3-D Spatialization and Localization, and Simulated Surround Sound with Headphones

LUCAS O'NEIL
BRENDAN CASSIDY

Aside from the amplitude and time cues that humans use to determine localization of sound, there are three other cues come into play to determine the source position of audio. A head-related transfer function (HRTF) can be used to reproduce most of the cues that will accomplish the 3D localization if the end result of the sound material can be assumed to be listened to on headphones (where the sound is directly 'injected' into the ear canal). Two methods were studied and implemented to reproduce the HRTF: a model of the human head, torso and pinna (part of the ear); and a collection of measured impulse responses directly inside the ear. The results were convincing spatialization effects for 3D with headphones.

In this study, a 5.1 stereo upmixer was researched and implemented in attempt to follow Dolby Pro Logic, an industry leading multichannel playback technology. The upmixer involved static and dynamic mixing gains, delay effects, filter effects, and subwoofer isolation. Two modes were implemented: *Pro Logic* mode for surround playback of movies or television audio, and *Music* mode for surround playback of stereo music.

By upmixing stereo audio to surround sound and by reproducing the effects of a head-related transfer functions for the placement of surround sound loudspeakers, simulated surround sound with headphones can be achieved. Using the HRTF and 5.1 Upmixer, such an effect was studied and implemented, with the results being a convincing surround sound experience.

TABLE OF CONTENTS

1 INTRODUCTION 1
1.1 SPATIAL PERCEPTION OF AUDIO 1
1.2 SURROUND SOUND 1
1.3 SIMULATED SURROUND SOUND WITH HEADPHONES 1
2 CONTEXT 1
2.1 PURPOSE 1
2.2 SCOPE 1
3 REFERENCES 1
4 PROBLEM STATEMENT 1
5 Analysis 2
5.1 3D WITH HEADPHONES 2
5.1.1 HRTF Modeling 2
5.1.2 Convolution with Impulse Response 3
5.1.3 Rotating Source Demo 4
5.2 5.1 SURROUND UPMIXING 4
5.2.1 Dolby Pro Logic II 4
5.2.2 Implementation of a 5.1 Upmixer 5
5.2.3 Static Gain Decoding 5
5.2.4 Autopanning 6
5.2.5 Delay Effects 6
5.2.6 Filtering Effects 6
5.2.7 Subwoofer Isolation 7
5.3 SIMULATED 5.1 SURROUND SOUND WITH HEADPHONES 7
6 RESULTS 7
6.1 3D WITH HEADPHONES 7
6.2 5.1 SURROUND UPMIXING 8
6.3 SIMULATED 5.1 SURROUND WITH HEADPHONES 8
7 DISCUSSION 9
7.1 3D WITH HEADPHONES 9
7.2 5.1 SURROUND UPMIXING 10
7.3 SIMULATED 5.1 SURROUND WITH HEADPHONES 10
8 7 CONCLUSIONS 10
Appendix A: MATLAB Source Listing A1

LIST OF TABLES AND FIGURES

Figure 1: Structural model of the head-torso-pinna system...2
Figure 2: Standard 5.1 surround sound loudspeaker layout. 4
Figure 3: Basic Pro Logic II decoder system block diagram. [4] 5
Figure 4: 5.1 surround upmixer system block diagram..5
Figure 5: Simulated 5.1 Surround with Headphones system block diagram......................7

Table 1: Parameters for amplitude and time delay of pinna reflections [1]. 3
Table 2: Binaural Audio for 3D with headphones 8
Table 3: Source and 5.1 Upmixed channels for Pro Logic and Music modes....................8
Table 4: Simulated 5.1 Surround with Headphones for various modes/methods.............9
Table 5: Isolated 5.1 Surround Channels spatialized for headphones using Music upmixing mode and measured HRIRs[2] 9
Table 6: Simulated 5.1 Surround with Headphones using Music upmixing mode and measured HRIRs[2] 9

1 INTRODUCTION

1.1 SPATIAL PERCEPTION OF AUDIO

The human aural system uses many cues to determine the source position of audio. Time and amplitude differences are two fundamental cues. However, in order to perceive 3D positions, other cues are needed. The compounded result of head shadowing, torso reflections and pinna reflections is known as the head-related transfer function, which describes how audio is altered before entering the ear canal.

1.2 SURROUND SOUND

Surround sound is a method of playing back audio with a collection of loudspeakers placed around the listener. This provides an effect of spatializing audio, as the source location of a sound can be placed around the listener, rather than simply from one or two loudspeakers.

There are many surround sound setups commonly used in the audio industry. Examples include 4.1, 5.1, 7.1 and 7.2; the number preceding a decimal refers to the number of loudspeakers placed around the listener while the number following the decimal refers to the number of subwoofers playing low frequencies.

For this study, a 5.1 stereo upmixer was researched and implemented in attempt to follow Dolby Pro Logic, an industry leading multichannel playback technology. Such an upmixer would isolate each of six 5.1 channels for a stereo (2 track) input.

1.3 SIMULATED SURROUND SOUND WITH HEADPHONES

By upmixing stereo audio to surround sound and by reproducing the effects of a head-related transfer functions for the placement of surround sound loudspeakers, simulated surround sound with headphones can be achieved. This report will discuss the study of and implementation of such an effect.

2 CONTEXT

2.1 PURPOSE

The purpose of this report is to discuss the study and implementation of 3D audio effects with headphones, 5.1 surround sound upmixing of stereo audio and simulated 5.1 surround with headphones.

2.2 SCOPE

The scope of this report is limited to the methods of achieving the aforementioned effects that were studied and implemented. It will discuss the relative advantages and disadvantages of the methods, as well as provide insight for possible improvements on the implementations.

3 REFERENCES

[1] Zolzer, Udo. DAFX. John Wiley & Sons. West Sussex, England: 2003. 149-159.
[2] HRTF Measurements of a KEMAR Dummy-Head Microphone. 18 July, 2000. <http://sound.media.mit.edu/KEMAR.html>.
[3] Dolby Laboratories Inc. Whitepaper. Dolby Surround Pro Logic Decoder. Roger Dressler.
[4] Dolby Laboratories Inc. Whitepaper. Dolby Surround Pro Logic II Decoder Principles of Operation

4 PROBLEM STATEMENT

The problems at hand in this study were twofold:

The first problem was to implement various methods of spatializing audio for headphones, specifically to model a head related transfer function using the head/torso/pinna model and to use measured head related impulse responses.

The second problem, which was the most significant part of this study and which involved significant research and engineering design decisions, was to develop a 5.1 surround sound upmixer and to combine the upmixer with the aforementioned spatialization effects to simulate surround sound with headphones. That is, a stereo audio recording should be upmixed to six distinct surround channels, and each of these six surround channels should then undergo spatialization to place them in the appropriate location, and the result should be downmixed back to stereo for headphones. The result should, in theory, give the listener the experience of listening to audio in surround sound using headphones.

After studying and implementing these effects, they were also to be demonstrated.

5 Analysis

5.1 3D WITH HEADPHONES

Aside from the amplitude and time cues (IID and ITD) that we use to determine localization of sound, there are three other cues come into play when determining the three dimensional specialization of the sound. Localization of the sound source in relation to elevation and whether it is in front or behind the listener is accomplished by reflections off of the shoulders and upper torso, the acoustical shadowing effect of the head, and the reflections due to the small ridges in the outer ear (called the pinna). These three effects can be modeled on the digital representation of the source signal. If the end result of the sound material can be assumed to be listened to on headphones (where the sound is directly 'injected' into the ear canal), the pinna-head-torso system can be simulated to reproduce most of the cues that will accomplish the 3D localization. The two methods used in this project were to use mathematical models of the filtering due to the head, torso and pinna (derived from the DAFX textbook [1]), and to use the recordings of the impulse responses from the MIT KEMAR setup [2]. The MATLAB file `HRTF.m` is called with the desired angles, as well as a string which tells it which method to use to accomplish the HRTF.

5.1.1 HRTF Modeling

The Head Related Transfer Function (HRTF) model is a set of filters and delays that simulate the natural effects of the head-torso-pinna system. An advantage of using this method to specialize is that it gives the possibility of applying continuous variation to the positions of the sound sources. The approach to the HRTF model used in this project is the same as the approach taken by Brown and Duda as outlined in DAFX [1]. The model is structurally divided into three parts: the Head Shadow and Interaural Time Differences, the Shoulder Echo, and the Pinna Reflection. A structural model of this system is shown in Figure 1.

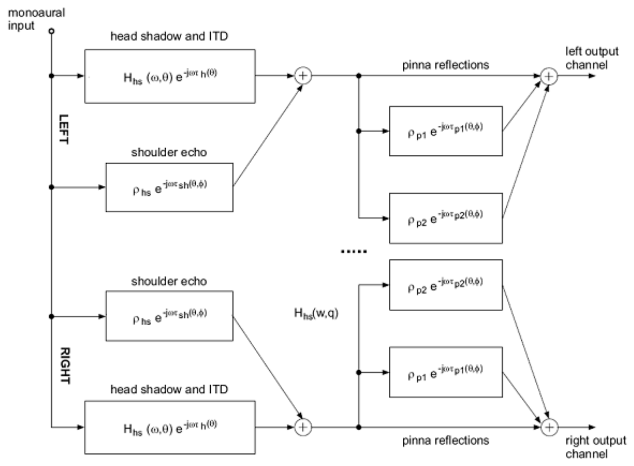
monoaural input
head shadow and ITD
$H_{hs}(\omega,\theta)\, e^{-j\omega\tau_h(\theta)}$
LEFT
shoulder echo
$\rho_{hs}\, e^{-j\omega\tau_{sh}(\theta,\phi)}$
pinna reflections
left output channel
$\rho_{p1}\, e^{-j\omega\tau_{p1}(\theta,\phi)}$
$\rho_{p2}\, e^{-j\omega\tau_{p2}(\theta,\phi)}$
.....
shoulder echo
$\rho_{hs}\, e^{-j\omega\tau_{sh}(\theta,\phi)}$
$H_{hs}(w,q)$
$\rho_{p2}\, e^{-j\omega\tau_{p2}(\theta,\phi)}$
$\rho_{p1}\, e^{-j\omega\tau_{p1}(\theta,\phi)}$
RIGHT
head shadow and ITD
$H_{hs}(\omega,\theta)\, e^{-j\omega\tau_h(\theta)}$
pinna reflections
right output channel

Figure 1: Structural model of the head-torso-pinna system.

The simulation of the Head Shadow and ITD is performed in the MATLAB file `hsfilter.m` (MATLAB code for this section in Appendix A), which was taken from the DAFX text [1]. The model begins by approximating the diffraction of the sound waves by the head, which can be represented by a first order continuous-time system. In other words, a pole-zero couple in the Laplace domain. A digital filter can be derived from this couple using bilinear transform, which results in the transfer function [1]:

$$H_{\text{hs}} = \frac{(\omega_0 + \alpha F_s) + (\omega_0 - \alpha F_s)z^{-1}}{(\omega_0 + F_s) + (\omega_0 - F_s)z^{-1}}$$

where α is a function of the inputted azimuth, θ, and is given by the relation[1]:

$$\alpha(\theta) = 1.05 + 0.95\cos\left(\frac{\theta}{150^\circ}180^\circ\right)$$

and ω_0 is the speed of sound divided by the effective radius of the head.
The ITD is obtained through an allpass filter with a group delay that is a function of the azimuth [1]:

$$\tau_{\text{h}}(\theta) = \begin{cases} -\frac{a}{c}\cos\theta & \text{if } 0 \le |\theta| < \frac{\pi}{2} \\ \frac{a}{c}\left(|\theta| - \frac{\pi}{2}\right) & \text{if } \frac{\pi}{2} \le |\theta| < \pi \end{cases}$$

In this method, the shoulder and torso effects are approximated by a singe echo. This is implemented as a delay that is proportional to the azimuth (θ) and angle of elevation (Φ). The relation for the delay [1], in seconds is:

$$\tau_{\text{h}} = 1.2\frac{180^\circ - \theta}{180^\circ}\left(1 - 0.00004\left((\phi - 80^\circ)\frac{180^\circ}{180^\circ + \theta}\right)^2\right) \text{ in ms ,}$$

The delay was performed by the MATLAB file `torso.m` (as shown in Appendix A).

Finally, the pinna reflections, which are the main cue for perception of elevation, are modeled by means of a tapped delay line. These echoes translate into notches in the frequency domain which have a position that is elevation dependant. The reflections were simulated in the file `pinna_reflections.m.` The formula for the time delay [1] (in seconds) is:

$$\tau_{pn} = A_n \cos(\theta/2)\sin(D_n(90^\circ - \phi)) + B_n$$

where the parameters are given in Table 1

n	ρ_{pn}	A_n[samples]	B_n[samples]	D_n
2	0.5	1	2	$\cong 1$
3	-1	5	4	$\cong 0.5$
4	0.5	5	7	$\cong 0.5$
5	-0.25	5	11	$\cong 0.5$
6	0.25	5	13	$\cong 0.5$

Table 1: Parameters for amplitude and time delay of pinna reflections [1].

Each of the three MATLAB files which simulate were run on each channel of the input sound source in the manner shown in Figure 1 above. To localize properly, the source is run through the head related transfer function twice for each position to place the source. Once with the azimuth and elevation angles and with the negative of the azimuth angle for the other channel.

5.1.2 Convolution with Impulse Response

The other method of localizing a signal is to convolve the input signal with a measured impulse response at the location of the desired output that already contains the head-torso-pinna effects. In order to accomplish this, a dummy upper torso can be constructed with appropriate dimensions and materials to get the same HRTF in the ear canals that humans do. Placing accurate microphones in the ear canal of the dummy and playing a wide frequency sweep at a number of locations around the dummy can be used to get an impulse response that corresponds to the HRTF. For this project, the KEMAR dummy recordings [2] were downloaded from MIT. The KEMAR files had a variety of stereo tracks corresponding to different angles for the azimuth and elevation. The desired angle (or the file closest to that angle) was chosen and the input sound file was convolved (using the MATLAB convolution function) with the KEMAR file matching that angle. Since convolution in the time domain is equivalent to multiplication in the frequency domain, convolving the sound file with an impulse response is equivalent to filtering it with the corresponding HRTF.

The convolution method has advantages and disadvantages compared to the mathematical HRTF method. One advantage is that the mathematical model uses some approximations that reduce its accuracy, the most noticeable being the shoulder reflection model of just one echo. On the other hand, a fundamental limitation to the Head Related Impulse Response method is the fact that the impulse responses can vary from person to person. The mathematical model can however be tweaked to accommodate different subjects by modifying the values in Table 1. As well, the model can select any angle of azimuth or elevation while the convolution method requires discrete recordings at each angle.

5.1.3 Rotating Source Demo

Both of these methods were successfully used on a mono source to create a stereo track that appeared to localize the sound source in various different locations somewhere within a 360^o sphere around the listener. Next, a demo that would demonstrate the ability to move the source around the listener's head and moved up and down at the same time was created. The effect was that the source was perceived to move in a spiral pattern from -90 to +90 degrees and back. In order to accomplish this effect a rotation frequency was chosen for the azimuth angle and the elevation angle as well as a number or angles per rotation. The sound file was then broken up into the appropriate number of blocks and the HRTF (either the mathematical model or the convolution method) was applied to each block. Since the HRTF includes delays, it resulted in the filtered block size being longer than the original. As such, the filtered blocks had to be overlap-added to create the output file. The MATLAB file `spin.m` (as shown in Appendix A) was used to call the HRTF model with the angle for the given block.

5.2 5.1 SURROUND UPMIXING

5.1 surround sound is a collection of five loudspeakers and a subwoofer (a larger speaker playing low frequency components at a higher power). Two speakers are placed on either side and in front of the listener, a third is placed directly front and center, and two more on either side in the rear. These are known as the left, right, center, surround left and surround right, respectively. The subwoofer is placed somewhere in front of the listener (typically to the center) but in theory should have no spatial distinction. A diagram presenting the standard layout of a 5.1 surround sound system is presented in Figure 2.

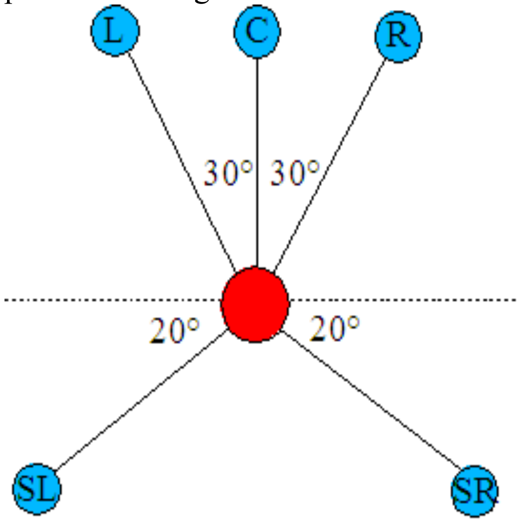

Figure 2: Standard 5.1 surround sound loudspeaker layout.

The purpose of this portion of the study was to implement a surround sound upmixer that would be capable of mixing a stereo audio track to 5.1 surround. The next section discussed Dolby Pro Logic II as an acceptable technology to attempt to recreate for this study.

5.2.1 Dolby Pro Logic II

An industry leader in surround sound audio encoders, decoders and systems is Dolby Laboratories Inc., whose technologies are prominently represented in the film industry. Dolby Pro Logic is a technology of Dolby Laboratories which is used to decode stereo audio into multichannel playback. Audio can be encoded in a two channel format that is intended specifically for multichannel decoding, or can simply be stereo audio which is then upmixed into multichannel playback.

The current technology used by Dolby Laboratories is their Pro Logic II, an improved version of the original Pro Logic. This technology is prominent in the multichannel audio industry and is a good standard for which to focus this study as there is information available from Dolby (as in [3] and [4]) which can be used as a guide for developing a 5.1 surround upmixer.

A basic visualization of the Pro Logic II decoder is presented in Figure 3.

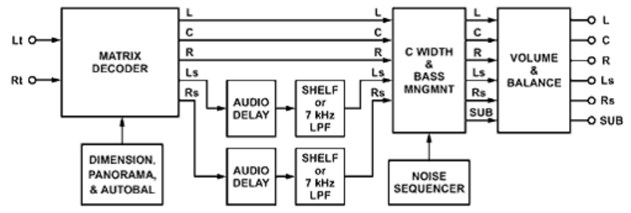

Figure 3: Basic Pro Logic II decoder system block diagram. [4]

It should be noted that Pro Logic decoders are primarily intended for decoding audio that was originally encoded in Pro Logic two channel format (typically used in VHS, DVD, TV and other video recordings), while the purpose of this study is to upmix a stereo audio track into surround sound. Pro Logic decoders also have decoding modes for such a purpose, so the resources in [3] and [4] provide insight into this. We will therefore make the distinction between two modes of the decoder: *Pro Logic* mode and *Music* mode, parallel to the distinctions made by Dolby (as well as a third *Movie* mode, which was not researched for this study).[4]

5.2.2 Implementation of a 5.1 Upmixer

As can be seen in Figure 3, a basic surround upmixer can be implemented with the following components: gain matrix decoder (which can include autopanning); delay effects; filtering effects; and subwoofer isolation. Such a technique was carried out in this study and is discussed in detail in the following sections.

The MATLAB source for this upmixing is presented in Appendix A in the function `upmix5.m`., while a block diagram depicting the 5.1 upmixer as it was implemented in this study is shown in Figure 4.

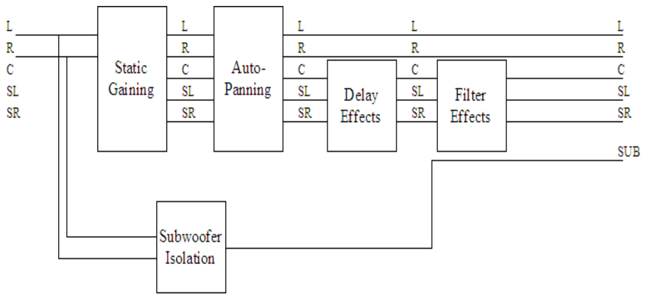

Figure 4: 5.1 surround upmixer system block diagram

5.2.3 Static Gain Decoding

The simplest method of decoding to five upmixed channels, and the starting point for more intelligent methods, is to mix the left and right stereo channels with a matrix of mixing gains. Each upmixed channel is a distinct mix of the two stereo channels.

The gain matrix is fairly intuitive with the exception of the surround (rear) channels. The left channel is the ungained left stereo channel, the right channel is similarly the ungained right stereo channel, and the center channel is an equal mix of the left and right stereo channels gained down 6dB. The value of 6dB differs from the Pro Logic gaining of 3dB specified in [3] and [4], as this value was recommended by Yan Li (aforementioned) in order to preserve the distinction between left and right, i.e. if the center channel were too loud, nearly everything would come from center and thus the surround effect would be lost.

The surround channels required a phase shift of 90 degrees for the left stereo and -90 degrees for the right stereo before mixing, and the gains were weighted to the left for left surround channel and to the right for right surround channel.

The resulting static gain matrix used in `upmix5.m` (as shown in Appendix A) is thus:

$$[l\ r\ c\ sl\ sr] = [l\ r]\begin{bmatrix} 1 & 0 & 0.5012 & j0.8165 & -j0.5774 \\ 0 & 1 & 0.5012 & j0.5774 & -j0.8165 \end{bmatrix}$$

5.2.4 Autopanning

An extension of the static gain matrix is to use a technique called autopanning [4]. Autopanning refers to the detection of the dominant channel (after undergoing upmixing as described in the previous section) and the subsequent panning to that channel. For example, if the right channel is dominant (having the largest RMS), then the right channel will be gained up while the remaining four channels will be gained down. This gaining should occur such that the overall energy (RMS) is preserved. Also, autopanning is only useful as a dynamic gaining technique: that is, the channel that is gained up at a given time should be the channel that is dominant at that time.

According to [4], Dolby Pro Logic applies autopanning effects in *Pro Logic* mode but does not in *Music* mode; therefore the same distinction was made in this implementation. The reasoning behind excluding *Music* mode is that generally the vocals are placed in the centre of a stereo track and therefore panning away from center can give undesirable effects. This was indeed confirmed in that autopanning seemed to make the vocals bounce back and forth, as will be discussed in the Results section (6.2).

The MATLAB source for this autopanning is presented in Appendix A in the function `dyn_5UpmixGains.m`, (as shown in Appendix A) which is called by `upmix5.m` for 20 ms blocks while in *Pro Logic* mode.

Seperate autopanning for sub-bands (i.e. Low, mid, and high frequency ranges) is more useful than for the entire frequency spectrum as a whole, as mentioned by Li. However, this requires more overhead and complexity and was unfortunately not implemented in this study, which will be discussed in the Discussion section (7.2).

5.2.5 Delay Effects

An important effect used in multichannel playback is time delaying. If the same signal arrives at the ear with a large enough delay, interaural time difference principals and the precedence effect play a role in the perception of the source location. In fact, a commonly used technique for amplification in theatres without distorting source location is to place loudspeakers behind the audience with an appropriate delay. The same principle is applied in surround sound in that the surround speakers are delayed in order to keep the source location in front of the listener, as for most cases this is ideal. As with autopanning, [4] indicates that a delay effect of 20 ms is used for Pro Logic mode only, while no delay is used for music mode. In music mode, the desired effect is to have the sound arrive at the listener's ear at the same time, giving an effect of coincident arrival.

On the recommendation of Li, a 5ms delay was applied to the center channel. This delay was applied in both upmixing modes.

5.2.6 Filtering Effects

In addition to delays, the surround channels also undergo filtering effects. In Pro Logic mode, the surround channels undergo lowpass filtering (at a cutoff of 7kHz). In Music mode, a shelving filter is used instead of a lowpass, which results in a more natural sound, emulating the absorption and reflection of high frequencies by a room's walls. [4]

5.2.7 Subwoofer Isolation

The above sections describe how to upmix each of five channels for surround sound, but have had no impact on the subwoofer channel. According to [4], the subwoofer channel is isolated from each of the five channels dynamically. As there was no more detail available as to how this is accomplished, this implementation simply isolated the subwoofer from the original stereo channel, on recommendation of Li.

The subwoofer channel was obtained by filtering both stereo channels with a lowpass filter of cutoff 300 Hz (which Li suggested as a good frequency to use), and then mixing them together, with a gain of 3 dB to equalize the overall sound.

5.3 SIMULATED 5.1 SURROUND SOUND WITH HEADPHONES

The final aspect of this study was to combine the 3D spatialization with headphones and the 5.1 surround upmixer to create a simulated surround with headphones effect. This effect was achieved by first upmixing a stereo audio file into 5.1 surround sound, and to then downmix with spatialization effects back to binaural audio for headphones.

The upmixing was carried out with the `upmix5.m` function given in Appendix A. This upmixing resulted in 6 monaural channels for each of the five louspeakers and the subwoofer. Each monaural channel, except the subwoofer, then underwent a HRTF for its spatial location (as depicted in Figure 3, with a zero degree elevation, i.e. level, with $\varphi = 0$). Each of the results gives the effect of an isolated loudspeaker in a 5.1 setup.

Because a HRTF implies binaural output, or a signal for both the left and right ear, this results in five pairs of signals, with each pair having a left and right component. The downmixed left binaural signal can be obtained simply by mixing the five individual left components together (and similarly for the right channel by mixing each of the five right components). The mixing is without gains as relative gaining is accomplished by the upmixer.

The subwoofer was excluded from spatialization as ideally the subwoofer has no spatial location. Although in a real surround system the subwoofer is physically placed somewhere, we found that a better effect was attained by not spatializing the subwoofer but instead by using equally gained (0dB) subwoofer signals in each the left and right binaural signal.

The simulated surround with headphones functionality is encapsulated in the MATLAB function `spatialize_5point.m` in Appendix A. A block diagram of the system is given in Figure 5.

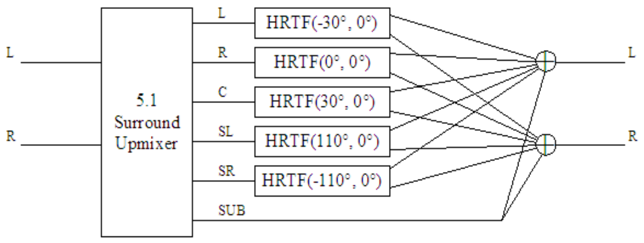

Figure 5: Simulated 5.1 Surround with Headphones system block diagram

6 RESULTS

6.1 3D WITH HEADPHONES

The final head-related transfer function is given in Appendix A in the MATLAB function `hrtf.m`, which calls either `hrtf_model.m` or `hrtf_kemar.m` for the head/torso/pinna model or for MIT's KEMAR dummy HRIR [2], respectively. Applying either of these HRTFs to a monaural track gives a convincing effect of 3D spatialization. In the authors' opinions it is difficult to discern between these two methods and to subjectively decide which, if either, is better.

The MATLAB function `spin.m` in Appendix A creates the effect of a moving source location purely for the purpose of demonstrating the spatialization effect. The effect is that the source location is moving in a spiral trajectory from a negative elevation (below the head) to a 90 degree elevation (directly above the head) in a clockwise manner, repeating for the duration of the input. The result is also very convincing.

Table 2 contains the resulting binaural audio files for a few static test positions, as well as for the effect given in `spin.m` (as shown in Appendix A), using both HRTF methods.

θ (to central plane)	**φ (elevation)**	**HRTF Method**	
		KEMAR	**MODEL**
0	0	Toms_diner_kemar_0_0.wav	Toms_diner_model_0_0.wav
0	50	Toms_diner_kemar_0_50.wav	Toms_diner_model_0_50.wav
30	20	Toms_diner_kemar_30_20.wav	Toms_diner_model_30_20.wav
140	0	Toms_diner_kemar_140_0.wav	Toms_diner_model_140_0.wav
-50	0	Toms_diner_kemar_-50_0.wav	Toms_diner_model_-50_0.wav
Spinning Effect		Toms_diner_spin_kemar.wav	Toms_diner_spin_model.wav

Table 2: Binaural Audio for 3D with headphones

6.2 5.1 SURROUND UPMIXING

The 5.1 surround upmixer (Appendix A, `upmixer.m`) reproduces the stereo effects quite accurately.

The *Pro Logic* upmixing mode is very good at isolating single source locations and autopanning to that location. Unfortunately, stereo recorded audio with vocals becomes very sensitive to this and the centered stereo vocals tend to move around with the autopanning. A solution to this is discussed in the Discussion section (7.2).

The *Music* upmixing mode is very good at giving a better sound that simple stereo. The removal of autopanning as well as the shelving filter provide a much more natural sound.

The subwoofer channel is very accurate to what would be a true subwoofer channel for a surround system; the 300Hz cutoff frequency is apparently a very good one for this isolation.

Each of the six 5.1 upmixed channels for an input are given in Table 3 for both upmixing modes.

Channel	**Mode**	
	Music	**Pro Logic**
Source	Whole Lotta Love.wav	
L	Whole Lotta Love_music_l_upmixed.wav	Whole Lotta Love_proLogic_l_upmixed.wav
R	Whole Lotta Love_music_r_upmixed.wav	Whole Lotta Love_proLogic_r_upmixed.wav
C	Whole Lotta Love_music_c_upmixed.wav	Whole Lotta Love_proLogic_c_upmixed.wav
SL	Whole Lotta Love_music_sl_upmixed.wav	Whole Lotta Love_proLogic_sl_upmixed.wav
SR	Whole Lotta Love_music_sr_upmixed.wav	Whole Lotta Love_proLogic_sr_upmixed.wav
Sub	Whole Lotta Love_music_sub_upmixed.wav	Whole Lotta Love_proLogic_sub_upmixed.wav

Table 3: Source and 5.1 Upmixed channels for Pro Logic and Music modes

To better demonstrate the upmixing, the following section presents spatialized surround sound with headphones.

6.3 SIMULATED 5.1 SURROUND WITH HEADPHONES

The simulated 5.1 surround with headphones MATLAB function `spatialize_5point.m` in Appendix A gives a convincing effect of spatialized multichannel playback.

To demonstrate the effect, an audio recording with dramatic stereo effects was used as input. *Led Zepplin*'s "Whole Lotta Love", which was used by radio broadcasters in the early days of

FM radio to demonstrate the effects of stereo broadcasting, was chosen as this input. Table 4 presents the resulting simulated 5.1 surround with headphones for the first 40 seconds of "Whole Lotta Love", using both *Pro Logic* and *Music* modes, each for both HRTF methods.

HRTF Method	Mode	
	Music	Pro Logic
Source (no effect)	Whole Lotta Love.wav	
KEMAR	Whole Lotta Love_kemar_music_surround.wav	Whole Lotta Love_kemar_proLogic_surround.wav
Model	Whole Lotta Love_model_music_surround.wav	Whole Lotta Love_model_proLogic_surround.wav

Table 4: Simulated 5.1 Surround with Headphones for various modes/methods

As was mentioned before, Pro Logic mode uses autopanning which can have undesirable effects on music with centered vocals. This is confirmed in the Pro Logic sound results, as we hear the vocals "jump" back and forth from left to right, as one side becomes dominant over another.

Listening to a single surround channel after spatialization gives the effect of listening to the corresponding "isolated loudspeaker". Table 5 presents each of the six 5.1 channels with the appropriate spatialization effects.

Channel	Isolated Spatialized Effect
Source (no effect)	
L	Whole Lotta Love_kemar_music_l.wav
R	Whole Lotta Love_kemar_music_r.wav
C	Whole Lotta Love_kemar_music_c.wav
SL	Whole Lotta Love_kemar_music_sl.wav
SR	Whole Lotta Love_kemar_music_sr.wav
Sub	Whole Lotta Love_kemar_music_sub.wav

Table 5: Isolated 5.1 Surround Channels spatialized for headphones using Music upmixing mode and measured HRIRs[2]

The simulated 5.1 surround with headphones effect was applied to two additional inputs: another clip of *Led Zepplin*'s "Whole Lotta Love", and a clip of *The Barenaked Ladies*' "If I Had a Million Dollars". The results are presented in Table 6.

Source	Simulated Surround Effect
Barenaked Ladies' "If I Had a Million Dollars"	bnl_clip_surround.wav
Led Zepplin's "Whole Lotta Love" (clip 2)	Whole Lotta Love2_surround.wav

Table 6: Simulated 5.1 Surround with Headphones using Music upmixing mode and measured HRIRs[2]

7 DISCUSSION

7.1 3D WITH HEADPHONES

Both the mathematical model and the HRIR convolution methods of localizing mono sources in headphones could successfully 'place' the source at the desired angle. Each of the methods had their advantages and disadvantages in implementation, but the end result seemed to be reasonably accurate for both. The mathematical HRIR model had some approximations that could be improved upon as well as tweaks to parameters to improve the result. The shoulder/torso reflection stood out as something that could be improved upon. In it's current implementation (as seen in DAFX [1]) simulates the reflections as a single echo. This could be improved by a more complicated reflections model. The HRIR convolution method had the advantage of not making physical approximations, but rather measuring the exact impulse response for the appropriate angle. However, this method results in discrete locations for all the angles, and thus a continuous sweep would not be possible. In its current form, if an angle is not present in a recorded impulse response, the closest one must be used. One method of improving this would be to interpolate the impulse response between surrounding closest impulse responses that are present.

7.2 5.1 SURROUND UPMIXING

The Dolby Pro Logic method of decoding 5.1 surround sound was used as a guideline to perform the upmixing of a stereo track. Gains, phase shifts, delays, and filtering were used in conjunction to separate the two stereo channels into 5 discrete tracks, as well as a low frequency 'subwoofer simulator' that would later have the HRTF applied to them. The resultant tracks at first sounded a bit 'tinny' and lacking in the low frequencies. This was due to crosstalk cancellation of the lower frequencies and phase shifts providing destructive interference due to the delays. Using a low pass filter to clone the low frequencies and then adding them back in after the upmixing fixed this problem.

The autopanning functionality in the Pro Logic mode of operation is used to keep the focus on the direction that the specific part of dialogue/music/ambient sound/etc should be coming from. The best way to implement this would be to break this down into sub bands and then pan that sub band of the frequency spectrum to the appropriate location. This could be used to filter out specific instruments in a music recording, as is done with frequency band keying in DAW software, to separate them in the ambient environment.

7.3 SIMULATED 5.1 SURROUND WITH HEADPHONES

After the upmixing was performed to separate the stereo track to its 5 remote locations, the HRTF was applied with the appropriate angles and the signals were downmixed back to stereo. This technique was successful in creating a more specialized version of the input while keeping the left/right localization from the source material. Using the Dolby Pro Logic 'music' mode sounded quite good when all the steps were completed, while the 'Pro Logic II' mode had some issues with the autopanning of the vocals. But that is to be expected since the vocals are typically meant to come from the center channel. The surround with headphones technique could be relatively easily accomplished by using the binaural to transaural transform as outlined in DAFX [1]. However, to accomplish this would require knowledge of the dimensions of the room, loudspeaker location, and listener location. As well, it could only work to its full effect for a single location in the room.

8 7 CONCLUSIONS

The 3D in headphones and simulated 5-speaker surround sound were successfully modeled using different methods of applying the Head Related Transfer Function and different upmixing applications inspired by the Dolby Pro Logic decoder. Using the two methods of HRTF, a point source was able to be simulated in a sphere around the listener's head which could clearly be heard as coming from an angle in front/behind and above/below. This effect was very noticeable in the demo of rotational panning.

The surround sound upmixing using some of the Dolby Pro Logic techniques was used to successfully create 5 discrete channels. The use of a 'subwoofer simulator' channel successfully alleviated the problem of loosing low frequencies to delay based interferences. Potential extensions to the material learned in the course of working on this project were also found. These include more exploration into autopanning, such as subband separation panning, more experimentation with shelving/filtering of the surround channels, and modifying some of the delays.

The techniques used in the HRTF section were used on the resultant surround channels and then downmixed to create a spatialized version of the input sound source. This project provided the opportunity to learn many of the details about how surround sound encoding and decoding works along with 3D in headphones. As well, many possible expansions for use in a lengthier, more in-depth project were discovered.

Appendix A: MATLAB Source Listing

```
%--------------------------------------------------------------------
%   hrtf
%
%   Applies a head-related transfer function to an input
%   monoaural audio track, using either measured HRIR (see
%   get_hrir.m for details), or using head/torso/pinna model,
%   depending on input mode specified
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           theta: angle (in degrees) in plane around head
%           phi: angle (in degreees) of elevation
%           mode: hrtf mode: 'kemar' for hrir, 'model' for model.
%   Returns:
%           y: [r l] stereo track in 3D for headphones
%--------------------------------------------------------------------
function [y] = hrtf(x, fs, theta, phi, mode);

%switch on mode, calling appropriate hrtf function
if ( strcmp(mode, 'kemar') )
    y = hrtf_kemar(x, fs, theta, phi);
elseif ( strcmp(mode, 'model') )
    y = hrtf_model(x, fs, theta, phi);
else
    y = x; %if no match, return input
end;
```

```
%---------------------------------------------------------------------
%   hrtf_model
%
%   Applies a head-related transfer function to an input
%   monoaural audio track using a head/torso/pinna model
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           theta: angle (in degrees) in plane around head
%           phi: angle (in degreees) of elevation
%   Returns:
%           y: [r l] stereo track in 3D for headphones
%---------------------------------------------------------------------
function [y] = hrtf_model(x, fs, theta, phi)

% 180 degrees gives a divide by zero, so if exactly 180 compensate
% for this by approximating with another (close) value
if (abs(theta) == 180)
    theta = 179 * sign(theta);
end

% Apply Head Shadowing to input (-theta for left ear)
r_hs = hsfilter(theta, fs, x);
l_hs = hsfilter(-theta, fs, x);

% Apply a torso delay to input (-theta for left ear)
r_sh = torso(x, fs, theta, phi);
l_sh = torso(x, fs, -theta, phi);

% Sum the head shadowed/torso delayed signals: This is the
% signal that makes it to the outer ear (pre pinna)
r_on_pinna = zeros(1, max(length(r_hs),length(r_sh)));
r_on_pinna(1:length(r_hs)) = r_on_pinna(1:length(r_hs)) + r_hs';
r_on_pinna(1:length(r_sh)) = r_on_pinna(1:length(r_sh)) + r_sh;

l_on_pinna = zeros(1, max(length(l_hs),length(l_sh)));
l_on_pinna(1:length(l_hs)) = l_on_pinna(1:length(l_hs)) + l_hs';
l_on_pinna(1:length(l_sh)) = l_on_pinna(1:length(l_sh)) + l_sh;

% Apply pinna reflections to the prepinna signals (-theta for left ear)
r = pinna_reflections(r_on_pinna, fs, theta, phi);
l = pinna_reflections(l_on_pinna, fs, -theta, phi);

% Pad shorter signal with zeros to make both same length
if ( length(r) < length(l) )
    r = [r zeros(1,length(l)-length(r))];
else
    l = [l zeros(1,length(r)-length(l))];
end;

% return final headphone stereo track
y = [r' l'];
```

```
function [output] = hsfilter(theta, Fs, input)
% hsfilter(theta, Fs, input)
%
% filters the input signal according to head shadowing
% theta is the angle with the frontal plane
% Fs is the sample rate

theta = theta + 90;
theta0 = 150 ;alfa_min = 0.05 ;
c = 334;  % speed of sound
a = 0.08; % radius of head
w0 = c/a;
alfa = 1+ alfa_min/2 + (1- alfa_min/2)* cos(theta/ theta0* pi) ;

B = [(alfa+w0/Fs)/(1+w0/Fs), (-alfa+w0/Fs)/(1+w0/Fs)] ;
    % numerator of Transfer Function
A = [1, -(1-w0/Fs)/(1+w0/Fs)] ;
    % denominator of Transfer Function
if (abs(theta) < 90)
 gdelay = - Fs/w0*(cos(theta*pi/180) - 1)  ;
else
 gdelay = Fs/w0*((abs(theta) - 90)*pi/180 + 1);
end;
a = (1 - gdelay)/(1 + gdelay);
    % allpass filter coefficient
out_magn = filter(B, A, input);
output = filter([a, 1],[1, a], out_magn);
```

```
%--------------------------------------------------------------------
%   torso
%
%   Applies a delay to input signal based on a single
%   shoulder reflection for given position, as a part
%   of a head related transfer function
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           theta: angle (in degrees) in plane around head
%           phi: angle (in degreees) of elevation
%   Returns:
%           y: delayed input, zero padded
%--------------------------------------------------------------------
function y = torso(x, fs, theta, phi)

% Calculate delay in ms
T_sh = 1.2*(180 - theta)/180 * (1 - 0.00004*((phi - 80)*180/(180 + theta)));

% Delay and return input signal
delay_sh = round(T_sh/1000*fs);
y = zeros(1, length(x) + delay_sh);
y(delay_sh+1:length(y)) = x;
```

```
%-------------------------------------------------------------------
%   pinna_reflections
%
%   Applies a series of delays based on modelling the pinna
%   reflections for a given position.
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           theta: angle (in degrees) in plane around head
%           phi: angle (in degreees) of elevation
%   Returns:
%           y: sum of input plus 5 delayed versions of input
%-------------------------------------------------------------------
function [y] = pinna_reflections(x, fs, theta, phi)

% compute angles in radians
theta = theta*pi/180;
phi = phi*pi/180;

% define constants
ro = [ 1 0.5 -1 0.5 -0.25 0.25];
A = [1 5 5 5 5];
B = [2 4 7 11 13];
D = [1 0.5 0.5 0.5 0.5];

% memory allocation
T_p = [0 0 0 0 0 0];
delay_p = [0 0 0 0 0 0];

% calculate 5 disting delays
for I = 2:6
    T_p(I) = A(I-1)*cos(theta/2)*sin(D(I-1)*(pi/2-phi)) + B(I-1);
    delay_p(I) = round(T_p(I)/1000 * fs);
end

% sum and return input plus 5 delayed copies
y = zeros(1,length(x)+max(delay_p));
temp_y = zeros(6, length(x)+max(delay_p));
for I = 1:6
    temp_y(I, (delay_p(I)+1):(delay_p(I)+length(x)) ) = x*ro(I);
    y = y + temp_y(I,:);
end
```

```
%--------------------------------------------------------------------
%   hrtf_kemar
%
%   Applies a head-related transfer function to an input
%   monoaural audio track using a HRIR obtained from
%   MIT media lab's measurements, as documented in get_hrir.m
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           theta: angle (in degrees) in plane around head
%           phi: angle (in degreees) of elevation
%   Returns:
%           y: [r l] stereo track in 3D for headphones
%--------------------------------------------------------------------
function [y] = hrtf_kemar(x, fs, theta, phi);

% get closest matching HRIR
h = get_hrir(theta, phi);

% convolve and return input with hrir
y(:,1) = conv(x, h(:,1));
y(:,2) = conv(x, h(:,2));
```

```
%-------------------------------------------------------------------
%   get_hrir
%
%   Returns a (stereo) head related impulse response which matches
%   closest the input angles. The IR is obtained from an MIT media
%   lab's HRTF Measurements of a KEMAR Dummy-Head Microphone which
%   can be found at http://sound.media.mit.edu/KEMAR.html
%
%   Input parameters
%           theta: angle (in degrees) in plane around head
%           phi: angle (in degreees) of elevation
%   Returns:
%           h: HRIR matching closest theta and phi
%-------------------------------------------------------------------
function [h] = get_hrir(theta, phi);

% define available locations:
% elevations: available phi values
elevations = [-40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90];
% theta_increments: spacing between thetas for corresponding phi
theta_increments =   [6.5 6 5 5 5 5 5 6 6.5 8 10 15 30 360];

% Determine which phi is best match based on search of
% values in elevations; record the index
diff = Inf;
elev_match = 1;
for I = 1:length(elevations)
    this_diff = abs(elevations(I)-phi);
    if ( this_diff <= diff )
        diff = this_diff;
        elev_match = I;
    end
end

% Calculate best theta match knowing the linear spacing
% between each theta
num_incr = round(abs(theta)/theta_increments(elev_match));
theta_match = floor(num_incr*theta_increments(elev_match));
while (theta_match>180)
    num_incr = num_incr-1;
    theta_match = floor(num_incr*theta_increments(elev_match));
end

% concatonate strings to make appropriate file name
% based on the HRIR file naming convention of
% H*e&&&a.wav in dir hrirs
% (*=phi in min digits; &&& = three digit phi, zero padded)

filename = strcat( 'hrirs\H', int2str(elevations(elev_match)) );
filename = strcat( filename, 'e');

tempstr = int2str(theta_match);
needed_zeros = 3-length(tempstr);
if (needed_zeros > 0)
    for I = 1:needed_zeros
        tempstr = strcat( '0',tempstr );
    end
end

filename = strcat( filename, tempstr );
filename = strcat( filename, 'a.wav');

% now read file with 'filename' into memory
[x fs nbits] = wavread(filename);

% if theta was negative, we are to the left of the frontal plane.
% only thetas to right (positive: 0 to 180) of plane are measured,
% but we can use these for left by swapping the l/r hrirs for the
% absolute value of theta (ie left HRTF is same as right HRTF but
% swapping stereo tracks)
h = zeros(size(x));
if (theta < 0)
    h(:,1) = x(:,2);
    h(:,2) = x(:,1);
else
    h(:,1) = x(:,1);
    h(:,2) = x(:,2);
end
```

```
%-------------------------------------------------------------------
%   spin
%
%   Applies a 3D with headphones spatialization effect
%   of the source audio spinning around the listener. The spinning
%   is applied to both the angle relative to the frontal plane (theta)
%   and the angle of elevation (phi)
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           theta_period: period of theta rotation
%           phi_period: period of phi rotation
%           mode: hrtf mode: 'kemar' for hrir, 'model' for model.
%   Returns:
%           y: [r l] stereo track in 3D for headphones
%-------------------------------------------------------------------
function y = spin (x, fs, theta_period, phi_period, hrtfMode)

% set constants
%theta_rotation_period = period;         %rotation around the head in seconds
%phi_rotation_period = 10;               %elevation change (up/down) in seconds
N_theta = 20;                           %number of blocks per rotation
%N_phi = N_theta;                        %number of blocks per rotation

%calculate the block size
samples_per_block = fs*theta_period/N_theta;

%Now we can calc blocks/rotation for phi with fixed block size
N_phi = fs*phi_period/samples_per_block %blocks per rotation

%create the vector of angles to be run through during the rotation period
thetas = 360*linspace(0,(N_theta-1)/N_theta,N_theta)-180;
phis = 180*linspace(0,(N_phi-1)/N_phi,N_phi)-90;

%initialize y
y=[];

%initialize starting block to 1
blockStart = 1;

%iterate through each block
for block_num = 1:floor(length(x)/samples_per_block)

    %determine theta and phi indeces
    I_theta = 1+mod(block_num, N_theta);
    I_phi = 1+mod(block_num, N_phi);        %round((N_phi+1)/2);

    %calc block length
    blockLength = min([samples_per_block (length(x) -
block_num*samples_per_block)]);

    %apply hrtf to this block for this theta and phi
    filtered_block = hrtf(x(blockStart:blockStart+blockLength-1), fs,
thetas(I_theta), phis(I_phi), hrtfMode);
```

```
    %crossfade in new filtered block to match previous overrun (don't do on first
iteration)
    if block_num ~= 1
        cross_up = linspace(0,1,overrun_length);
        filtered_block(1:overrun_length) =
filtered_block(1:overrun_length).*cross_up;
    end

    % get sizes of vectors
    y_size = size(y);
    filtered_block_size = size(filtered_block);
    overrun_length = filtered_block_size(1) - (blockLength-1);

    %crossfade out overrun (from delay)
    cross_down = linspace(1,0,overrun_length);
    filtered_block(blockLength:filtered_block_size(1)) =
filtered_block(blockLength:filtered_block_size(1)).*cross_down;

    %add filtered block to running total
    y = [y' zeros(blockStart-1+filtered_block_size(1)-y_size(1), 2)']' +
[zeros(blockStart-1, 2)' filtered_block']';         %[y; temp];

    % increment starting sample for next block
    blockStart = blockStart+samples_per_block;
end
```

```
%--------------------------------------------------------------------
%   upmix5
%
%   Upmixes input stereo track to 5.1 surround
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           mode: upMix mode: 'proLogic' or 'music'
%   Returns:
%           l: left channel
%           r: right channel
%           c: center channel
%           sl: surround left channel
%           sr: surround right channel
%           sub: subwoofer channel
%--------------------------------------------------------------------
function [l, r, c, sl, sr, sub] = upmix5(x, fs, mode)

%--------------------------------------------------------------
% DEFINE CONSTANTS, SIGNALS
%--------------------------------------------------------------

% Mixing gains for left and right stereo tracks for
% each of the six upmixed channels
gains = [1 0 10^(-10/20) 0.8165 0.5774 10^(3/20);
         0 1 10^(-10/20) 0.5774 0.8165 10^(3/20)];

% delays for each of the six upmixed channels
delays = [0 0 5e-3 20e-3 20e-3 0];

%get stereo channels
stereo_left = x(:,1);
stereo_right = x(:,2);

%--------------------------------------------------------------
% LOW PASS FILTERING (SIMULATING SUBWOOFER)
%--------------------------------------------------------------

% get filter coeffs
% [N_sub, Wn_sub] = ellipord(1000/(fs/2), 1100/(fs/2), .1, 60);
% [b_sub, a_sub] = ellip(N_sub,.1,60,Wn_sub,'low');
[b_sub, a_sub] = cheby1(5, .5, 300/22050, 'low');

%filter each left and right tracks
subl = filter(b_sub, a_sub, stereo_left);
subr = filter(b_sub, a_sub, stereo_right);



%--------------------------------------------------------------
% MIX STEREO TO SURROUND WITH DEFINED GAINS
%--------------------------------------------------------------

% front 3 channels are simple gains
l = gains(1,1)*stereo_left + gains(2,1)*stereo_right;
```

```
r = gains(1,2)*stereo_left + gains(2,2)*stereo_right;
c = gains(1,3)*stereo_left + gains(2,3)*stereo_right;

% surround channels require phase shifts
% shift stereo tracks
left_p90 = real((hilbert(stereo_left) - stereo_left)/i);
right_n90 = real((hilbert(-1*stereo_right) - (-1*stereo_right))/i);
%mix shifted tracks
sl_pre_shelf = gains(1,4)*left_p90 + gains(2,4)*right_n90;
sr_pre_shelf = gains(1,5)*left_p90 + gains(2,5)*right_n90;

% sub is simple gaining
sub = gains(1,6)*subl + gains(2,6)*subr;

% clear tracks from memory that are no longer needed
clear stereo_left; clear stereo_right;
clear left_p90; clear right_p90;
clear subl; clear subr;



%---------------------------------------------------------------
% PAN TO STRONGEST SIGNAL DYNAMICALLY (PROLOGIC MODE)
%---------------------------------------------------------------
if (strcmp(mode, 'proLogic'))
    % Iterate through in 20 ms blocks

    %find block constants
    blockSize = round(20e-3*fs);
    numBlocks = ceil(length(x)/blockSize);

    for I = 1:numBlocks

        %find start/end samples
        blockStart = (I-1)*blockSize+1;
        blockEnd = blockStart+min(blockSize-1,length(x)-blockStart);

        %find dynamic gains for this 20 ms block
        [upmixGains iMaxRMS] = dyn_5UpmixGains( l(blockStart:blockEnd),
r(blockStart:blockEnd), c(blockStart:blockEnd), sl_pre_shelf(blockStart:blockEnd),
sr_pre_shelf(blockStart:blockEnd) );
        focus(I, 1:2) = [iMaxRMS(1) length(iMaxRMS)];

        %Apply Panning to this block by gaining
        l(blockStart:blockEnd) = l(blockStart:blockEnd)*upmixGains(1);
        r(blockStart:blockEnd) = r(blockStart:blockEnd)*upmixGains(2);
        c(blockStart:blockEnd) = c(blockStart:blockEnd)*upmixGains(3);
        sl_pre_shelf(blockStart:blockEnd) =
sl_pre_shelf(blockStart:blockEnd)*upmixGains(4);
        sr_pre_shelf(blockStart:blockEnd) =
sr_pre_shelf(blockStart:blockEnd)*upmixGains(5);

    end;
    % %TEST CODE
    % L_STRONG =  find( focus(:, 1) == 1 )
    % R_STRONG =  find( focus(:, 1) == 2 )
    % C_STRONG =  find( focus(:, 1) == 3 )
    % SL_STRONG = find( focus(:, 1) == 4 )
    % SR_STRONG = find( focus(:, 1) == 5 )
    % MULTI_STRONG = find( focus(:, 2) > 1 )

    % clear uneeded signals
    clear focus;
end;


%--------------------------------------------------------------
% APPLY DELAYS TO CHANNELS
%--------------------------------------------------------------
l = delay(l, fs, delays(1));
r = delay(r, fs, delays(2));
c = delay(c, fs, delays(3));
if (strcmp(mode, 'proLogic'))
    sl_pre_shelf = delay(sl_pre_shelf, fs, delays(4));
    sr_pre_shelf = delay(sr_pre_shelf, fs, delays(5));
end;
sub = delay(sub, fs, delays(6));


%--------------------------------------------------------------
% FILTERING FOR SURROUND CHANNELS
%--------------------------------------------------------------
if (strcmp(mode, 'proLogic'))
    %--------------------------------------------------------------
    % LOW PASS FILTERING FOR SURROUND CHANNELS (PROLOGIC MODE)
    %--------------------------------------------------------------
    [N_lp, Wn_lp] = elliprod(7e3/(fs/2), 7.5e3/(fs/2), .1, 60);
    [b_lp,a_lp] = ellip(N_lp,.1,60,Wn_lp,'low');
    sl = filter(b_lp, a_lp, sl_pre_shelf);
    sr = filter(b_lp, a_lp, sr_pre_shelf);
elseif (strcmp(mode, 'music'))
    %--------------------------------------------------------------
    % SHELVING FILTER FOR SURROUND CHANNELS (MUSIC MODE)
    %--------------------------------------------------------------
    [b_lp, a_lp] = shelf1(fs, 4e3, -20, 'hc');
    sl = filter(b_lp, a_lp, sl_pre_shelf);
    sr = filter(b_lp, a_lp, sr_pre_shelf);
end
```

```
%--------------------------------------------------------------------
%   dyn_5UpmixGains
%
%   Calculates gains for each of 5 surround channels such that
%   The dominant channel(s) is(are) gained up, while the remaining are
%   gained down, preserving the total energy of all 5 tracks.
%
%   Input parameters
%           l: left channel
%           r: right channel
%           c: center channel
%           sl: surround left channel
%           sr: surround right channel
%   Returns:
%           upmixGains: gains for each of 5 surround tracks
%           iMaxRMS: the indexes (relative to input order) of the
%               dominant channel(s)
%--------------------------------------------------------------------
function [ upmixGains, iMaxRMS ] = dyn_5UpmixGains ( l, r, c, sl, sr )

% factor for gains such that rms gains gain_up/gain_down = scaling_factor
scaling_factor = 2;

% calc rms's
rms = [ mean(l.^2) mean(r.^2) mean(c.^2) mean(sl.^2) mean(sr.^2) ];

% determine indeces of dominant(s) and weaker channels
iMaxRMS = find ( rms == max(rms) );
iNotMaxRMS = find ( rms < max(rms) );

% calc rms gains such that energy (total rms) is preserved
g = sym('g');
rmsGains( iMaxRMS ) = g;
rmsGains( iNotMaxRMS ) = g/scaling_factor;
g = solve( sum(rmsGains.*rms) - sum(rms) );

% find and return channel gains
upmixGains = eval(sqrt( eval(rmsGains) ));


% %TEST CODE
% origSumRMS = sum( [rms] )
%
% y_l = upmixGains(1)*l;
% y_r = upmixGains(2)*r;
% y_c = upmixGains(3)*c;
% y_s = upmixGains(4)*s;
%
% newRMS = [ mean(y_l.^2) mean(y_r.^2) mean(y_c.^2) mean(y_s.^2) mean(y_s.^2) ];
%
% newSumRMS = eval(sum( newRMS ))
```

```
%-----------------------------------------------------------------------
%   delay
%
%   Applies a delay to a vector
%
%   Input parameters
%           x: Input signal (monoaural)
%           fs: sample rate of x in samples per second
%           t: delay to apply in seconds
%   Returns:
%           y: delayed input
%-----------------------------------------------------------------------
function [y] = delay(x, fs, t)

% find delay in sampls
sample_delay = round(t*fs);

% delay input
y = [zeros(sample_delay, 1)' x']';
```

```
%--------------------------------------------------------------------
%   shelf1
%
%   First order shelving filter
%
%   Input parameters
%           fs: sample rate of x in samples per second
%           fc: cutoff freq in Hz
%           G: gain of desired shelved frequencies
%           type: 'hc' for highcut, currently no others
%   Returns:
%           [b, a]: filter coefficients
%--------------------------------------------------------------------
function [b, a] = shelf1(fs, fc, G, type)

% find K
K = tan(pi*fc/fs);

% for high cut
if (type == 'hc')

    %find constants
    Vo = 10^(G/20);
    Ho = Vo - 1;
    ac = (Vo*tan(pi*fc/fs) - 1)/(Vo*tan(pi*fc/fs) + 1);

    % find and return coefficients
    b = [ (2 + Ho*(1-ac)) (2*ac + Ho*(ac-1)) ];
    a = [ 2 2*ac ];
end;
```

```
%-------------------------------------------------------------------
%   spatialize_5point
%
%   Upmixes input stereo track to 5.1 surround and mixes all
%   channels with HRTF spatialization
%
%   Input parameters
%           x: Input signal
%           fs: sample rate of x in samples per second
%           hrtfMode: hrtf mode: 'kemar' for hrir, 'model' for model.
%           upMixMode: upMix mode: 'proLogic' or 'music'
%   Returns:
%           y: mixed 5.1 spatialized
%           l: left channel spatialized
%           r: right channel spatialized
%           c: center channel spatialized
%           sl: surround left channel spatialized
%           sr: surround right channel spatialized
%           sub: subwoofer channel
%-------------------------------------------------------------------
function [y, l, r, c, sl, sr, sub] = spatialize_5point(x, fs, hrtfMode, upMixMode)

%-------------------------------------------------------------
% DEFINE CONSTANTS
%-------------------------------------------------------------
theta = [-30 30 0 -110 110];
phi = [0 0 0 0 0];

%-------------------------------------------------------------
% UPMIX TO SURROUND SOUND
%-------------------------------------------------------------
[up_l up_r up_c up_sl up_sr sub] = upmix5(x, fs, upMixMode);
clear x;

%-------------------------------------------------------------
% SPATIALIZE EACH UPMIXED CHANNELS (not subwoofer)
%-------------------------------------------------------------
hrtf_up_l = hrtf(up_l, fs, theta(1), phi(1), hrtfMode);
clear up_l;
hrtf_up_r = hrtf(up_r, fs, theta(2), phi(2), hrtfMode);
clear up_r;
hrtf_up_c = hrtf(up_c, fs, theta(3), phi(3), hrtfMode);
clear up_c;
hrtf_up_sl = hrtf(up_sl, fs, theta(4), phi(4), hrtfMode);
clear up_sl;
hrtf_up_sr = hrtf(up_sr, fs, theta(5), phi(5), hrtfMode);
clear up_sr;

%-------------------------------------------------------------
% DOWNMIX TO STEREO
%-------------------------------------------------------------

% find lengths of each track
lengths = [ length(hrtf_up_l(:,1)) length(hrtf_up_r(:,1)) length(hrtf_up_c(:,1))
length(hrtf_up_sl(:,1)) length(hrtf_up_sr(:,1)) length(sub)];
max_size = max( lengths );
```

```
% memory allocation
y = zeros(max_size, 2);

% mix zero padded signals to output
y(1:lengths(1), :) = y(1:lengths(1), :) + hrtf_up_l;
l = hrtf_up_l;
clear hrtf_up_l;
y(1:lengths(2), :) = y(1:lengths(2), :) + hrtf_up_r;
r = hrtf_up_r;
clear hrtf_up_r;
y(1:lengths(3), :) = y(1:lengths(3), :) + hrtf_up_c;
c = hrtf_up_c;
clear hrtf_up_c;
y(1:lengths(4), :) = y(1:lengths(4), :) + hrtf_up_sl;
sl = hrtf_up_sl;
clear hrtf_up_sl;
y(1:lengths(5), :) = y(1:lengths(5), :) + hrtf_up_sr;
sr = hrtf_up_sr;
clear hrtf_up_sr;
y(1:lengths(6), :) = y(1:lengths(6), :) + [sub sub];
```

```
%--------------------------------------------------------------------
%   main_Spin
%
%   Test file for spin
%   Saves result to file
%
%--------------------------------------------------------------------

clear all;
close all;

filename = 'Toms_diner';

[x fs nbits] = wavread(filename);

y = spin ([x' x' x' x']', fs, 2, 10, 'kemar');

wavwrite(y, fs, nbits, strcat(filename, '_spin'));
```

```
%--------------------------------------------------------------------
%   main_Upmixed
%
%   Test file for upmix5
%   Saves result to file
%
%--------------------------------------------------------------------

clear all;
close all;

filename = 'Whole Lotta Love';

[x fs nbits] = wavread(filename);

[l r c sl sr sub] = upmix5(x,fs,'kemar','music');

wavwrite(l, fs, nbits, strcat(filename, '_l_upmixed'));
wavwrite(r, fs, nbits, strcat(filename, '_r_upmixed'));
wavwrite(c, fs, nbits, strcat(filename, '_c_upmixed'));
wavwrite(sl, fs, nbits, strcat(filename, '_sl_upmixed'));
wavwrite(sr, fs, nbits, strcat(filename, '_sr_upmixed'));
wavwrite(sr, fs, nbits, strcat(filename, '_sub_upmixed'));
```

```
%-------------------------------------------------------------------
%   main_Surround
%
%   Test file for spatialize_5point
%   Saves result to file
%
%-------------------------------------------------------------------

clear all;
close all;

filename = 'Whole Lotta Love2';

[x fs nbits] = wavread(filename);

[y l r c sl sr sub] = spatialize_5point(x,fs,'kemar','music');

% wavwrite(l, fs, nbits, strcat(filename, '_l'));
% wavwrite(r, fs, nbits, strcat(filename, '_r'));
% wavwrite(c, fs, nbits, strcat(filename, '_c'));
% wavwrite(sl, fs, nbits, strcat(filename, '_sl'));
% wavwrite(sr, fs, nbits, strcat(filename, '_sr'));
wavwrite(sub, fs, nbits, strcat(filename, '_sub'));
wavwrite(y*.5, fs, nbits, strcat(filename, '_surround'));
```