthesis. Similarly, Directional Audio Coding (DirAC) [164] takes an analysis-synthesis approach. Input is analysed (again, in the frequency domain) for directional and diffuse properties (the results of which can be stored as metadata). The approach was developed for binaural representation in [116].

The approach used here is the more traditional multi-channel binaural paradigm of virtual loudspeakers reproducing a multi-channel output. The novelty introduced is the flexibility and complete user control. Sweet spot reproduction is not assumed, as in [155]. Also, any multi-channel reproduction algorithm is possible; the user is not limited in this way. The flexibility essentially transforms the tool into a multi-channel algorithm audition tool. Any algorithm can be tested from a point of view of non-ideal loudspeaker placement, off-centre (Ambisonics is tested in this way in [118])/dynamic listener locations, room size, etc.

In MPEG binaural surround technology [28], the virtual-loudspeaker paradigm is utilised (and updated). In a process that optimises the process by using parametric HRTFs, low bit rates with surround information for headphones is achieved.

Although the primary focus of the system proposed is not auralisation accuracy, commercial auralisation tools are perhaps worth mentioning, in the context of virtual multi-channel processing. Such a discussion is inherently somewhat postulation, due to the closed nature of the source code. CATT is a good example of the state of the art [51]. It uses B-format impulses to allow dynamic user walkthroughs. For binaural reproduction, a virtual-Ambisonic, sweet-spot approach is taken. Odeon also appears to offer similar, if less detailed processing [156]. VRSonic's tools appear to take a direct approach to HRTF auralisation, thus increasing processing cost [219].

6.9 MultiBin

The virtual multi-channel approach is implemented as MultiBin. At the onset, a generic approach to processing was applied. Usage scenarios are therefore completely flexible: a single source can be moved around a listener in a desired room, or a complex multi-channel algorithm can be tested for a slightly off-centre/moving listener. Implementation and usage are discussed below.

Head tracking is accounted for, the significance of which is discussed in [199] in the context of modelling error in binaural systems. Interestingly, head tracking was not found to have a significant effect on localisation for speech sources in [19], although its addition does appear to greatly reduce reversals. An example of implementation of head tracking is discussed in [212]; the authors use a circular set of infrared diodes on the head, which are picked up by a WII remote. Head tracking can be ‘plugged in’ to MultiBin as `hrtfearly` includes it as a parameter (interaction with hardware needs to be considered; implementation using Open Sound Control [158], for example).

6.9.1 Implementation

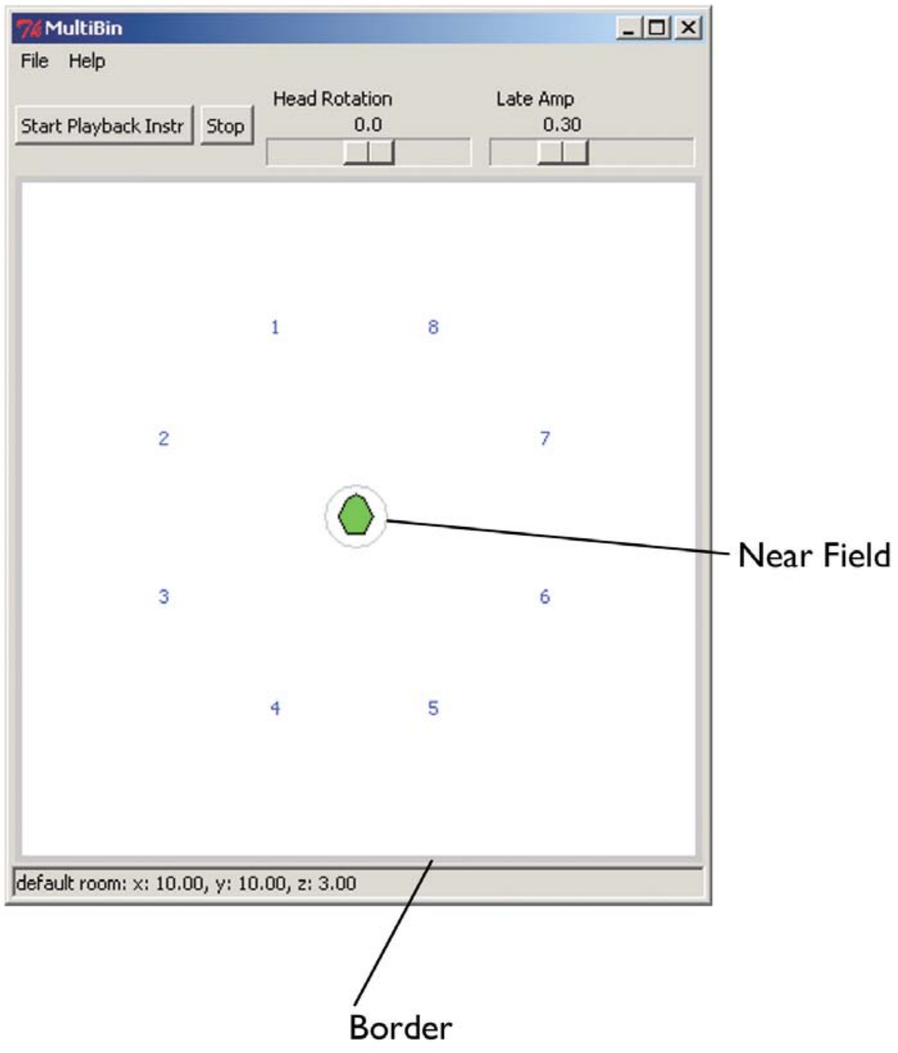
Python was chosen to implement (initially to promptly prototype, but the language proved very suitable for the task) the MultiBin application [169]. The nature of the language allows for immediacy and flexibility in development (it is a dynamically-typed, interpreted language). Crucially, Python code can interact with Csound code through the *csnd* module, which wraps Csound API functions. Also, an abundance of libraries exist for various tasks, including GUI development, which is relevant here.

The language is flexible, allowing various approaches to the task. An object-oriented style is taken. The main class is, however, quite specific in its task, so generic object design is somewhat sacrificed in this more heuristic approach. The specific nature of the application merits this. The *Tkinter* library is used for GUI development (it provides an interface to the Tk GUI toolkit [210]). Several didactic websites have been referenced in the development of the application, most significantly [171, 170 and 208, 209 for *Tkinter*].

Application code will now be discussed. As before, code is heavily commented. Trivial aspects of development are not discussed. A broad overview of developing a *Tkinter* application is offered, highlighting the idiosyncrasies of this particular implementation, in the context of the goal of the application: flexible multi-channel/generic source binaural processing and exposition of the new Csound opcodes.

The main `application` class inherits from *Tkinter's* `Frame` class. The constructor of the main class then calls the constructor of its parent. The `frame` is the basic rectangular window, on which the complex layout can be built. An instance of Csound is setup and associated with the instantiated object. The appropriate csd is compiled and started. The `self` argument is a reference to self/the current instance of the class, and is used throughout to initialise and access member variables/methods. The GUI is then setup. A menu is created, added, and expanded. The various menu choices are bound to commands (member functions), to be discussed shortly. The various options for display are documented in the referenced citations. Similarly, buttons are created for playback. A `scale` widget is used for both head rotation and the level of late reverb. These widgets are all placed on the master/root widget. Canvas and status labels are also added.

The `grid` method of widget organisation is employed; a row/column approach. Default values for member variables are setup; the size of the canvas depends on the y value of the room size (shoebox shapes are assumed). Correct relative room wall ratios are maintained. The useful canvas area is defined as the area in the room 10 centimetres from the walls (a source is assumed to occupy some physical space; this also avoids image sources being at the same location as 'real' sources). This usable rectangle will be highlighted using a grey border (and will inherently change with room size). Defaults are as per the Csound opcodes. Figure 6.1, below illustrates the main processing screen and border (the numbers represent sound sources).



default room: x: 10.00, y: 10.00, z: 3.00

Figure 6.1: MultiBin

Some care is required to make the head control/visual representation as interactive as desired. Each point on the polygon that represents the head is understood as a complex number, to allow for rotation. Near-field HRTF processing is dealt with by not processing sources inside a 45 cm radius, as discussed in chapter 5. As with the processing area, the relative size of this area changes with the room dimensions. This implies that sources entering this area from the left and moving to the right will not appear to be located at the right of the listener until they exit this area. Hence, a clear visual cue is offered (see figure 6.1, above). The location of the head is sent to Csound, as well as an initial value of 0 for the head rotation and .3 for the late reverb level. A score message is sent using the `InputMessage` API function, turning on Csound instrument 100. This instrument essentially parses the aforementioned values, passing them to global variables and adding some portamento to avoid zipper noise; this is a consequence of the pixelated nature/limited resolution of the input canvas, the portamento essentially adds a smoothing low-pass filter.

The final task for the initialisation function/constructor is to bind specific mouse actions to objects/widgets on the canvas. Selecting, dragging, and deselecting objects are all bound to specific member methods. Canvas objects are tagged with the string ‘drag’ (the head object with ‘draghead’, as it is treated slightly differently). The `WM_DELETE_WINDOW` protocol is used to call a function when the user closes the window (which essentially destroys the master widget and stops Csound). The initialisation function is presented below. Note that Python interprets code segments by indentation.

```
def __init__(self, master = None):
      master.title("MultiBin")

      Frame.__init__(self, master)

      #csound setup: turn on, wait for input!
      self.cs = csnd.Csound()
      self.cs.Compile("finaltable.csd")
      self.perf = csnd.CsoundPerformanceThread(self.cs)
      self.perf.Play()

      #create a Menu base
      self.menu = Menu(self)
      #add it
      self.master.config(menu = self.menu)
      #create menu
      self.filemenu = Menu(self.menu)
      #file menu
      self.menu.add_cascade(label = "File", menu = self.filemenu)
      #this choice is required before processing begins!
      self.filemenu.add_command(label = "New Scene(==Restart)",
                                command = self.newscene)
```

```
self.filemenu.add_command(label = "New Source", command =
                          self.newsrc)
self.filemenu.add_separator()
self.filemenu.add_command(label = "'Ideal' Stereo",
                          command = self.stereo)
#ambisonics: layout 4 from bformdec1
self.filemenu.add_command(label = "Ambi - Octogon",
                          command = self.ambi4)
self.filemenu.add_command(label = "VBAP - 8 Channel",
                          command = self.vbap8)
self.filemenu.add_separator()
#clear
self.filemenu.add_command(label = "Clear Recent",
                          command = self.clearrecent)
self.filemenu.add_command(label = "Clear All",
                          command = self.clearall)
self.filemenu.add_separator()
self.filemenu.add_command(label = "Exit",
                          command = self.end)
self.helpmenu = Menu(self.menu)
self.menu.add_cascade(label = "Help", menu = self.helpmenu)
self.helpmenu.add_command(label = "About...",
                          command = self.about)

#button for source on/off, dial for head rotation
self.playbut = Button(master, text = "Start Playback Instr",
                      command = self.play)
self.stopbut = Button(master, text = "Stop",
                      command = self.stop)

#scale widget for head rotation
self.rotscale = Scale(master, orient = HORIZONTAL,
                      label = "Head Rotation",
                      from_ = -90.0, to = 90.0,
                      length = 120, cursor = "exchange",
                      resolution = .1,
                      command = self.headrotate)

#scale for output level of late reverb
self.latescale = Scale(master, orient = HORIZONTAL,
                       label = "Late Amp", from_ = 0,
                       to = .99, length = 120,
                       resolution = .01,
                       command = self.lateamp)

#status: hints for user
self.statusstring = "default room: x: 10.00, y: 10.00, z:
                     3.00"
self.status = Label(master, text = self.statusstring,
                    relief = SUNKEN, anchor = W)

#canvas, default size for first run...
self.canvas = Canvas(master, bg = "grey", width = 400,
                     height = 400)

#setup grid...
self.playbut.grid(row = 0, column = 0)
self.stopbut.grid(row = 0, column = 1)
self.rotscale.grid(row = 0, column = 2)
self.latescale.grid(row = 0, column = 3)
self.canvas.grid(row = 1, column = 0, columnspan = 4)
self.status.grid(row = 2, column = 0, columnspan = 4,
                 sticky = E + W)

#set reverb...
self.latescale.set(0.3)

#defaults
#roomarray data order: rmx, rmy, rmz, wlh, wll, wl1, wl2,
 wl3, flh, fll, fl1, fl2, fl3, clh, cll, cl1, cl2, cl3
self.roomarray = [10.0, 10.0, 10.0, 3.0, .3, .1, .75, .95,
                  .9, .6, .1, .95, .6, .35, .2, .1, 1.0,
                  1.0, 1.0]

self.sizex = 400.0
self.sizey = self.sizex / self.roomarray[0] *
             self.roomarray[1]
#useful canvas area...needs to be min .1m from wall
#canvas goes from 1 - 401, add extra 1 to bottom right
 corner as rect is contained within this point
self.rect = self.canvas.create_rectangle((.1 /
      self.roomarray[0]) * self.sizex + 1,
      (.1 / self.roomarray[1]) * self.sizey + 1,
      self.sizex + 2 - (.1 / self.roomarray[0]) * self.sizex,
      self.sizey + 2 - (.1 / self.roomarray[1]) * self.sizey,
      fill = "white", outline = "grey")
#initialise
self.headx = self.sizex / 2
self.heady = self.sizey / 2
#initialise to zero...for rotation
self.rot = []
for i in range (7):
      self.rot.append(complex(0, 0))
#head best as last object, most recent will be selected
  first!
self.range = self.sizex / self.roomarray[0] * .45
self.oval = self.canvas.create_oval(self.headx - self.range,
            self.heady - self.range, self.headx +
            self.range, self.heady + self.range,
            outline = "grey")
#points for polygon of head...
self.head = self.canvas.create_polygon(self.headx + 5,
            self.heady + 10, self.headx + 10, self.heady,
            self.headx + 5, self.heady - 10, self.headx,
            self.heady - 13, self.headx - 5, self.heady - 10,
            self.headx - 10, self.heady, self.headx - 5,
            self.heady + 10, fill = "green",
            tags = "draghead", outline = "black")
self.cs.SetChannel("xhead", self.roomarray[0] / 2)
self.cs.SetChannel("yhead", self.roomarray[1] / 2)
self.cs.SetChannel("rot", 0)
self.cs.SetChannel("lateamp", 0.3)
self.perf.InputMessage("i100 0 -1 %f %f" %(self.roomarray[0]
                       / 2, self.roomarray[1] / 2))
#link 'drag' to functions
self.canvas.tag_bind('drag','<B1-Motion>', self.move)
self.canvas.tag_bind('drag','<ButtonPress>', self.select)
self.canvas.tag_bind('drag','<ButtonRelease>',
                     self.deselect)
self.canvas.tag_bind('draghead','<B1-Motion>',
                     self.movehead)
#closing window also ends csd...
```

```
      self.master.protocol("WM_DELETE_WINDOW", self.end)
```

Widget/object movement on the canvas is perhaps the most significant process in the application, so will be discussed next. The `select` method essentially gets the location of the event triggering the function call, and changes the colour of the active object. The object under the mouse pointer (`current`) is changed to a red colour to illustrate selection.

```
def select(self, event):
      loc = event.widget
      item = loc.find_withtag("current")
      #red if selected!
      loc.itemconfig(item, fill = "red")
```

Similarly, deselecting an object restores its blue colour.

```
def deselect(self, event):
      loc = event.widget
      item = loc.find_withtag("current")
      loc.itemconfig(item, fill = "blue")
```

Source movement is a little more complex. Once again, an event triggers the function. The location of this event is stored, with the window location changed to a canvas location (the canvas can be scrolled). The coordinates of the object are then replaced. It is crucial to maintain a structured system for labelling sources as they are added/removed from the canvas. Items on the canvas are numbered sequentially as they are created. Therefore, the active area rectangular is item 1, the near-field circle item 2 and the head item 3 (see figure 6.1 above). Therefore, the relative source number with regard to sources on screen is the *item number – 3*. Removing items from the canvas does not reset the internal *Tkinter* counter. Therefore, removal of sources needs to be considered. Users can remove all items to clear the current canvas, or remove the most recently created objects. The unique ID of the object is thus calculated. A count variable also stores the number of objects active on the canvas. In this way, the internal object counter can be used with the active object counter to ensure the correct unique ID is stored. The `SetChannel` function can then be used to send a message to Csound, which has been setup to listen for information on particular named channels using the `chnget` opcode. The x and y coordinate values are sent on channels labelled with the particular source number. The final point worth noting is that y values are inverted for agreement between *Tkinter* and the Csound opcodes. An appropriate accompanying csd file will be discussed shortly.

```
def move(self, event):
      #widget that called the event
      loc = event.widget
      #set, in case of canvas scroll...
      x = loc.canvasx(event.x)
      y = loc.canvasy(event.y)
      #find_withtag returns list (tuple) of matching items,
       in order created:
      #only 1 item will be returned here (or if they are at same
       loc, most recently created)...
      item = loc.find_withtag("current")
      #text only has 2 coords...
      loc.coords(item, x, y)
      #choose value based on default room sizes/inputted size
      #head is item 2, oval 3, rectangle 1, then sources in order
       created...
      #if > no of items active, subtract no of removed...also
       subtract total removed...
      if item[0] - 3 - self.allremoved > self.count:
            chno = item[0] - 3 - self.allremoved - self.removed
      else:
            chno = item[0] - 3 - self.allremoved

      self.cs.SetChannel("xsrc%d" %chno, (x / self.sizex) *
                         self.rmx)
      #send inverted y to csound...
      y = self.sizey - y
      self.cs.SetChannel("ysrc%d" %chno, (y / self.sizey) *
                         self.rmy)
```

The head will always be on the same ‘channel’, so a simpler move function suffices. The coordinates of the head are updated, considering any rotation. The near-field limitation cue also follows the head movement. Again, Csound receives information on head movements from the software bus.

```
#a simpler move function, as head will always be on same channel
def movehead(self, event):
      #widget that called the event
      loc = event.widget
      #set, in case of canvas scroll?...
      self.headx = loc.canvasx(event.x)
      #invert y throughout
```

```
        self.heady = loc.canvasy(event.y)
        #y = self.sizey - y
        item = loc.find_withtag("current")[0]
        #head coords...include existing measured rotation...
        loc.coords(item, self.headx + self.rot[0].real, self.heady +
                   self.rot[0].imag, self.headx + self.rot[1].real,
                   self.heady + self.rot[1].imag,
                   self.headx + self.rot[2].real,
                   self.heady + self.rot[2].imag,
                   self.headx + self.rot[3].real,
                   self.heady + self.rot[3].imag,
                   self.headx + self.rot[4].real,
                   self.heady + self.rot[4].imag,
                   self.headx + self.rot[5].real,
                   self.heady + self.rot[5].imag,
                   self.headx + self.rot[6].real,
                   self.heady + self.rot[6].imag)
        #move range
        self.canvas.coords(self.oval, self.headx - self.range,
                           self.heady - self.range,
                           self.headx + self.range,
                           self.heady + self.range)
        self.cs.SetChannel("xhead", (self.headx / self.sizex) *
                           self.roomarray[0])
        #send inverted y to csound...
        y = self.sizey - self.heady
        self.cs.SetChannel("yhead", (y / self.sizey) *
                           self.roomarray[1])
```

Csound playback is dealt with by turning on a source playback instrument indefinitely. A member Boolean keeps track of playback status.

```
def play(self):
        self.perf.InputMessage("i1 0 -1")
        self.playing = 1

def stop(self):
        if self.playing:
              self.perf.InputMessage("i-1 0 -1")
              self.playing = 0
```

Head rotation is dealt with by reading the angle variable from the slider and sending the value to Csound. Complex numbers are used to manipulate the polygon. Essentially, each vertex of the polygon is represented by a point on the complex plane. The location of the head on the canvas (its central point) is used as an offset. A complex representation of the angle is used to rotate each point.

```
def headrotate(self, event):
        degrees = self.rotscale.get()
        self.cs.SetChannel("rot", degrees)
        #rotate polygon...use complex maths here, as it more elegant
        offset = complex(self.headx, self.heady)
        radangle = math.radians(degrees)
        compangle = cmath.exp(radangle * 1j)
        #angle, from centre of non rotated polygon, rotate by point at
         0, 0, add offset again at end...
        self.rot[0] = compangle * (complex(self.headx + 5,
                      self.heady + 10) - offset)
        self.rot[1] = compangle * (complex(self.headx + 10,
                      self.heady) - offset)
        self.rot[2] = compangle * (complex(self.headx + 5,
                      self.heady - 10) - offset)
        self.rot[3] = compangle * (complex(self.headx,
                      self.heady - 13) - offset)
        self.rot[4] = compangle * (complex(self.headx - 5,
                      self.heady - 10) - offset)
        self.rot[5] = compangle * (complex(self.headx - 10,
                      self.heady) - offset)
        self.rot[6] = compangle * (complex(self.headx - 5,
                      self.heady + 10) - offset)
        #add offset again...
          self.canvas.coords(self.head, self.rot[0].real + self.headx,
                             self.rot[0].imag + self.heady,
                             self.rot[1].real + self.headx,
                             self.rot[1].imag + self.heady,
                             self.rot[2].real + self.headx,
                             self.rot[2].imag + self.heady,
                             self.rot[3].real + self.headx,
                             self.rot[3].imag + self.heady,
                             self.rot[4].real + self.headx,
                             self.rot[4].imag + self.heady,
                             self.rot[5].real + self.headx,
                             self.rot[5].imag + self.heady,
                             self.rot[6].real + self.headx,
                             self.rot[6].imag + self.heady)
```

Amplitude control of the late reverberation is straightforward:

```
def lateamp(self, event):
        vol = self.latescale.get()
        self.cs.SetChannel("lateamp", vol)
```

When the window is closed, the Csound performace is stopped (the main thread waits for the processing thread to finish before proceeding) and the main root widget is destroyed.

```
def end(self):
        self.perf.Stop()
        self.perf.Join()
        self.master.destroy()
```

Adding a new source is dealt with using a number of member functions. The `newspeaker` function is a generic function used by other, more specific solutions. It takes an x, y coordinate as its arguments. Firstly, the number of active sources is incremented; then the visualisation of the object is placed on the canvas. A simple text number was considered the most intuitive representation. Therefore, the audio associated with each source is clear. This representation also performs well from an interactivity point of view.

A string is prepared as the Csound score command used to turn on the processing instrument: an instance of `hrtfearly`. All parameters are passed to the opcode. The source location is set on the appropriate listening channel and the score message is sent. The global reverb instrument (using `hrtfreverb`) is turned on if not already on.

```
#generic: new speaker
def newspeaker(self, x, y):
      self.count = self.count + 1
      no = str(self.count)

      self.canvas.create_text(x, y, text = no, fill = 'blue',
                              tags = 'drag')

      #invert y for csound
      y = self.sizey - y

      S = 'i101.{0} 0 -1 {0} {1} {2} {3} {4} {5} {6} {7} {8} {9}
           {10} {11} {12} {13} {14} {15} {16} {17} {18} {19} {20}'
           .format(self.count, (x / self.sizex) * self.roomarray[0],
            (y / self.sizey) * self.roomarray[1], self.roomarray[0],
            self.roomarray[1], self.roomarray[2], self.roomarray[3],
            self.roomarray[4], self.roomarray[5], self.roomarray[6],
            self.roomarray[7], self.roomarray[8], self.roomarray[9],
            self.roomarray[10], self.roomarray[11],
            self.roomarray[12], self.roomarray[13],
            self.roomarray[14], self.roomarray[15],
            self.roomarray[16], self.roomarray[17])

      #set channel
      self.cs.SetChannel("xsrc%d" %self.count, (x / self.sizex) *
                         self.roomarray[0])
      self.cs.SetChannel("ysrc%d" %self.count, (y / self.sizey) *
                         self.roomarray[1])
      self.perf.InputMessage(S)

      #turn on global late reverb, if not already on (may have added
      and removed sources!)
      if self.count == 1:
            if self.reverb == 0:
                  self.perf.InputMessage("i102 0 -1")
                  self.reverb = 1
                  print "reverb"
```

This generic member function is used in all cases of source addition. An instance of the `location` class is used to add a single source. This class inherits from the tkSimpleDialog.Dialog class. The location class essentially overrides the appropriate methods of the parent class for use here. The body method describes the GUI, adding a choice for direct or angular source location input. Again, the grid method is used to setup the GUI (note an array is used to elegantly store and process elements). The angle/distance entry option is focused upon by default. Each of the radio buttons calls a function. The rect function en/disables the appropriate inputs for rectangular input. The polar function acts similarly for angle/distance input. The radio buttons are linked by association with the same variable. The apply method override essentially defines the functionality of the ‘ok’ button: check a data flag, read the Entry boxes and return a list of appropriate location information, including the mode of acquisition.

```
class location(tkSimpleDialog.Dialog):

      flag = 0

      def body(self, master):

            self.mode = IntVar()
            self.r1 = Radiobutton(master, text = "Angle, Distance
                                  Input", value = 1, variable =
                                  self.mode, command = self.rect)
            #need to do this separately to return correct type
             self.r1...
            self.r1.grid(row = 0, column = 0, columnspan = 2)
            Label(master, text="OR").grid(row = 0, column = 2)
            Radiobutton(master, text = "X, Y Coordinates",
                        value = 2, variable = self.mode,
                        command = self.polar).grid(row = 0,
                        column = 3, columnspan = 2)
            Label(master, text="Note: All values must fit on
                  canvas/in room specified, and will be truncated
                  accordingly!").grid(row = 1, columnspan = 4)
            Label(master, text="Distance from Centre:").grid(row =
                                                  2, column = 0)
            Label(master, text="Angle:").grid(row = 3, column = 0)
            Label(master, text="X:").grid(row = 2, column = 3)
            Label(master, text="Y:").grid(row = 3, column = 3)
            self.r1.select()

            self.e = []
            for i in range (4):
                  self.e.append(Entry(master, width = 10))
            self.e[0].insert(0, "1")
            self.e[1].insert(0, "0")
            self.e[2].insert(0, "100")
            self.e[3].insert(0, "100")
            for i in range (2):
                  self.e[i].grid(row = i + 2, column = 1)
            for i in range (2):
                  self.e[i + 2].grid(row = i + 2, column = 4)

            for i in range (2):
                  self.e[i + 2].configure(state = DISABLED)

            #initial focus
            return self.e[0]

      def apply(self):
            self.flag = 1
            dist = float(self.e[0].get())
            angle = float(self.e[1].get())
            x = float(self.e[2].get())
            y = float(self.e[3].get())
            temp = self.mode.get()
            #fill in array...
            if temp == 1:
                  self.locationdata = 1, dist, angle
            elif temp == 2:
                  self.locationdata = 2, x, y

      def rect(self):
            for i in range (2):
                  self.e[i].configure(state = NORMAL)
            for i in range (2):
                  self.e[i+ 2].configure(state = DISABLED)

      def polar(self):
            for i in range (2):
                  self.e[i].configure(state = DISABLED)
            for i in range (2):
                  self.e[i+ 2].configure(state = NORMAL)
```

The `newsrc` method uses this instance of the `location` class to add a source (if details have been submitted; avoiding problems with the ‘cancel’ button). The mode is extracted from the returned data, a polar source is dealt with by another method, a direct source is stored and validated (error checking is performed here, where other class parameters are known). Finally, the `newspeaker` method is used as above.

```
def newsrc(self):
      #default add source: in front of listener
      self.src = location(self)
      if self.src.flag:
            mode = self.src.locationdata[0]
            if mode == 1:
                  self.newpolar(self.src.locationdata[2],
                                self.src.locationdata[1])
            #simpler scenario...
            elif mode == 2:
                  self.srcx = self.src.locationdata[1]
                  self.srcy = self.src.locationdata[2]
            #invert y for pixel location
            self.srcy = self.sizey - self.srcy
            #validate location, leave other validation up to
             csound...wall params...room limits...
            #src must be minimum 10 cm from wall, as per csound
            if self.srcx > self.sizex - (.1 / self.roomarray[0]) *
               self.sizex:
                  self.srcx = self.sizex - (.1 / self.roomarray[0])
                              * self.sizex
            if self.srcx < (.1 / self.roomarray[0]) * self.sizex:
                  self.srcx = (.1 / self.roomarray[0]) * self.sizex
            if self.srcy > self.sizey - (.1 / self.roomarray[1]) *
               self.sizey:
                  self.srcy = self.sizey - (.1 / self.roomarray[1])
                              * self.sizey
            if self.srcy < (.1 / self.roomarray[1]) * self.sizey:
                  self.srcy = (.1 / self.roomarray[1]) * self.sizey

            self.newspeaker(self.srcx, self.srcy)
```

A polar source input is dealt with by the `newpolar` method. The location of the point is calculated on the unit circle, the distance and room geometry then decide where it should be located on the canvas.

```
def newpolar(self, angle, distance):
      radangle = math.radians(angle)
      #angle measured from centre, clockwise...
      #sin gives x coord...cos y...
      #srcx and srcy are temp variables used for each source
      self.srcx = math.sin(radangle)
      self.srcy = math.cos(radangle)
      #radius: same if calculated from x or y params, as per setup
      mult = self.sizex / self.rmx * distance
      self.srcx *= mult
      self.srcy *= mult
      #relative to listener/centre
      self.srcx += self.sizex / 2
      self.srcy += self.sizey / 2
```

The next three methods implement loudspeaker setups for multi-channel setups; content for which can be easily generated in Csound (examples will be given below). An Ambisonic scenario is created by setting up eight virtual loudspeakers (an octagon layout) with an appropriate layout to the geometry of the room [42]. Similarly, an eight-channel VBAP scenario can be setup [50]. Finally, an 'ideal' sweet spot stereo setup is offered (a user may then move out of the sweet spot, an insightful exercise), which could be used, for example, to emulate a control room.

```
#each default setup will call a unique function...
#need different menu for different possible ambi, vbap setups in
csound...
#mode 4 Ambisonics
def ambi4(self):
      self.clearall()
      #local variables here
      #angles: anticlockwise
      ang = -22.5
      if self.roomarray[0] < self.roomarray[1]:
            dist = self.roomarray[0] / 3
      else:
            dist = self.roomarray[1] / 3
      for i in range(1, 9):
            self.newpolar(ang, dist)
            #invert y for pixel location
            self.srcy = self.sizey - self.srcy
            self.newspeaker(self.srcx, self.srcy)
            ang -= 45

def vbap8(self):
      self.clearall()
      #from manual example...
      ang = 15
      if self.roomarray[0] < self.roomarray[1]:
            dist = self.roomarray[0] / 3
      else:
            dist = self.roomarray[1] / 3
      for i in range(1, 9):
            self.newpolar(ang, dist)
            #invert y for pixel location
            self.srcy = self.sizey - self.srcy
            self.newspeaker(self.srcx, self.srcy)
            if i == 4:
                ang += 30
            else:
                ang += 50

#simple, externalised stereo
def stereo(self):
      self.clearall()
      if self.roomarray[0] < self.roomarray[1]:
            dist = self.roomarray[0] / 3
      else:
          dist = self.roomarray[1] / 3
      self.newpolar(30, dist)
      #invert y for pixel location
      self.srcy = self.sizey - self.srcy
      self.newspeaker(self.srcx, self.srcy)
      self.newpolar(-30, dist)
      self.srcy = self.sizey - self.srcy
      self.newspeaker(self.srcx, self.srcy)
```

Methods to clear sources from the canvas are defined next. The most recent canvas item is cleared by `clear` (a message is printed for the user). Again, this is a generic method, used by `clearrecent` to clear the most recent source, and `clearall` to clear the canvas completely (this method also stops playback). The variables keeping track of the number of removed sources (to be used with the source interactivity, as above) are updated appropriately.

```
def clear(self):
      #last element of array = [-1]
      recent = self.canvas.find_all()[-1]
      self.canvas.delete(recent)
      self.perf.InputMessage("i-101.%d 0 -1" %self.count)
      print "i-101.%d 0 -1 SENT" %self.count

def clearrecent(self):
      print self.count
      if self.count > 0:
            self.clear()
            self.count -= 1
            self.removed += 1

def clearall(self):
      self.stop()
      while self.count > 0:
            self.clear()
            self.count -= 1
            self.allremoved += 1
      self.allremoved += self.removed
      self.removed = 0
```

A new scene can also be created. The method implementing this functionality: `newscene` again uses a class based on the `Dialog` class from `tkSimpleDialog`. The 'cancel' scenario is treated in the same manner as the `location` class. The class, `room`, uses familiar tools to generate the GUI. A radio button is used to optionally include complex surface parameters (default input simply involves room geometry). The associated function simply dis/allows access to these complex parameters. The 'ok' `apply` method simply reads all data, storing it in an array.

```
#dialog for room creation
class room(tkSimpleDialog.Dialog):

      flag = 0

      def body(self, master):
            self.complex = IntVar()
            Checkbutton(master, variable = self.complex, text =
                        "Show Complex Params?", command =
                        self.check).grid(row = 0, column = 0)
            Label(master, text = "Room X:").grid(row = 2,
                                                  column = 0)
            Label(master, text = "Room Y:").grid(row = 3,
                                                 column = 0)
            Label(master, text = "Room Z:").grid(row = 4,
                                                 column = 0)

            #empty list
            self.r = []
            for i in range (3):
                  self.r.append(Entry(master, width = 10))
            self.r[0].insert(0, "10")
            self.r[1].insert(0, "10")
            self.r[2].insert(0, "3")
            for i in range (3):
                  self.r[i].grid(row = i + 2, column = 1)
            Label(master, text = "High Ab Coef:").grid(row = 1,
                                                       column = 2)
            Label(master, text = "Low Ab Coef:").grid(row = 2,
                                                      column = 2)
            Label(master, text = "Band 1(cf: 250Hz):").grid(row = 3,
                                                      column = 2)
            Label(master, text = "Band 2(cf: 1000Hz):").grid(row =
                                                      4, column = 2)
            Label(master, text = "Band 3(cf: 4000Hz):").grid(row =
                                                      5, column = 2)
            Label(master, text = "Walls").grid(row = 0, column = 3)

            self.w = []
            for i in range (5):
                  self.w.append(Entry(master, width = 10))
            self.w[0].insert(0, ".3")
            self.w[1].insert(0, ".1")
            self.w[2].insert(0, ".75")
            self.w[3].insert(0, ".95")
            self.w[4].insert(0, ".9")
            for i in range (5):
                  self.w[i].grid(row = i + 1, column = 3)

            Label(master, text = "Floor").grid(row = 0, column = 4)
            self.f = []
            for i in range (5):
                  self.f.append(Entry(master, width = 10))
            self.f[0].insert(0, ".6")
            self.f[1].insert(0, ".1")
            self.f[2].insert(0, ".95")
            self.f[3].insert(0, ".6")
            self.f[4].insert(0, ".35")
             for i in range (5):
                self.f[i].grid(row = i + 1, column = 4)

            Label(master, text = "Ceiling").grid(row = 0,
                                                 column = 5)
            self.c = []
            for i in range (5):
                  self.c.append(Entry(master, width = 10))
            self.c[0].insert(0, ".2")
            self.c[1].insert(0, ".1")
            self.c[2].insert(0, "1.0")
            self.c[3].insert(0, "1.0")
            self.c[4].insert(0, "1.0")
            for i in range (5):
                  self.c[i].grid(row = i + 1, column = 5)
```

```
		#disable extra parameters by default
		for i in range (5):
			self.w[i].configure(state = DISABLED)
			self.f[i].configure(state = DISABLED)
			self.c[i].configure(state = DISABLED)

		#initial focus
		return self.r[0]

	def apply(self):
		#avoid error on cancel with flag...
		self.flag = 1
		self.rmdata = []
		for i in range (3):
			self.rmdata.append(float(self.r[i].get()))
		for i in range (5):
			self.rmdata.append(float(self.w[i].get()))
		for i in range (5):
			self.rmdata.append(float(self.f[i].get()))
		for i in range (5):
			self.rmdata.append(float(self.c[i].get()))

	def check(self):
		if self.complex.get() == 0:
			for i in range (5):
				self.w[i].configure(state = DISABLED)
			self.f[i].configure(state = DISABLED)
			self.c[i].configure(state = DISABLED)
		else:
			for i in range (5):
				self.w[i].configure(state = NORMAL)
				self.f[i].configure(state = NORMAL)
				self.c[i].configure(state = NORMAL)
```

The `newscene` method stops playback, and declares an instance of the `room` class. Once the ‘ok’ button has been selected, all sources are cleared, late reverb is turned off, the global channel instrument is turned off and all room data is stored. The canvas is then reconfigured and all channels and variables are reset accordingly.

```
#get parameters...
def newscene(self):
	#stop playback
	self.stop()
	self.rm = room(self)
	#if not cancel...
	if self.rm.flag:
		self.clearall()
		#turn off reverb
		if self.reverb == 1:
			self.perf.InputMessage("i-102 0 -1")
			self.reverb = 0
			print "reverb off"
		#globals off
		self.perf.InputMessage("i-100 0 -1")

		for i in range(18):
```

```
            self.roomarray[i] = self.rm.rmdata[i]

        self.sizey = self.sizex / self.roomarray[0] *
                     self.roomarray[1]
        self.canvas.configure(width = self.sizex, height =
                              self.sizey)
        #useful canvas area...needs to be min .1m from wall
        self.canvas.coords(self.rect, (.1 / self.roomarray[0]) *
                           self.sizex + 1, (.1 /
                           self.roomarray[1]) * self.sizey + 1,
                           self.sizex + 2 - (.1 /
                           self.roomarray[0]) * self.sizex,
                           self.sizey + 2 - (.1 /
                           self.roomarray[1]) * self.sizey)

        self.canvas.itemconfigure(self.rect, fill = "white",
                                  outline = "grey")
        #initialise
        self.headx = self.sizex / 2
        self.heady = self.sizey / 2
        #reset rotation
        self.rotscale.set(0)
        #reset amp
        self.latescale.set(0.3)
        #move 'head' to middle
        self.canvas.coords(self.head, self.headx + 5,
                           self.heady + 10, self.headx + 10,
                           self.heady, self.headx + 5,
                           self.heady - 10, self.headx,
                           self.heady - 13, self.headx - 5,
                           self.heady - 10, self.headx - 10,
                           self.heady, self.headx - 5,
                           self.heady + 10)
        #near field range
        self.range = self.sizex / self.roomarray[0] * .45
        self.canvas.coords(self.oval, self.headx - self.range,
                           self.heady - self.range,
                           self.headx + self.range,
                           self.heady + self.range)
        #head back to middle!
        self.cs.SetChannel("xhead", self.roomarray[0] / 2)
        self.cs.SetChannel("yhead", self.roomarray[1] / 2)
        #rotation back to 0
        self.cs.SetChannel("rot", 0)
        self.statusstring = ("x: %.2f, y: %.2f z: %.2f"
                             %(self.roomarray[0],
                               self.roomarray[1],
                               self.roomarray[2]))
        self.status.config(text = self.statusstring)
        self.status.update_idletasks()
        # globals back on
        self.perf.InputMessage("i100 0 -1 %f %f"
                               %(self.roomarray[0] / 2,
                               self.roomarray[1] / 2))
```

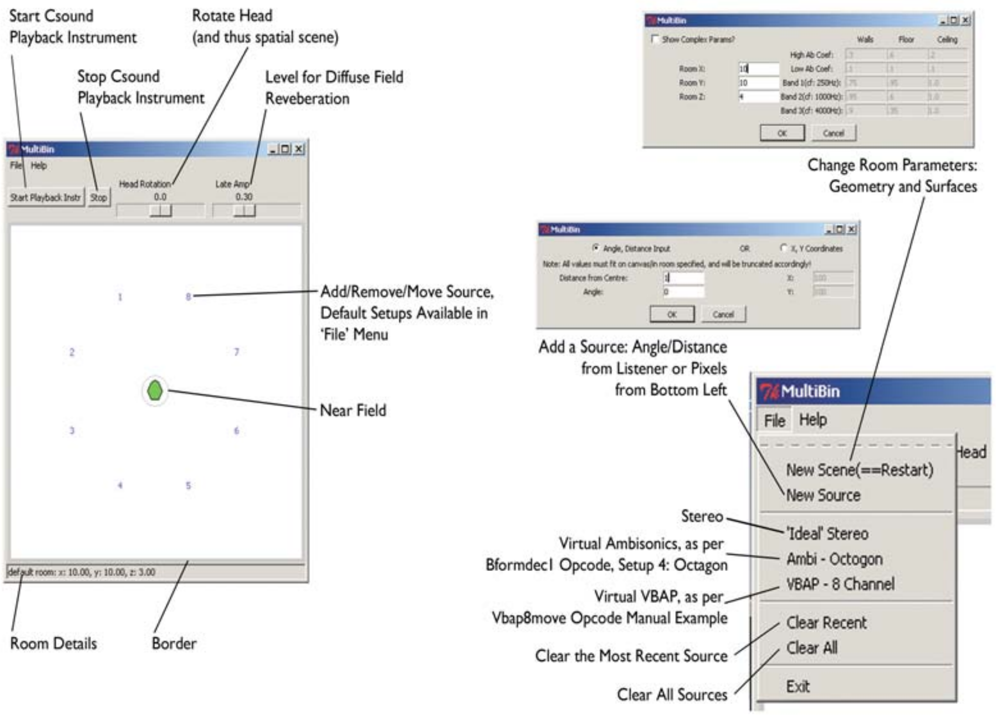
Initialising an instance of the main class involves creating a `Tk` root widget. This then becomes the `master` widget. The main loop is then processed.

```
app = Application(Tk())
```

```
app.mainloop()
```

6.9.2 MultiBin Instructions

A brief user guide for MultiBin is now offered, including some examples of suitable csd files. As discussed, the application can be used to create a virtual shoebox room, with multi-band surface filters. Any number of sources can be placed in this room. Sources are represented by numbers on the canvas, which correspond to the channel on which they are listening for audio from Csound. Source and listener position are dynamically controllable by selecting and dragging. A particular application is the audition of multi-channel material. Ambisonics, VBAP and Stereo are offered as defaults; others can be setup using the new source options. Usage is summarised in figure 6.2.



Start Csound Playback Instrument
Rotate Head (and thus spatial scene)
Stop Csound Playback Instrument
Level for Diffuse Field Reveberation
MultiBin
File Help
Start Playback Instr
Stop
Head Rotation
0.0
Late Amp
0.30
1
8
2
7
3
6
4
5
Add/Remove/Move Source, Default Setups Available in 'File' Menu
Near Field
Room Details
Border
Change Room Parameters: Geometry and Surfaces
Add a Source: Angle/Distance from Listener or Pixels from Bottom Left
File Help
New Scene(==Restart)
New Source
'Ideal' Stereo
Ambi - Octogon
VBAP - 8 Channel
Clear Recent
Clear All
Exit
Head
Stereo
Virtual Ambisonics, as per Bformdec1 Opcode, Setup 4: Octagon
Virtual VBAP, as per Vbap8move Opcode Manual Example
Clear the Most Recent Source
Clear All Sources
OK
Cancel

default room: x: 10.00, y: 10.00, z: 3.00

Show Complex Params?

			Walls	Floor	Ceiling
Room X:	10	High Ab Coef:	[illegible]	[illegible]	[illegible]
Room Y:	10	Low Ab Coef:	[illegible]	[illegible]	[illegible]
Room Z:	4	Band 1(cf: 250Hz):	[illegible]	[illegible]	[illegible]
		Band 2(cf: 1000Hz):	[illegible]	[illegible]	[illegible]
		Band 3(cf: 4000Hz):	[illegible]	[illegible]	[illegible]

[✓ Angle, Distance Input] OR X, Y Coordinates

Note: All values must fit on canvas/in room specified, and will be truncated accordingly!

Distance from Centre: 1 X: [illegible]

Angle: 0 Y: [illegible]

Figure 6.2: MultiBin usage

Python really only supplies the control, GUI and interaction with Csound. All audio processing is performed by Csound. `hrtfearly` and `hrtfreverb` are the core processing tools. Users can follow the templates below to generate appropriate content. Specifically, users need to supply details of the playback instrument in an *instrument1.inc* file. No other alteration is needed. The MultiBin application uses *MultiBin.csd* for all other processing. *instrument1.inc* should simply create/playback the audio to be controlled by MultiBin (examples are given below). In *MultiBin.csd*, instrument 100 receives head location, a head rotation value and a diffuse-field reverberation level. Portamento is added to all control values (suitable defaults are offered); due to the discrete pixel-based control, as above. The `port` opcode takes the signal, a timing control (the half-time of the function) and an initial value.

```
instr 100

;global channel instruments, same for each instrument...
khx chnget "xhead"
khy chnget "yhead"
krot chnget "rot"
kamp chnget "lateamp"
gkhxp port khx, .15, p4
gkhyp port khy, .15, p5
gkrotp port krot, .15
;slightly lower port ok for amp...
gkampp port kamp, .05, 0.3

endin
```

Instrument 101 spatially processes the direct source and early reflections for each source. It listens for appropriate x and y source location values, as well as the appropriate audio source (defined in the external instrument 1). Again, appropriate portamento is added. All audio is added to a global variable for processing by the diffuse field instrument. Global variables store the low and high frequency reverb time, as well as the mean free path, again to be used by the diffuse field instrument. The outputs from MultiBin are then used by the `hrtfearly` opcode (at a sampling rate of 44.1 kHz, two-dimensional processing and eight buffer crossfades).

```
instr 101

Schnlx sprintf "xsrc%d", p4
Schnly sprintf "ysrc%d", p4
Schnin sprintf "in%d", p4

ksx chnget Schnlx
ksy chnget Schnly

ksxp port ksx, .15, p5
ksyp port ksy, .15, p6

ain   chnget Schnin
;add all for late o/p
gainput = ain + gainput

aearlyl, aearlyr, gilow, gihigh, gimfp hrtfearly ain, ksxp, ksyp,
      p9/2, gkhxp, gkhyp, p9/2, "datal.raw", "datar.raw", 0, 8,
      44100, giorder, 0, gkrotp, p7, p8, p9, p10, p11, p12, p13,
      p14, p15, p16, p17, p18, p19, p20, p21, p22, p23, p24
    outs    aearlyl, aearlyr

endin
```

Finally, instrument 102 processes the late reverberant diffuse field. It uses the global outputs from the `hrtfearly` instrument for low and high frequency reverb time and mean free path. It also uses the global audio bus, which it zeros at each pass. The appropriate delay is also added to the late reverb here.

```
instr 102

al, ar, idel hrtfreverb gainput, gilow, gihigh, "datal.raw",
      "datar.raw", 44100, gimfp, giorder
alatel delay al * gkampp, idel
alater delay ar * gkampp, idel
      outs alatel, alater

;zero global in
gainput = 0

endin
```

Global variables for low and high frequency reverb time, mean free path, audio bus and processing order (set at 1) are also setup in the orchestra, as well as the required stereo output. In the score, processing is turned on for 10 minutes by default.

Instrument 1 is declared in *MultiBin.csd*:

```
instr 1

#include "instrument1.inc"

endin
```

Therefore, *instrument1.inc* is expected, and defines audio input. This is the only required user input, and is designed with simplicity in mind. For example, four wav files, read using `loscil` can be setup as below:

```
i1 ftgen 1,0,0,1,"sample.wav",0,0,0
i2 ftgen 2,0,0,1,"speech.wav",0,0,0
i3 ftgen 3,0,0,1,"pianoarp3.wav",0,0,0
i4 ftgen 4,0,0,1,"noise.wav",0,0,0

a1    loscil 20000, 1, 1, 1, 1
a2    loscil 20000, 1, 2, 1, 1
a3    loscil 20000, 1, 3, 1, 1
a4    loscil 20000, 1, 4, 1, 1

; fade out...
k1 linsegr 1,1,1,0.1,0

chnset a1*k1, "in1"
chnset a2*k1, "in2"
chnset a3*k1, "in3"
chnset a4*k1, "in4"
```

It is important to highlight that channel label numbers match the numbers on the GUI: GUI source 1 listens on the channel 'in1' (more accurately, its reverb instruments listen for their matching channels), etc. An Ambisonic setup is simpler, as it just reads a pre-prepared file:

```
a1, a2, a3, a4, a5, a6, a7, a8 soundin "ambi.wav"
; fade out...
k1 linsegr 1,1,1,0.1,0

chnset a1*k1, "in1"
chnset a2*k1, "in2"
chnset a3*k1, "in3"
chnset a4*k1, "in4"
chnset a5*k1, "in5"
chnset a6*k1, "in6"
chnset a7*k1, "in7"
chnset a8*k1, "in8"
```

So, add an Ambisonic setup, and press play! VBAP works similarly:

```
a1, a2, a3, a4, a5, a6, a7, a8 soundin "vbap.wav"
; fade out...
k1 linsegr 1,1,1,0.1,0

chnset a1*k1, "in1"
chnset a2*k1, "in2"
chnset a3*k1, "in3"
chnset a4*k1, "in4"
chnset a5*k1, "in5"
chnset a6*k1, "in6"
chnset a7*k1, "in7"
chnset a8*k1, "in8"
```

Note that real-time processing of eight sources, all with first order reflections and late reverb may not be possible, depending on processing power. Simply change the order of processing to 0 in *MultiBin.csd* to process only direct sources and late reverberation. Note, due to opcode optimisation, that static scenes will be significantly more efficient. The Ambisonic and VBAP wav files were created with *simpleambi.csd* and *simplevbap.csd* in the 'Chapter6' folder respectively. It is hoped that this discussion illustrates the ease with which arbitrary source material may be employed.

6.9.3 Creative Use

The application, although generic, is primarily intended for creative use. Spatial audio is becoming an ever more relevant aspect of composition (as stated by Blauert, 'there is no such thing as non-spatial hearing' [21]). In [14], spatial composition (and 'spatial choreography') is discussed. Historically, location based polychoral music is referenced as a first example of spatial music, followed by location-based composition (essentially reverberation based), leading to the fine degree of control of spatialisation afforded by modern techniques. The dichotomy of the task of creative use of technology is highlighted; comprehension, correct use and optimisation of spatialisation can be an extremely technical endeavour. The composer must take care not to loose sight of the ultimate creative goal. It is, however, this author's opinion that the developing nature of the research area also demands creativity on the behalf of the technical researcher; there is not yet one all-encompassing solution.

In [38], the authors also discuss using HRTFs compositionally. Perceptual difficulties with HRTF processing (reversals, externalisation) are discussed from the point of view of the composers ‘spatial tessitura’ and ideas like spatial modulation are suggested. Only HRTF processing is used here; the findings are similar to those of this work which inspired and informed the development of binaural reverberation. The author (in [38]) specifically mentions that reverberation (even when added before spatialisation, in a more artificial processing chain than the idealisation realised here) greatly improves externalisation. The benefits of HRTF processing over amplitude panning in headphones are epitomised succinctly in [38], where amplitude panning is described as creating a potential imbalance or ‘vacuum’.

Primarily, MultiBin serves as a functional composition aid. Multi-channel works can be auditioned on headphones, removing time and expense from multi-channel composition (which may have previously led to composers discounting multi-channel composition by necessity). As discussed, audition of any multi-channel setup is possible (even WFS, as is presented in [218]), in a dynamic, user driven context. Virtual WFS also allows the processing of a dry source only, with the application of less computationally intensive environmental processing (replacing the need for multiple impulse responses).

Any form of spatial sound design is obviously another application. The interactivity offered also implies that MultiBin, with appropriate sources, can be considered as an autonomous binaural work.

6.10 Conclusion

An overview of multi-channel spatialisation is offered in this chapter, as well as an insight into how multi-channel algorithms can be processed binaurally. This approach is implemented in the MultiBin application, which allows dynamic source and listener locations in a user-defined environment. In contrast to other techniques, it does not attempt to minimise or optimise the multi-channel representation. Nor does it assume sweet spot user location. Uses of the application are discussed, focusing on the dynamic audition of multi-channel loudspeaker setups.

MultiBin exploits the integrated architecture of the reverb opcodes; the user-defined room parameters informing both the early reflections as well as the overall late reverb. It is hoped that the application will encourage creative experimentation in multi-channel audio (and indeed spatial audio more generally), which would perhaps not have been possible with hardware and other restrictions that have now been overcome.

Chapter 7. Conclusions

Overall conclusions will now be drawn. From a point of view of novel contributions of this work, four distinct aspects should be considered: (i) algorithm development, (ii) analysis, insight and critique of the minimum-phase HRTF assumption and phase unwrapping threshold technique, (iii) transparency of algorithm implementation, directly addressing the often neglected subtleties involved, and (iv) application of the techniques developed in a flexible multi-channel binaural tool.

Algorithm development was discussed in chapters 2 (HRTF modelling and interpolation) and 5 (binaural reverberation). Two new approaches to HRTF interpolation, Phase Truncation and a Functional Model were presented. Both aim to use empirical data directly, with no obligation to prepare, compress or process empirical HRTF data. A thorough literature review motivates the techniques used. Phase Truncation uses a nearest measured phase paradigm with user-definable efficient brief crossfades to avoid any discontinuities. The Functional Model extracts a low-frequency spectral scaling factor from a HRTF dataset and uses it to improve ITD accuracy (a theoretically psychoacoustically-ideal model). Although initially this data extraction appears to imply a caveat to direct data use, the scaling curves of the generic MIT HRTF dataset can be used generally if required. Crucially, the algorithms both perform better than a minimum-phase based approach in testing.

Binaural reverberation development was again motivated by a comprehensive literature review. The standard approach of processing early reflections and later reverberation separately is employed. Improved accuracy is afforded to early reflections and an FDN model is adopted and developed for later reverberation.

Interaural coherence is added to the parametric scenario presented. Details of dynamic FDN initialisation and use are also offered.

The assumption that HRTFs can be represented as minimum-phase plus delay systems was reviewed and challenged in chapter 2. The approach is clearly problematic for a number of source locations. Both objective and subjective testing confirm the problems with using the approximation in a spatialisation context.

Similarly, the typically employed threshold based phase unwrapping technique was scrutinised in the context of HRTF processing. The literature suggests that the method is often used. However, it is by no means infallible. Several examples of failures of the algorithm when applied to the MIT HRTF dataset were presented. It is concluded that use of the method with a complex signal set such as a HRTF database should be carefully considered.

From an implementation point of view, fine detail of the algorithms presented was offered. Great care has been taken to prepare transparent, usable, efficient solutions. Both command-line and real-time solutions were offered for the HRTF algorithms. Csound implementations were developed (and indeed early incarnations of the solutions have been in use for a number of years) in the hope that the work will not remain purely theoretical, but will be widely disseminated. The reverberation opcodes epitomise the development paradigm employed, offering an integrated solution with a minimal number of required parameters. Although valid as independent processes (the late-reverberation opcode is particularly useful in this context), the later reverberation process can directly use outputs of the early reflection opcode. Furthermore, advanced parameters can be utilised by more expert users.

The multi-channel binaural tool presented in chapter 6 further highlights the usability of the algorithms presented. It offers tangible applications of the new suite of binaural processes, primarily the audition of multi-channel setups in headphones. The novel flexibility offered by the user friendly proof-of-concept software presented illustrates the potential of the paradigm.

The vast literature on the topics covered was found to lack detail at times, and even appear contradictory. Overall, it is hoped that a sense of clarity, perspective and focus has been offered. Ultimately, it is hoped that the software development and publications will encourage creative work in the area of spatial audio. The audition of multi-channel setups in headphones is the obvious primary use of the algorithms; however, perhaps binaural composition now also merits further attention.

7.1 Possibilities for Further Development

The multi-disciplinary and ever-expanding nature of virtual environmental processing implies several potential outlets for further development. Although comprehensive and autonomous in nature, the algorithms presented could benefit from further research in fields such as source directionality. As discussed in [187], this is a complex process, dictated strongly by modes in musical instruments. On a related topic, source shape could be considered. Auralisation of arbitrary shapes, as opposed to point sources is discussed in [9]. From the point of view of the MultiBin application, loudspeaker responses could be considered [39]. Also of interest for the MultiBin application is the possibility of three-dimensional GUI environments, moving towards multi-modal virtual-reality simulation [157]. This, in turn, suggests potential application in several fields, such as medicine (spatial rehabilitation), architecture (pre-design of structures), multimedia (next-generation entertainment), etc. Ultimately, however, further processing leads to increased processing requirements. The systems developed offer efficient and insightful solutions to the core requirements of virtual environmental processing.

From a programming point of view, a more generalised approach offering a library and API that would allow configuration for use with other datasets is being considered. From the point of view of the existing Python code (the MultiBin application), perhaps a system to allow the parsing of a more general source instrument could be considered.

Bibliography

1 Ajdler, T., Faller, C., Sbaiz, L. and Vetterli, M. Interpolation of Head Related Transfer Functions Considering Acoustics, AES, 118th Convention, 2005

2 Ajdler, T., Faller, C., Sbaiz, L. and Vetterli, M. Sound Field Analysis along a Circle and Its Applications to HRTF Interpolation, JAES, 56 (3), 2008

3 Algazi, V., Avendano, C. and Duda, R. Estimation of a Spherical-head Model from Anthropometry, JAES, 49(6), 2001

4 Algazi, V., Duda, R. and Thompson, D. The CIPIC Database, IEEE, WASPAA, 2001

5 Algazi, V., Duda, R. and Thompson, D. The Use of Head-and-Torso Models for Improved Spatial Sound Synthesis, AES, 113th Convention, 2002

6 Allen, J. and Berkley, D. Image Model for Efficiently Simulating Small-room Acoustics, JASA, 65 (4), 1979

7 Avendano, C., Duda, R. and Algazi, V. Modelling the Contralateral HRTF, AES, 16th Conference, 1999

8 Baalman, M. Application of Wave Field Synthesis in Electronic Music and Sound Installations, LAC, 2004

9 Baalman, M. *swonder3Dq*: Auralisation of 3d Objects with Wave Field Synthesis, LAC, 2006

10 Baalman, M. Updates of the WONDER Software Interface for using Wave Field Synthesis, LAC, 2005

11 http://www2.ak.tu-berlin.de/~mbaalman/, accessed August 2010

12 Baalman, M., Hohn, T., Schampijer, S. and Koch, T. Renewed Architecture of the sWONDER Software for Wave Field Synthesis on Large Scale Systems, LAC, 2007

13 Bates, E., Kearney, G., Boland, F. and Furlong, D. Localization Accuracy of Advanced Spatialization Techniques in Small Concert Halls, ASA, 153rd Meeting, 2007

14 Begault, D. 3-D Sound for Virtual Reality and Multimedia, NASA, 2000

15 Begault, D. Audible and Inaudible Early Reflection: Thresholds for Auralization System Design, AES, 100th Convention, 1996

16 Begault, D. Auditory and Non-auditory Factors that Potentially Influence Virtual Acoustic Imagery, AES, 16th Conference, 1999

17 Begault, D. Perceptual Effects of Synthetic Reverberation on Three-Dimensional Audio Systems, JAES, 40 (11), 1992

18 Begault, D., Godfroy, M., Miller, J., Roginska, A., Anderson, M. and Wenzel, E. Design and Verification of HeadZap, a Semi-automated HRIR Measurement System, AES, 120th Convention, 2006

19 Begault, D., Wenzel, E. and Anderson, M. Direct Comparison of the Impact of Head Tracking, Reverberation and Individualized Head-Related Transfer Functions on the Spatial Perception of a Virtual Speech Source, JAES, 49 (10), 2001

20 Bitzer, J. and Extra, D. Artificial Reverberation: Comparing Algorithms by Using Binaural Analysis Tools, AES, 121st Convention, 2006

21 Blauert, J. Spatial Hearing, MIT Press, Massachusetts, 1997

22 Borish, J. An Auditorium Simulator for Domestic Use, JAES, 33 (5), 1985

23 Borish, J. Extension of the Image Model to Arbitrary Polyhedra, JASA, 75 (6), 1984

24 Borβ, C. A VST Reverberation Effect Plugin based on Synthetic Room Impulse Responses, DAFx, 2009

25 Boulanger, R. (ed.) The Csound Book, MIT Press, Massachusetts, 2000

26 Boulanger, R. and Lazzarini, V. (eds.) The Audio Programming Book, MIT Press, Massachusetts, 2010

27 Bregman, A. Auditory Scene Analysis, MIT Press, Massachusetts, 1990

28 Breebaart, J., Herre, J., Villemoes, L., Jin, C., Kjörling, K., Plogsties, J. and Koppens, J. Multi-channel Goes Mobile: MPEG Surround Binaural Rendering, AES, 29th Conference, 2006

29 Breebaart, J. and Kohlrausch, A. The Perceptual (Ir)relevance of HRTF Magnitude and Phase Spectra, AES, 110th Convention, 2001

30 Breebaart, J., Nater, F. and Kohlrausch, A. Parametric Binaural Synthesis: Background, Applications and Standards, International Conference on Acoustics, 2009

31 Burkhard, M. and Sachs, R. Anthropometric Manikin for Acoustic Research, JASA, 58(1), 1975

32 Busson, S., Nicol, R. and Katz, B. Subjective Investigations of the Interaural Time Difference in the Horizontal Plane, AES, 118th Convention, 2005

33 Carty, B. Artificial Simulation of Audio Spatialisation: Developing a Binaural System, Maynooth Musicology, 1, 2008

34 Carty, B. Binaural Processing: A Sample Application, in Boulanger, R. and Lazzarini, V. (eds.), The Audio Programming Book, MIT Press, 2010

35 Carty B. HRTFmove, HRTFstat, HRTFmove2: Using the new HRTF Opcodes, The Csound Journal, 2008

36 Cheng, C. Visualisation, Measurement, and Interpolation of Head-related Transfer Functions (HRTFs) with Applications in Electroacoustic Music, Doctoral Dissertation, University of Michigan, 2001

37 Cheng, C. and Wakefield, G. Introduction to Head-related Transfer Functions (HRTFs): Representation of HRTFs in Time, Frequency, and Space, AES, 107th Convention, 1999

38 Cheng, C. and Wakefield, G. Moving Sound Source Synthesis for Binaural Electroacoustic Music using Interpolated Head-related Transfer Functions (HRTFs), CMJ, 25 (4), 2001

39 http://www.clfgroup.org/, accessed August 2010

40 Cook, P. Music, Cognition, and Computerized Sound, MIT Press, Massachusetts, 1999

41 Corteel, E. and Caulkins, T. Sound Scene Creation and Manipulation using Wave Field Synthesis, IRCAM

42 http://www.csounds.com/manual/html/bformdec1.html, accessed August 2010

43 http://www.csounds.com/manual/html/eqfil.html, accessed August 2010

44 http://www.csounds.com/manual/html/csound5extending.html, accessed August 2010

45 http://www.csounds.com/manual/html/ControlFltkIntro.html, accessed August 2010

46 http://www.csounds.com/manual/html/hrtfmove.html, accessed August 2010

47 http://www.csounds.com/manual/html/hrtfmove2.html, accessed August 2010

48 http://www.csounds.com/manual/html/hrtfstat.html, accessed August 2010

49 http://www.csounds.com/manual/html/tone.html, accessed August 2010

50 http://www.csounds.com/manual/html/vbap8move.html, accessed August 2010

51 Dalenbäck, B. and Strömberg, M. Real Time Walkthrough Auralization-The First Year, Institute of Acoustics, 28 (2), 2006

52 de Vries, D. and Baan, J. Auralization of Sound Fields by Wave Field Synthesis, AES, 106th Convention, 1999

53 Dodge, C. and Jerse, T. Computer Music: Synthesis, Composition and Performance, Schirmer, NY, 1997

54 Doukhan, D. and Sédès, A. CW_binaural~: A Binaural Synthesis External for Pure Data, PD Convention, 2009

55 Duda, R. and Martens, W. Range Dependence of the Response of a Spherical Head Model, JASA, 104(5), 1998

56 Duraiswami, R., Zotkin, D. and Gumerov, N. Interpolation and Range Extrapolation of HRTFs, International Conference on Acoustics, Speech and Signal Processing, 2004

57 Evans, M., Angus, J. and Tew, A. Analysing Head-related Transfer Function Measurements using Surface Spherical Harmonics, JASA, 104(4), 1998

58 http://www.eventide.com/AudioDivision.aspx, accessed August 2010

59 Everest, F. Master Handbook of Acoustics, McGraw-Hill, 2000

60 Faller, C. and Merimaa, J. Source Localization in Complex Listening Situations: Selection of Binaural Cues based on Interaural Coherence, JASA, 116 (5), 2004

61 Farina, A. and Ugolotti, E. Automatic Measurement System for Car Audio Applications, AES, 104th Convention, 1998

62 http://www.fftw.org/, accessed August 2010

63 http://www.fftw.org/benchfft/, accessed August 2010

64 http://www.fftw.org/fftw3_doc/, accessed August 2010

65 Freeland, F., Biscainho, L. and Diniz, P. Efficient HRTF Interpolation in 3D Moving Sound, AES, 22nd Conference, 2002

66 Frigo, M. and Johnson, S. The Design and Implementation of FFTW3, IEEE Proceedings, 93 (2), 2005

67 Fuster, L., López, J. González, A. and Faus, P. Time and Frequency Domain Room Compensation Applied to Wave Field Synthesis, DAFx, 2005

68 Gardner, W. 3-d Audio using Loudspeakers, PhD Dissertation, MIT, 1997

69 Gardner, W. Efficient Convolution without Input/Output Delay, AES, 97th Convention, 1994

70 Gerzon, M. Ambisonics in Multichannel Broadcasting and Video, JAES, 33 (11), 1985

71 Gerzon, M. Periphony: With-height Sound Reproduction, JAES, 21 (1), 1973

72 Gilkey, R. and Anderson, T. Binaural and Spatial Hearing in Real and Virtual Environments, Laurence Erlbaum Associates, New Jersey, 1997

73 Goodwin, M. and Jot, J. Binaural 3-d Audio Rendering based on Spatial Audio Scene Coding, AES, 123rd Convention, 2007

74 Grantham, D. Detection and Discrimination of Simulated Motion of Auditory Targets in the Horizontal Plane, JASA, 79 (6), 1986

75 Griesinger, D. Practical Processors and Programs for Digital Reverberation, AES, 7th International Conference, 1989

76 Hacihabiboğlu, H., Günel, B and Kondoz, A. Head-related Transfer Function Filter Interpolation by Root Displacement, IEEE, WASPAA, 2005

77 Hartung, K., Braasch, J. and Sterbing, S. Comparison of Different Methods for the Interpolation of Head-related Transfer Functions, AES, 16th Conference, 1999

78 Hoffmann, P. and Møller, H. Audibility of Spectral Differences in Head-related Transfer Functions, AES 120th Convention, 2006

79 Hollerweger, F. An Introduction to Higher-order Ambisonic, http://flo.mur.at/writings, 2008, accessed August 2010

80 Howard, D. and Angus, J. Acoustics and Psychoacoustics, Focal Press, Oxford, 2006

81 Huopaniemi, J. Virtual Acoustics and 3d Sound in Multimedia Signal Processing, PhD Dissertation, Helsinki University of Technology, 1999

82 Huopaniemi, J. and Karjalainen, M. Review of Digital Filter Design and Implementation Methods for 3-d Sound, AES, 102nd Convention, 1997

83 Huopaniemi, J., Zacharov, N. and Karjalainen, M. Objective and Subjective Evaluation of Head-related Transfer Function Filter Design, AES, 105th Convention, 1998

84 Hynninen, J. and Zacharov, N. GuineaPig-A Generic Subjective Test System for Multichannel Audio, AES, 106th Convention, 1999

85 ITU-R BS. 1284-1 General Methods for the Subjective Assessment of Sound Quality

86 ITU-R BS. 1534-1 Method for the Subjective Assessment of Intermediate Quality Level of Coding Systems

87 Jensen, R. and Welti, T. The Importance of Reflections in a Binaural Impulse Response, AES, 114th Convention, 2003

88 Jot, J. An Analysis/Synthesis Approach to Real-time Artificial Reverberation, IEEE, ICASSP, 1992

89 Jot, J. Efficient Models for Reverberation and Distance Rendering in Computer Music and Virtual Audio Reality, ICMC, 2007

90 Jot, J., Cerveau, L. and Warusfel, O. Analysis and Synthesis of Room Reverberation Based on a Statistical Time-frequency Model, AES, 103rd Convention, 1997

91 Jot, J. and Chaigne, A. Digital Delay Networks for Designing Artificial Reverberators, AES, 90th Convention, 1991

92 Jot, J., Larcher, V. and Pernaux, J. A Comparative Study of 3-d Audio Encoding and Rendering Techniques, AES, 16th Conference, 1999

93 Jot, J., Larcher, V., Warusfel, O. Digital Signal Processing Issues in the Context of Binaural and Transaural Stereophony, AES, 98th Convention, 1995

94 Jot, J., Walsh, M. and Philip, A. Binaural Simulation of Complex Acoustic Scenes for Interactive Audio, AES, 121st Convention, 2006

95 Jot, J. and Wardle, S. Approaches to Binaural Synthesis, AES, 105th Convention, 1998

96 Kaman, Z. and Oppenheim, A. Computation of the One-dimensional Unwrapped Phase, IEEE, DSP Conference, 2007

97 Karjalainen, M. and Järvaläinen, H. More about this Reverberation Science: Perceptually Good Late Reverberation, AES, 111th Convention, 2001

98 Kearney, G., Bates, E., Boland, F. and Furlong, D. A Comparative Study of the Performance of Spatialization Techniques for a Distributed Audience in a Concert Hall Environment, AES, 31st Conference, 2007

99 Kearney, G., Masterson, C., Adams, S. and Boland, F. Towards Efficient Binaural Room Impulse Response Synthesis, EAA Auralization Symposium, 2009

100 Kelley, A. and Pohl, I. A Book on C, Benjamin Cummings Publishing, CA, 1995

101 Kendall, G. A 3d Sound Primer: Directional Hearing and Stereo Reproduction, CMJ, 19 (4), 1995

102 Kendall, G. and Martens, W. Simulating the Cues of Spatial Hearing in Natural Environments, ICMC, 1984

103 Kendall, G., Martens, W., Freed, D., Ludwig, D. and Karstens, R. Image Model Reverberation from Recirculating Delays, AES, 81st Convention, 1986

104 Kendall, G., Martens, W. and Wilde, M. A Spatial Sound Processor for Loudspeaker and Headphone Reproduction, AES, 8th Conference, 1990

105 Kistler, D. and Wightman, F. A Model of Head-related Transfer Functions based on Principal Components Analysis and Minimum-phase Reconstruction, JASA, 91(3), 1992

106 Krokstad, A., Strøm, S. and Sørsdal, S. Calculating the Acoustical Room Response by the use of a Ray Tracing Technique, JSV, 8 (1), 1968

107 Kudo, A., Hokari, H. and Shimada, S. A Study on Switching of the Transfer Functions Focusing on Sound Quality, Acoustical Science and Technology, 26 (3), 2005

108 Kuhn, G. Model for the Interaural Time Difference in the Azimuthal Plane, JASA, 62(1), 1977

109 Kulkarni, A. and Colburn, H. Infinite-impulse-response Models of the Head-related Transfer Function, JASA, 116 (4), 2004

110 Kulkarni, A., Isabelle, S. and Colburn, H. On the Minimum-phase Approximation of Head-related Transfer Functions, IEEE, ICASSP, 1995

111 Kulkarni, A., Isabelle, S. and Colburn, H. Sensitivity of Human Subjects to Head-related Transfer Function Phase Spectra, JASA, 105 (5), 1999

112 Kuster, M. Multichannel Room Impulse Response Rendering on the Basis of Undetermined Data, JAES, 57 (6), 2009

113 Kuttruff, K. Auralization of Impulse Responses Modeled on the Basis of Ray-Tracing Results, JAES, 41 (11), 1993

114 Kuttruff, H. Room Acoustics, Spon Press, London, 2009

115 Laasko, T., Välimäki, V., Karjalainen, M. and Laine, U. Splitting the Unit Delay, IEEE Signal Processing Magazine, 1996

116 Laitinen, M. and Pulkki, V. Binaural Reproduction for Directional Audio Coding, IEEE, WASPAA, 2009

117 Landone, C. and Sandler, M. 3-d Sound Systems: A Computationally Efficient Binaural Processor, IEE, Audio and Music Technology, 1998

118 Landone, C. and Sandler, M. Applications of Binaural Processing to Surround Sound Reproduction in Large Spaces, IEEE, Circuits and Systems, 2000

119 Landone, C. and Sandler, M. Digital Filtering for 3d Binaural Sound, IEE, Digital Filters, 1998

120 Larcher, V., Jot, J., Guyard, J. and Warusfel, O. Study and Comparison of Efficient Methods for 3D Audio Spatialization, AES, 108th Convention, 2000

121 Lazzarini, V. Extensions to the Csound Language: from User-Defined to Plugin Opcodes and Beyond, LAC, 2005

122 http://www.lexiconpro.com/, accessed August 2010

123 http://www.mega-nerd.com/libsndfile/, accessed August 2010

124 http://recherche.ircam.fr/equipes/salles/listen/index.html, accessed August 2010

125 Lorho, G. and Zacharov, N. Subjective Evaluation of Virtual Home Theatre Sound Systems for Loudspeakers and Headphones, AES, 116th Convention, 2004

126 Malham, D. Spatial Hearing Mechanisms and Sound Reproduction, http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm, 2008, accessed August 2010

127 Malham, D. and Myatt, A. 3-d Sound Spatialization using Ambisonic Techniques, CMJ, 19 (4), 1995

128 http://www.mathworks.com/products/matlab/, accessed August 2010

129 http://www.mathworks.com/access/helpdesk/help/techdoc/ref/unwrap.html, accessed August 2010

130 Matsumoto, M., Yamanaka, S. Tohyama, M., and Nomura, H. Effect of Arrival Time Correction on the Accuracy of Binaural Impulse Response Interpolation, JAES, 52 (1/2), 2004

131 McGovern, S. A Model for Room Acoustics, 2003

132 McGovern, S. Fast Image Method for Impulse Response Calculations of Box-shaped Rooms, Applied Acoustics, 70 (1), 2009

133 Meesawat, K. and Hammershøi, D. An Investigation on the Transition from Early Reflections to a Reverberation Tail in a BRIR, ICAD, 2002

134 Meesawat, K. and Hammershøi, D. The Time when the Reverberation Tail in a Binaural Room Impulse Response Begins, AES, 115th Convention, 2003

135 Mehrgardt, S. and Mellert, V. Transformation Characteristics of the External Human Ear, JASA, 61 (6), 1977

136 Menzel, D., Wittek, H., Theile, G and Fastl, H. The Binaural Sky: A Virtual Headphone for Binaural Room Synthesis, Tonnmeister Symposium, 2005

137 Menzer, F. and Faller, C. Binaural Reverberation using a Modified Jot Reverberator with Frequency-dependent Interaural Coherence Matching, AES, 126th Convention, 2009

138 Menzer, F. and Faller, C. Obtaining Binaural Room Impulse Responses from B-format Impulse Responses, AES, 125th Convention, 2008

139 Minnaar, P., Christensen, F., Møller, H., Olesen, S. and Plogsties, J. Audibility of All-pass Components in Binaural Synthesis, AES, 106th Convention, 1999

140 Minnaar, P., Plogsties, J. and Christensen, F. Directional Resolution of Head-related Transfer Functions Required in Binaural Synthesis, JAES, 53 (10), 2005

141 Minnaar, P., Plogsties, J., Olesen, S., Christensen, F., and Møller, H. The Interaural Time Difference in Binaural Synthesis, AES, 108th Convention, 2000

142 http://sound.media.mit.edu/resources/KEMAR.html, accessed August 2010

143 Moore, B. An Introduction to the Psychology of Hearing, Emerald, Bingley, 2004

144 Moore, F. Elements of Computer Music, Prentice Hall, NJ, 1990

145 Moorer, J. About this Reverb Business, CMJ, 3 (2), 1979

146 Mouba, J. and Marchand, S. A Source Localization/Separation/Respatialization System based on Unsupervised Classification of Interaural Cues, DAFx, 2006

147 Murphy, D., Beeson, M., Shelley, S., Moore, A. and Southern, A. Hybrid Room Impulse Response Synthesis in Digital Waveguide Mesh based Room Acoustics Simulation, DAFx, 2008

148 Musil, T., Noisternig, M. and Höldrich, R. A Library for Realtime 3d Binaural Sound Reproduction in Pure Data (PD), DAFx, 2005

149 Nam, J., Abel, J. and Smith, J. A Method for Estimating Interaural Time Difference for Binaural Synthesis, AES, 125th Convention, 2008

150 Naylor, G. and Rindel, J. Predicting Room Acoustical Behaviour with the ODEON Computer Model, ASA, 124th Meeting, 1992

151 Nettinsmeier, J. AMBI@Home-The Search for Extra-frontal Intelligence, LAC, 2008

152 Nielsen, S. Auditory Distance Perception in Different Rooms, JAES, 41 (10), 1993

153 Nishino, T., Ikeda, M., Takeda, K. and Itakura, F. Interpolating Head Related Transfer Functions, Western Pacific Regional Acoustics Conference, 2000

154 Nishino, T., Kajita, S., Takeda, K. and Itakura, F. Interpolating Head Related Transfer Functions in the Median Plane, IEEE, WASPAA, 1999

155 Noisternig, M., Musil, T., Sontacchi, A. and Höldrich, R. A 3d Real Time Rendering Engine for Binaural Sound Reproduction, ICAD, 2003

156 ODEON Room Acoustics Modelling Software Product Data

157 Olaiz, N., Arumí, P., Mateos, T. and Garcia, D. 3d-audio with CLAM and Blender's Game Engine, LAC, 2009

158 http://opensoundcontrol.org/, accessed August 2010

159 Oppenheim, A. and Schafer, R. Discrete-time Signal Processing, Prentice-hall, New Jersey, 1999

160 Pallant, J. SPSS Survival Manual, Open University, Press, Buckingham, 2001

161 Perrott, D. and Musicant, A. Minimum Auditory Movement Angle: Binaural Localization of Moving Sound Sources, JASA, 62 (6), 1977

162 Peterson, K., d_fftroutine.c, Pure Data 0.41.4-extended, MIT Media Lab, 1986

163 Plogsties, J., Olesen, S., Minnaar, P., Christensen, F. and Møller, H. Audibility of All-pass Components in Head-related Transfer Functions, AES, 108th Convention, 2000

164 Pulkki, V. Spatial Sound Reproduction with Directional Audio Coding, JAES, 55 (6), 2007

165 Pulkki, V. Virtual Sound Source Positioning Using Vector Base Amplitude Panning, JAES, 45 (6), 1997

166 Pulkki, V. and Karjalainen, M. Localization of Amplitude-panned Virtual Sources 1: Stereophonic Panning, JAES, 49 (9), 2001

167 Pulkki, V., Karjalainen M. and Huopaniemi, J. Analyzing Virtual Sound Source Attributes using a Binaural Auditory Model, JAES, 47 (4), 1999

168 Pulkki, V. and Merimaa, J. Spatial Impulse Response Rendering: A Tool for Reproducing Room Acoustics for Multi-channel Listening, Waves Inc.

169 http://www.python.org/, accessed August 2010

170 http://www.python.org/doc/, accessed August 2010

171 http://www.swaroopch.com/notes/Python, accessed August 2010

172 Queiroz, M. and de Sousa, G. Structured IIR Models for HRTF Interpolation, ICMC, 2010

173 http://compmus.ime.usp.br/hrtfinterpolation, accessed August 2010

174 Rakerd, B. and Hartmann, W. Localization of Sound in Rooms, 2: The Effect of a Single Reflecting Surface, JASA, 78 (2), 1985

175 Regalia, P. and Mitra, S. Tunable Digital Frequency Response Equalization Filters, IEEE, ICASSP, 1987

176 Röber, N., Andres, S. and Masuch, M. HRTF Simulations through Acoustic Raytracing, Technischer Report, Fakultät für Informatik,
177 Otto-von-Guericke Universität, Magdeburg, 2006

178 Rocchesso, D. Introduction to Sound Processing, http://profs.sci.univr.it/~rocchess/SP/sp.pdf, 2003, accessed August 2010

179 Rocchesso, D. The Ball within the Box: A Sound-Processing Metaphor, CMJ, 19 (4), 1995

180 Rubak, P. Headphone Signal Processing System for Out-of-head Localization, AES, 90th Convention, 1991

181 Rumori, M. Girafe-A Versatile Ambisonics and Binaural System, Ambisonics Symposium, Graz, 2009

182 Rumsey, F. and McCormick, T. Sound and Recording, Focal Press, Oxford, 2002

183 Runkle, P., Blommer, M. and Wakefield, G. A Comparison of Head Related Transfer Function Interpolation Methods, IEEE, WASPAA, 1995

184 Sandvad, J. Dynamic Aspects of Auditory Virtual Environments, AES, 100th Convention, 1996

185 Sandvad, J. and Hammershøi, D. Binaural Auralization. Comparison of FIR and IIR Filter Representation of HIRs, AES, 96th Convention, 1994

186 Savioja, L. Modeling Techniques for Virtual Acoustics, PhD Thesis, Helsinki University of Technology, 2000

187 Savioja, L., Huopaniemi, J., Lokki, T. and Väänänen, R. Creating Interactive Virtual Acoustic Environments, JAES, 47 (9), 1999

188 Schafer, R. and Rabiner, L. A Digital Signal Processing Approach to Interpolation, IEEE Proceedings, 61 (6), 1973

189 Schroeder, M. Natural Sounding Artificial Reverberation, JAES, 10 (2), 1962

190 Seo, J., Shim, H., Yoo, J. and Sung, K. Artificial Reverberation Algorithm to Control Distance and Direction of Sound Source for Multi-channel Audio System, AES, 119th Convention, 2005

191 Shaw, E. Acoustical Features of the Human External Ear, in Gilkey, R. and Anderson, T. Binaural and Spatial Hearing in Real and Virtual Environments, Laurence Erlbaum Associates, New Jersey, 1997

192 Slaney, M. An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank, Apple Computer, 1993

193 https://ccrma.stanford.edu/~jos/pasp/pasp.html, accessed August 2010

194 https://ccrma.stanford.edu/~jos/sasp/sasp.html, accessed August 2010

195 https://ccrma.stanford.edu/~jos/sasp/FFT_versus_Direct_Convolution.html#21440, accessed August 2010

196 https://ccrma.stanford.edu/~jos/sasp/Phase_Interpolation_Peak.html, accessed August 2010

197 http://sndobj.sourceforge.net/, accessed August 2010

198 http://support.ircam.fr/forum-ol-doc/spat/3.0/spat-3-ref/co/spat-3.html, accessed August 2010

199 Sontacchi, A., Noisternig, M., Majdak, P. and Höldrich, R. An Objective Model of Localization in Binaural Sound Reproduction Systems, AES, 21st Conference, 2002

200 Sporer, T. Wave Field Synthesis-Generation and Reproduction of Natural Sound Environments, DAFx, 2004

201 Spors, S., Rabenstein, R. and Ahrens, J. The Theory of Wave Field Synthesis Revisited, AES, 124th Convention, 2008

202 Stautner, J. and Puckette, M. Designing Multi-Channel Reverberators, CMJ, 6 (1), 1982

203 Steiglitz, K. A Digital Signal Processing Primer, Addison-wesley, CA, 1996

204 Stewart, R. and Murphy, D. A Hybrid Artificial Reverberation Algorithm, AES, 122nd Convention, 2007

205 Stewart, R. and Sandler, M. Real-time Panning Convolution Reverberation, AES, 123rd Convention, 2007

206 Thiele, G. Wave Field Synthesis-A Promising Spatial Audio Rendering Concept, DAFx, 2004

207 Theile, G. and Wittek, H. Wave Field Synthesis-A Promising Spatial Audio Rendering Concept, Journal of the Institute of Image and Television Engineers, 2007

208 http://effbot.org/tkinterbook/, accessed August 2010

209 http://infohost.nmt.edu/tcc/help/pubs/tkinter/, accessed August 2010

210 http://www.tcl.tk/, accessed August 2010

211 Tsakostas, C. and Floros, A. Real-time Spatial Representation of Moving Sound Sources, AES 123rd Convention, 2007

212 Ubilla, M., Domingo, M. and Cadiz, R. Head Tracking for 3d Audio using the Nintendo WII Remote, ICMC, 2010

213 Välimäki, V. and Laasko, T. Suppression of Transients in Time-varying Recursive Filters for Audio Signals, IEEE, ICASSP, 98

214 Väljamäe, A., Larsson, P., Västfjäll, D. and Kleiner, M. Auditory Presence, Individualized Head-Related Transfer Functions, and Illusory Ego-Motion in Virtual Environments, Presence Workshop, 2004

215 Viste, H. Binaural Localization and Separation Techniques, PhD Dissertation, Lausanne EPFL, 2004

216 Viste, H. and Evangelista, G. Binaural Source Localization, DAFx, 2004

217 Viste, H. and Evangelista, G. On the Use of Spatial Cues to Improve Binaural Source Separation, DAFx, 2003

218 Völk, F., Konradl, J. and Fastl, H. Simulation of Wave Field Synthesis, Acoustics, 2008

219 http://vrsonic.com/, accessed August 2010

220 http://www.waves.com/, accessed August 2010

221 Wenzel, E., Arruda, M., Kistler, D. and Wightman, F. Localization using Non-individualised Head-related Transfer Functions, JASA, 94 (1), 1993

222 Wenzel, E. and Foster, S. Perceptual Consequences of Interpolating Head-related Transfer Functions during Spatial Synthesis, IEEE, WASPAA, 1993

223 Wenzel, E., Miller, J. and Abel, J. A Software-based system for Interactive Spatial Sound Synthesis, ICAD, 2000

224 Wenzel, E., Miller, J. and Abel, J. Sound Lab: A Real-time, Software-based System for the Study of Spatial Hearing, AES, 108th Convention, 2000

225 Wightman, F. and Kistler, D. The Dominant Role of Low-frequency Interaural Time Differences in Sound Localization, JASA, 91 (3), 1992

226 Wittek, H. Perceptual Differences between Wavefield Synthesis and Stereophony, PhD Dissertation, University of Surrey, 2007

227 http://sourceforge.net/projects/swonder/, accessed August 2010

228 Woodworth, R. and Schlosberg, G. Experimental Psychology, Holt, Rinehard and Winston, New York, 1962

229 Xiang, P., Camargo, D. and Puckette, M. Experiments on Spatial Gestures in Binaural Sound Display, ICAD, 2005

230 Zacharov, N. and Huopaniemi, J. Results of a Round Robin Subjective Evaluation of Virtual Home Theatre Sound Systems, AES, 107th Convention, 1999

231 Zhang, M., Tan, K. and Er, M. Three-Dimensional Sound Synthesis Based on Head Related Transfer Functions, JAES, 46 (10), 1998

232 Zieliński, S., Rumsey, F. and Bech, S. On Some Biases Encountered in Modern Audio Quality Listening Tests-A Review, JAES, 56 (6), 2008

233 Zotkin, D., Duraiswami, R. and Davis, L. Creation of Virtual Auditory Spaces, IEEE, ICASSP, 2002

234 Zotkin, D., Duraiswami, R. and Davis, L. Rendering Localized Spatial Audio in a Virtual Auditory Space, IEEE Transactions on Multimedia, 6 (4), 2004

235 Zotkin, D., Hwang, J. Duraiswami, R. and Davis, L. HRTF Personalization using Anthropometric Measurements, IEEE, WASPAA, 2003