M.Eng Project Report

Implementation of Simple Sound Spatialization Algorithm in Android (v1.0)

Jong Hwi Lee (jl658)

Team OpenComm

Cornell University

May 23, 2010

Contents

1 ABSTRACT 2

2 INTRODUCTION 3

2.1 OpenComm Project 3
2.2 Android Platform 3
2.3 Sound Spatialization and Head-related Transfer Function 3
2.4 Related Works 4

3 DEVELOPMENT ENVIRONMENT 5

3.1 Eclipse with ADT 5
3.2 Operating Systems and Hardware Specification 5

4 BACKGROUND PHYSICS 6

4.1 Assumptions 6
4.2 Interaural Time Difference 6
4.3 Volume Difference 7

5 APPROACH & CHALLENGES 8

5.1 Byte Manipulation 8
5.2 Use of AudioTrack 8
5.3 Use of Two AudioTrack Objects 8

6 IMPLEMENTATION 9

6.1 Configuration 9
6.2 Implementation of AudioPlayer Class 9
6.2.1 Constructor 9
6.2.2 play() 9
6.2.3 pause() 9
6.2.4 stop() 9
6.2.5 write() 9
6.2 Sample Usage 10

7 FUTURE WORKS 11

7.1 Raw Audio Stream Manipulation & Use of Experiment Result 11
7.2 Full 3D Spatialization 11
7.3 Stress Test 11
7.4 Trial on Android Phone 11

1. Abstract

Simple audio spatialization API was implemented on Google Android platform. The API enables users to give a sense of direction of any 2D direction in front and distance from the input audio stream, when a listener hears the sound from a headphone. For the purpose of OpenComm project, it is optimized to fit to the situation of audio conference, thus focus less on the accuracy of spatialization and is designed to be able to quickly play multiple audio streams simultaneously and clearly, each with different direction and distance. Head-related transfer function was studied for better spatialization.

This report includes the physics behind it and its implementation using Android SDK, examines the achievements and challenges undergone so far, and analyzes the problems that occurred during the project.

The introduction and general use of Android and existing OpenComm code is omitted from this report, because the content heavily overlaps corresponding sections in OpenComm Design Documents & Developer Handover. Interested readers are recommended to read it.

2. Introduction

2.1 OpenComm Project

OpenComm is the project name at Cornell University under Professor Graeme Bailey. The goal of this project is to develop a stable open-source VoIP conference application on Android platform. One of its prominent features is that each participant in audio conference has their "physical" – though virtual - location as in a real conference, and their voice comes to the listener as if they are talking from that virtual location. The sound spatialization algorithm is used in OpenComm project for this purpose.

2.2 Android Platform

Android is a mobile operating system developed by Google, Inc. It uses a modified Linux kernel and runs on smart phones that support this platform. Google provides the standard development kit (SDK) on Android platform, which runs on a variant of Java virtual machine called Dalvik and includes an emulator on major operating systems. This project uses version 2.1 of Android SDK released on January 12, 2010.

In 2.1 of Android SDK, the package android.media includes useful Java classes needed for audio manipulation. As Dalvik lacks the Java Sound API in Sun Microsystem's standard Java virtual machine, this project mainly utilizes Android's AudioTrack class as an alternative.

The detailed information about Android platform and the use of emulator can be found in "Opencomm Design Documentation & Developer Handover" by Jha, Kobelski, Lin, or in the following links.

Android Developers Official Website: http://developer.android.com

Android Community: http://androidcommunity.com

2.3 Sound Spatialization & Head-related Transfer Function

Head-related transfer function (HRTF), sometimes known as anatomical transfer function (ATF), is the study on how people perceives sound and localize the source from their two ears. Even though humans

have two ears, they are able to localize 3D sounds, and the exact way differs from the shape of head and ears. Complete 3D sound simulation is difficult, as it depends on a lot of factors: individual difference, temperature, humidity, reverberation of the location, the pitch of the sound, and so on. Still, approximate 3D sound simulation is widely used in games and movies.

2.4 Related Works

When it comes to Android platform, however, there seems to be no previous work on sound spatialization. However, there were a number of experiments done on 3D sound spatialization.

Periphonic Sound Spatialization in Multi-User Virtual Environments by Florian Hollerweger

HRTF Measurements of a KEMAR Dummy-Head Microphone by Bill Gardner and Keith Martin, MIT Media Lab (comprehensive measurement data from different angles)

Head Related Transfer Function by Brett Patane and Eric Brumer, for ECE 576 course project at Cornell University (coded in Velilog)

3. Development Environment

3.1 Eclipse with ADT

The API was developed in Eclipse 3.5.2 with ADT. ADT (Android Development Tools) is an eclipse plug-in that facilitates application development on Android platform. A project that will be used in Android platform can be created in Eclipse from ADT. It generates code skeletons, automatically builds necessary files and constants, and links Eclipse and Android Emulator so that the application can be tested in it before actually ported into an Android phone. The following website contains the guide for ADT installation.

ADT on Eclipse: http://developer.android.com/guide/developing/eclipse-adt.html

3.2 Operating Systems and Hardware Specification

The API was tested in Android Emulator on Windows XP SP 3 (x86) and Windows 7 (x64), in a PC with built-in soundcard. They were not virtual OS's. The CPU and motherboard specification is described below.

CPU: AMD Athlon 64 X2 2.6GHz Dual-Core Black Edition Model ADO5000DSWOF

Motherboard: GIGABYTE GA-MA69G-S3H, with audio chipset Realtek ALC889A

4. Background Physics

4.1 Assumptions

Because of the limit of Dalvik and mobile devices in terms of OS support, CPU speed, and memory amount, the algorithm covers only the front of the listener in 2D plane, because simulation of full 3D sound effect requires much more detailed calculation. Also, this project only considers the two most important factors of sound spatialization: the time and the difference of sound intensity between two ears.

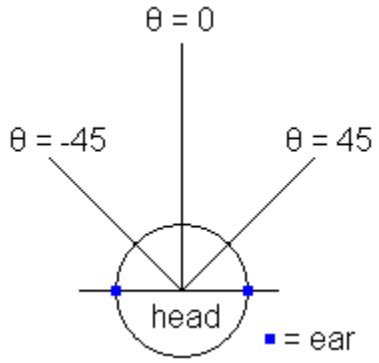
There are a few more assumptions to simplify the environment. First of all, the head shape is assumed to be a sphere of radius 11.0cm, and the ears are considered to be points lying on the opposite ends of the sphere (thus no outer ear). Also, sound travels through dry air, in which the speed of sound is calculated from the formula

$$s_{air} = \left(331.3 + (0.606\ ^{\circ}C^{-1} * T)\right)\ m/s$$

where T is the temprature in celcius. The temperature is assumed to be 20℃, and the speed of sound is therefore 343.42m/s.

4.2 Interaural Time Difference

When the angle θ is defined from the center of the face to the right, it becomes positive to the right side of the nose, and negative to the left side of the nose. For example, when the angle completely lies on a half line from the center of the head to the left ear, θ becomes -90 degree.



The interaural time between two ears follows the formula:

$$\text{interaural time delay } = \frac{\text{radius of head}}{(\text{speed of sound})} * \left(\theta + \sin(\theta)\right)$$

Negative delay means that the left ear first hears the sound. Here, θ is radian.

4.3 Volume Difference

According to distance law, sound pressure (volume) is inversely proportional to the distance. In the implementation, the volume starts from 1.0 (max) and stays the same till threshold distance (set to 15cm), and decreases as (distance - threshold) increases.

The difference in distance between two ears is calculated from the interaural time delay. Since the speed of sound was assumed to be constant, from the time delay the difference of distance can be obtained, and from it the actual distances of each ear from sound source can be calculated.

When the distance from the center of head to the sound source is denoted as d,

$$\text{distance differece} = \text{interaural time delay} * \text{speed of sound}$$

$$\text{distance to left ear} = \text{d} + \frac{\text{distance difference}}{2}$$

$$\text{distance to right ear} = \text{d} - \frac{\text{distance difference}}{2}$$

The volume to each side is set accordingly.

5. Approach & Challenges

5.1 Raw Audio Stream Manipulation

In the lowest level, sound is represented as raw stream of bytes. The initial approach for the spatialization algorithm utilizes byte-level manipulation. This required digital signal processing and above all, no proper way to convert stereo sound into Java byte stream could be found, which means that even though sound is spatialized, it was not possible to play it. An alternative to raw stream manipulation was needed.

5.2 Use of AudioTrack

The second approach was to use Android's AudioTrack class, because of its setStereoVolume() method. By the method, it was possible to set each side to different volumes. However, it still does not give reasonable extent of spatialization because interaural time difference was not considered yet.

5.3 Use of Two AudioTrack Objects

The third approach was to use two AudioTrack objects with some interval in play time. Each object handles one of left or right sides only, and the volume can be independently set by setStereoVolume() method (in this case, the volume of one side should be 0.0). If AudioTrack's play() method were executed not simultaneously, it can give a sense of time difference between two sound streams. The implemented code uses Java's Thread.sleep() method to play with interval.

6. Implementation

6.1 Configuration

The API does not require any pre-configuration to use. However, in the future work, it is being planned to let users be able to configure a set of assumed parameters (head shape and radius, temperature and speed of sound).

6.2 Implementation of AudioPlayer Class

The API consists of one class: AudioPlayer. It contains two AudioTrack objects, which are instantiated according to section 6.1 of OpenComm Design Documentation & Developer Handover.

6.2.1 Constructor

Method signature: AudioPlayer(float degree, float distance)

The constructor has two parameters: the angle in degree, the distance in meter. The constructor calculates both AudioTrack's volumes and the interplay time.

6.2.2 play()

Method signature: void play()

This method calls play() of both AudioTrack objects, with corresponding interval in between.

6.2.3 pause()

Method signature: void pause()

This method calls pause() of both AudioTrack objects, with corresponding interval in between.

6.2.4 stop()

Method signature: void stop()

This method calls stop() of both AudioTrack objects, with corresponding interval in between.

6.2.5 write()

Method signature: void write(short[] data), or void write(byte[] data)

This method calls the same method of AudioTrack objects and writes the PCM audio data into them. It is possible to write data while it is being played.

6.3 Sample Usage

Below code plays sample file when it is played 1m away and 45 degree right from the listener.

```
AudioPlayer player = new AudioPlayer(45.0f, 1.0f);
FileInputStream fis = new FileOutputStream(new File(“sample.wav”));
byte[] arr = new byte[BUFFER_SIZE];
while(fis.available() > 0) {
        int read = fis.read(arr);
        player.write(arr, 0, read);
}
player.play();
```

7. Future Works

7.1 Raw Audio Stream Manipulation & Use of Experiment Result

For more accurate spatialization, we can use previous research data on spatialization, in related work sections. They have comprehensive experiment result from different distances and angles with different sound. One disadvantage is that the amount of required memory to store all mapping from (distance, angle) state may be too expensive for a mobile device.

7.2 Full 3D Spatialization

Not likely, but it is possible that full 3D spatialization of sound sources helps listeners distinguish between conference participants, especially when it is packed with many (> 30) people in which simple 2D spatialization may not be enough.

7.3 Stress Test

The API was tested for no more than 5 simultaneous operations. It needs to be also stress tested, possibly up to > 100 participants in a conference.

7.4 Trial on Android Phone

Throughout the development, Android emulator was used for this project, and byte stream from a file, not an actual stream from mic, was the only sound source attempted because for some reason the emulator crashed whenever microphone was tried. Trial of this API on Android Phone is necessary to be shipped to be actually used as a stable application, both locally and over the network.