Dirt Cheap 3D Spatial Audio

Eric Klein[1]   Greg S. Schmidt[1][2]   Erik B. Tomlin[1]   Dennis G. Brown[1]

[1]Virtual Reality Laboratory, Naval Research Laboratory, Washington, D.C.
[2]ITT Advanced Engineering & Sciences, Alexandria, VA

Abstract

The advent of affordable sophisticated audio solutions for the home PC market has paved the way for low-cost true three-dimensional spatial audio, which can produce a sound effect from any 3D position source. The home entertainment market has guided the development of audio solutions for PCs through hardware support for stereo and Dolby®Surround Sound versions 4.1, 5.1, and 7.1. However, these solutions provide sound panning only in a planar configuration, typically around the user's entertainment room. The drivers for these home-market cards (e.g. Direct Sound) do not have support for non-planar configurations. There are other cards and drivers available that provide true 3D spatial audio, but they are geared for the professional user market and have steep price tags over one thousand dollars. We explain how to use inexpensive consumer-level hardware and free software for Linux to build a true 3D spatial audio system.

1 Introduction

Many computer systems set up for advanced gaming include Dolby®Surround Sound. The typical speaker configurations are 4.1 (4 speakers and 1 subwoofer), 5.1, and 7.1. This system is designed for all speakers to be located on a plane centered at the listener, and thus it is not possible to have a sound truly be emitted from above or below the listener–although some systems attempt to simulate that effect. Imagine a game scenario where a monster is climbing down a wall above and behind the player, while at the same time, a mouse is scrambling across the floor behind the listener. In a planar surround system, the sound effects for both the monster and the mouse would come from the rear speakers, making it hard to distinguish the actual locations of the sound sources. With true 3D spatial audio, the monster's sound effects could be played from speakers located to the back-upper-left and the mouse's sound from speakers located to the back-lower-left and back-lower-right. The player will have a much better feel for what is creating the sound and where the sound is coming from then with traditional Dolby®Surround Sound. Now the player can arm the rocket launcher and turn towards the back-upper-left directly and blast the monster–no need to aim towards the harmless mouse running across the chamber floor behind.

Spatial sound has been available for several years and is primarily employed in immersive virtual environments. The systems are not in mass-scale production and often must be installed by professionals, making them costly and out of the reach of most home users. We have devised a low-cost true 3D spatial audio solution that requires only inexpensive consumer-level hardware and open source software. This solution allows arbitrary placement of speakers, not necessarily co-planar like other systems. Our 3D spatial audio solution is the first that we are aware to provide true 3D sound at such a low cost. We describe the hardware and software we used and how we set up, configured, and tested the 3D spatial audio solution.

2 Background on 3D Spatial Audio

Preliminary technology for 3D spatial audio, Fantasound [2], was first developed for the movie industry in the late 1930s by Disney. Over the years a great deal of work has been done to advance the field, especially by the Dolby corporation. In the last few decades, researchers enabled personal computers to emit spatial audio. Today spatial audio is commonplace in modern computer games. Home systems typically use headphones or a planar array of speakers, usually in a preset configuration, such as Dolby®Surround Sound 5.1.

Headphones present a unique opportunity to provide inexpensive 3D audio. Algorithms that use head-related transfer functions (HRTFs) [10] can create convincing 3D spatial audio on headphones using a simple stereo sound card. HRTFs use data about how sound is transformed by the user's body (especially the shape of the ears) for mapping sounds with 3D positional sources. The technique relies heavily on applying different time delays for each ear. Ultimately, we decided not to use headphones, because we needed a system that scaled easily to many users. It was far more practical and cost efficient to use speakers.

There are a number of high-cost professional-grade hardware packages available (e.g., RME Hammerfall series [9], M-Audio Delta series [4] and Lake Audio [3]) that provide true 3D spatial audio. Each package has a cost exceeding one thousand dollars, boasts high sound quality, and has a large array of features aimed at the professional market.

1. REPORT DATE **OCT 2005**	2. REPORT TYPE	3. DATES COVERED **00-00-2005 to 00-00-2005**
4. TITLE AND SUBTITLE **Dirt Cheap 3D Spatial Audio**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Virtual Reality Laboratory,4555 Overlook Ave. SW,Washington,DC,20375**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Linux Journal, Oct 2005.

14. ABSTRACT
The advent of affordable sophisticated audio solutions for the home PC market has paved the way for low-cost true three-dimensional spatial audio, which can produce a sound effect from any 3D position source. The home entertainment market has guided the development of audio solutions for PCs through hardware support for stereo and Dolby R Surround Sound versions 4.1, 5.1, and 7.1. However these solutions provide sound panning only in a planar configuration, typically around the user?s entertainment room. The drivers for these home-market cards (e.g. Direct Sound) do not have support for non-planar configurations. There are other cards and drivers available that provide true 3D spatial audio, but they are geared for the professional user market and have steep price tags over one thousand dollars. We explain how to use inexpensive consumer-level hardware and free software for Linux to build a true 3D spatial audio system.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**10**	

Although the acoustic quality of these packages is undoubtedly higher than that of our low-cost 3D audio solution in terms of audio clarity and fidelity, both options provide true 3D spatial audio.

When we started putting together a spatial audio system, there was no inexpensive hardware and software combination to produce true 3D spatial audio. While there are software APIs that allow arbitrary (not necessarily co-planar) positioning of sound sources (for example, Microsoft®DirectSound® [5], the open source community's Advanced Linux Sound Architecture (ALSA) [1], OSS [8], and OpenAL [7]), the low-level drivers only officially support the co-planar 4.1, 5.1, and 7.1 speaker positions mentioned earlier. There is no way to tell the drivers that the speakers have been moved to an alternate configuration, for example, with speakers above or below the listener. So even though software developers could position a sound above or below the user's head, the low-level drivers still assumes the sound was emitted in a circle around the user's head. The bulk of the true 3D spatial audio support comes from customized APIs.

Tommi Ilmonen at the Helsinki University of Technology (HUT) developed a 3D spatial audio API called Mustajuuri [6] that is built upon the ALSA drivers. The Mustajuuri API implements Vector Base Amplitude Panning (VBAP), introduced by Ville Pulkki [12], as the underlying 3D spatial audio model. In short, VBAP is the algorithm responsible for moving a sound across a 3D array of speakers and making the sound appear to come from a specific direction. VBAP selects the three speakers closest to the virtual sound position and calculates the required volumes for each speaker. See Figure 1 for an example of how VBAP panning works. Mustajuuri also simulates depth for audio by using time delays and distance attenuation. This makes it possible to position a sound anywhere in space relative to the listener. Mustajuuri has already been used to produce 3D spatial audio using high-end audio cards, but up until now has not supported low-end audio cards.

3 Hardware Selection and Setup

The hardware needed to set up a low-cost 3D spatial audio system includes a commodity sound card with certain features, speakers, and audio cables. We describe our choices for hardware components and the steps needed to set up the hardware. Throughout the discussion, refer to Figure 2 for an illustration of the hardware interconnections, speaker placement, and wiring.

The first thing to consider is the number of speakers that are needed to produce 3D spatial audio for a specific application. A minimally encompassing setup produces sound from all directions about the user (left-right, front-back, and up-down). The speakers can be placed in any configura-
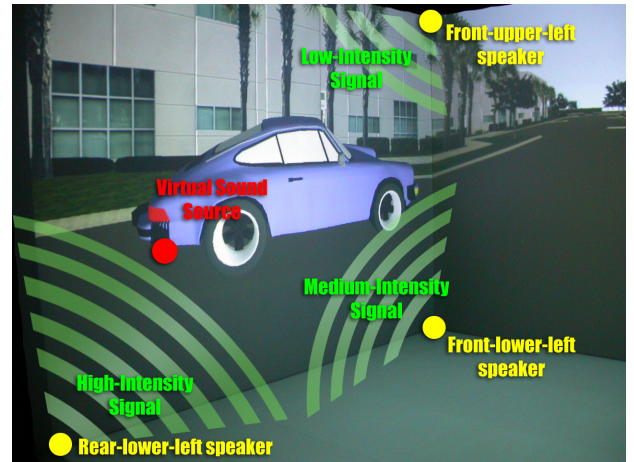

Figure 1: View of 3D spatial audio test case in the immersive room. Visual depictions show from which speakers the sound is coming for the current view. For each sound, the three speakers closest to the virtual sound source are used to play the sound. Their volumes are varied based on the distance from the speaker and a number of other factors.

tion, but setting up the 3D audio panning functions is not as simple for irregular configurations. We decided to use eight speakers in a cubic configuration (each speaker at a vertex of the cube) as shown in Figure 2. There is nothing special about the speakers needed for this task–the choice is a matter of budget and taste. We used eight amplified commercial-grade speakers for the simple reason that we already had them in our lab.

The eight speakers require a sound card that can produce eight channels of audio. Of the low-cost commodity audio cards, the only candidates are the 7.1 cards. We chose the Creative Labs Audigy 2 card which we found available (at the time of writing) for as low as $90 USD. Although it is possible to produce eight independent channels of audio using more expensive sound cards, the Audigy 2 card is the only commodity card we are aware of that has drivers in place to support what we are doing.

It is important to understand the outputs from the card. In a typical Dolby®Surround Sound 7.1 speaker arrangement, there are two front speakers (left and right), two side speakers, two rear speakers, a center speaker (that sits in front above the video screen), and a subwoofer channel (the ".1" speaker). The Audigy 2 ZS has three analog output jacks (1/8-inch mini-phone), labeled 1, 2, and 3, providing line-level outputs for the eight speakers. Jack 1 is three-pole, meaning it carries three signals–two signals drive the front left and right speakers and the third is ground. Jacks 2 and 3 are four-pole, each carrying four signals. Jack 2
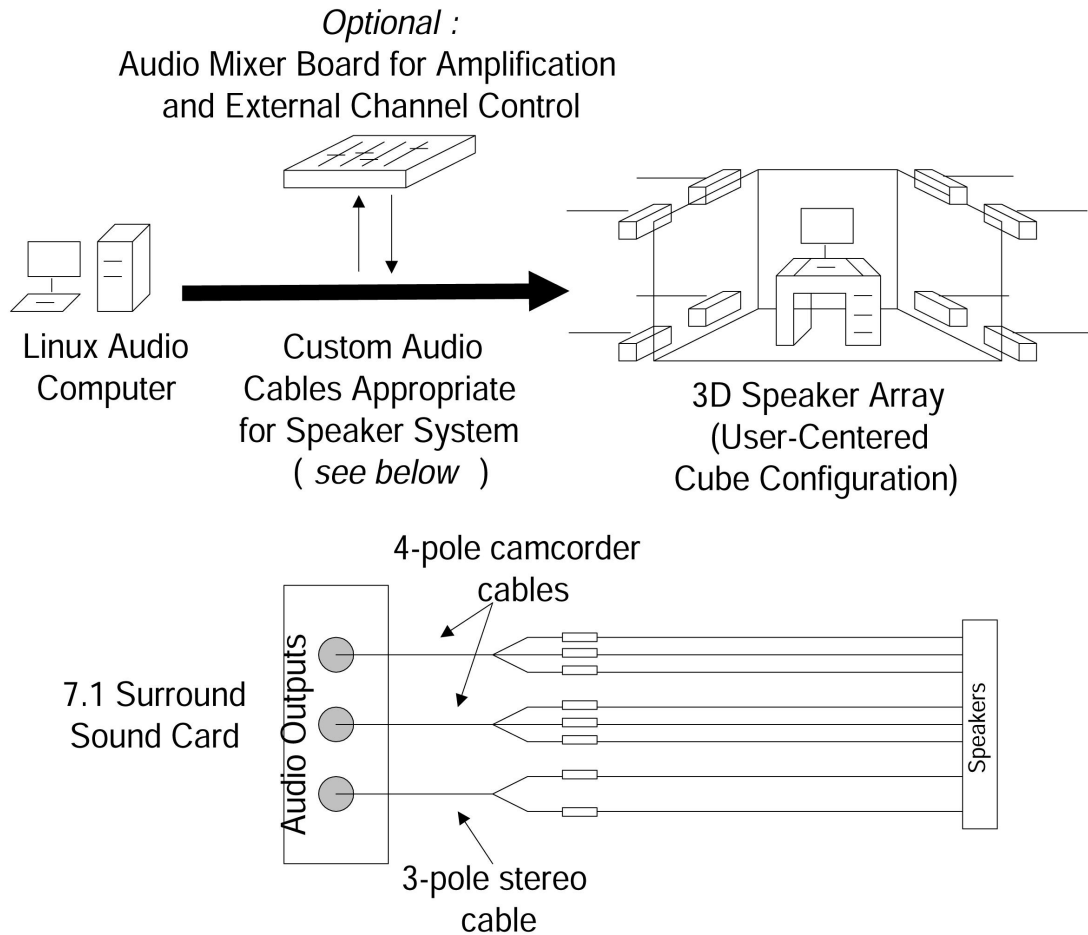
Figure 2: Audio Hardware Setup

drives the rear speaker pair and a side speaker, while jack 3 drives the subwoofer, the center speaker, and the remaining side speaker. One final consideration is that these signals are unamplified line-level, so the speakers need to be the amplified type that accept line-level inputs, or a separate amplifier (or set of amplifiers) should be used between the sound card and the speakers.

The next step is to install the speakers. Many speakers designed for surround usage include mounts, but speaker mounts are available commercially for a number of other speaker types. In our application, we already had the cubic infrastructure in place and used custom mounts to attach the speakers to the cube. The simpler the speaker configuration (placing them on a regular shape, using 45-and 90-degree angles, etc), the simpler the software configuration will be– that process will be explained in a later section.

Finally, the speakers must be connected to the audio card. How one connects speakers to the Audigy 2 will depend on the type of speaker and amplification used. A trip to a favorite electronics store should yield any necessary connectors. In our case, our speakers each have a two-pole 1/4-inch phone jack, so we needed to split the three combined outputs of the sound card into eight separate signals. For Jack 1, we used an easily-available 1/8-inch-stereo-to-dual-RCA adapter. For Jacks 2 and 3, we found a similar adapters with four poles and three RCA connectors. These adapters are most commonly used with camcorders (the three signals used for composite video and stereo audio). These adaptors gave us eight separate RCA connectors, and after obtaining eight long RCA-to-1/4-inch-mono cables, we were set.

In our final configuration, we used an Alesis Studio 32 mixer board. This device fits inline between the audio card's outputs and speakers' inputs and allows fine tuning of the volume levels. Although the board made it a little easier to test and tune the audio, it wasn't truly necessary as the same adjustments can be made in software.

4 Software Selection and Setup

The software solution for low-cost 3D spatial audio is best described by the layered hierarchy shown in Figure 3. The software layers required to interface with the sound cards include low-level audio drivers and a 3D spatial audio API. We focused our primary development efforts under Linux because of easy access to the source code for low-level audio drivers and the overall support community for developers working on projects like this one.
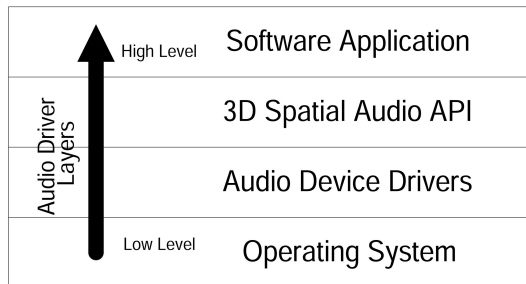

Figure 3: Audio Driver Layers

For the driver layer, we chose ALSA, which was mentioned previously. ALSA provides audio and Musical Instrument Digital Interface (MIDI) functionality to the Linux operating system and supports many types of audio hardware ranging from consumer sound cards to professional multichannel audio interfaces. We selected ALSA since it appeared to require the least effort to generate the eight channels we needed for 3D spatial audio. Until we modified the ALSA driver to access all eight channels, it only supported 6 channels (5.1) on the Audigy 2. These changes have been incorporated into ALSA, but may or may not be in a release version at publication time. In that case, one can get the latest source and build it–be sure to include the "emu10k1" sound card argument when using the "./configure" script so that the ALSA driver will recognize the Audigy card.

After the driver is set up, the 3D spatial sound API can be installed. It will distribute sound effects from a given 3D position to the appropriate audio channels. Although there are quite a few APIs to choose from, we chose Mustajuuri, mentioned in the background section. The Mustajuuri software works with ALSA and provides 3D panning over an arbitrary array of speakers using the VBAP algorithm as described previously. The Mustajuuri API provides all of the features needed for a basic 3D positional sound system and is fairly easy to extend. Over the course of this project, we made several minor source code modifications and they are included in the October 2004 release.

Mustajuuri does its magic via a module called the Mixer, which mixes multiple sound sources (sound files, microphone inputs, or other sources) into individual audio streams. These streams are then piped into a panning module, which is responsible for routing each input signal to the appropriate speakers, setting the correct gain and time delay at each speaker, and mixing multiple streams meant for the same speaker into a single stream to be sent to that speaker. It does the routing and gain calculations based on VBAP, and some additional gain and delay calculations based on distance. The result is that each incoming sound source to the panning module leaves to a set of three speakers, and the resulting sound appears to come from a specific 3D position in space. Doppler shifting is also simulated.

Once Mustajuuri is compiled and installed, there are several tasks that must be performed to configure the software to work with the given 3D speaker array.

Task 1: Configuring ALSA

ALSA needs to know how to communicate with all eight channels of the audio card. This would normally be achieved by simply using the device named "surround71", however it is not fully compatible with the spatial sound API Mustajuuri. Mustajuuri requires support for input channels. The device "surround71" supports eight output channels, but not any input channels. Therefore it is necessary to define a new device that has eight output channels and some input channels.

In order to meet this requirement, an "asymmetric" device is defined. The device is called "asymmetric" because the number of input and output channels are not necessarily the same. Note that the number of input channels are not explicitly stated. ALSA determines the number of input channels automatically and assigns the maximum (Audigy card has two).

To configure ALSA, add the following text into the file "/etc/asound.conf" (or create the file if necessary). This file holds information about user defined devices, and so we use the following text to add an asymmetric device called "eightout".

```
ctl.eightout {
    type hw
    card 0
}

pcm.eightout {
    type asym
    playback.pcm {
        type route
        slave.pcm surround71
        ttable.0.0 1
        ttable.1.1 1
```

```
            ttable.2.2 1
            ttable.3.3 1
            ttable.4.4 1
            ttable.5.5 1
            ttable.6.6 1
            ttable.7.7 1
        }
        capture.pcm {
            type hw
            card 0
        }
    }
```

Next, an environment variable must be set to allow Mustajuuri to talk to the audio card through ALSA. Set the following environment variable:

```
export MJ_AUDIO_CONF=
"Input=2=hw:0,0 | Output=8=eightout"
```

Once this is done, Mustajuuri should be able to output audio through all eight channels of the audio card.

Task 2: Configuring the Mustajuuri Mixer Panel
Mustajuuri uses a mixer-board style GUI for sending input audio streams to a speaker array, combining them or just passing them through intact. The input streams can either come from sound files or from live sources (such as a microphone). The GUI lays out several strips of channels that can be assigned different functions that are applied in a sequential process. Some example functions are input, send (to speaker), amplitude gain, panning and synthesizer. The gain and panning modify how the audio is distributed to individual output audio channels.

The Mixer Panel configuration we use can be seen in Figure 4, which uses two mixer strips. The first has two interesting channels: a synthesizer channel, which manages the sound files, and a panning module, which handles the VBAP-based panning across speakers. The second strip is used to manage remote connections from external applications and does not accept an audio stream as input. It sends commands to the synthesizer and the VBAP module.

To create a similar configuration, launch Mustajuuri and create a new mixer from the "File" menu. This mixer will have several strips already, and all of these strips will be essentially blank. The number of strips and the number of modules per strip can be changed using the "Edit" menu if needed. Modules can be assigned by clicking with the mouse on a particular slot. To adjust the module's properties, just click on the blue link defining the module's type (i.e. "Synth1", "Mixer Input", etc.). The "Strip X" button at the top of a strip can be used to modify and remove the modules in any slot in that strip. All mixer configuration changes are saved by using the save options from the "File"
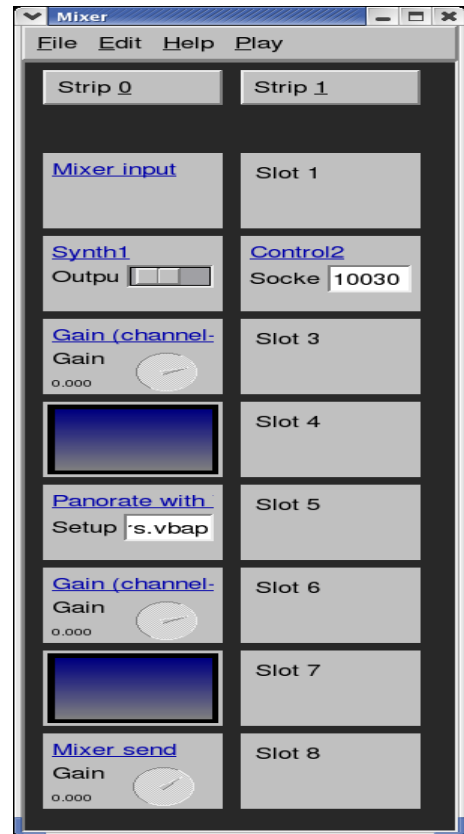

Figure 4: Screenshot of the Mustajuuri Mixer GUI (Tommi Ilmonen [6])

menu. The resulting configuration file (e.g., SpatialAudio.mj) is specified on the command line (e.g., "./Mustajuuri SpatialAudio.mj") when Mustajuuri is called.

Task 3: Specifying Speaker Placement
In order to use VBAP, it is necessary for Mustajuuri to know the locations of the speakers in the 3D array. Mustajuuri does this through a configuration file that is specified as part of the VBAP panning module setup (this module was created as part of **Task 2**). This file specifies the azimuth and elevation angles (in degrees) for each speaker relative to the listener. Since our system uses eight speakers arranged in a cube configuration, our configuration file is specified as follows:

```
3        # dimensionality
# Azimuth, followed by elevation.
# E.g.,0 0 would be straight ahead.
-45 45   # Front upper left
45 45    # Front upper right
-135 45  # Back  upper left
135 45   # Back  upper right
```

```
-45 -45  # Front lower left
45 -45   # Front lower right
-135 -45 # Back  lower left
135 -45  # Back  lower right
```

This configuration file is used by the main configuration file for the Mustajuuri Mixer. It should be noted that this file assumes that all speakers are equidistant from the listener. If this is not the case, it will be necessary to adjust the gain and delay for each speaker (using the Mustajuuri Mixer) manually. In our system, this was not necessary and it would involve significant work if it were necessary. The easiest solution is to try to place all speakers equidistant from the ideal listening position (i.e. the center of the environment).

Task 4: Configuring Sound File Loader

In order to use a sound file from Mustajuuri, that sound file must be known to Mustajuuri at load time. The mechanism used to do that is a configuration file that specifies all sound files that might possibly be used by Mustajuuri. This configuration file is used by the synthesizer mixer module (created in **Task 2**). A sample configuration file that loads three sound files follows. Once a file is created, any name can be assigned, and make sure the synthesizer module points to that file.

```
unusevoices *-stk
polyphony 48

sample audioeffect1.wav
sample audioeffect2.wav
sample sudioeffect3.wav
```

The polyphony line specifies the maximum number of audio files that will be loaded by Mustajuuri (so it should be at least as large as the number of audio files listed in this file). The last three lines specify three sample audio WAV files. Any audio files specified here must be placed in a directory that is specified separately as part of the synthesizer module configuration. Note that the "unusevoices" line is a somewhat more advanced setting, but should help improve efficiency somewhat.

Task 5: Configuring Mustajuuri for Remote Control

Mustajuuri is designed to act as a standalone program to manipulate audio, not as a library to be linked against with another application. In order to control Mustajuuri from another application, as is the case with our project, two steps are required. The first involves setting up Mustajuuri to listen for control commands over the network (this task). The second includes writing a simple API in the main application to talk to Mustajuuri (the next task).

To get Mustajuuri to accept commands from the network, a network module must be loaded. This network module is the only way for an external application to control Mustajuuri, even if both the application and Mustajuuri are running on the same machine. Adding this module is very simple and only requires configuring which port Mustajuuri will listen to (the default port is 10030). This module, which should have been created in **Task 2** above, will automatically communicate with the synthesizer module and the VBAP panning modules, if they are in the mixer.

Task 6: Interfacing with Mustajuuri API

In order to control Mustajuuri from an application, it is necessary to add code included with the Mustajuuri API into the application. We present example code segments showing how to connect to a remote audio server, play audio specified coming from a given 3D position, and change the position of a sound source and listener position.

The first code segment shows the initial commands to connect the application to the remote Mustajuuri audio server and initialize it. If Mustajuuri is running on another machine, change the address to reflect this. To change the default port (10030) that Mustajuuri listens to, this is accomplished by specifying the new port in the address string (e.g., "mjserver.mydomain.com:12345").

The two objects that we created, one instance each of AC_Control and AC_VrControl, will be used later to send commands to Mustajuuri.

```
// connecting to a remote server
  #include <ac_vr_control.h>

  AC_Control acControl =
    new AC_Control();
  char* mjServerAddress = "127.0.0.1";
  if(!acControl->init(mjServerAddress))
  {
    // error handling code here
  }

  AC_VrControl acVrControl =
    new AC_VrControl( acControl );
```

The next code segment shows how to specify the position of the source of the audio and play it. One interesting thing to note here is that the variable "outputChannel" identifies the intended sound source to work with. The number of supported sound sources was specified in the synthesizer module from **Task 2**, and "outputChannel" should be between 0 and the number of sources minus one. The variable "soundFilename" should not have a path as part of the filename. The filename should be one of the files listed in the configuration file created as part of **Task 4** (the sound file

loader configuration file). Lastly, the soundLevel is essentially the initial gain level for the new sound. This will need to be experimented with to find an appropriate setting.

```
AC_Vector3 location(
   positionX,
   positionY,
   positionZ);

int soundId =
   acVrControl->playSample(
      outputChannel,
      soundFilename,
      soundLevel,
      location,
      true,
      0, 0);
```

The last code segment shows how to reposition the sound source location and the orientation and position of the listener. The "outputChannel" variable refers to the sound source that is desired to be moved, and will be the same value that was used to call "playSample" from the previous example. The "listenerRotation" matrix specifies the orientation of the listener relative to the world, and the "worldRotation" matrix specifies the orientation of the world relative to the speakers.

```
// reposition a source of sound
  AC_Vector3 location(
    positionX,
    positionY,
    positionZ);

  acVrControl->moveSource(outputChannel,
    0.05, location);

// reposition the listener orientation
  AC_Matrix3 listenerRotation(
    ... listener rotation matrix ... );
  AC_Matrix3 worldRotation(
    ... world rotation matrix ... );

  acVrControl->setTransformations(
    location, listenerRotation,
    worldRotation, 0.05);
```

5 Hardware Testing and Calibration

We tested the hardware design of our 3D spatial audio system by integrating the hardware with our 4-wall immersive virtual reality room at the Virtual Reality Laboratory, part of the Naval Research Laboratory in Washington D.C. We arranged the speakers in a cube array and placed them at
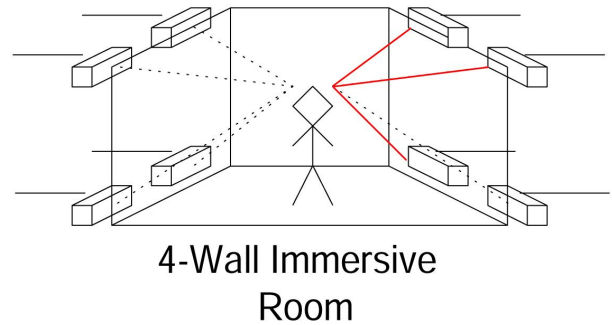

4-Wall Immersive Room

Figure 5: Immersive room depiction showing placement of speakers in a cube array and audio coming from the user's front-upper-right direction (lines are colored red).

the corners of the immersive roon as shown in Figure 5. We designated a 1.2 GHz Redhat Linux machine as an audio server and installed in it an Audigy 2 ZS card. We connected the speakers using the cabling described before, and tried the system both with and without the mixing board mentioned earlier.

CPU utilization on this machine while Mustajuuri was running with 3 audio sources in motion was generally less than 20%, and the memory usage was negligible. Further savings could of course be realized by using optimized compiler settings, rather than the debug settings we used.

We tested the outputs from each speaker to determine the range of intensities that could be played on each channel. We listened to each speaker individually for sound quality, sound balance, and percussive resonance. The easiest aspect to listen for is the sound balance between treble and bass. If one of the two is obviously higher than the other, adjust the necessary frequency filters as needed. For example, if there is too much bass, decrease the bass and/or increase the treble. If there seems to be excessive low end or high end noise, adding a low-pass or high-pass filter may be necessary.

Another easy aspect to listen for is speaker distortion. Simply put, if the speakers are so loud that the sound produced is bad, lower the volume of the speaker. If a given set of speakers will not produce quality sound at an acceptable volume, it may be necessary to get more powerful speakers.

One of the hardest aspects to listen for is the resonance of percussive sounds generated by a speaker. This quality is basically how much the sound echoes from where the speaker is located. Adjustments have to be made if it sounds like a speaker is reverberating with percussive sounds. Depending on the quality of the speaker, and the quality of the mixer board, this problem may be corrected to some degree by continuing to filter the signal. For excessively bad cases, hard objects such as exposed metal, concrete, hard plastic,

and even glass should be covered with a sound dampening material like cloth or foam.

Once each speaker is calibrated, the entire setup has to be balanced. It can either be done with devices designed to measure acoustic levels or by listening to the speaker from the pre-determined center of the 3D speaker array. Either way, the gain of each speaker should be adjusted until the same audio intensity level is received from each channel. Keep in mind that the outputs for each channel of the audio card were customized by the manufacturer for the intensity requirements for each type of speaker (satelite, center, and subwoofer) normally attached in a surround configuration, and the intensity output for each type differs. There are many published methods for dealing with this problem, but we went with the low-tech solution of having someone stand and listen at the center of our speaker array. We set the software control to maximum gain and adjusted the mixer board based on feedback from the listener. Remember that these changes can be made in software with the ALSA drivers and Mustajuuri if a mixing board is not available.

6   Software Testing

We tested the software by integrating sound into an existing in-house simulation platform, BARS-Utopia [11], that operates on a Linux visualization cluster (ORAD Incorporated, orad-ny.com) which drives our immersive room. BARS-Utopia supports several virtual world databases, interaction methods, and spatial audio. However, no support was available for interacting with the Mustajuuri API in particular, so we implemented a plugin to bridge the BARS-Utopia spatial audio support with Mustajuuri. BARS-Utopia already contains all of the information needed by Mustajuuri, such as sound source positions, listener position and orientation, and sound source creating/deletion notifications–the plugin simply translates that data into a form that Mustajuuri understands.

When the plugin was completed, we tested and debugged the new system. The primary software adjustments we made were to the attenuation level of the audio channel outputs. Mustajuuri uses a simple attenuation model and requires some manual tweaking for the expected environment (i.e. outdoor, indoor, time of year, etc.). In the real world, sound attenuation rates are quite complicated and are influenced by factors such as temperature, humidity, and the frequency makeup of the sound.

We tested the sound system by implementing several scenarios, each with a different scene dataset and different audio effects attached to an animated object. Before the audio objects were animated, we evaluated several volume levels and several distances away for each object. Figure 1 shows a simple scenario we designed and tested the sound effects of a car. When we finished testing the volume and distance effects, we generated an animated path for the car to follow.

Figure 6 shows a more complex scenario with three audio sources (tank, jet, and helicopter)—the jet is off the screen and not shown in view. We performed some simple tests to see how many sound sources interacted together. It was of primary importance that the jet (typically far away) not sound too quiet and the tank and helicopter (typically closer to the camera) not dominate the aureal bandwidth, so some minor tweaking resulted for both the far and near objects' attenuation parameters.

7   System Validation

After all of the testing and calibration was completed, we performed two informal, qualitative user tests that would help us validate our new low-cost spatial audio system. The first test evaluated how the new sound system configuration with eight speakers compared with our previous planar configuration containing four speakers. The prior configuration simply used the four speakers on the top of the cube array. We realize that directly comparing these two configurations is somewhat biased, due to the placement of the 4-speaker array being located above the user's head–it would be fairer to compare against a 4-speaker array located at the height of the user. However, by using the top four speakers, we were able to switch between the two configurations without dismantling our installation.

We performed the experiment by asking a few test subjects to stand in the middle of the immersive room and listen to sounds played for each configuration. We played different sequences of audio on both speaker configurations and made use of the full range of speakers available. The subjects were not told which configurations were being used, nor in which order the pairs of configurations were presented. Several iterations of the pairs of configurations were tried for each subject. After each pair was presented, the subjects rated the two systems. Admittedly, this was not a scientific test as is evidenced by several unaddressed biases, but all test subjects clearly preferred the 8-speaker configuration.

The second user test evaluated how well the listener is able localize the source of the audio using the 8-speaker configuration. Again, the subjects were tested and each were asked to stand in the center of the immersive room. Each subject was presented with several sounds played one at a time and originating from different positions surrounding the subject. The subjects were asked to point in the direction of the sound source, as they heard it. The visual system was not running, so the users did not get visual cues as to the sound source's location. The subjects were able to localize the sounds with a high degree of accuracy, especially with respect to elevation.
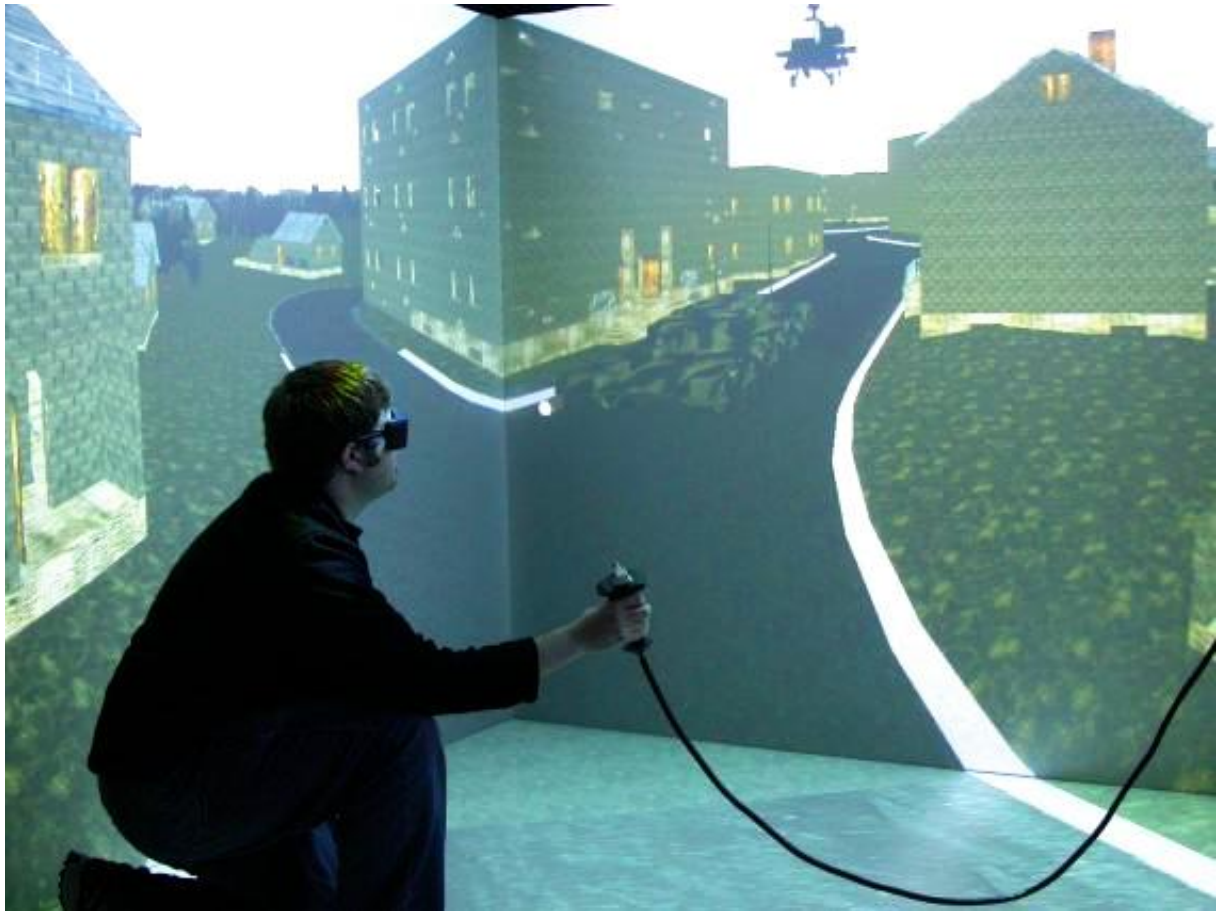
Figure 6: Shows a user interacting with a scene with multiple sound sources (a tank, helicopter and a jet (not visible)). The nearest speakers are determined for each sound source and the output is mixed at each speaker (if overlapping).

The implementation of our 3D spatial audio system integrated with our immersive room really enhanced the simulation and training demos we have. Our completed system has dramatically improved the sense of immersion when running the demos. A simulation user easily perceives helicopters and jets flying overhead and a tank rumbling down one of the many streets nearby in the virtual world. The perception of depth from the source of audio is conveyed very accurately and also includes doppler effects. Our system is a step above a 4-speaker solution we had previously using the Microsoft®DirectSound®API. It was also a good replacement for the very capable but outdated and unsupported 8-speaker solution we had running using another very expensive hardware and software platform.

8 Conclusions and Future Work

We have devised a true 3D spatial audio solution that is low cost and has comparable quality to high-end expensive commercial systems. The 3D spatial audio solution allows sound effects to be generated from all directions surrounding a user, not only planar directions. We accomplished this feat by using only commodity hardware and open source software. We feel this feature, now available at an affordable price, will create numerous options for game and virtual reality system developers.

We feel our system will lead the way for others to devise similar solutions with current and future commodity audio equipment. The developer needs only to purchase a Dolby®Surround Sound 7.1 audio card, four pairs of low-cost speakers, and audio cables. We spent less than $150 USD on hardware (Audigy 2 audio card and audio cables) since we already had speakers available. From start to finish, including hardware and software debugging, configuring and testing, we spent less than a month developing the low-cost 3D spatial audio system. We feel that using this document as a guide, it should be possible to implement

this system in less than a week. Obviously, if a currently unsupported audio card is used, there will be time and expertise needed to write a driver for it.

Although the system currently meets our needs quite nicely, these features would be nice to add to the 3D spatial sound API in the future:

- **Directional Sound Cones:** Directional sound cones are a mechanism to provide directional sound with the strongest intensities propagated along the central axis of the cone and weakest towards the edges. Since many sound sources are directional by nature (e.g., sound emanated from a megaphone), directional sound cones would allow these sound sources to be more accurately generated. Also, since some major APIs (such as Microsoft®DirectSound®) offer sound cones, it would be nice to offer such a feature.

- **Additional Environmental Reverberation Effects:** Although there are a number of very simple environmental reverberation effects available in the common 3D spatial audio APIs, supporting more sophisticated effects such as sound reflection and absorption off of different surfaces would greatly enhance the listening experience. This is an area of ongoing research, and the Mustajuuri system would be a good testbed for trying out new techniques.

- **Enhanced Sound Attenuation Level:** The current distance attenuation models for sound in Mustajuuri are quadratic by nature, thus are fairly simple. In the real world, sound attenuation is much more sophisticated and depends on heat, humidity, sound frequency, and many other factors. For example, the low frequency sounds generally carry much farther than high frequency. Accounting for these complexities could help significantly in providing distance cues.

References

[1] Advanced Linux Sound Architecture (ALSA). www.alsa-project.org. Last accessed December 14, 2004.

[2] Fantasound. http://en.wikipedia.org/wiki/Fantasound. Last accessed December 14, 2004.

[3] Lake Audio. www.lakeaudio.com. Last accessed December 14, 2004.

[4] M-Audio. www.m-audio.com. Last accessed December 14, 2004.

[5] Microsoft DirectX: Home Page. www.microsoft.com/windows/directx. Last accessed December 14, 2004.

[6] Mustajuuri: Audio Application and Toolkit. www.tml.hut.fi/~ilmonen/mustajuuri. Last accessed December 14, 2004.

[7] Open Audio Library (OpenAL). www.openal.org. Last accessed December 14, 2004.

[8] Open Sound System (OSS). www.opensound.com. Last accessed December 14, 2004.

[9] RME Intelligent Audio Solutions. www.rme-audio.com. Last accessed December 14, 2004.

[10] Richard O. Duda. 3-D Audio for HCI. interface.cipic.uc.davis/CIL_tutorial/3D_home.htm, 2000. Last accessed December 14, 2004.

[11] S. Julier, Y. Baillot, M. Lanzagorta, D. Brown, and L. Rosenblum. Bars: Batlefield augmented reality system. In *NATO Symposium on Information Processing Techniques for Military Systems*, Istanbul, Turkey, October 2000.

[12] V. Pulkki. *Spatial Sound Generation and Perception by Amplitude Panning Techniques*. PhD thesis, Helsinki University of Technology, 2001.