Android Audio Processing

A PROJECT REPORT

SUBMITTED ON COMPLETION OF THE PROJECT

'ANDROID AUDIO PROCESSING'

UNDER THE

STUDENTS-UNDERGRADUATE RESEARCH GRADUATE EXCELLENCE PROGRAM 2022

BY

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR (U.P)

Submitted by:

Saransh Shivhare (2230618)

Under the guidance of

Prof. Vipul Arora

DEPT. OF ELECTRICAL ENGINEERING

IIT KANPUR (U.P)

MAY 2022-JULY 2022

CERTIFICATE

This is to certify that the project entitled "Android Audio Processing" submitted by Saransh Shivhare (2230618) as a part of Student Undergraduate Research and Graduate Excellence Program 2022 offered by the Indian Institute of Technology, Kanpur, is a bonafide record of the work done by him under my guidance and supervision at the Indian Institute of Technology, Kanpur from 13th May, 2022 to 14th July, 2022

Dr. Vipul Arora
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur

ACKNOWLEDGEMENT

Firstly, I will thank IIT Kanpur for giving me such an opportunity in which I learned a lot, and thanks again for giving me an opportunity to work under the supervision of such an inspiring mentor, Prof Vipul Arora. I am extremely grateful to Sir for encouraging me at every moment. He also motivated me to work on a relevant problem of Tone Detection, which I am looking forward to work upon in my upcoming semesters. I am thankful to him for guiding and mentoring the project. I am thankful to Suraj Jaiswal(Project Associate) for helping me understand the concepts and theoretical aspects related to this work. This research-based internship was under constant support and inspiration from my parents, who supported me throughout by being a source of motivation for me. Thanks to everyone who supported and encouraged me.

Saransh Shivhare

ABSTRACT

This research project focuses reducing the audio playback latency problems observed in devices with the Android operating system. It involves building a front-end android application for signal processing and machine learning tools.The android application has Oboe library implementation for recording audio with minimum latency.
Recorded multiple samples to analyze the dependency of audio latency on sample rate, amplitude gain, burst size and audio buffer length. Further made modification in buffer length and calibrated the latency value on a specific range according to the audio latency obtained on testing the application on SAMSUNG and REDMI systems .

Contents

1. INTRODUCTION

1.1 Overview

This research project focuses on reducing and stabilizing the audio playback latency problems observed in devices with the Android operating system. The project involves research as well as software development aspects. On the research aspect, it involves a thorough analysis of the C++ Oboe library and its working mechanism.

The sample application developed in the project has a recorder and player implementation using Oboe Library. The aim was to create a low latency audio recorder and player to ensure better user experience and results of the recordings in the Narottam Application. (Narottam is a music learning Android Application to bridge the gap between music teachers and students).

2. METHOD

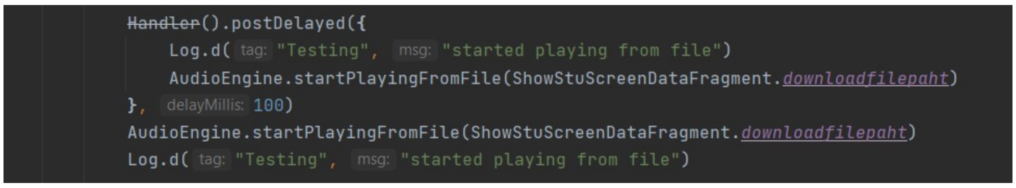
2.1 About C++ Oboe Library

Oboe is an open-source C++ library designed to help build high-performance audio apps on Android. Oboe provides a single native API that works in Android 4.1 (API level 16) and higher.
Use of Oboe is to gain the following benefits:

- Achieve the lowest latency. Oboe helps your application achieve the lowest-possible audio latency for a given device and Android version combination.
- Use the best available native library. On devices running Android API 8.1 (API level 27) and higher, Oboe uses AAudio. For devices running lower versions, Oboe uses OpenSL ES.
- Avoid audio bugs. Oboe includes workarounds for some known audio issues that manifest on specific devices or versions of Android. Using Oboe helps your application avoid these issues without having to implement or test your own solutions.

2.2 Use of Handlers in Java/Kotlin code

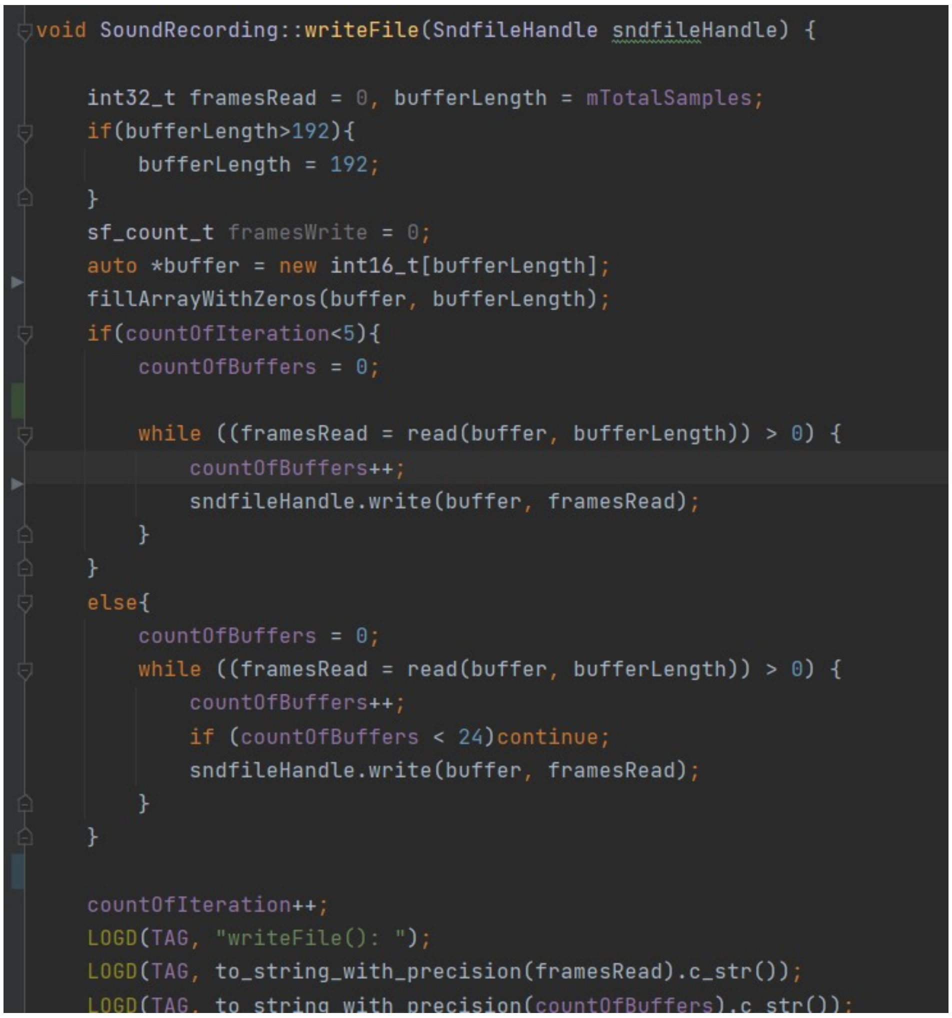
A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler it is bound to a Looper. It will deliver messages and runnables to that Looper's message queue and execute them on that Looper's thread.
The handler has been used in the application to ensure a pause of some milliseconds between the sound recording and playback functions to the average latency observed in the application



```
~~Handler~~().postDelayed({
    Log.d( tag: "Testing",  msg: "started playing from file")
    AudioEngine.startPlayingFromFile(ShowStuScreenDataFragment.downloadfilepaht)
},  delayMillis: 100)
AudioEngine.startPlayingFromFile(ShowStuScreenDataFragment.downloadfilepaht)
Log.d( tag: "Testing",  msg: "started playing from file")
```

2.3 Adjusting initial buffers

An audio buffer holds a single buffer of audio data in its mData field. The buffer can represent two types of audio: A single, monophonic, noninterleaved channel of audio. Interleaved audio with the number of channels set by the mNumberChannels field. In computer based audio systems a certain amount of latency, known as audio buffering, is necessary to ensure that playback, recording and processing results in an error-free audio stream. However, if the buffer size is set too low, then crackles, static noise, pops or dropouts may occur.
According to the multiple samples analyzed, skipped some of the initial buffers as per the observation made to obtain the lowest possible latency. Below image



```
void SoundRecording::writeFile(SndfileHandle sndfileHandle) {

    int32_t framesRead = 0, bufferLength = mTotalSamples;
    if(bufferLength>192){
        bufferLength = 192;
    }
    sf_count_t framesWrite = 0;
    auto *buffer = new int16_t[bufferLength];
    fillArrayWithZeros(buffer, bufferLength);
    if(countOfIteration<5){
        countOfBuffers = 0;

        while ((framesRead = read(buffer, bufferLength)) > 0) {
            countOfBuffers++;
            sndfileHandle.write(buffer, framesRead);
        }
    }
    else{
        countOfBuffers = 0;
        while ((framesRead = read(buffer, bufferLength)) > 0) {
            countOfBuffers++;
            if (countOfBuffers < 24)continue;
            sndfileHandle.write(buffer, framesRead);
        }
    }

    countOfIteration++;
    LOGD(TAG, "writeFile(): ");
    LOGD(TAG, to_string_with_precision(framesRead).c_str());
    LOGD(TAG, to_string_with_precision(countOfBuffers).c_str());
```

3. RESULTS

3.1 Reduced and Stabilized Latency Values

The audio latency observed after the modifications done in the application was quite low. The following images depicts the improvement in value and stability in the audio latency of the player and recorder.The latency values in the *Figure 3.1* is highly unstable and has maximum delay of about 800ms. The latency values in the *Figure 3.2* is nearly 50ms with a variation range of 0ms to 10ms.

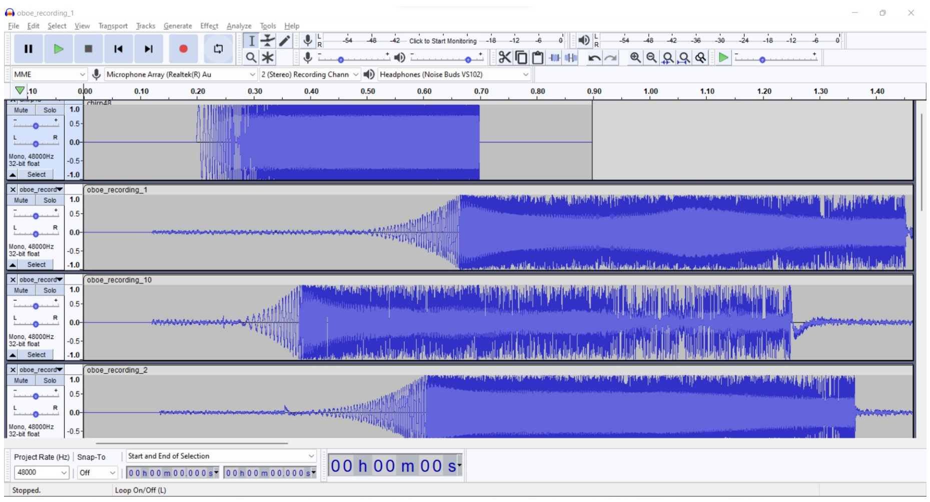


Figure 3.1: Results after initial implementation

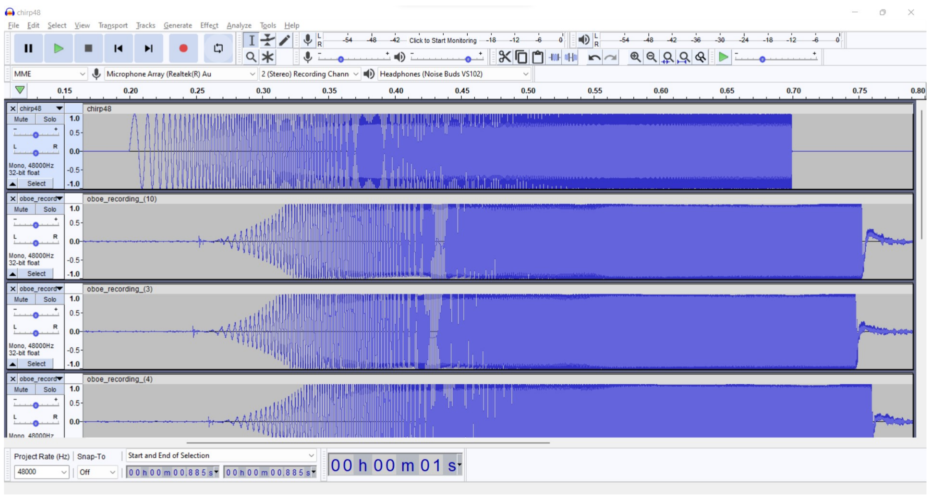


Figure 3.2: Results after modifications

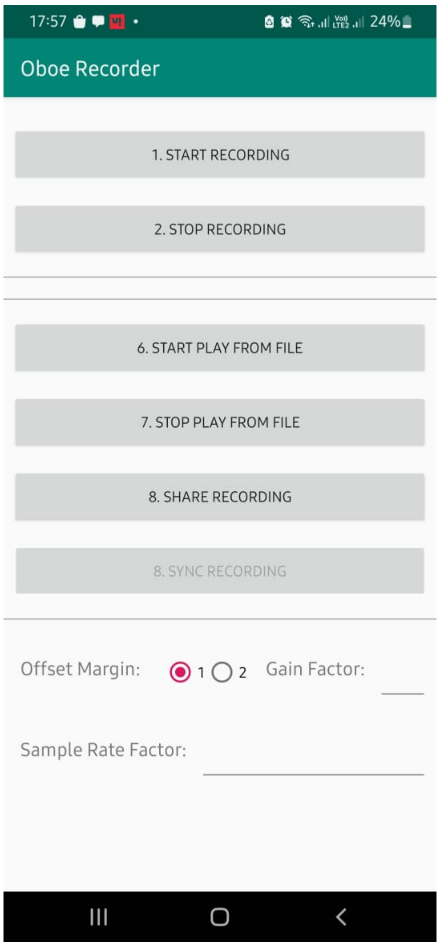
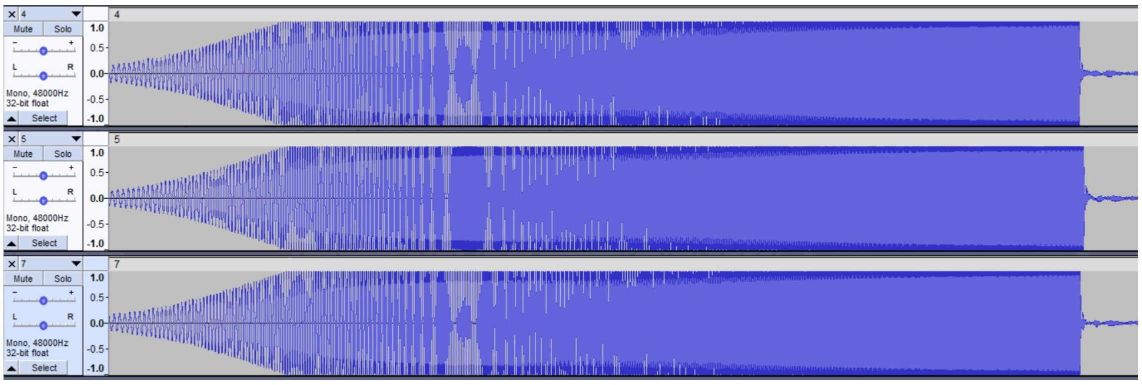
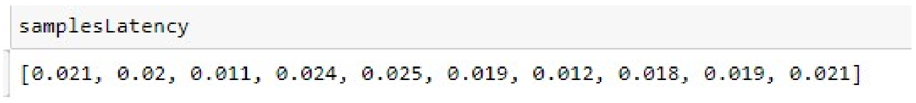


Figure 3.3: Frontend of the Sample Application



Some samples of waveform obtained after modifications in audio buffers read



```
samplesLatency
[0.021, 0.02, 0.011, 0.024, 0.025, 0.019, 0.012, 0.018, 0.019, 0.021]
```

Figure 3.4: Latency values (in *s*) of continuous 10 samples calculated

3.2 Conclusion & Summary

The project commenced with the development of the frontend of the android application and analyzing the working mechanism of Oboe Library for high quality audio. After properly analyzing the library, moved forward to implement it in the sample application. Though the initial latencies were quite high and unstable. Later on, analyzed multiple samples by varying the sample rate, buffer length, bursts size, etc.
Further implemented the handler functions and skipped the initial buffers to introduce delay in code execution as per the average latency observed in the SAMSUNG and REDMI systems. This optimized the audio latency by nearly 88% i.e. max latency reduced to nearly 20ms. Later on, implemented the optimized audio recording and playing version in the Narottam Android Application, a music learning platform.

Bibliography

[1] https://developer.android.com/ndk/guides/audio/audio-latency

[2] https://github.com/google/oboe

[3] https://developer.android.com/docs