Music to the Ears:
Creating Multi-Dimensional Sound for a Virtual Environment

18-551, Spring 2001
Group 2 Final Report

Matthew Juhasz (mdj@andrew.cmu.edu),
Vivek Krishna (vkrishna@andrew.cmu.edu),
Julie Peoples (jpeoples@andrew.cmu.edu)

Table of Contents

I. Introduction

In people's everyday interaction with their environment, the way in which their five senses add subtle information about the surrounding environment plays a fundamental role in how they react to that environment. Until recently, individuals interested in creating a virtual environment neglected most of the other four senses in favor of developing the most realistic visual experience possible. However, it is hard to truly immerse oneself in a virtual world if only one sense is being stimulated. In order to create a solution to this problem, the purpose of our project was to create a multi-dimensional sound field, where sound can be placed at various positions in that field by the user. With this multi-dimensional sound field, a virtual environment can be created where the user is made to believe that sound is coming from various angles around the user's head. The addition of sound that can be placed in different positions in the user's field of hearing allows the user to be able to locate sound sources that are beyond the user's field of vision, thus increasing the amount of interaction with the environment. When this sound field is combined with a visual environment, a virtual environment can be created where the user is utilizing two senses, instead of just one. This creates a more realistic virtual environment.

1.A. Applications

The question then becomes "where would such a virtual sound environment be useful?" There are actually several different applications where such a system would come in handy. One such application is flight simulators. When training to fly, there is more to observe that just what can be seen. By adding multi-dimensional sound to the simulator, such as in a radar warning system, the trainees are able to more realistically

create actual flight conditions. The same type of situation can be applied to virtual surgery. In addition to seeing what is occurring in front of them, doctors-in-training also need to be able to keep track of the sounds that are occurring in surgery, such as machine warnings being set off and observations being made by nurses that are beyond the doctors' field of sight. Additionally, for those people who enjoy being completely immersed in a movie, the addition of a virtual sound environment allow the user to experience that sensation at home by allowing, not only the person's eyes to be tricked, but also his or her ears. However, the largest market for this type of a product is within the computer gaming industry. The popularity of virtual environment games and the desire for these games to immerse the user's senses as much as possible has created an industry where a virtual auditory environment is a lucrative addition to a software program.

1.B. Papers and Algorithms

Several papers have been written on virtual audio systems. For our purposes, we were most concerned with the articles dealing with how to create the illusion of placing audio anywhere around a user's head, and how to improve that sound. With these goals in mind, we found several papers and studies that related directly to our project. "Modeling HRTFs" by Richard Duda details the theory behind and how to use HRTFs (head related transfer functions, which are covered in more detail later in this report). From the article "Refined Algorithm of 3-D Sound Synthesis" by Zhang, Tan, and Er, we got a method for reducing the HRTF front-back reversal rates. In other words, this article gave us an algorithm for making the difference between sounds coming from in front of the user and sounds coming from in back of the user more noticeable. One of our most

important sources of information came from the study on the "KEMAR" dummy head at the MIT Media Labs (http://sound.media.mit.edu/Kemar.htm). Their study included actual HRTF data files that could be convolved with a sound file to simulate the sound coming from different angles. The measurements were taken in 5° increments around the head in the horizontal plane and in 10° elevations from -40° to 90°. These files were crucial for us being able to carry out our project.

1.C. Head Related Transfer Functions

With the discovery of all of this information on head related transfer functions (HRTFs) and their crucial role in virtual sound systems, the next logical questions are "what are HRTFs and how do they factor into the design of a virtual sound environment?" Basically, HRTFs describe the frequency response of a human head, depending on where the sound is in relation to the ear, both in the horizontal plane and the vertical elevation. The response also changes according to what the frequency components of the original sound file are and how good the sound quality of the original file is, as seen in our discussion of sound quality later in this report. Different HRTFs are needed for each ear, since each ear is at a different angle to the virtual position of the sound. Once the different HRTFs needed have been determined, they are convolved with the original sound file in order to give the impression of the sound coming from a particular angle and/or elevation.

Our particular HRTFs measurements came from the MIT Media Labs "KEMAR" dummy head experiment. The "KEMAR" dummy head was a simulated model of an average head. Microphones were placed inside the ears of the model head in order to

measure the in-ear response to pseudo-random impulses being played through loudspeakers at various angles and elevations.

1.D. Existing Software/Hardware and Prior Work

As far as work done in previous years in 18-551 is concerned, we believe that this is the first project of its type to be attempted. However, within the business realm, there are several companies that have developed both hardware and software versions of a virtual sound system. We also found commercial examples of the work that we were planning on doing at Sensaura (http://www.sensaura.co.uk/wse/Home/home.html) and SRS Laboratories (http://www.srslabs.com). Both of these companies provided us with software versions of working virtual sound systems, although, due to the patents and copyrights, we were unable to obtain any details on how they implemented their systems. Thus, we were responsible for writing our own software for the EVM processing and GUI. Pre-written code that we were able to utilize was the Texas Instruments assembly code for a Radix-4 FFT (IFFT). We also used the basic PC to EVM transfer code from Lab 2 and the DMA transfer code from Lab 3.

II. Project History

II.A. Initial Research

Before we could actually start on our project, we first had to research the subjects of both virtual audio systems and HRTFs. It was at this stage of our project that we found the information on how to implement HRTFs to provide the most realistic virtual environment. It was also during this initial research period that we discovered the HRTF files from the MIT Media Lab and found several companies who had already

implemented a virtual audio system similar to what we were attempting (although, we were unable to get any details from these companies).

II.B. Matlab Simulations

Matlab proved to be invaluable to us, not only for verifying that the FFT in our code was working correctly, but also for refining our algorithm, and converting our HRTFs and sound files into forms that could be used more easily in our project. We utilized Matlab throughout our project. At the beginning of our project, we used Matlab to evaluate the algorithms that we were using. Matlab generated the results of the FFT that we should have been seeing generated by the code, and then we checked these results against the results that were actually produced by the code. Later on in the development of the project we used Matlab to read in all of the HRTF files that we used into one file so that they could be easily accessed from the program that we wrote. In addition, Matlab was also used to convert the input .wav files into floating point data files, which was the input data type used by the program.

II.C. Transferring Information

Our first priority in writing the code for our project was to make sure that we could transfer data from the PC to the EVM and back again. In order to write this piece of code, we borrowed from the code we worked on in Lab 2 of our class. In order to get the data from the PC to the EVM, we utilized synchronous PCI/FIFO transfers

We spent some time becoming more familiar with the process of transferring information, so that we would be able to transfer both our sound files and the HRTFs. Before we worried about how the .wav files sounded after they had been processed, we first had to ensure that we could transfer a sound from a .wav file on the PC to the EVM,

have it play back, and have it sound like the sound file before we worked on processing it with the FFTs.

II.D. FFT Calculations on Transferred Data

We did not worry about performing an FFT of the signal (or performing an FFT of the HRTFs) until after we had finished writing the code that could transfer the sound file to the EVM board and play it back to the PC sound card. The radix-4 FFT (IFFT) source code/assembly code that we used to perform the calculations was from TI. We felt that we were able to switch to using frequency-domain calculations instead of doing time-domain calculations because we were manipulating the strength of different frequencies present in the input signal. For more information on the additional speed increases associated with switching from time-based calculations to FFT-based calculations, see the "Profiling of Optimization Steps" section of this report. In order to do processing of the FFT on-chip, it was necessary to use the dma_copyblock function to move data into the EVM internal memory from external memory. We decided to break the input signal into pieces of 1024 samples when we moved the data in order to allow enough room for all FFT-related data on-chip. Fitting all of the FFT-related data on-chip gave the best performance for processing the sound file.

II.E. Full Sound Processing for Each Channel

For processing the input sounds, we needed to perform an FFT to modify the sound's frequency spectrum, and we enhanced the HRTFs using the front-back reversal rate algorithm found in the article "Refined Algorithm of 3-D Sound Synthesis" by Zhang, Tan, and Er. The front-back reversal rate algorithm allowed us the ability to make the effects of the HRTFs more noticeable by exaggerating the differences in the

frequency responses between front and rear HRTFs. We fine-tuned the output by listening for distortion and adjusting volume levels and multiplier-constants in the EVM code. There were multiplier-constants in the EVM code because we were working in floating-point values while manipulating the input sound (>-1 and <1), and we needed to convert the output to the range -32768 to 32767 as this is the range of integer values that the EVM audio codec uses to output a signal.

II.F. Interrupts

After we knew that the sounds could be transferred to the EVM and convolved with HRTFs to create a sound coming from a particular direction, we could then work on manipulating the code so that whenever the virtual position of the sound or the distance or sound source changes, the sound will stop, recalculate the output sound with the new data and begin playing the output again. We determined that the best way of accomplishing this was through interrupts. The ways in which we used the interrupts and the issues that we had with interrupts are discussed further in this report.

II.G. GUI Integration

Once we knew that the sound files could be sent over to the EVM, convolved with the HRTFs, and interrupted so that they could be reevaluated with new inputs, we could then integrate our user interface with the code. The GUI was written with Visual C++, and the idea for our particular design of GUI was taken from the user interface from Sensaura. A detailed description of our GUI can be read about in the Demonstration section of our report.

II.H. Optimization

The process of optimization was actually an ongoing process that occurred throughout the work being done on the project. For more information on how we optimized the code for speed, see the section on Profiling the Optimization Steps. We also worked on optimizing the code for size. More information on this is found in our discussion of dealing with memory constraints in the next section of this report.

III. Problems and How They Were Solved

III.A. Distortion in Output Sound

One problem that we faced with our output sound was distortion. This distortion caused the distracting scratches and pops to be added to the original input sound throughout the sound and at the end of the clip. Redefining the lengths of some of the global variables and adjusting the constants we used to vary the output volume solved this problem.

III.B. Corrupted PCI Transfers

Another problem that we faced with our project concerned the corruption of the PCI transfers. If the PC CPU or the PC-side software was busy with anything else, and the PCI transfer had already started, then the transfer just kept going, even if some data was late. Then, this late data would either corrupt later data, or just be ignored, which caused the gaps in the output sound and/or pops. We could not find a clear solution to this, as the amount of noise/bad data wasn't too bad and its location in the data stream was always random. Our first attempt at solving this problem was to insert FOR loops into the PC-side code just before beginning a PCI transfer to force the PC to wait. By doing this, the PC software should have finished all other instructions before starting the

FOR (wait) loop, and should have been ready to transfer. This addition of code completely eliminated the noise generated by freeing up the EVM board and ensuring that the board was ready to receive the entire PCI transfer.

III.C. Interrupts

One of the problems that we encountered with interrupts was learning how to use them. Our first priority was figuring out which interrupts we needed to use. We discovered that the only interrupt command that we could find for triggering the EVM board from the PC side was DSPINT. After we included the DSPINT command, the interrupt would work once, but then would not trigger anymore after that. We finally discovered the solution to this solution in one of the TI manuals. The reason that DSPINT would work only once was because it was not being reset after triggering like the audio codec interrupts used in Lab 1. Thus, we needed to add the hpi_reset_dspint() command from the hpi.h library. Once this command was inserted after the call to DSPINT, the interrupts worked every time they were called.

III.D. Memory Allocation

We faced several different issues with the memory in the course of working on our project, most of which were related to space constraints in one form or another. There were four primary problems that we dealt with over the course of working on the project that were related to memory allocation. They are described in detail below.

III.D.i. Program size

One of the first ways that we attempted to deal with the memory space constraints was to reduce the size of the program. There were a couple of different ways that we went about doing this. The first way was to decrease the code size by trying to combine

certain operations. We also decreased the overall size of the code by cutting out the libraries that we were not using. Another process that we used was to re-use code through the creation of functions. For those operations that were occurring more than once within the code (such as performing FFTs), we combined the code into a function and then called that function instead of rewriting the code.

III.D.ii. Fitting Data On-chip

Another problem that we had related to the space constraints in memory allocation was fitting the necessary data on-chip. We found ourselves unable to allocate the entire ONCHIP_DATA. We would find ourselves trying to assign contents to the ONCHIP_DATA, and we knew that we had less than the 64K of memory used, but only 32K would be allocated to our data. The rest of the contents that did not fit into the 32K of space would signal an over-allocation of memory space and the software would crash. We solved this problem by allocating two different ONCHIP_DATA memory storages. The first ONCHIP_DATA storage had an origin of 0x80000000 and was 0x00008000 long. The second ONCHIP_DATA storage had an origin of 0x80008000 and was also 0x00008000 long. Although this method prevented us from putting any contents in ONCHIP_DATA that were longer than 0x00008000, we had not been able to do that previously, and this method of assigning the ONCHIP_DATA allowed us to more fully use the on-chip memory. More information on the exact amounts of memory used is seen later on in this report.

III.D.iii. Compiler Failed to Catch Allocation Errors

One problem that we found very frustrating was when the compiler did not catch allocation errors. This happened when we were very close to the on-chip memory

allotment (such as having filled 0xff00 bytes out of the 0x10000 allowed in the ONCHIP_P). The memory would be overwritten in the process of compiling the program, but because the on-chip memory allotment was not actually exceeded by us, the compiler did not list any allocation errors. Thus, we were getting inconsistent answers for our FFT output because information was being overwritten, but we were not getting warnings that the exceeded memory was the cause. It was not until we brought the memory use in the ONCHIP_P down to f200 that these inconsistencies with the FFT values were replaced by the correct values.

III.D.iv. Value Truncated at 0x…

Value truncated at 0xXXXXXXXX (the XXXXXXXX is replaced by a number of different hex values) was an error code that we saw quite frequently when we compiled our program. It occurred when our project exceeded any of the memory allocations. Optimizing the code to reduce the size solved this problem.

III.E. GUI Problems Concerning Visual C++

There were several problems that came about in regards to the GUI. The first issue was in the actual process of writing it. Visual C++ has a multitude of libraries that all correlate to functions that are similar, but different enough that they require separate libraries. Thus, it was very time consuming work to find the correct library for some of the functions that we wanted to include in our GUI, such as the ball on the circle surrounding the user's virtual head that can be clicked and dragged to the next location from which the user would like to hear the sound coming from. Another problem occurred when the GUI code was brought into the lab. Code that worked fine on computers outside the lab would send errors concerning the way that certain items were

called. One of the largest problems concerning the GUI was with the software itself, though. Visual C++ showed a high tendency to crash at the slightest provocation, which was a great source of annoyance because of the necessity of dealing with the problem so often on a daily basis. Visual C++ was also problematic in the fact that it displayed a weakness for causing memory leaks that would eventually force us to restart the computer. Parts of the code had to rewritten in order to manage this problem more efficiently, but we would still need to keep an eye on the percentage of memory being used on the PC.

III.F. EVM Software

We also experienced a problem with the EVM software displaying a tendency to either crash or freeze. The software had to be reset nearly every time that we ran the program or else we would receive an error message stating that the software had failed. A larger problem that we faced was when the EVM software froze. This required us to restart the computer. The EVM software was most likely to freeze when we did a board reset (pressed the white button on the C67 board) at the wrong time. The wrong time seemed to be when the EVM software was running (the software had to be closed in order to be able to do a board reset and not crash the computer).

III.G. FFT-Based Convolution

One problem that we had to deal with concerning the FFT convolutions was being able to get the correct output. We were able to use Matlab to determine what answers we should have been getting, so we knew that the answers that the code was outputting were wrong, but we were unsure as to the cause. Usually the situation was that the first output would be correct, but then the output would be incorrect and every other output would be

a zero. We were able to get the FFT-based convolution to work by doing a couple of different things. First, since we couldn't trigger an interrupt while the assembly code was running, we moved the processing of the initial sound to before we started playing it back. Also, we made sure that the project settings were correct - making sure the compiler/linker/assembler options were set as specified in the labs, set for the C67x specifically.

III.H. Real-Time Processing, Assembly Code, and Interrupts

Finally, we dealt with the issue of how the interrupts interacted with the radix-4 FFT (IFFT) assembly code that we were using from TI. The problem stemmed from the fact that the assembly code was too optimized. Interrupts cannot interrupt code that has loops less than 6 cycles long or that uses more than one register. Before we discovered the hpi_reset_dspint() function, the interrupts would not work with the TI assembly code because the code had so many commands being done in parallel that the interrupt command could never find a break in the code in which to interrupt the code. The large amount of parallelizing drove the loops in the assembly code to be under 6 cycles, and the assembly code was using two registers. For these reasons, we decided to use the C equivalent of the assembly code in order to utilize a less-optimal code that would have the necessary breaks in processing that were necessary to enact an interrupt.

Using the C code instead of the assemble code had problems of its own, though. Real-time processing was completely out of the question when using the C code. The process of changing the angle that a sound was coming from would take over 30 seconds to process. After we had completed the project and were able to look for additional ways to optimize the code, we attempted to implement the assembly code again. With the

addition made previously of the hpi_reset_dspint(), the assembly code could be interrupted, thus we permanently replaced the C code with the assembly code version of the radix-4 FFT. With the switch to assembly code, our performance increased from over 30 seconds to process a sound's new angle to less than 5 seconds. The numerical improvement in terms of cycles per second of sound is shown later in this report.

IV. Our Solution and Results

IV.A. Data Flow

IV.A.i. System Level

From the macroscopic view of our project, the creation of our virtual audio system can be seen simply as the movement of sound from the sound card on the PC to the EVM back to the sound card and through the headphones, as can be seen in Figure 1 below. The sound, which can come either in the form of a prerecorded .wav file or as a sound through the microphone which is then saved as a .wav file, is 0-7 seconds in duration, 44.1 kHz, 16-bit, and monaural. The sound then goes to the EVM, where it is convolved with the HRTFs to become a sound that appears to be coming from a virtual location. This processing is done for each ear, so by the time the sound gets back to the sound card in the PC, it has become a stereo sound instead of the monaural sound that it entered the EVM as. From the sound card, the new sound, which now has a virtual location, can then be heard through a pair of headphones, preferably a pair of in-ear

headphones, to fully experience the virtual audio environment.

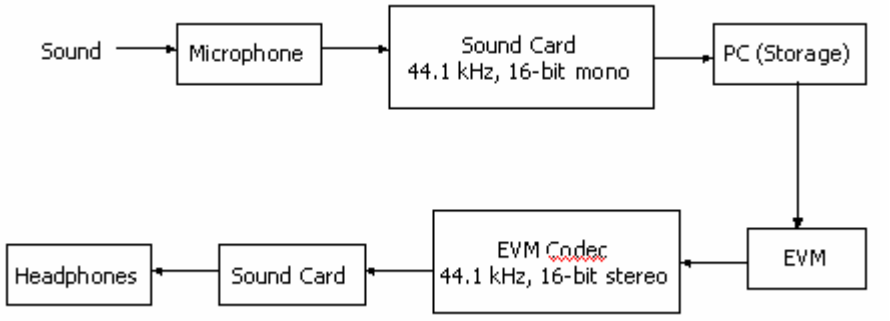

Figure 1

IV.A.ii Board Level

So if Figure 1 represents the brief overview of how the virtual audio system works, what are the details of what is occurring on the EVM board? As seen in Figure 2 below, once the sound file leaves the PC and enters the EVM board, there several different steps that the sound goes through before being sent back to the PC sound card. The first area of the EVM Board that the sound file reaches is external memory. From external memory, the sound file is brought into the on-chip memory by means of DMA block transfers. Once the sound is on-chip, the EVM Processing Algorithm is enabled to convolve the sound file with the HRTF and thus produce the illusion of the sound coming from a particular direction. The newly processed sound is then passed on to the audio codec on the EVM board before going out to the sound card in the PC.

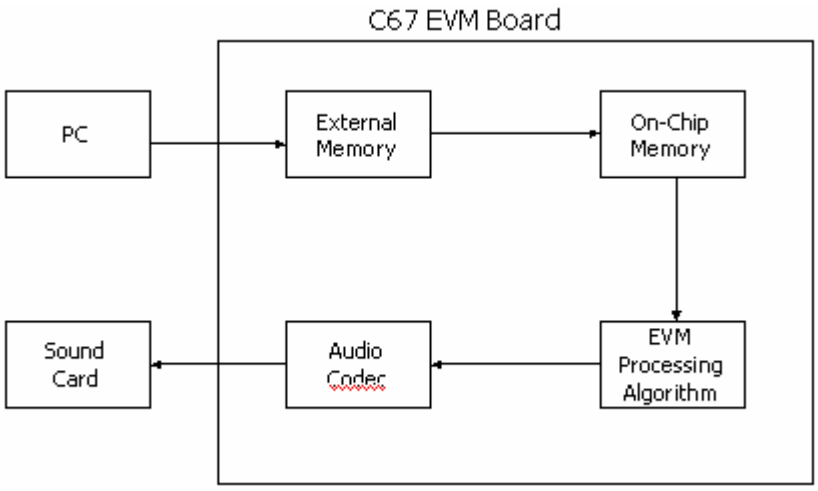

Figure 2

IV.A.iii. Algorithm

Now that the board layout has been described, what exactly occurs during the EVM processing algorithm? There are actually two processes going on at the same time, as seen in Figure 3. The sound file that was brought over from the PC is FFTed. The HRTF for the particular angle and elevation that the user desires for the particular sound file is filtered through an equation that generates a more noticeable difference between the sounds directly in front of the user and sounds directly in back of the user; then, this modified HRTF is also FFTed. Once both of these inputs are FFTed, they are put through a process of complex multiplication in order to modify the sound file so that the HRTF for a particular angle and elevation shifts the virtual location of the original sound file. Once the HRTF and the sound file are multiplied together, the new file is run through an IFFT to bring the sound file back to a form where it sounds like the original file. Then, the results are placed into a sample accumulator until enough of them are accumulated to block transfer them to the output buffer. Once all of the results are in the output buffer, the file is sent to the audio codec.

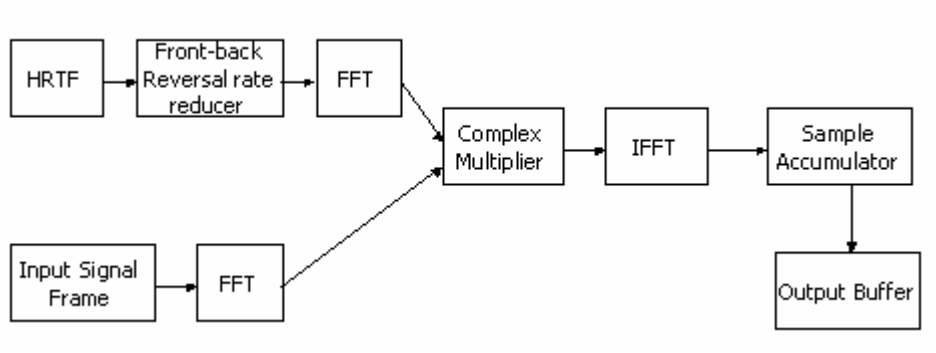

Figure 3

IV.B. Demonstration

The demonstration of our project basically consisted of using the GUI to load a sound, move the sound, and observe the change in sound through the headphones provided. The graphical interface provided three ways of manipulating the sound: the user could move the sound around 360° on a horizontal plane, move the sound's elevation from -40° to 90°, and move the sound from 1-9 meters away from the head. In addition, the user could also reset the angle, which would return the sound to 0° (directly in front of the virtual location of the user's head). This would enable the user to more easily compare the differences between a sound directly in front of him or her and a sound at the angle that the user had previously chosen. Also, we included a button that would enable the user to pick which .wav file he or she would like to listen to. Figure 4 below is the image of our GUI.

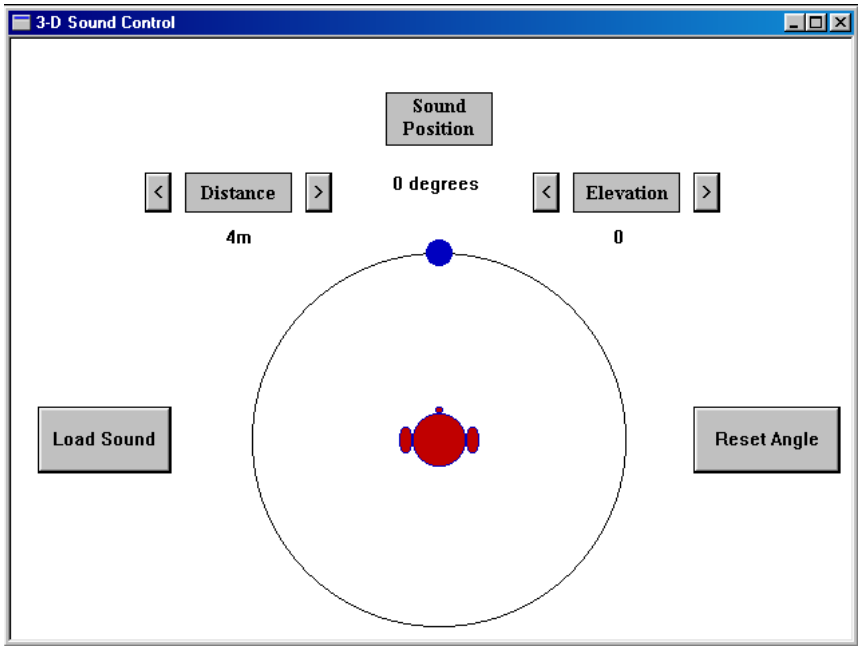

Figure 4

As one can see, the angle was manipulated by means of a ball that could be clicked on and dragged to the desired angle on the circle that surrounds the user's virtual head. The distance and elevation were manipulated by means of arrow keys that moved the elevations and distances up or down depending on which arrows the user pushed.

IV.C. Comparison to Commercial Software Programs

The primary commercial software program that we compared to our own program was Sensaura's computer simulated virtual audio environment. Our program actually provides more variables than Sensaura's, such as a greater range of distances to experiment with, and the ability to change the sound's elevation. We may be a bit biased to say that we believed our program to be better, but we can definitely state that our system was comparable to that of that particular commercial software program.

IV.D. Sound Quality and Equipment

We discovered that degree to which the user could detect a difference in the sound location was dependent on a couple of different factors. The first was the type of headphones used. Originally, we were using large headphones that completely covered the ears. However, since the HRTFs were generated using microphones inside of the KEMAR dummy head (from the MIT Media Lab), we decided to try in-ear headphones, which placed the sound source closer to the position that the microphones would have been in the KEMAR dummy head. This change produced a more convincing virtual audio sound. A user was also more likely to detect a difference in where the sounds appeared to be coming from if the sound played had a frequency spectrum similar to white noise. The closer that a sound was to white noise, the greater the number of frequencies that were represented in the original sound. The more frequencies that were in the original sound meant that more frequencies were being manipulated when the location changed, thus the change in location was more pronounced.

IV.E. Unconvincing Elevation Option

Regardless of the sound that we used, though, the elevation option was never very convincing. Changing the elevation would alter the frequencies, but these alterations would result in the .wav file sounding more "tinny" as the elevation was increased (this was caused by certain frequencies dropping out). This increase in "tinnyness" was not convincing as a change in elevation, though.

IV.F. Profiling of Optimization Steps

As seen in the chart below, there were several steps in the optimization process that led to sizable decreases in the amount of process time it took to process commands. Our earliest design of the project had us using time-domain convolution. The time-

domain convolution was fairly easy to implement, but it also caused a very high process time. Switching from time-based to FFT-based convolution dropped over 20 million cycles per second of sound off of our process time. Another decision that dropped about 5 million cycles per second of sound off of our process time was the use of DMA block transfers for putting the sound files on-chip for processing. However, the best way in which we optimized our project for speed was by using the radix-4 FFT assembly code that was hand optimized by TI instead of using the exact same code in C code. This decision saved us over 50 million cycles per second of sound off of our processing time.

Optimization Process	Process Time (cycles per sec. of sound)
Time-domain convolution	95,345,000
FFT-based convolution	74,346,000
DMA block transfers for on-chip processing (input/output paging)	69,798,000
TI-optimized radix-4 FFT routine	18,260,000

Chart 1

IV.G. Memory Usage Results

The following chart details how, where, and to what extent we used our memory sources.

Name	origin	length	used	attributes	**Contents**
ONCHIP_P	00000000	000010000	0000f200	R X	**Program**
SDRAM1	03000000	000400000	00400000	RW	**Dynamic Variables**
ONCHIP_D	80000000	000008000	00007b98	RW	**.far Variables**
ONCHIP_D	80008000	000008000	000070b4	RW	**Global Variables**

Chart 2

As discussed in the Problems section of our report, it was very difficult to find the room to place all of our code. Ideally, we would have wanted to place everything on-chip in order to make the processes as fast as possible, but memory constraints forced us to be

more creative with the way in which we allocated what went where. We decided what had to be on-chip and what we could handle being off-chip by the frequency and amount of time that particular components of the program were used. By this guideline, the actual program had to be ONCHIP_PROG, and the global variables and .far variables were ONCHIP_DATA. Because they were called only infrequently, we were able to afford to put the dynamic variables in SBRAM1, off-chip. Once we decided what had to be where in memory, we had to make sure that they fit. How we accomplished this is discussed in the Problems section of this report.

V. Conclusion

Overall, our goal of being able to create an audio virtual environment that would allow a user to choose where he or she would like to hear a sound coming from was achieved. In fact, we were able to stretch beyond our original goal of just being able to manipulate the angle around the head, and were also able to allow the user to manipulate both elevation and distance. As the market for virtual environments increases, the market for software and hardware that increase the interaction that a user can have with the virtual environment will also increase. Thus, this project is very applicable in regards to what the current market is demanding.

```
/*****************************************************************/
/* 18-551 Group 2 EVM code                                       */
/*****************************************************************/

/*---------------------------------------------------------------*/
/* INCLUDES AND LOCAL DEFINES                                    */
/*---------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <mcbsp.h>              /* mcbsp devlib */
#include <common.h>
#include <mcbspdrv.h>           /* mcbsp driver */
#include <board.h>              /* EVM library */
#include <codec.h>              /* codec library */
#include <mathf.h>              /* math library */
#include <intr.h>               /* interrupt library */
#include <pci.h>                /* PCI communication library */
#include <dma.h>
#include <hpi.h>                    /* HPI communications */

/*---------------------------------------------------------------*/
/* GLOBAL VARIABLES (On chip)                                    */
/*---------------------------------------------------------------*/
#define BUFFER_LEN 2048
#define FFT_LEN 1024
#define B_LEN 512
#define CONV_LEN 1024

int xindex=2;                       /* Index for transmission ISR */
int count=0;                              /* Position in external
memory */
int i = 0;
int rev_i;
int dev2;

float *x;            /* input sound  */
int *y;              /* output sound */
int sound_size;      /* length of the input sound */
int transfer;        /* transfer type */
int distance;        /* sound source distance from head */
int intcall;
far float hrtf_left[FFT_LEN*2];       /* hrtf for the left ear */
far float hrtf_right[FFT_LEN*2];     /* hrtf for the right ear */
far float fft[FFT_LEN*2];           /* fft of frame of input sound */
far int buffer[FFT_LEN];           /* dma transfer buffer for paging */
float temp_left[FFT_LEN*2];
float temp_right[FFT_LEN*2];
float temp1,temp2;
int revtable[FFT_LEN];         /* Look-up table for digit-reversing */
float w[FFT_LEN*2];               /* FFT twiddle factors (complex
interleaved) */


/* Function prototypes */
int dma_copy_block(void *src, void *dest, int numBytes, int chan);
```

```
void cfftr4_dif(float* x, float* w, short n);  /* Prototype for FFT
routine */
void fftconv();

/*---------------------------------------------------------------*/
/* FUNCTIONS                                                     */
/*---------------------------------------------------------------*/
interrupt void sendISR(void)
{
  INTR_DISABLE(CPU_INT14);
  intcall = 1;
}


interrupt void xmitISR(void)
{
  MCBSP0_DXR = y[xindex++];
}


/* This function creates the lookup table for digit reversing.  After
   it is run, revtable[n] equals the pairwise digit-reversal of n.
   n is the size of the FFT this table will be used for.*/
void mkrevtable(int n) {
  int bits, i, j, r, o;

  bits= (31 - _lmbd(1, n))/2;  /* _lmbd(1,n) finds leftmost 1 bit in n
*/
  for(i=0; i<n; i++) {
    r=0; o=i;
    _nassert(bits>=3);
    for(j=0; j<bits; j++) {
      r <<= 2;
      r |= o & 0x03;
      o >>= 2;
    }
    revtable[i] = r;
  }
}

/* Function for filling the table of FFT twiddle factors.  n is the
   size of the FFT to be used */
void fillwtable(int n) {
      int i;

      for (i = 0; i < n; i++) {
        w[2*i] = cosf(2*PI*i/n);
        w[2*i+1] = sinf(2*PI*i/n);
      }
}


/* FFT of input, complex multiplied by FFT of hrtf, then swapped real
and imaginary for IFFT and IFFTed; done for each ear */
void fftconv() {
  for (i = 0; i < FFT_LEN; i++) {
   rev_i = revtable[i];
```

```
   temp_left[2*i+1] = fft[2*rev_i] * hrtf_left[2*rev_i] -
fft[2*rev_i+1] * hrtf_left[2*rev_i+1];
   temp_left[2*i] = fft[2*rev_i+1] * hrtf_left[2*rev_i] + fft[2*rev_i]
* hrtf_left[2*rev_i+1];
   temp_right[2*i+1] = fft[2*rev_i] * hrtf_right[2*rev_i] -
fft[2*rev_i+1] * hrtf_right[2*rev_i+1];
   temp_right[2*i] = fft[2*rev_i+1] * hrtf_right[2*rev_i] +
fft[2*rev_i] * hrtf_right[2*rev_i+1];
  }
  cfftr4_dif((float *)temp_left,  (float *)w, FFT_LEN);
  cfftr4_dif((float *)temp_right, (float *)w, FFT_LEN);
}


/***** MAIN ********/
int main(void)
{
  int i;
  Mcbsp_dev dev;                /* Serial port device */
  intcall = 0;
  evm_init();                   /* Standard board initialization */
  HPI_RESET_DSPINT();           /* Reset DSPINT interrupt */
  mcbsp_drv_init();
  if (!(dev=mcbsp_open(0))) {
    return(ERROR);
  }
  mcbsp_setup(dev);
  pci_driver_init();           /* Call before using any PCI code */
  DMA_AUXCR = 0x00000010;   /* Set priority of HPI over CPU to avoid
crashing */
  dev2 = pci_fifo_open();        /* Open FIFO */
  mkrevtable(FFT_LEN);      /* Make the digit-reversal look-up table */
  fillwtable(FFT_LEN);      /* Make the table of FFT twiddle factors */
  distance = 1;

  /* Send over the default sound and hrtfs */
  pci_message_sync_send(1, TRUE);
  pci_fifo_sync_receive(dev2,(unsigned int *)&sound_size,
1*sizeof(int));
  x = (float *)malloc(310000 *2*sizeof(float));
  pci_message_sync_send(1, TRUE);
  pci_fifo_sync_receive(dev2,(unsigned int *)x, /* 2* */ sound_size *
sizeof(float));
  pci_message_sync_send(1, TRUE);
  pci_fifo_sync_receive(dev2,(unsigned int *)&x[sound_size], /*2 * */
sound_size * sizeof(float));
  pci_message_sync_send(1, TRUE);
  pci_fifo_sync_receive(dev2,(unsigned int *)hrtf_left, 2 * FFT_LEN *
sizeof(float));
  pci_message_sync_send(1, TRUE);
  pci_fifo_sync_receive(dev2,(unsigned int *)hrtf_right, 2 * FFT_LEN *
sizeof(float));

  /* Allocate y and initialize it to all zeros */
  y = (int *)malloc((310000+512)*sizeof(int));
  for (i = 0; i < sound_size+512; i++) {
     y[i] = 0;
```

```
  }
  for (i = FFT_LEN; i<2*FFT_LEN; i++) {
    fft[i] = 0.0;
  }

  cfftr4_dif((float *)hrtf_left, (float *)w, FFT_LEN);
  cfftr4_dif((float *)hrtf_right, (float *)w, FFT_LEN);


  /******************* configure CODEC *********************/
  /* EXIT_ERROR is a macro which jumps to exit_err if the function
     returns an ERROR */
  EXIT_ERROR(codec_init());
  codec_change_sample_rate(44100, TRUE);
  /* A/D 0.0 dB gain, turn off 20dB mic gain, sel (L/R)LINE input */
  EXIT_ERROR(codec_adc_control(LEFT,0.0,FALSE,LINE_SEL));
  EXIT_ERROR(codec_adc_control(RIGHT,0.0,FALSE,LINE_SEL));
  /* mute (L/R)LINE input to mixer */
  EXIT_ERROR(codec_line_in_control(LEFT,MIN_AUX_LINE_GAIN,TRUE));
  EXIT_ERROR(codec_line_in_control(RIGHT,MIN_AUX_LINE_GAIN,TRUE));
  /* D/A 0.0 dB atten, do not mute DAC outputs */
  EXIT_ERROR(codec_dac_control(LEFT, 0.0, FALSE));
  EXIT_ERROR(codec_dac_control(RIGHT, 0.0, FALSE));
  /***************** setup interrupt routines ******************/
  intr_init();
  /* Hook up serial transmit interrupt to CPU Interrupt 14 */
  intr_map(CPU_INT14,ISN_XINT0);
  INTR_CLR_FLAG(CPU_INT14);     /* Clear any old interrupts */
  intr_hook(xmitISR,CPU_INT14); /* Hook our own xmitISR into chain for
14 */
  /* Repeat the same process for the receive interrupt */

/* Map NMI */
  intr_map(CPU_INT4,ISN_DSPINT);
  INTR_CLR_FLAG(CPU_INT4);
  intr_hook(sendISR,CPU_INT4);
  /* End of NMI */

  /* Enable all necessary interrupts */
  INTR_ENABLE(CPU_INT_NMI);    /* Non-maskable interrupt */
  INTR_ENABLE(CPU_INT4);
  INTR_ENABLE(CPU_INT14);
  INTR_GLOBAL_ENABLE();         /* Controls whether ANY interrupts
function */
  /******************* Turn on the serial port ***********************/
/* MCBSP_ENABLE(dev->port,MCBSP_RX|MCBSP_TX); */

for (count = 0; count < 2*sound_size ; count+=FFT_LEN) {
    dma_copy_block(&x[count],&fft,FFT_LEN*sizeof(float),0);
    for (i = FFT_LEN; i<2*FFT_LEN; i++) {
      fft[i] = 0.0;
    }
    while (DMA0_XFER_COUNTER != 0);
    dma_copy_block(&y[count/2],&buffer,FFT_LEN/2*sizeof(float),0);
    for (i = FFT_LEN/2; i<FFT_LEN; i++) {
       buffer[i] = 0.0;
    }
```

```
    cfftr4_dif((float *)fft, (float *)w, FFT_LEN);
    fftconv();
    while (DMA0_XFER_COUNTER != 0);
    for (i = 0; i < FFT_LEN; i++) {
      rev_i = revtable[i];
      buffer[i] += 0xffff0000 & ((short int) (temp_left[2*rev_i+1] *
32767 / FFT_LEN) << 16);
      buffer[i] += 0x0000ffff & (short int) (temp_right[2*rev_i+1] *
32767 / FFT_LEN);
    }
    dma_copy_block(&buffer,&y[count/2],FFT_LEN*sizeof(float),0);
    while (DMA0_XFER_COUNTER != 0);
}

  /****************** Turn on the serial port **********************/
  MCBSP_ENABLE(dev->port,MCBSP_RX|MCBSP_TX);



while(TRUE) {
  if (xindex > sound_size)  /*  if we reach the end of output start at
the beginning */
     xindex = 2;

  if (intcall !=0){   /* called if an interrupt was triggered by the PC
*/
     intcall = 0;
     transfer = 2;

  /* receive transfer type */
  pci_fifo_sync_receive(dev2,(unsigned int *)&transfer, 1*sizeof(int));

   if (transfer==1) {  /* transfer over new input sound */
      pci_fifo_sync_receive(dev2,(unsigned int *)&sound_size,
1*sizeof(int));
      pci_fifo_sync_receive(dev2,(unsigned int *)x, sound_size *
sizeof(float));
      pci_fifo_sync_receive(dev2,(unsigned int *)&x[sound_size],
sound_size * sizeof(float));
  }
  else if (transfer==5) {  /* transfer over new hrtfs */
      pci_fifo_sync_receive(dev2,(unsigned int *)hrtf_left, 2 * FFT_LEN
* sizeof(float));
      pci_fifo_sync_receive(dev2,(unsigned int *)hrtf_right, 2 *
FFT_LEN * sizeof(float));
      cfftr4_dif((float *)hrtf_left, (float *)w, FFT_LEN);
      cfftr4_dif((float *)hrtf_right, (float *)w, FFT_LEN);
  }
  else if (transfer==3) {  /* transfer over new distance value */
      pci_fifo_sync_receive(dev2,(unsigned int *)&distance,
1*sizeof(int));
  }

  /* reset all of y to zeros */
  for (i = 0; i < sound_size+512; i++) {
      y[i] = 0;
  }
```

```
  /* calculate the output sound and place it in y */
  for (count = 0; count < 2*sound_size ; count+=FFT_LEN) {
    dma_copy_block(&x[count],&fft,FFT_LEN*sizeof(float),0);
    for (i = FFT_LEN; i<2*FFT_LEN; i++) {
       fft[i] = 0.0;
    }
    while (DMA0_XFER_COUNTER != 0);
    dma_copy_block(&y[count/2],&buffer,FFT_LEN/2*sizeof(float),0);
    for (i = FFT_LEN/2; i<FFT_LEN; i++) {
       buffer[i] = 0.0;
    }
    cfftr4_dif((float *)fft, (float *)w, FFT_LEN);
    fftconv();
    while (DMA0_XFER_COUNTER != 0);
    for (i = 0; i < FFT_LEN; i++) {
      rev_i = revtable[i];
      buffer[i] += 0xffff0000 & ((short int) (temp_left[2*rev_i+1] *
(int)(32767/(sqrtf(distance))) / FFT_LEN) << 16);
      buffer[i] += 0x0000ffff & (short int) (temp_right[2*rev_i+1] *
(int)(32767/(sqrtf(distance))) / FFT_LEN);
    }
    dma_copy_block(&buffer,&y[count/2],FFT_LEN*sizeof(float),0);
    while (DMA0_XFER_COUNTER != 0);
  }

    xindex = 2;           /* set playback back to the beginning */
    HPI_RESET_DSPINT(); /* Reset DSPINT interrupt */
    INTR_ENABLE(CPU_INT14);   /* May not need this */
  }
}

    exit_err:
return 0;
}

/* McBSP stands for Multi-Channel Buffered Serial Port.  It is build
   onto the C67 processor itself, and is how the codec communicates
   with the processor.  This function sets up the serial port for
   communication with the codec, and should never need to be modified
*/
int mcbsp_setup(Mcbsp_dev dev)
{
  /* Structure with all configuration parameters for serial port */
  Mcbsp_config mcbspConfig;

  memset(&mcbspConfig,0,sizeof(mcbspConfig)); /* Initialize everything
to 0 */

  mcbspConfig.loopback              = FALSE;

  mcbspConfig.tx.update             = TRUE;
  mcbspConfig.tx.clock_polarity     = CLKX_POL_RISING;
  mcbspConfig.tx.frame_sync_polarity= FSYNC_POL_HIGH;
  mcbspConfig.tx.clock_mode         = CLK_MODE_EXT;
  mcbspConfig.tx.frame_sync_mode    = FSYNC_MODE_EXT;
  mcbspConfig.tx.phase_mode         = SINGLE_PHASE;
```

```
  mcbspConfig.tx.frame_length1      = 0;
  mcbspConfig.tx.word_length1       = WORD_LENGTH_32;
  mcbspConfig.tx.frame_ignore       = FRAME_IGNORE;
  mcbspConfig.tx.data_delay         = DATA_DELAY0;

  mcbspConfig.rx.update             = TRUE;
  mcbspConfig.rx.clock_polarity     = CLKR_POL_FALLING;
  mcbspConfig.rx.frame_sync_polarity= FSYNC_POL_HIGH;
  mcbspConfig.rx.clock_mode         = CLK_MODE_EXT;
  mcbspConfig.rx.frame_sync_mode    = FSYNC_MODE_EXT;
  mcbspConfig.rx.phase_mode         = SINGLE_PHASE;
  mcbspConfig.rx.frame_length1      = 0;
  mcbspConfig.rx.word_length1       = WORD_LENGTH_32;
  mcbspConfig.rx.frame_ignore       = FRAME_IGNORE;
  mcbspConfig.rx.data_delay         = DATA_DELAY0;

  /* Pass entire structure to mcbsp_config, a library function which
     sets registers according to the contents of the structure */
  if(mcbsp_config(dev,&mcbspConfig) != OK) {

        return(ERROR);
  }

  return(OK);
}

/* dma_copy_block: Copies numBytes from src to dest using DMA channel
      chan.  chan can be 0 or 1.  this function is ASYNCHRONOUS!  You
must
      poll DMA0_TRANSFER_COUNT (or DMA1_TRANSFER_COUNT) to see when the
      transfer is complete */
/* Only valid for numBytes < 4 * 0xFFFF */
int dma_copy_block(void *src, void *dest, int numBytes, int chan)
{
  unsigned int dma_pri_ctrl=0;
  unsigned int dma_tcnt=0;

  /* Give DMA priority over CPU, and increment src and dest
      after each element */
  dma_pri_ctrl = 0x01000050;

  /* One frame, and we're using 4 byte elements */
  dma_tcnt = 0x00010000 | (numBytes/4);

  /* Write to DMA channel configuration registers */
  dma_init(chan,
           dma_pri_ctrl,
           0,
           (unsigned int) src,
           (unsigned int) dest,
           dma_tcnt);

  DMA_START(chan);
  return(OK);
}
```

```
// 18-551 - 3-D Sound Control GUI
//
// Allows user to manipulate sound location
// by changing angle, elevation and distance

#include <afx.h>
#include <afxcmn.h>
#include <afxdlgs.h>
#include <afxwin.h>
#include "evm6xdll.h"
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <string.h>
#include <windows.h>
#include <winuser.h>
//#undef LOAD_FILE //Use if we are debugging EVM code
#define IDB_BUTTON 100
#define PI 3.1415926535
#define HRTF_LEN 1024
#define LOAD_FILE
#define EVM_FILE "c:\\551Group2\\echo.out" // EVM code (executable)
#ifdef LOAD_FILE
int load_file(HANDLE hBd, LPVOID hHpi);
#endif

/* EVM Global variables */
  char filename[718][36];
  int location[14] =
{0,60,120,192,264,336,408,480,540,600,645,681,705,717}; // Elements in
hrtfs
  int spacing[14] = {60 ,60 ,72 ,72 ,72 ,72 ,72 ,60 ,60 ,45 ,36 ,24
,12, 1}; // # of angles
  char s_buffer[80];
  float hrtf_array[718][1024]; // Array for storing all hrtfs
  HANDLE hBd    = NULL;
  HANDLE h_event;
  EVM6XDLL_MESSAGE send;
  LPVOID hHpi = NULL;
  int sound_len;
  float *x;
  float hrtf_left[HRTF_LEN*2],hrtf_right[HRTF_LEN*2];
/*                                  */

/* Variables used for interface controls */
  int test;
  int changeangle = 0;
  int i = 0;
  int j = 1;
  int soundloc = 0;
  int xloc;
  int yloc;
  int oldx;
  int oldy;
  int lessx;
```

```
  int lessy;
  int morex;
  int morey;
  int nextsl;
  char *lastinput;
  char s[100] = "";
  char *li2 = "";
  CString file1 = "";
  int filelen = 0;
  bool headdrag;
  float data1[800000];
  CDC *pDC;
/*                                                              */
  // Application Class declaration
class CHelloApp : public CWinApp
{
public:
  virtual BOOL InitInstance();
  void main();
};

// Instance of the application class
CHelloApp HelloApp;

class DrawStuff : public CDC
{
public:
      DrawStuff();
      int k;
};


// Main window class
class CHelloWindow : public CFrameWnd
{
  CStatic* cs;
  CStatic* cs2;
  CStatic* cs3;
  CButton *button;
  CButton *button2;
  CButton *button3;
  CButton *button4;
  CButton *button5;
  CButton *button6;
  CFont *font;

public:
  CHelloWindow();
  CButtonWindow();
  afx_msg void OnSize(UINT, int, int);
  afx_msg void HandleButton1();
  afx_msg void HandleButton2();
  afx_msg void HandleButton3();
  afx_msg void HandleButton4();
  afx_msg void HandleButton5();
  afx_msg void HandleButton6();
  afx_msg void OnPaint();
```

```
  afx_msg void OnDraw(CDC*);
  afx_msg void OnLButtonDown(UINT, CPoint);
  afx_msg void OnLButtonUp(UINT, CPoint);
  afx_msg void OnMouseMove(UINT, CPoint);
  DECLARE_MESSAGE_MAP()

};

// The functions below handle button clicks
void CHelloWindow::HandleButton1() // Change Distance
{
      MessageBeep(-1);
      unsigned long ulLength, message;
      int transfersel[1];
      int dist[1];
      if (j>1)
      {
            j--;
            CDC* pDC;
            pDC = GetDC();
            char *t = itoa(j, s, 10);
            strcpy(s,t);
            strcat(s,"m");
            BeginWaitCursor();
            evm6x_hpi_generate_int(hHpi);
            h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
            transfersel[0] = 3;
            dist[0] = j;
            /* SEND TYPE OF TRANSFER */
            WaitForSingleObject( h_event, INFINITE );
          evm6x_retrieve_message(hBd, &message);
            ulLength = 1*sizeof(int);
            if (!evm6x_write(hBd, (unsigned long *)transfersel,
&ulLength)) {
            exit(7);
            }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER TYPE */

      /* SEND NEW DISTANCE */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);
      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)dist, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND NEW DISTANCE */

            for (int g = 0; g < 300000000; g++); // Pause before
allowing new input
            EndWaitCursor();
```

```
            pDC->DrawText(s,(int)(log10(j))+2,
CRect(160,140,270,180),DT_NOCLIP);
            ReleaseDC(pDC);
      }
}

void CHelloWindow::HandleButton2() // Change Distance
{
      MessageBeep(-1);
      unsigned long ulLength, message;
      int transfersel[1];
      int dist[1];
      if (j<9)
      {
            j++;
            CDC* pDC;
            pDC = GetDC();
            char *t = itoa(j, s, 10);
            strcpy(s,t);
            strcat(s,"m");
      BeginWaitCursor();
      evm6x_hpi_generate_int(hHpi);
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      transfersel[0] = 3;
      dist[0] = j;
      /* SEND TYPE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);
      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)transfersel, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER TYPE */

      /* SEND NEW DISTANCE */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);
      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)dist, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND NEW DISTANCE */

            for (int g = 0; g < 300000000; g++); // Pause before
allowing new input
            EndWaitCursor();
            pDC->DrawText(s,(int)(log10(j))+2,
CRect(160,140,270,180),DT_NOCLIP);
            ReleaseDC(pDC);
      }
}
```

```
void CHelloWindow::HandleButton3()  // Change Elevation
{
      MessageBeep(-1);
      unsigned long ulLength, message;
      int elev;
      int transfersel[1];
      if (i>-40)
      {
            BeginWaitCursor();
            i-=10;
            elev=i;
            int startloc=location[(elev+40)/10];
            changeangle = 360 / spacing[(elev+40)/10];
            int found = 0;
            int at = 0;
            while (found!=1)
            {
                  if ((abs(soundloc-at) < 5) || (at+changeangle>=360))
                  {
                        found = 1;
                        soundloc = at;
                  }
                  at += changeangle;
            }
            for (int i2 = 0; i2 < HRTF_LEN; i2++)
            {
                  hrtf_left[2*i2] =
hrtf_array[startloc+(soundloc/changeangle)][i2];
                  hrtf_left[2*i2+1] = 0;
            }
            for (i2 = 0; i2 < HRTF_LEN; i2++)
            {
                  hrtf_right[2*i2] = hrtf_array[startloc+(((360-
soundloc)%360)/changeangle)][i2];
                  hrtf_right[2*i2+1] = 0;
            }
      evm6x_hpi_generate_int(hHpi);
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      transfersel[0] = 5;

      /* SEND SIZE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);

      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)transfersel, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER SIZE */

      /* Sending hrtfs */
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
```

```
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_left, &ulLength)) {
            exit(3);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_right, &ulLength)) {
            exit(5);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
    /* End of sending hrtfs */

            CDC* pDC;
            pDC = GetDC();
            char *t = itoa(i, s, 10);
            strcpy(s,t);
            strcat(s,"    ");
            if (i>=0)
            {
                  pDC->DrawText("
",1,CRect(445,140,560,180),DT_NOCLIP);
                  pDC->DrawText(s,(int)(log10(i))+4,
CRect(450,140,560,180),DT_NOCLIP);
            }
            else
            pDC->DrawText(s,(int)(log10(i))+6,
CRect(450,140,560,180),DT_NOCLIP);
            ReleaseDC(pDC);
            for (int g = 0; g < 300000000; g++); // Pause before
allowing new input
            EndWaitCursor();
            Invalidate(TRUE);
      }
}

void CHelloWindow::HandleButton4()  // Change Elevation
{
      MessageBeep(-1);
      unsigned long ulLength, message;
      if (i<90)
      {
            BeginWaitCursor();
            i+=10;
            int elev = 0;
            int transfersel[1];
            elev=i;
            int startloc=location[(elev+40)/10];
            changeangle = 360 / spacing[(elev+40)/10];
            int found = 0;
            int at = 0;
            while (found!=1)
            {
                  if ((abs(soundloc-at) < 5) || (at+changeangle>=360))
```

```
                  {
                        found = 1;
                        soundloc = at;
                  }
                  at += changeangle;
            }
            for (int i2 = 0; i2 < HRTF_LEN; i2++)
            {
                  hrtf_left[2*i2] =
hrtf_array[startloc+(soundloc/changeangle)][i2];
                  hrtf_left[2*i2+1] = 0;
            }
            for (i2 = 0; i2 < HRTF_LEN; i2++)
            {
                  hrtf_right[2*i2] = hrtf_array[startloc+(((360-
soundloc)%360)/changeangle)][i2];
                  hrtf_right[2*i2+1] = 0;
            }
      evm6x_hpi_generate_int(hHpi);
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      transfersel[0] = 5;

      /* SEND SIZE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);

      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)transfersel, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER SIZE */

      /* Sending hrtfs */
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_left, &ulLength)) {
            exit(3);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_right, &ulLength)) {
            exit(5);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
    /* End of sending hrtfs */

            CDC* pDC;
            pDC = GetDC();
            char *t = itoa(i, s, 10);
```

```
            strcpy(s,t);
            strcat(s,"    ");
            if (i>=0)
            {
                  pDC->DrawText("
",1,CRect(445,140,560,180),DT_NOCLIP);
                  pDC->DrawText(s,(int)(log10(i))+4,
CRect(450,140,560,180),DT_NOCLIP);
            }
            else
            pDC->DrawText(s,(int)(log10(i))+6,
CRect(450,140,560,180),DT_NOCLIP);
            ReleaseDC(pDC);
            for (int g = 0; g < 300000000; g++); // Pause before
allowing new input
            EndWaitCursor();
            Invalidate(TRUE);
      }
}

void CHelloWindow::HandleButton5() // Change Input Sound (load new .BIN
file)
{
      unsigned long ulLength, message;
      int transfersel[1];
      int q = 0;
      MessageBeep(-1);
      CWnd* mainwnd = GetParent();
      CFileDialog dlg(TRUE, NULL, "C:\\551Group2\\sounds\\*.bin",
OFN_ENABLESIZING, "*.bin", mainwnd);
      q = dlg.DoModal();
      if (q==IDOK)
      {
            CFile newfile, newfile2;
            newfile.Open(dlg.m_ofn.lpstrFile,
CFile::modeRead|CFile::shareDenyNone|CFile::typeBinary);
            sound_len = ((newfile.ReadHuge(data1, 310000*4))/4);
            x = (float *)realloc(x, sound_len*2*sizeof(float));
            for (int i2 = 0; i2 < sound_len; i2++) {
                  x[2*i2] = (data1[i2]);
                  x[2*i2+1] = 0;
            }
      BeginWaitCursor();
      evm6x_hpi_generate_int(hHpi);
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      transfersel[0] = 1;

      /* SEND TYPE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);

      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)transfersel, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
```

```
      }
      /* END OF SEND TRANSFER TYPE */

      /* SEND SIZE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
      /* we got a message, now read it */
    evm6x_retrieve_message(hBd, &message);
      /* Set the length of the transfer */
      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)&sound_len, &ulLength)) {
            exit(1);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER SIZE */

      /* SEND WAVE FILE DATA */
      /* wait for event signaling a message from the DSP */
      WaitForSingleObject( h_event, INFINITE );
      /* we got a message, now read it */
      evm6x_retrieve_message(hBd, &message);
      /* Set the length of the transfer */
      ulLength = sound_len*sizeof(float);
      for (int f = 0; f < 30000; f++);
      if (!evm6x_write(hBd, (unsigned long *) x, &ulLength)) {
            exit(1);
      }
      if ( ulLength != sound_len*sizeof(float)) { /* Actual amount sent
is in ulLength */
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);;
      ulLength = sound_len*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) &x[sound_len],
&ulLength)) {
            exit(1);
      }
      if ( ulLength != sound_len*sizeof(float)) {
      }

      /* END OF WAVEFILE TRANSFER */

            filelen = 10;
            newfile.Close();
            CDC* pDC;
            pDC = GetDC();
            for (int g = 0; g < 300000000; g++); // Pause before
allowing new input
            EndWaitCursor();
            ReleaseDC(pDC);
      }
}

void CHelloWindow::HandleButton6() // Set default angle of zero degrees
{
      float ftemp;
```

```
      int i2, transfersel[1];
      unsigned long ulLength, message;
      char * hr;
      char *hrtf1;
    char *hrtf2;
      FILE  *h1, *h2;
      int ang = 0;
      int elev = 0;
      int startloc=location[(elev+40)/10];
      soundloc = 0;
      Invalidate(TRUE);
      for (i2 = 0; i2 < HRTF_LEN; i2++) {
        hrtf_left[2*i2] = hrtf_array[4][i2];
        hrtf_left[2*i2+1] = 0;
      }
      for (i2 = 0; i2 < HRTF_LEN; i2++) {
        hrtf_right[2*i2] = hrtf_array[4][i2];
        hrtf_right[2*i2+1] = 0;
      }
      BeginWaitCursor();
      evm6x_hpi_generate_int(hHpi);
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      transfersel[0] = 5;
      /* SEND SIZE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);

      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)transfersel, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER SIZE */

      /* Sending hrtfs */
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_left, &ulLength)) {
            exit(3);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_right, &ulLength)) {
            exit(5);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
    for (int g = 0; g < 300000000; g++); // Pause before allowing new
input
      EndWaitCursor();
      /* End of sending hrtfs */
```

```
}

// OnPaint - Handles the WM_PAINT message from Windows.
void CHelloWindow::OnPaint()
{
  xloc = (int)(320+(140*sin(2*PI*soundloc/360)));
  yloc = (int)(300-(140*cos(2*PI*soundloc/360)));
  PAINTSTRUCT ps;
  CDC* pDC = BeginPaint(&ps);
  int angle = soundloc;
  if (angle > 180)
  angle = 360 - angle;
  char *q = itoa(angle, s, 10);
  strcpy(s,q);
  strcat(s," degrees");
  strcat(s,"   ");
  pDC->DrawText(s, 11, CRect(285, 100, 400, 130), DT_NOCLIP);
  char *t = itoa(i, s, 10);
  strcpy(s,t);
  strcat(s,"    ");
  if (i>=0)
      {
            pDC->DrawText(" ",1,CRect(445,140,560,180),DT_NOCLIP);
            pDC->DrawText(s,(int)(log10(i))+4,
CRect(450,140,560,180),DT_NOCLIP);
      }
  else
  pDC->DrawText(s,(int)(log10(i))+6, CRect(450,140,560,180),DT_NOCLIP);
  t = itoa(j, s, 10);
  strcpy(s,t);
  strcat(s,"m");
  pDC->DrawText(s,(int)(log10(j))+2, CRect(160,140,270,180),DT_NOCLIP);
  LOGBRUSH LBrush;
  LBrush.lbStyle = BS_SOLID;
  LBrush.lbColor = RGB(192,0,0);
  LBrush.lbHatch = HS_CROSS;
  CBrush brush,brush2, brush3;
  LOGPEN Logpen;
  Logpen.lopnStyle = PS_SOLID;
  Logpen.lopnWidth.x = 1;
  Logpen.lopnColor = RGB(0, 0, 192);
  CPen pen, pen2,pen3;
  brush.CreateBrushIndirect(&LBrush);
  pen.CreatePenIndirect(&Logpen);
  CBrush* pOldBrush = (CBrush*)pDC->SelectObject(&brush);
  CPen* pOldPen = (CPen*)pDC->SelectObject(&pen);
  pDC->Ellipse(300, 280, 340, 320);  // Draw Head
  pDC->Ellipse(290, 290, 300, 310);  // Draw Left Ear
  pDC->Ellipse(340, 290, 350, 310);  // Draw Right Ear
  pDC->Ellipse(317, 275, 323, 280);  // Draw Nose

  pOldBrush = pDC->SelectObject(pOldBrush);
  pOldPen = pDC->SelectObject(pOldPen);
  LBrush.lbStyle = BS_NULL;
  LBrush.lbColor = RGB(0,0,0);
  Logpen.lopnColor = RGB(0, 0, 0);
  brush2.CreateBrushIndirect(&LBrush);
```

```
  pen2.CreatePenIndirect(&Logpen);
  pOldBrush = pDC->SelectObject(&brush2);
  pOldPen = pDC->SelectObject(&pen2);
  pDC->Ellipse(180, 160, 460, 440); // Draw Circle around head

  LBrush.lbStyle = BS_SOLID;
  LBrush.lbColor = RGB(0,0,192);
  LBrush.lbHatch = HS_CROSS;
  Logpen.lopnStyle = PS_SOLID;
  Logpen.lopnWidth.x = 1;
  Logpen.lopnColor = RGB(0, 0, 192);
  brush3.CreateBrushIndirect(&LBrush);
  pen3.CreatePenIndirect(&Logpen);
  pOldBrush = pDC->SelectObject(&brush3);
  pOldPen = pDC->SelectObject(&pen3);
  pDC->Ellipse(xloc-10, yloc-10, xloc+10, yloc+10);  // Draw Location
of Sound
  EndPaint(&ps);
}

// Handles redraws of Main Window
void CHelloWindow::OnDraw(CDC* pDC)
{
  int angle = soundloc;
  if (angle > 180)
  angle = 360 - angle;
  char *q = itoa(angle, s, 10);
  strcpy(s,q);
  strcat(s," degrees");
  strcat(s,"   ");
  pDC->DrawText(s, 11, CRect(285, 100, 400, 130), DT_NOCLIP);
  char *t = itoa(i, s, 10);
  strcpy(s,t);
  strcat(s,"    ");
  if (i>=0)
      {
            pDC->DrawText(" ",1,CRect(445,140,560,180),DT_NOCLIP);
            pDC->DrawText(s,(int)(log10(i))+4,
CRect(450,140,560,180),DT_NOCLIP);
      }
  else
  pDC->DrawText(s,(int)(log10(i))+6, CRect(450,140,560,180),DT_NOCLIP);
  t = itoa(j, s, 10);
  strcpy(s,t);
  strcat(s,"m");
  pDC->DrawText(s,(int)(log10(j))+2, CRect(160,140,270,180),DT_NOCLIP);
}

// Handles resizing of Main Window
void CHelloWindow::OnSize(UINT nType, int cx,
      int cy)
{
  MoveWindow (0, 0, 640, 480);
  xloc = (int)(320+(140*sin(2*PI*soundloc/360)));
  yloc = (int)(300-(140*cos(2*PI*soundloc/360)));
  PAINTSTRUCT ps;
  CDC* pDC = BeginPaint(&ps);
```

```
  int angle = soundloc;
  if (angle > 180)
  angle = 360 - angle;
  char *q = itoa(angle, s, 10);
  strcpy(s,q);
  strcat(s," degrees");
  strcat(s,"   ");
  pDC->DrawText(s, 11, CRect(285, 100, 400, 130), DT_NOCLIP);
  char *t = itoa(i, s, 10);
  strcpy(s,t);
  strcat(s,"    ");
  if (i>=0)
  {
      pDC->DrawText(" ",1,CRect(445,140,560,180),DT_NOCLIP);
      pDC->DrawText(s,(int)(log10(i))+4,
CRect(450,140,560,180),DT_NOCLIP);
  }
  else
  pDC->DrawText(s,(int)(log10(i))+6, CRect(450,140,560,180),DT_NOCLIP);
  t = itoa(j, s, 10);
  strcpy(s,t);
  strcat(s,"m");
  pDC->DrawText(s,(int)(log10(j))+2, CRect(160,140,270,180),DT_NOCLIP);
  LOGBRUSH LBrush;
  LBrush.lbStyle = BS_SOLID;
  LBrush.lbColor = RGB(192,0,0);
  LBrush.lbHatch = HS_CROSS;
  CBrush brush,brush2, brush3;
  LOGPEN Logpen;
  Logpen.lopnStyle = PS_SOLID;
  Logpen.lopnWidth.x = 1;
  Logpen.lopnColor = RGB(0, 0, 192);
  CPen pen, pen2,pen3;
  brush.CreateBrushIndirect(&LBrush);
  pen.CreatePenIndirect(&Logpen);
  CBrush* pOldBrush = (CBrush*)pDC->SelectObject(&brush);
  CPen* pOldPen = (CPen*)pDC->SelectObject(&pen);
  pDC->Ellipse(300, 280, 340, 320);  // Draw Head
  pDC->Ellipse(290, 290, 300, 310);  // Draw Left Ear
  pDC->Ellipse(340, 290, 350, 310);  // Draw Right Ear
  pDC->Ellipse(317, 275, 323, 280);  // Draw Nose

  pOldBrush = pDC->SelectObject(pOldBrush);
  pOldPen = pDC->SelectObject(pOldPen);
  LBrush.lbStyle = BS_NULL;
  LBrush.lbColor = RGB(0,0,0);
  Logpen.lopnColor = RGB(0, 0, 0);
  brush2.CreateBrushIndirect(&LBrush);
  pen2.CreatePenIndirect(&Logpen);
  pOldBrush = pDC->SelectObject(&brush2);
  pOldPen = pDC->SelectObject(&pen2);
  pDC->Ellipse(180, 160, 460, 440); // Draw Circle around head

  LBrush.lbStyle = BS_SOLID;
  LBrush.lbColor = RGB(0,0,192);
  LBrush.lbHatch = HS_CROSS;
  Logpen.lopnStyle = PS_SOLID;
```

```
  Logpen.lopnWidth.x = 1;
  Logpen.lopnColor = RGB(0, 0, 192);
  brush3.CreateBrushIndirect(&LBrush);
  pen3.CreatePenIndirect(&Logpen);
  pOldBrush = pDC->SelectObject(&brush3);
  pOldPen = pDC->SelectObject(&pen3);
  pDC->Ellipse(xloc-10, yloc-10, xloc+10, yloc+10);  // Draw Location
of Sound
  EndPaint(&ps);
}

// Handles clicking of mouse button on sound object
void CHelloWindow::OnLButtonDown(UINT nFlags, CPoint point)
{
  CDC* pDC;
  pDC = GetDC();
  if ((point.x >= xloc-20) && (point.x <= xloc+20) && (point.y >= yloc-
20)
        && (point.y<= yloc+20))
  {
      if (!headdrag)
      {
            headdrag=TRUE;
            oldx = xloc;
            oldy = yloc;
            nextsl = soundloc-changeangle;
            if (nextsl < 0)
            nextsl = nextsl+360;
            lessx = (int)(320+(140*sin(2*PI*nextsl/360)));
            lessy = (int)(300-(140*cos(2*PI*nextsl/360)));
            nextsl = soundloc+changeangle;
            if (nextsl >= 360)
            nextsl = nextsl-360;
            morex = (int)(320+(140*sin(2*PI*nextsl/360)));
            morey = (int)(300-(140*cos(2*PI*nextsl/360)));
      }
  }

  ReleaseDC(pDC);
}

// Handles unclicking of mouse button on sound object (for click-and-
drag)
void CHelloWindow::OnLButtonUp(UINT nFlags, CPoint point)
{
      if (headdrag)
      {
            CWaitCursor wait();
            unsigned long ulLength, message;
            headdrag=FALSE;
            int transfersel[1];
            int elev=i;
            int startloc=location[(elev+40)/10];
            for (int i2 = 0; i2 < HRTF_LEN; i2++)
            {
                  hrtf_left[2*i2] =
hrtf_array[startloc+(soundloc/changeangle)][i2];
```

```
                  hrtf_left[2*i2+1] = 0;
            }

            for (i2 = 0; i2 < HRTF_LEN; i2++)
            {
                  hrtf_right[2*i2] = hrtf_array[startloc+(((360-
soundloc)%360)/changeangle)][i2];
                  hrtf_right[2*i2+1] = 0;
            }
      BeginWaitCursor();
      evm6x_hpi_generate_int(hHpi);
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      transfersel[0] = 5;

      /* SEND SIZE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
    evm6x_retrieve_message(hBd, &message);

      ulLength = 1*sizeof(int);
      if (!evm6x_write(hBd, (unsigned long *)transfersel, &ulLength)) {
            exit(7);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
      }
      /* END OF SEND TRANSFER SIZE */

      /* Sending hrtfs */
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_left, &ulLength)) {
            exit(3);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_right, &ulLength)) {
            exit(5);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
      }
    /* End of sending hrtfs */

            for (int g = 0; g < 300000000; g++); // Pause before
allowing new input
            EndWaitCursor();
            Invalidate(TRUE);
      }
}

// Handles movement of mouse in Main Window (for Click-and-Drag)
void CHelloWindow::OnMouseMove(UINT nFlags, CPoint point)
{
      CDC* pDC;
```

```
      pDC = GetDC();
      if (headdrag)
      {
            if ((point.x >= lessx-(int)(changeangle*1.25)) && (point.x
<= lessx+(int)(changeangle*1.25)) && (point.y >= lessy-
(int)(changeangle*1.25))
                  && (point.y <= lessy+(int)(changeangle*1.25)))
            {
                  oldx = xloc;
                  oldy = yloc;
                  InvalidateRect(CRect(oldx-10, oldy-10, oldx+10,
oldy+10),TRUE);
                  xloc = lessx;
                  yloc = lessy;
                  soundloc = soundloc - changeangle;
                  if (soundloc < 0)
                  soundloc = soundloc+360;

                  int angle = soundloc;
                  if (angle > 180)
                  angle = 360 - angle;
                  char *q = itoa(angle, s, 10);
                  strcpy(s,q);
                  strcat(s," degrees");
                  strcat(s,"   ");
                  pDC->DrawText(s, 11, CRect(285, 100, 400, 130),
DT_NOCLIP);

                  nextsl = soundloc-changeangle;
                  if (nextsl < 0)
                  nextsl = nextsl+360;
                  lessx = (int)(320+(140*sin(2*PI*nextsl/360)));
                  lessy = (int)(300-(140*cos(2*PI*nextsl/360)));
                  nextsl = soundloc+changeangle;
                  if (nextsl >= 360)
                  nextsl = nextsl-360;
                  morex = (int)(320+(140*sin(2*PI*nextsl/360)));
                  morey = (int)(300-(140*cos(2*PI*nextsl/360)));
                  xloc = (int)(320+(140*sin(2*PI*soundloc/360)));
                  yloc = (int)(300-(140*cos(2*PI*soundloc/360)));
                  pDC->Ellipse(xloc-10, yloc-10, xloc+10, yloc+10);  //
Draw Location of Sound
            }
            else if ((point.x >= morex-(int)(changeangle*1.25)) &&
(point.x <= morex+(int)(changeangle*1.25)) && (point.y >= morey-
(int)(changeangle*1.25))
                  && (point.y <= morey+(int)(changeangle*1.25)))
            {
                  oldx = xloc;
                  oldy = yloc;
                  InvalidateRect(CRect(oldx-10, oldy-10, oldx+10,
oldy+10),TRUE);
                  xloc = morex;
                  yloc = morey;
                  soundloc = soundloc + changeangle;
                  if (soundloc >= 360)
                  soundloc = soundloc-360;
```

```
                int angle = soundloc;
                if (angle > 180)
                angle = 360 - angle;
                char *q = itoa(angle, s, 10);
                strcpy(s,q);
                strcat(s," degrees");
                strcat(s,"   ");
                pDC->DrawText(s, 11, CRect(285, 100, 400, 130),
DT_NOCLIP);
                nextsl = soundloc-changeangle;
                if (nextsl < 0)
                nextsl = nextsl+360;
                lessx = (int)(320+(140*sin(2*PI*nextsl/360)));
                lessy = (int)(300-(140*cos(2*PI*nextsl/360)));
                nextsl = soundloc+changeangle;
                if (nextsl >= 360)
                nextsl = nextsl-360;
                morex = (int)(320+(140*sin(2*PI*nextsl/360)));
                morey = (int)(300-(140*cos(2*PI*nextsl/360)));
                xloc = (int)(320+(140*sin(2*PI*soundloc/360)));
                yloc = (int)(300-(140*cos(2*PI*soundloc/360)));
                pDC->Ellipse(xloc-10, yloc-10, xloc+10, yloc+10);  //
Draw Location of Sound
            }
      }
            ReleaseDC(pDC);
}

// The Message Map -- Assigns functions to Windows Events
BEGIN_MESSAGE_MAP(CHelloWindow, CFrameWnd)
      ON_BN_CLICKED(IDB_BUTTON, HandleButton1)
      ON_BN_CLICKED(IDB_BUTTON+1, HandleButton2)
      ON_BN_CLICKED(IDB_BUTTON+2, HandleButton3)
      ON_BN_CLICKED(IDB_BUTTON+3, HandleButton4)
      ON_BN_CLICKED(IDB_BUTTON+4, HandleButton5)
      ON_BN_CLICKED(IDB_BUTTON+5, HandleButton6)
      ON_WM_SIZE()
    ON_WM_PAINT()
      ON_WM_LBUTTONDOWN()
      ON_WM_LBUTTONUP()
      ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()


// Initial commands run by application (program starts here)
BOOL CHelloApp::InitInstance()
{
/* Setting default values  */
  i = 0;
  j = 1;
  changeangle = 5;
  soundloc = 0;
  strcat(s,"1m");
  headdrag = 0;
  oldx = 0;
  oldy = 0;
  morex = 0;
```

```
  morey = 0;
  lessx = 0;
  lessy = 0;
  nextsl = 0;
  lastinput = " ";
/*                                      */

/* Variables for EVM stuff */
      unsigned long ulLength, message;
    FILE    *fi, *fi2, *fi3, *files;
      int i2;
      float ftemp;
      char *infile = "C:\\551Group2\\sounds\\LoopyMusic.bin";
      char *infile2 = "C:\\551Group2\\hrtfs\\L000e000a.bin";
    char *infile3 = "C:\\551Group2\\hrtfs\\L000e000a.bin";
      char *infile4 = "C:\\551Group2\\hrtfs\\largearray.bin";
      char name;
      int j2;
/*                          */

/* intialize EVM code */
      hBd = evm6x_open(0, 1);
    hHpi = evm6x_hpi_open(hBd);
      #ifdef LOAD_FILE
      load_file(hBd, hHpi);
 #else
      /* Set DMA AUX priority greater than CPU priority, so we
            don't lock the PCI bus */
    hpi_write_word(0x01840070 /*Addr*/, 0x00000010 /*Data*/);
      /* Reset the mailboxes and FIFO */
      hpi_write_word(0x0170003C, 0x0e000000);
  #endif
  evm6x_close(hBd);
/*                               */

/* READING IN BIG HRTF ARRAY */
      files = fopen(infile4,"rb");

      for (int w = 0; w < 718; w++)
      {
            for (int w2 = 0; w2 < 1024; w2++)
            {
            fread(&ftemp,sizeof(float),1,files);
            hrtf_array[w][w2] = ftemp;
            }
      }
    fclose(files);
/*                                                 */

      fi = fopen(infile,"rb");
      if (fi == NULL) {
    }

      /*  Read input file to end-of-file */
    i2 = 0;
      fread(&ftemp, sizeof(float), 1, fi);
      sound_len = 231546;
```

```
      x = (float *)malloc(sound_len*2*sizeof(float));
      for (i2 = 0; i2 < sound_len; i2++) {
      fread(&ftemp, sizeof(float), 1, fi);
        x[2*i2] = ftemp;
        x[2*i2+1] = 0;
      }
    fclose(fi);

      /* READING IN HRTF Files */
      for (i2 = 0; i2 < HRTF_LEN; i2++) {
            hrtf_left[2*i2] = hrtf_array[location[4]][i2];
            hrtf_left[2*i2+1] = 0;  }
      for (i2 = 0; i2 < HRTF_LEN; i2++) {
            hrtf_right[2*i2] = hrtf_array[location[4]][i2];
            hrtf_right[2*i2+1] = 0; }
      /* END OF READ FILES */

      hBd = evm6x_open(0, 0);          /* Open board */
      if((hHpi = evm6x_hpi_open(hBd)) == NULL)
            exit(1);
      evm6x_hpi_write_single(hHpi, 0x0e000000, 4, 0x0170003C);
      if ( hBd == INVALID_HANDLE_VALUE ) {
            exit(1);
      }
      /* Setup the Windows event that comes when a message is received
*/
      h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );
      if (h_event==NULL) {
      }

      /* SEND SIZE OF TRANSFER */
      WaitForSingleObject( h_event, INFINITE );
      send = sound_len;
      /* wait for event signaling a message from the DSP */
      WaitForSingleObject( h_event, INFINITE );
      /* we got a message (synchronization)*/
    evm6x_retrieve_message(hBd, &message);
      /* Set the length of the transfer */
      ulLength = 1*sizeof(int);
      /* Actually send the stuff. */
      if (!evm6x_write(hBd, (unsigned long *)&sound_len, &ulLength)) {
            exit(1);
      }
      if ( ulLength != 1*sizeof(int)) { /* Actual amount sent is in
ulLength*/
            exit(1);
      }
      /* END OF SEND TRANSFER SIZE */

      /* SEND WAVE FILE DATA */
      /* wait for event signaling a message from the DSP */
      WaitForSingleObject( h_event, INFINITE );
      /* we got a message, now read it */
      evm6x_retrieve_message(hBd, &message);
      /* Set the length of the transfer */
      ulLength = sound_len*sizeof(float);
```

```
      for (int f = 0; f < 30000; f++); // Pause before allowing new
instructions to run
      if (!evm6x_write(hBd, (unsigned long *) x, &ulLength)) {
            exit(1);
      }
      if ( ulLength != sound_len*sizeof(float)) { /* Actual amount sent
is in ulLength*/
            exit(1);
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = sound_len*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) &x[sound_len],
&ulLength)) {
            exit(1);
      }
      if ( ulLength != sound_len*sizeof(float)) {
            exit(1);
      }
      /* END OF SEND WAVE FILE DATA */
      /* Sending hrtfs */
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_left, &ulLength)) {
            exit(1);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
            exit(1);
      }
      WaitForSingleObject( h_event, INFINITE );
      evm6x_retrieve_message(hBd, &message);
      ulLength = 2*HRTF_LEN*sizeof(float);
      if (!evm6x_write(hBd, (unsigned long *) hrtf_right, &ulLength)) {
            exit(1);
      }
      if ( ulLength != 2*HRTF_LEN*sizeof(float)) {
            exit(1);
      }
    /* End of sending hrtfs */
/*       END of initial code  */

/* Initialize new instance of Main Window and update it */
  m_pMainWnd = new CHelloWindow();
  m_pMainWnd->ShowWindow(m_nCmdShow);
  m_pMainWnd->UpdateWindow();
  m_pMainWnd->MessageBox("Welcome to the Control GUI.", "3-D Sound
Control",
      MB_ICONINFORMATION | MB_OK);
  pDC = m_pMainWnd->GetDC();
/*
            */
  return TRUE;
}

// The constructor for the Main Window class
CHelloWindow::CHelloWindow()
```

```
{
// Define Main Window
  Create(NULL,"3-D Sound Control",
  WS_OVERLAPPEDWINDOW|WS_VISIBLE,
  CRect(0,0,640,480),NULL, NULL, 0, NULL);
// Create a new 36 point Arial font for use in Window
      font = new CFont;
      font->CreateFont(18,0,0,0,700,0,0,0,
                ANSI_CHARSET,OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
                DEFAULT_PITCH|FF_DONTCARE,
                "times");
// Cause the label to use the new font
      cs->SetFont(font);

    CRect r;
    CRect r2;
    CRect r3;
    CRect r4;
      CRect r5;
      CRect r6;

// Set sizes for graphical items
      r.SetRect(100, 100, 120, 130);
      r2.SetRect(220, 100, 240, 130);
      r3.SetRect(390, 100, 410, 130);
      r4.SetRect(510, 100, 530, 130);
      r5.SetRect(20,275, 120, 325);
      r6.SetRect(510,275, 620, 325);

// Create graphical (text) buttons
      button = new CButton();
      button->Create("<",
            WS_CHILD|WS_VISIBLE|SS_CENTER|BS_PUSHBUTTON,
            r,
            this,
            IDB_BUTTON);
      button2 = new CButton();
      button2->Create(">",
            WS_CHILD|WS_VISIBLE|SS_CENTER|BS_PUSHBUTTON,
            r2,
            this,
            IDB_BUTTON+1);
      button3 = new CButton();
      button3->Create("<",
            WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON|SS_CENTER,
            r3,
            this,
            IDB_BUTTON+2);
      button4 = new CButton();
      button4->Create(">",
            WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON|SS_CENTER,
            r4,
            this,
            IDB_BUTTON+3);
      button5 = new CButton();
```

```
      button5->Create("Load Sound",
            WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON|SS_CENTER,
            r5,
            this,
            IDB_BUTTON+4);
      button6 = new CButton();
      button6->Create("Reset Angle",
            WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON|SS_CENTER,
            r6,
            this,
            IDB_BUTTON+5);

// Create Window Text boxes
  cs = new CStatic();
  cs->Create("Sound Position",
    WS_CHILD|WS_VISIBLE|SS_CENTER|WS_BORDER,
    CRect(280,40,360,80),this);

  cs2 = new CStatic();
  cs2->Create("Distance",
    SS_CENTERIMAGE|SS_CENTER|WS_CHILD|WS_VISIBLE|WS_BORDER,

    CRect(130,100,210,130),this);

  cs2->SetFont(font);


  cs3 = new CStatic();
  cs3->Create("Elevation",
    SS_CENTERIMAGE|SS_CENTER|WS_CHILD|WS_VISIBLE|WS_BORDER,

    CRect(420,100,500,130),this);

  cs3->SetFont(font);
}


DrawStuff::DrawStuff() // Temporary class::instance for debugging
{
k = 0;
}



void CHelloApp::main() // Optional "Main" function (debugging)
{
}
/* END of classes and main functions*/

/* EVM-based functions and definitions*/
/* Write a single 32-bit word to EVM memory */
void hpi_write_word(ULONG addr, ULONG data)
{
      ULONG ulLength = 4;

      if (!evm6x_hpi_write(hHpi, &data, &ulLength, addr)) {
            exit(1);
```

```
      }
      if (ulLength != 4 ) {
            exit(1);
      }
}


#ifdef LOAD_FILE
/* Load the .out file and run the program. */
int load_file(HANDLE hBd, LPVOID hHpi)
{
  if ( !evm6x_reset_board(hBd) ) exit(2);
  if ( !evm6x_reset_dsp(hBd,HPI_BOOT) ) exit(3);
  if ( !evm6x_init_emif(hBd, hHpi) ) exit(5);

  /* Load program */
  if (!evm6x_coff_load(hBd,hHpi,EVM_FILE,0,0,0)) exit(8);

  /* Set HPI priority and reset mailboxes */
  hpi_write_word(0x01840070 /*Addr*/, 0x00000010 /*Data*/ );
  hpi_write_word(0x0170003C, 0x0e000000);

  if (!evm6x_unreset_dsp(hBd)) exit(10);
  if (!evm6x_set_timeout(hBd,1000)) exit(11);
 return(0);
}

#endif
```