Bachelor Thesis

Datateknik
Computer Engineering

JChord
An Android application helping people learn guitar chords

Jiwen Zhou
MID SWEDEN UNIVERSITY
Computer Science and Technology

Examiner: Ulf Jennehag, ulf.jennehag@miun.se
Supervisor: Tingting Zhang, Tingting.Zhang@miun.se
Author: Jiawen Zhou, jizh1500@student.miun.se
Degree programme: Final Project/ Examensarbete, 15 credits
Main field of study: Sound analysis and Android coding
Semester, year: Fourth cycle, 2016
Abstract

Nowadays, a lot of interesting and useful and imaginative applications are springing to Android software market. And for guitar fans, some related apps bring great connivence to them, like a guitar tuner can save people from carrying a entity tuner all the time, some apps can simulate a real guitar, and some apps provide some simple lessons allowing people to learn some basic things. But these apps which can teach people, they can't really “monitor ” people, that is, they just give some instructions and hope people would follow them. So my project is to design an app which can detect if users are playing wrong and right real-timely. Guitar chords are always the first for new guitar beginners to learn, and a chord is a set of notes combined together in a regulated way ( get from the music theory having millions of developing ), and 'pitch' is the term for determining if the note different from other notes or noise, so the problem here is to manage the multi-pitch analysis in real time. And it's necessary to know some basics of digital signal processing ( DSP ) because digital signals are always more convenient for computers to analyze compared to analog signals. Then I found an audio processing Java library – TarsosDSP, and try to apply it to my Android project.

Key words: Multi-pitch analysis, Android, DSP, Music theory, TarsosDSP
Acknowledgements

Thanks to Tingting Zhang for supervising my work and helping me to make decision on the subject of my thesis project.
# Table of Contents

Abstract........................................................................................................................................... iii  
Acknowledgements ......................................................................................................................... iv  
Terminology ....................................................................................................................................... vii  
1 Introduction................................................................................................................................. 1  
1.1 Why this app?......................................................................................................................... 1  
1.2 Background .......................................................................................................................... 1  
1.2.1 Guitar Chords.................................................................................................................... 1  
1.3 Challenge.............................................................................................................................. 2  
1.4 Overall aim............................................................................................................................ 2  
1.5 Scope....................................................................................................................................... 2  
1.6 Concrete and verifiable goals............................................................................................... 2  
1.7 Outline...................................................................................................................................... 3  
2 Theory.......................................................................................................................................... 4  
2.1 Music Theory........................................................................................................................ 4  
2.1.1 The essence of the sound................................................................................................... 4  
2.1.2 Guitar and frequencies...................................................................................................... 4  
2.2 Multi-pitch detection and analysis...................................................................................... 6  
2.3 More facts about guitar chords............................................................................................ 8  
2.4 Digital signal processing (DSP)........................................................................................... 8  
2.5 Single pitch detection ........................................................................................................... 10  
2.5.1 Introduction to pitch detection.......................................................................................... 10  
2.5.2 Different pitch detection algorithms................................................................................ 11  
2.5.3 Why choose MPM (McLeod Pitch Method) ...................................................................... 11  
2.6 DSP and Android................................................................................................................ 11  
2.6.1 Sampled sound using Android....................................................................................... 11  
3 Methodology............................................................................................................................... 14  
3.1 Concrete goals achievement................................................................................................. 14  
3.2 TarsosDSP............................................................................................................................ 14  
4 Design.......................................................................................................................................... 16  
4.1 User interfaces design........................................................................................................... 16  
4.2 How we can say two frequencies are equal?..................................................................... 21  
4.3 Other design.......................................................................................................................... 22  
5 Results.......................................................................................................................................... 24  
6 Conclusions................................................................................................................................... 25  
6.1 Future work............................................................................................................................ 25  
References....................................................................................................................................... 26  
Appendix A: Code pieces of JChord .............................................................................................. 28  
Code piece 1: Grid View implementation...................................................................................... 28  
Code piece 2: Chord Class............................................................................................................. 28
Code piece 3: Specify “dot” to be visible ........................................29
Code piece 4: Change “dot”’s color ..................................................30
Code piece 5: Compare pitches..........................................................30
## Terminology

### Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chord</td>
<td>A guitar chord is a set of notes played on a guitar. [1]</td>
</tr>
<tr>
<td>Pitch</td>
<td>Pitch is the quality that makes it possible to judge sounds as &quot;higher&quot; and &quot;lower&quot; in the sense associated with musical melodies. Pitch can be determined only in sounds that have a frequency that is clear and stable enough to distinguish from noise. [2]</td>
</tr>
<tr>
<td>Fret</td>
<td>Metal stripes (usually nickel alloy or stainless steel) embedded along the fretboard and located at exact points that divide the scale length in accordance with a specific mathematical formula. Pressing a string against a fret determines the strings' vibrating length and therefore its resultant pitch. [3]</td>
</tr>
<tr>
<td>Scale</td>
<td>A scale is any set of musical notes ordered by fundamental frequency or pitch. A scale ordered by increasing pitch is an ascending scale, and a scale ordered by decreasing pitch is a descending scale. [10]</td>
</tr>
<tr>
<td>Tablature</td>
<td>A form of musical notation indicating instrument fingering rather than musical pitches. [17]</td>
</tr>
</tbody>
</table>

### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP</td>
<td>Digital Signal processing</td>
</tr>
<tr>
<td>A/D</td>
<td>Analog to digital conversion</td>
</tr>
<tr>
<td>MPM</td>
<td>McLeod Pitch Method, a fast, accurate and robust method for finding the continuous pitch in monophonic musical sounds. It uses a special normalized version of the Squared Difference Function (SDF) coupled with a peak picking algorithm. [21]</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse code modulated bytes</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Why this app?

Guitar, an exemplification of the phrase “a labor of love” has become one of the most popular musical instruments nowadays. Guitar is originated in Spain, having a history for more than 4000 years, is “a long, fretted neck, flat wooden soundboard, ribs, and a flat back, most often with incurved sides” (From Wikipedia). Guitars are always relatively affordable instrument for common people, you can even get a decent one for a low price whereas with some other instruments like the violin or piano you may be forced to rent one to learn on. The most magic thing is guitar can imitate almost every other musical instruments, incredibly versatile! It's choral like a piano, can play as much as notes as you want, though not as strongly so, with a few chords, a little practice and a decent voice, you can play and sing and have a good time with others. And it's expressive and vocal like a sax. You can also get many sounds ranging from legato almost violin tones to percussion. Many people love guitar due to it's beautiful sounds it makes. And you can also sing while playing the guitar, it covers a pretty broad range of notes across what people ordinarily sing in. It's really common nowadays for singers and bands to apply guitar to their songs, and guitars are indispensable in almost every bands. And for us common people, it's rather portable, can be easily carried anywhere anytime. Just put one back on your back and you can just ride your motorcycle to meet with your friends, fun and cool, really help livening up the atmosphere among friends and can show your personal charm.

But although guitar is said by many people that easy to learn, as far as I learned guitar and getting comments from my friends who just started to learn guitar, all proved it's harder than expected. First reason is most of us can't afford to buy a guitar with good quality and these guitars are more difficult to practice, same chords or melody may need more force to press on the strings, then new beginners would spend more time practicing than others and they would think it's because they aren't born for guitars and then lose faith in themselves gradually. The second reason is that some of us can't hire some guitar teacher due to lacking regular spare time or money and it's really hard to learn guitar by ourselves. Yeah, we can learn some basic things through the Internet – online free videos. But it's never guaranteed that what we learned is correct because the videos are dead and 'they' can't 'point out' we are wrong like 'Hey, it's not like that, look me again'.

And nowadays, plenty of tuning apps are flocking to the Android apps market, making it really convenient for people to tune their guitars whenever and wherever just use their phones, so I am thinking maybe I can design an app for new guitar learners to learn some basic guitar things-- like chords. And hoping the final app could tell people which fret [3] (Pressing a string against a fret
determines the strings' vibrating length and therefore its resultant pitch) of the string they press wrong or right.

1.2 Background

1.2.1 Guitar Chords

First, let's have a look at what a “chord” is. From Wikipedia, “A guitar chord is a set of notes played on a guitar. A chord's notes are often played simultaneously, but they can be played sequentially in an arpeggio— a musical technique where notes in a chord are played or sung in sequence, one after the other, rather than ringing out simultaneously.”[1] So first I concentrated on the key words – “a set of notes”, “simultaneously”.

1.3 Challenge

By then I searched online and found there's already a similar app for IOS platform named ”Uberchord”[5] and there's none app for Android, so I emailed the Support of them and try to get some 'technical' help. But after some days, they replied “All existing technologies are far from the quality that we needed so we started to invent our own. It took more than two years full time research.” and they do have a plan to port the app to other platforms but they themselves don't know when they would time for that. So sadly I can't get some existing algorithm of the “open source self”.

So “Multi-pitch analysis”, this topic is extremely challenging and existing methods do not match human ability in either accuracy or flexibility.

But I do find a Java open source project on Github, it's the only existing Java project implementing a multi-pitch analysis algorithm from the paper [6]. I test it but unfortunately the results aren't satisfied with my project's needs. So after spending a lot of time on this topic, in the end, I have to change the strategy to implement my app in a short time.

So I change to just detect each string’s frequency separately if matching the chord’s each string’s frequency, the reason I will explain also in Chapter 2. So then my task is to figure out what is DSP and pitch detection and try to justify them to be applied to my project. And then I found a Java library for audio processing, named “TarsosDSP”[7], but they don't have a API document for Android package, so I have to figure out how to implement this API to my project.

1.4 Overall aim

The project's overall aim is to design an android app for people to learn guitar chords themselves real-timely and easily.
1.5 Scope

My project is focus on implementing DSP theory with Android code. And at the same time, try to implement pitch-detection algorithm in my project. The future work will also be discussed in the later chapter.

1.6 Concrete and verifiable goals

(1) Study basic music theory.

(2) Make guitar special features clear, like each fret of the guitar the frequency is what.

(3) Study DSP, pitch-detection algorithms.

(4) Apply the Java library for audio processing – TarsosDSP to my project.

1.7 Outline

Chapter 1 introduces the reason why I want to design such an app and some background information on music theory, guitar and DSP. Chapter 2 describes all theory which involved behind my project, is the extension of Chapter 1. Chapter 3 is how I design the user interfaces and how to apply the “detection” things. Chapter 4 presents how the of the concrete goals. Chapter 5 is some testing results of the app and where I evaluated. Then in the end, summarization of the whole project work and prospect of the future work would be given.
2 Theory

2.1 Music Theory

2.1.1 The essence of the sound

First, what is a sound? Again from Wikipedia -- sound is a vibration that propagates as a typically audible mechanical wave of pressure and displacement, through a medium such as air or water. And all sound could be divided into musical tones and noise. Musical tones are regular waves compared to noise--unpleasant, loud, or interferes with hearing. They sound melodious, sweet and harmonious, such as sound performed from musical instruments and broadcast music.

All sound is composed of several variations with different frequencies. Among them, the one who has the lowest frequency is so-called fundamental frequency, all others are called harmonic overtones. Why musical tones can be distinguished from noise is in that all the frequency of the harmonic overtones of the musical tone are always the integer multiples of the fundamental frequency. While for noise, the frequency of the harmonic overtones of the musical tone are not the integer multiples of the fundament frequency [9].

From the perspective of physics, single musical tone consists of three main attributes: frequency, amplitude and spectrum. The musical tones emitted by instruments first reflected to the auditory nerve center through auditory system of human ears, then causes the listener's subjective feeling. This feeling then creates the psychological form of a single musical note, which are also three elements: pitch, loudness and timbre. These three are closely related to the three physical attributes that are easily to be determined objectively mentioned before[8]. So these three features make single musical tone could be measured and analyzed with physical methods. To specify, the pitch of the tone (pitch) of single music note is determined by the fundamental frequency of the note; the loudness of a single tone (sound intensity) is the energy flow reach the ear through air vibration; timbre is more complex, is formed by the constitution of spectrum (the proportion of harmonic components) of the single music note[9].

2.1.2 Guitar and frequencies

So after 2.1.1, we can see that for guitar, when we pluck a string, the variations doesn’t consist of just one single frequency, then naming a ‘defining’ frequency could be difficult. But from section 2.3 of [9], it says that the vibration of a string consists of multi-order harmonies components. When we flick a string, although the position would be different, then the frequency components of the vibration would be different — harmonic overtone would be different, but the fundamental frequency would stay unchanged. So, we care about the frequency of fundamental wave, can differentiate pitch from pitch.
“A440 or A4, which has a frequency of 440 Hz, is the musical note A above middle C and serves as a general tuning standard for musical pitch.” (from wikipedia) And normal guitars all follow the standard tuning, the chords I used in this app is under the standard tuning too. And another thing we should know about guitar is what are frets. I first mentioned in section 1.1 -“And hoping the final app could tell people which fret [3] of the string they press wrong or right.” Figure 2.1 is the picture from wikipedia and shows all components of a normal guitar.

![Figure 2.1 Components of the guitar](image)

No. 18 is the strings, normally a guitar has 6 strings with different degree thickness. And No. 4 is the so-called frets. No. 20 the thing like a blackboard is called fretboard, covered with frets. Each open string has a specific pitch frequency under standard tuning and guitar won't be so popular if they only can perform 6 musical notes (we don't care other tunings here). And these frets are really magical and amazing in that the guitar can have much more tones just to press a string against different frets! After centuries of history, the frets just locate at exact points that divide the scale [10] length in accordance with a specific mathematical formula [3]. Normally, a guitar would have 19, 21 or 24 frets, but for common use, 12 frets are enough.

And Table 1 is the first 12 frets' frequency of the normal guitars. “Fret 0” represents open string of the string, and “fret 1” means to press the first fret of the string.
Table 1 First 12 frets' frequency of the normal guitar

<table>
<thead>
<tr>
<th>Fret</th>
<th>String 6</th>
<th>String 5</th>
<th>String 4</th>
<th>String 3</th>
<th>String 2</th>
<th>String 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>82.4</td>
<td>110.0</td>
<td>146.8</td>
<td>195.0</td>
<td>246.9</td>
<td>329.6</td>
</tr>
<tr>
<td>1</td>
<td>87.3</td>
<td>116.5</td>
<td>155.6</td>
<td>207.6</td>
<td>251.5</td>
<td>349.2</td>
</tr>
<tr>
<td>2</td>
<td>92.5</td>
<td>123.5</td>
<td>164.8</td>
<td>220.0</td>
<td>277.2</td>
<td>370.0</td>
</tr>
<tr>
<td>3</td>
<td>98.0</td>
<td>130.8</td>
<td>174.6</td>
<td>233.1</td>
<td>293.6</td>
<td>392.0</td>
</tr>
<tr>
<td>4</td>
<td>103.8</td>
<td>136.6</td>
<td>185.0</td>
<td>246.9</td>
<td>311.1</td>
<td>415.3</td>
</tr>
<tr>
<td>5</td>
<td>110.0</td>
<td>145.8</td>
<td>196.0</td>
<td>261.6</td>
<td>329.5</td>
<td>440.0</td>
</tr>
<tr>
<td>6</td>
<td>116.5</td>
<td>155.6</td>
<td>207.6</td>
<td>277.2</td>
<td>349.2</td>
<td>466.1</td>
</tr>
<tr>
<td>7</td>
<td>123.5</td>
<td>164.8</td>
<td>220.0</td>
<td>293.6</td>
<td>370.0</td>
<td>493.8</td>
</tr>
<tr>
<td>8</td>
<td>130.8</td>
<td>174.6</td>
<td>233.1</td>
<td>311.1</td>
<td>392.0</td>
<td>523.2</td>
</tr>
<tr>
<td>9</td>
<td>138.6</td>
<td>185.0</td>
<td>246.9</td>
<td>329.6</td>
<td>415.3</td>
<td>554.3</td>
</tr>
<tr>
<td>10</td>
<td>146.8</td>
<td>196.0</td>
<td>261.6</td>
<td>349.2</td>
<td>440.0</td>
<td>587.3</td>
</tr>
<tr>
<td>11</td>
<td>155.6</td>
<td>207.6</td>
<td>277.2</td>
<td>370.0</td>
<td>466.1</td>
<td>622.2</td>
</tr>
<tr>
<td>12</td>
<td>164.8</td>
<td>220.0</td>
<td>293.6</td>
<td>392.0</td>
<td>493.8</td>
<td>669.2</td>
</tr>
</tbody>
</table>

Then let's know more about guitar chords. Figure 2.2 is the C major chord in open position. We know that a guitar chord is a set of notes (always more than three) played on a guitar [1], and it is any harmonic set of usually three or more notes that is heard as if sounding simultaneously. Why guitar chord are formed this way is not our emphasis. So if we want to detect a chord, we need to detect each note simultaneously -- multi-pitch detection.

![C major chord in open position](image)

Figure 2.2 C major chord in open position

### 2.2 Multi-pitch detection and analysis

About this topic, I do spend a lot of time on it from the beginning of deciding my thesis project in the end of January and till the beginning of April this year, around 2 months, and it ends that I figured out it's impossible for me finish it in such a short time alone.

I will briefly talk about this. On the whole, this task can be divided into three parts[11]:

---

6
“Level 1 (easiest) - Multi-pitch Estimation is to collectively estimate pitch values of all concurrent sources at each individual time frame, without determining their sources.

Level 2 (intermediate) - Note Tracking is to estimate continuous pitch segments that typically correspond to individual notes or syllables. Note that each pitch contour comes from one source but each source can have many contours (e.g. one contour per musical note or spoken word).

Level 3 (hardest) - Multi-pitch Estimation and Streaming is to estimate pitches and stream them into a single pitch trajectory over an entire conversation or music performance for each of the concurrent sources. The trajectory is much longer than those estimated at the second level, and contains a number of discontinuities that are caused by silence, non-pitched sounds and abrupt frequency changes. This level is also called timbre tracking.”

There are some papers like [12][13][14][15], talking about algorithms on multi-pitch analysis. All of them are complicated and I try to search on the Internet if there existing any APIs. And shortly afterwards, I found the “Uberchord” but in the end, I can't get any help.

But at the same time, I found a open-source Java project called ”PolyphonicPitchDetection” on Github [16] which implements the paper's [6] algorithm, and I test it in Eclipse. Figure 2.3 is the user interface of the project.

![Figure 2.3 UI of the “PolyphonicPitchDetection”](image)

Because the project is really noise-sensitive and I can't screenshot the exactly point so I recorded a video about how I test this project, link. And here are the results of testing:
• All strings can be detected alone clearly.
• But noise can affect detection greatly.
• 2, 3 or 4 strings are flicked simultaneously can be detected clearly.
• 5 or 6 strings can’t.
• The program can detect at most 8 pitch at the same time but can’t be stable and clearly.

So clearly, the results are disappointed and aren't satisfied with my project's needs. Together with the e-mail “Uberchord” sent to me, then I decided to change my strategy.

2.3 More facts about guitar chords

Then I was thinking for our beginners to learn guitar, we seldom would practice chords by picking all the strings at the same time, we would try to remember the tablature (a form of musical notation indicating instrument fingering rather than musical pitches) of the chord by playing the notes of the chord one by one, in sequence, rather than performing all the notes at the same time, this performing method is called “arpeggio”. And again I look carefully into the “chord” definition on Wikipedia, then I found: “Chords need not actually be played together: arpeggios and broken chords (these involve the notes of the chord played one after the other, rather than at the same time) may, for many practical and theoretical purposes, constitute chords.”

And my app is aimed to help beginners to learn chords so I think the problem here is not detecting all strings at the same time. So I decided to just detect each string’s frequency separately if matching the chord’s each string’s frequency.

2.4 Digital signal processing (DSP)

Then it's the whole thing about single pitch detection stuff. First I would introduce basic stuff on DSP.

From the book *Understanding Digital Signal Processing*,

The term signal processing refers to the science of analyzing time-varying physical processes. As such, signal processing is divided into two categories, analog signal processing and digital signal processing. The term analog is used to describe a waveform that’s continuous in time and can take on a continuous range of amplitude values. [4]

And to analyze on computers, digital signals are always easier for manage and are more robust to noise. So the first step is to convert collected analog signal -- continuous signal to digital signal. This concerns a concept called sampling. Look below Figure 2.4 is a common continuous sine wave which is one of the most simple audio signals, because the signal is continuous, we can see that there are an infinite amount of points between \([0, 2\pi]\), makes it impossible to represent within a digital system. So sampling helps us to represent a continuous signal with a sequence of discrete data values, like Figure 2.5.
But after sampling, according to most of the theory of DSP, the samples are considered to be value-continuous. So next step, quantization is needed-- to constrain a continuous or large set of values (e.g. floating-point number) to a smaller set of (typically $2^8, 2^{12}, 2^{16}$, the exponent value is also called quantization level) values. And quantization can reduce the loss of information greatly. Figure 2.6 is different versions of quantization.
Then after the basic transformations of the data, then we can do some meaningful analysis.

Time-domain analysis is to analyze the data over a time period, to see how the signals change over time and variables are measured against time. While frequency-domain analysis is to see how much signals lie in the frequency range, theoretically signals are composed of many sinusoidal signals with different frequencies, so its actually composed of infinite sinusoidal signal (fundamental and odd harmonics frequencies). Figure 2.7 is same signal from time-domain view and frequency-domain view. The most important concept is the transformation—convert a time domain function to a frequency domain function and vice versa. And the most common transformation used is the Fourier transformations. So then after frequency-domain analysis, we can find our pitch.

Figure 2.7 Time-domain VS. Frequency-domain

### 2.5 Single pitch detection

From Wikipedia: “Pitch detection is designed to estimate the pitch or fundamental frequency of a quasi-periodic or oscillating signal, usually a digital recording of speech or a musical note or tone. This can be done in the time domain or the frequency domain or both the two domains.” So basically, pitch can be reliably matched to fundamental frequency (F0). And there are already a lot of algorithms designed. After previous section given some basic information on DSP, in this section, I will talk about how to do specific pitch detection briefly.

#### 2.5.1 Introduction to pitch detection

From paper [18], "Accurate and reliable measurements of the pitch period from only one acoustic signal are often extremely difficult. " One reason is that the signal wave formats always change across the time, in other words, pitch changes over time. Secondly, noise would influence a lot, it would be hard to
distinguish between tones and noise. But luckily, so far, there are a lot of methods developed although so much difficulty exists along the way.

2.5.2 Different pitch detection algorithms

Basically, there are three main kinds of algorithms on pitch detection:

A. Time-domain approaches

B. Frequency-domain approaches

C. Spectral/temporal approaches

Time-domain approaches includes zero-crossing method, parallel processing technique, autocorrelation, Square Difference Difference (SDF), Average Magnitude Difference Function (AMDF) [19].

Spectrum Peak Methods, Phase Vocoder, Harmonic Product Spectrum, Subharmonic-to-Harmonic Ratio, Autocorrelation via FFT are frequency-domain approaches [19].

The “C” approaches have Cepstrum, Wavelets and Linear Predictive Coding (LPC) [19].

2.5.3 Why choose MPM (McLeod Pitch Method)

And when I try to apply “TarsosDSP” into my project, I choose the method MPM which is especially “to find the continuous pitch in monophonic musical sounds.” [20] I will introduce more about TarsosDSP in Chapter 3.

2.6 DSP and Android

In this part, we will briefly discuss how to turn DSP theory into Android code following a manual [21], but it's talking about DSP and Java, not Android.

2.6.1 Sampled sound using Android

From section 1.2.3, we all know we have to convert analog signals to digital signals. The first step is to sample. And in order to make information carried by the signal loss less, there are two significant properties we have to care about: sampling rate and bit depth [22].

According to Nyqvist-Shannon sampling theorem [23], the sampling rate (in Hz) measuring samples per second should be at least twice the maximum frequency of the signal. And I googled that guitar can't perform higher frequency than 5kHz, so in this application, I just set the sampling rate to be 16kHz.
The bit depth is the number of bits of information in each sample, decides the resolution of each sample. Usually we will use signed integers of 16 bits. Take Figure 2.8 as an example, the sampling rate defines the horizontal interval size of the grid, the bit depth defines how many bits in a sample. Higher bit depths, the more dynamic range for the signal.

![Figure 2.8 Comparison of different bit depths](image)

In Android, we use “AudioRecorder” to define a object, “ to manage the audio resources for Java applications to record audio from the audio input hardware of the platform”. Let's see what needs to be certain to define a “AudioRecorder”.

```java
AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes)
```

So here are 5 parameters should be determined. The “audioSource” asks us to decide physical source of audio signal, then in my app, I just set to be “MediaRecorder.AudioSource.MIC”, let the cellphone's microphone to be the source.

The “sampleRateInHz” I set is “160000”, we just mentioned before.

The “channelConfig” describes the configuration of the audio channels, int values, I set to be “AudioFormat.CHANNEL_IN_MONO”, just one mono channel.

The “audioFormat” is “the format in which the audio data is to be returned”. And “bufferSizeInBytes” is “the total size (in bytes) of the buffer where audio data is written to during the recording”.

To determine these 2 parameters, let's go back to the DSP theory.

---

1 [AudioRecorder](https://developer.android.com/reference/android/media/AudioRecord.html)
“It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency.”

At the same time recording the audio, we need a buffer to store the data, and it recommender to use float data type to define the buffer. Then it's need to convert the float buffer into pulse code modulated (PCM) bytes.

“PCM is a method to digitally represent sampled analog signals. It is the standard form of digital audio in computers, Compact Discs, digital telephony and other digital audio applications. In a PCM stream, the amplitude of the analog signal is sampled regularly at uniform intervals, and each sample is quantised to the nearest value within a range of digital steps.

A PCM stream has two basic properties that determine the stream's fidelity to the original analog signal: the sampling rate, which is the number of times per second that samples are taken; and the bit depth, which determines the number of possible digital values that can be used to represent each sample.” [24]

So the “audioFormat” set to be “AudioFormat.ENCODING_PCM_16BIT” because bit depth is usually 16 bits.

Then for “bufferSizeInBytes”, we can get the minimum buffer size required for the successful creation of an AudioRecord object by using:

```java
int bufferSizeInBytes = AudioRecord.getMinBufferSize(sampleRateInHz, channelConfig, audioFormat);
```

---

3 Methodology

In this part, first I will state the methods and process to complete the project. Then I will also talk about how I evaluate my project.

3.1 Concrete goals achievement

To achieve goal (1) – to study basic music theory, I googled a lot.

To achieve goal (2) – to guitar special features clear, like each fret of the guitar the frequency is what, still google.

To achieve goal (3) – to study DSP and pitch-detection algorithms, I read some papers which I go in detail in Chapter 2.

To achieve goal (4) – to apply the TarsosDSP to my project, I read the manual, but have to figure what to apply with Android codes because they just have documents in Java.

3.2 TarsosDSP

TarsosDSP implements a lot of pitch detection algorithms. So apply it to get the pitch of the sound is really convenient. This API prefers to do operation on sound in blocks. Figure 3.1 is the processing pipeline of how TarsosDSP works.

![Figure 3.1 A schematic representation of the TarsosDSP processing pipeline](image)

The “AudioDispatcher” chops incoming audio in blocks of a requested number of samples, with a defined overlap. Then the blocks are encapsulated in an “AudioEvent” object which contains a pointer to the audio. The “AudioDispatcher” would send the “AudioEvent” through a series of “AudioProcessor” objects which execute an operation on audio. The core of the algorithms are contained in these AudioProcessor.[21]

Code 3.1 is how I apply TarsosDSP. Line 69 I get the audio buffer, line 72 I create a “AudioEvent” object. Line 74 I choose the MPM algorithm. Then in line 75 I send the “AudioEvent” to the processor, and a pitch in line 78. Then I compare the pitch with the wanted pitch with the method “checkIfMatchString”, this method is stated in Code piece 5.
private boolean analysisAudio(final int stringNumber) {
    boolean result = false;
    byte recordedBuffer[] = record.getRecordedBuffer(); // get recorded audio data
    int bufferReadResult = record.getBufferSize();
    be.hogen.tarsos.dsp.AudioFormat audioFormat = new be.hogen.tarsos.dsp.AudioFormat(sr, 16, 1, true, false);
    AudioEvent audioEvent = new AudioEvent(audioFormat, bufferReadResult);
    audioEvent.setFloatBufferWithNysedBuffer(recordedBuffer);
    audioProcessor = new PitchProcessor(PitchProcessor.PitchEstimationAlgorithm.MM, sr, bufferSize, handler);
    audioProcessor.process(audioEvent);
    PitchDetectionResult presult = new PitchDetectionResult();
    handler.handlePitch(presult, audioEvent);
    float pitch = presult.getPitch();
    Log.e("Pitch is ","-pitch");
    if(checkIfMatch5String(pitch,stringNumber,chord.getFretValGet(stringNumber))){
        Log.d("ANALYZE", "Correct note!");
        return true;
    } else {
        Log.d("ANALYZE", "Incorrect note...");
        return false;
    }
}
4 Design

I will divide this part into two main parts: the user interfaces design and how I apply “TarsosDSP” API into my project.

4.1 User interfaces design

I design three man UI, Figure 4.1 is the welcome interface of the app and have the only button with text “Pick a chord”, asking users to press the button to choose a chord they want to practice.

Figure 4.1 Welcome interface

Figure 4.2 is the grid view in Android Studio, would be filled with buttons with chords' names on them. After the Code piece 1, it would look like Figure 4.3 on real phones. Because I just choose three chords for the app, so there are only 3 buttons here: Major C, Minor A and Minor E.
JChord – An Android application for people learn guitar chords
Jiwen Zhou
2016-05-16

Figure 4.2 Grid View in Android Studio

Figure 4.3 Grid View on real phones
Figure 4.4 is the final chord view user interface in Android Studio. Figure 4.5 points out which parts are “TextView” and what information they supposed to show.
Figure 4.6 points out which parts are “ImageView”. Actually each fret of a string have a “dot” image, but the ‘dot’s are visible only when the fret of the string should be pressed according to specific chord. To do this, I first create a class – “Chord” (Code piece 2), then I set “dot.setVisibility(View.VISIBLE);” according to the specific chord in “if” clause if the fret is satisfied with the condition (Code piece 3).

Then I set a special trick here, if the detected pitch is not equal to the wanted pitch, I set the dot colour multiple with “red” and multiple with “green” when it's correct, Figure 4.7 and Figure 4.8.

And I also add a “ProgressBar” to show which string is playing by the user and the string is to being listened by the app, Figure 4.9 below.
Figure 4.7 “Dot” to be red if users’ finger is wrongly pressed

Figure 4.8 “Dot” to be green if users’ finger is correctly pressed
4.2 How we can say two frequencies are equal?

After getting the detected pitch, the next step is to compare it with the one we wanted. But then here comes a question: In which scale two frequencies in music field can be said to be equal? I need to set a specific error amount for pitch comparison.

This question really confused me and then I try to google and bingo, there is already a global scale for this – “Cent”:

“The cent is a logarithmic unit of measure used for musical intervals. Twelve-tone equal temperament divides the octave into 12 semitones of 100 cents each. Like a decibel’s relation to intensity, a cent is a ratio between two close frequencies.

Since a frequency raised by one cent is simply multiplied by this constant cent value, and 1200 cents doubles a frequency, the ratio of frequencies one cent apart is precisely equal to 21/1200, the 1200th root of 2, which is approximately 1.0005777895.”[26]

So I just set the error amount as “±1 Hz”. And detected pitch should satisfy the following formula:

\[
Frets[\text{StringNumber}][\text{FretNumber}] - 1\text{Hz} \leq \text{Pitch} \\
\text{Pitch} \leq Frets[\text{StringNumber}][\text{FretNumber}] + 1\text{Hz}
\]

Frets[\text{StringNumber}][\text{FretNumber}] in the formula indicates the pitch value of the “FretNumber”th of fret of the “StringNumber”th of the string. And Code piece 5 is the method I create for pitches with float values.
4.3 Other design

It's quite complicated to record some data using “AudioRecorder”, involving how to know a buffer is filled, when to stop recording, how to write the recorded data to a wav file etc. I didn't have enough time for understanding how to use “AudioRecorder”, so in my own app, I just use the sample code from the webpage [25], and change some properties I stated in section 2.5.1.

Because the app would detect one string each time, so I put the detection process for each string into a thread, inside a “for” loop. Code 4.1 is the implementation codes.

![Code 4.1 Whole “Detection” part](image)

Line 206, to loop for 6 times because there are 6 strings. Line 222 the method would return a boolean “result” indicating if the detected pitch is matched with the wanted pitch, “true” if mated, “wrong” if not. And the little trick here is line 229 and 244. Line 229 I set the thread to sleep for 2s when the result is wrong and what I consider is that at this time, user need to fix their finger's position on the string, so I set here 2s for them to have longer time to adjust their finger. And when the result is correct, the user don't need to move their finger, so less
time needed and the user experience is better – can immediately check if next string is correct.
5 Results

In this chapter, I will show you how I decide the dot's color changing mode during the testing.

The dot I found from the Internet has grey as it's original color, and I hope my app could tell people when they are wrongly plucking the string or correctly. Then I design a method for dot to change color called “fingerDotColorSet”, and find a method in “ImageView” class:

```java
final void setColorFilter(int color, PorterDuff.Mode mode)
```

But the “Android Developer” document doesn't explain how to choose among the “PorterDuff.Mode”s and I googled and find there are two modes act as a color filters, other for picture filters--'PorterDuff.Mode.MULTIPLY' and 'PorterDuff.Mode.SCREEN'. Then I test with the two filters and didn't record the process. For 'PorterDuff.Mode.SCREEN', it would show the new 'color' shape and for default, the class 'Color' all have a square shape. For 'PorterDuff.Mode.MULTIPLY', it will overlap two color but will still keep the the original shape of the target image. Code 5.1 is the final code.

```java
private void fingerDotColorSet(int i, boolean isCorrect) {
    if(chord.fretValGet(i)<0)
        return;
    ImageView dot = (ImageView) findViewById(fretsIds[i][chord.fretValGet(i)-1]);
    if(isCorrect)
        dot.setColorFilter(Color.GREEN, PorterDuff.Mode.MULTIPLY); //green if correct
    else
        dot.setColorFilter(Color.RED, PorterDuff.Mode.MULTIPLY); //red if wrong
}
```

Code 5.1 Final code of changing dots’ color

And two screenshots show the final effect in Chapter 4, Figure 4.7 and 4.8. And I recorded a video of the testing and representation of the app, link.
6 Conclusions

This chapter is the final conclusion of the project and some discussion of future work.

This project is to design an Android app for new guitar learners to learn basic guitar chords easily. The app would detect the sound made by the user when user choose the specific chord in real time and tell them if their fingers press the right fret of the string by changing the fret dot's color and showing some text on the screen. Because I am a guitar fan myself so I know what a chord is and then I searched a lot about how to detect guitar chords and find it's about “Multi-pitch analysis” and then I focused on this topic and in the end just to find I can't implement this technique to my project by myself in such a short time. Then I had to change the technique to “Single pitch analysis”. But it's quite complicated to turn DSP and pitch-detection theory to Android code, so I searched a lot on the Internet for code examples, especially for how to get a specified captured buffer with “AudioRecorder”. And fortunately I found a audio processing API – TarsosDSP and it really saved me to implement the pitch detection algorithm myself. So in the end, I managed to complete this project. And the app testing results are satisfied.

6.1 Future work

Clearly, this app can be extended if could apply some “Multi-pitch Analysis”. Actually when I found that the “PolyphonicPitchDetection” project's testing results are not satisfied, I tried to modify some codes like adding a filter to filter some noise by trying to understand the structure of the project which has no documents to explain how it implement the algorithm in paper [6]. But then I received the e-mail from the “Uberchord” and I was worried about whether I can finish my project in time and at that time, I just started to read the codes of the “PolyphonicPitchDetection”, so I quit this attempt in the end. So if this project could be modified, my project would be applied the “Multi-pitch Analysis” technique. And we could extend this project by implementing some algorithms ourselves.
References


JChord – An Android application for people learn guitar chords
Jiawen Zhou


Published 2003-03-02. Retrieved 2015-05-17


Published 2011-12-03. Retrieved 2016-01-08.


Appendix A: Code pieces of JChord

Code piece 1: Grid View implementation

```java
public View getView(final int position, View convertView, ViewGroup parent) {
    Button button; //create a new button for each item referenced by the Adapter
    if(convertView == null) {
        button = new Button(mContext);
        button.setText(chords[position].nameGet());
        button.setTextColor(Color.rgb(0, 0, 0));
        button.setBackgroundColor(Color.argb(50, 255, 255, 255));
        button.setClickable(true);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mContext.startChordActivity(chords[position]); //Grid中，进入chordView
            }
        });
    } else {
        button = (Button) convertView;
    }
    return button;
}
```

Code piece 2: Chord Class
Code piece 3: Specify “dot” to be visible
private void fingerDotsInit() {
    fretIds = new int[6][5];
    fretIds[0][0] = R.id.dot01;
    fretIds[0][1] = R.id.dot02;
    fretIds[0][2] = R.id.dot03;
    fretIds[0][3] = R.id.dot04;
    fretIds[0][4] = R.id.dot05;
    fretIds[1][0] = R.id.dot11;
    fretIds[1][1] = R.id.dot12;
    fretIds[1][2] = R.id.dot13;
    fretIds[1][3] = R.id.dot14;
    fretIds[1][4] = R.id.dot15;
    fretIds[2][0] = R.id.dot21;
    fretIds[2][1] = R.id.dot22;
    fretIds[2][3] = R.id.dot24;
    fretIds[3][0] = R.id.dot31;
    fretIds[3][1] = R.id.dot32;
    fretIds[3][2] = R.id.dot33;
    fretIds[3][3] = R.id.dot34;
    fretIds[4][0] = R.id.dot41;
    fretIds[4][1] = R.id.dot42;
    fretIds[4][2] = R.id.dot43;
    fretIds[4][3] = R.id.dot44;
    fretIds[4][4] = R.id.dot45;
    fretIds[5][0] = R.id.dot51;
    fretIds[5][1] = R.id.dot52;
    fretIds[5][2] = R.id.dot53;
    fretIds[5][3] = R.id.dot54;
    fretIds[5][4] = R.id.dot55;
    if(chord == null) {
        Log.d("ERR", "Chord is null");
        return;
    }
    for(int i=0; i<6; i++) {
        Log.d("DATA", "chord sample val + chord.fretValGet()=1");
        if(chord.fretValGet() > 0) {
            ImageView dot = (ImageView) findViewById(fretsIds[i + chord.fretValGet() - 1]);
            dot.setColorFilter(new PorterDuff.Mode.MULTIPLY);
            dot.setVisibility(View.VISIBLE);
        }
    }
}

Code piece 4: Change “dot”s color

private void fingerDotColorSet(int i, boolean isCorrect) {
    if(chord.fretValGet() > 0) {
        ImageView dot = (ImageView) findViewById(fretsIds[i + chord.fretValGet() - 1]);
        if(isCorrect)
            dot.setColorFilter(Color.GREEN, PorterDuff.Mode.MULTIPLY);//green if correct
        else
            dot.setColorFilter(Color.RED, PorterDuff.Mode.MULTIPLY);//red if wrong
    }
}

Code piece 5: Compare pitches

private boolean checkITMatchString(float pitch, int stringNumber, int fretNumber) {
    Log.d("ANALYZE", "Check string " + stringNumber + " fret " + fretNumber);
    int res1 = Float.compare(pitch, stringBims[stringNumber][fretNumber]-1); //true
    int res2 = Float.compare(pitch, stringBims[stringNumber][fretNumber]+1); //false
    if(res1 == 0 || res2 == 0)
        return true;
    return false;
}