Proceedings of the Peradeniya University Research Sessions, Sri Lanka, Vol.12, Part II, 30th November 2007

Implementation of an Electronic Tuner in J2ME using Fast Fourier Transform

I. Herath and R.G. Ragel*

Department of Computer Engineering, Faculty of Engineering, University of Peradeniya

Introduction

An electronic tuner is a device used by musicians to tune instruments. There, a musician plays a note and the display of the tuner tells the musician how much the played note has deviated from the desired note in terms of frequency. Most of these devices, such as *KORG Chromatic Tuner CA-30*, are standalone equipments which are solely used to tune musical instruments. It would be advantageous if this functionality could be integrated into existing electronic appliances like mobile phones as these devices are equipped with all the necessary hardware components to be used as an electronic tuner. In addition, mobile software applications are becoming increasingly popular and authors of this paper uses this opportunity to develop a tuner application to be deployed in mobile phones.

Section two of this paper focuses on related researches on this area and section three describes the methodology used during the development process. Section four shows the results obtained and section five discusses the issues related to development. Finally, section six concludes the paper.

Related work

In the first part of this section we will brief two academic researches and in the latter part we will focus on few similar industrial applications. Zaykovskiy and Schmitt (2007) presented front end implementation of speech recognition systems for mobile devices. The signal analysis of these systems is being carried out at the server side where the mobile client does not interfere with signal processing. The shortcoming of this sort of application is the communication overhead between the client and the server. The Java signal analysis application by Clausen *et al.* (1998) can only be deployed on desktop computers and any other similar or advanced electronic appliances due to the functions used in this particular program.

There are quite a number of mobile applications such as, *Guitar Tuner* (Kohn, 1997-2007), *Tiny Tuner* (GetJar.com, 2004-2007), and *Chromatic Guitar Tuner* (4pockets.com) are available. The *Guitar Tuner* and *Tiny Tuner* are simple applications which are only capable of playing a note so that musicians can hear the note and adjust their instruments. However *Chromatic Guitar Tuner* is a sophisticated application which is capable of analyzing the frequencies and estimating the closest notes. The limitation of this application is that it can only be deployed on high end mobile devices such as *Pocket PCs*.

Therefore the aim of this project is to develop a mobile application which could be deployed in most of mobile phones. It was a challenging task to implement frequency analysis in low end mobile devices such as *Java enabled mobile phones* due to their limited functionality. The authors of this paper have succeeded in this by finding a mechanism to implement *Fast Fourier Transform* (FFT) in *Java 2 Micro Edition* (J2ME).

Development and implementation

The major part of the product is the frequency calculation. The FFT has to be applied to obtain the frequency of the sound wave from which the notes are identified. In order to perform FFT the sound wave needed to be sampled. But the chosen platform, J2ME, does not accompany an Application Programming Interface (API) to deal with sampling the sound. Even though this limitation exists, J2ME was chosen as the development platform due to the higher number of mobile phone models supported by this platform. Then the authors used the only available option, which is to sample the sound wave within the application. The sound wave is sampled and stored as a wave file, which comprises of header (*chunk descriptor*), "*fmt*" sub chunk and "*data*" sub chunk as shown in Figure 1. As depicted in Figure 1, the "*fmt*" chunk has eight fields which are represented in little endian format except the field chunk ID. The fields which are important from this chunk for the

FFT are *SampleRate* and *BitsPerSample*. The value represented in *BitsPerSample* field is being used in deciding the number of bytes to read from data chunk per reading. The value in the sample rate field is being used in calculating the frequency. When the frequencies are known, it is mapped into the closest notes and displayed to the user. The first phase of our application samples the sound wave by recoding it as an audio clip in wave file format as shown in Figure 1.

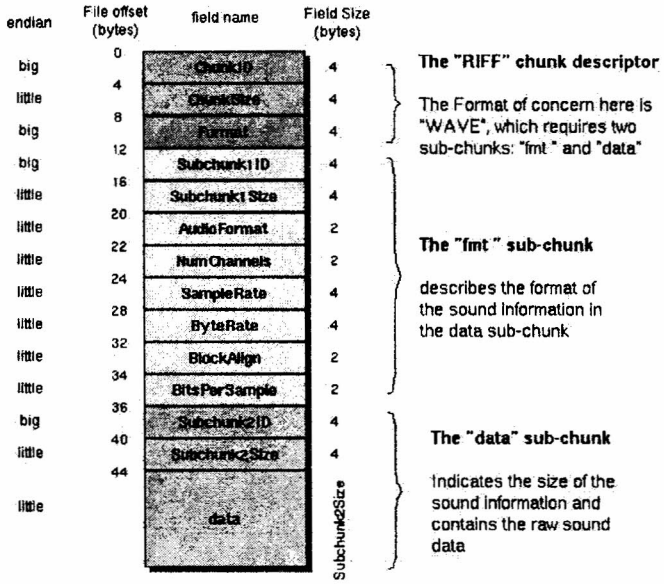

Figure 1. The WAVE File Format (taken from (Wilson, 2003))

```
Main:
       While there is data in data chunk
              Call Subroutine1 with 256 samples
                                  End while

      For (i = 0 to 256)
           FindMax (|totalCos[i] + totalCos[i]|)
           location = i
      End
      Frequency = SampleRate * location / (2*255)
End Main

Subroutine1: Input data[]
      For (i=0 to 256)
            For (j=0 to 256)
                X = (-2 * n * i * j ) / 256
                cosPart [i] = cosPart [i] + data [j] * Cos (X)
                sinPart [i] = sinPart [i] + data [j] * Sin (X)
            End For
            totalCos [i] = totalCos [i] + cosPart [i]
            totalSin [i] = totalSin [i] + sinPart [i]
      End For
End Subroutine1
```

Figure 2. Algorithm for Obtaining the Frequency

The second phase of our application analyses the audio clip in wave file format from where the information is extracted. The values *SampleRate* and *BitsPerSample* are read from "*fmt*" chunk and are stored as application variables. As in Figure 1, the data chunk comprises of a chunk ID (*Subchunk2ID*), chunk size (*Subchunk2Size*) and data. The value chunk size is being used to determine the size of the data to be read. Then 256 samples are being read to an array called data which is then being processed by *Subroutine1* in the algorithm shown in Figure 2. This process is being carried out until there are no samples to be read. Then as shown in Figure 2, amplitude of the total *cos* and *sin* values are being calculated by the *FindMax* method and the location of the highest amplitude, i.e. between 1 and 255, is determined. Then the frequency is being calculated as shown in the algorithm.

Results

The *playTone()*function (an inbuilt function of the API) in the *Manager* class is being used to play the notes A, B, C, D, E, F, G and sound is captured via the microphone. The captured data are then being sent to our application to get the frequency of the played notes. Following results were obtained.

Table 1. Frequencies obtained for notes played with Mobile Media API

Note	A	B	C	D	E	F	G
Frequency (Hz)	220	246	261	293	329	349	391

Discussion

Our application uses *cos* and *sin* functions from the *Math library*. Therefore this application can only be deployed on mobile phones with Connected Limited Device Configuration (CLDC) 1.1 (SunMicrosystems, 2006), where the *cos* and *sin* functions are implemented in the *Math library*. This limits the usability of our application as these functions are not implemented in the *Math library* of CLDC 1.0 (SunMicrosystems, CLDC Library API Specification 1.0, 2006). Therefore it is necessary to take appropriate measurements to make our application run on CLDC 1.0. The FFT processing time is comparably high due to the floating point arithmetic used during the process. This reduces the performance of the application. Further research and development is required to overcome these issues.

Conclusions

This paper presents the mechanism behind the implementation of *FFT* in *J2ME* for an electronic tuner application and to the best of our knowledge this is the first time such an effort has succeeded (SunMicrosystems, develop a Musical Tuner, 1994 - 2006).

References

4pockets.com. (n.d.). *4pockets.com*. Retrieved 07 30, 2007, from PocketPC Applications: Chromatic Guitar Tuner: http://www.4pockets.com/product_info.php?p=39

Clausen, A., Spanias, A., Xavier, A. and Tampi, M. (1998) A Java signal analysis tool for signal processing experiments, *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*. Seattle, 1998. 1849-1852.

GetJar.com. (2004-2007) *Tiny Tuner / free download*. Retrieved 07 30, 2007, from Free Java, Symbian, Palm and Pocket PC Software: http://www.getjar.com/products/5314/Tiny Tuner

Kohn, M. (1997-2007) *Guitar Tuner*. Retrieved 07 30, 2007, from Michael Kohn: http://www.mikekohn.net/j2me/guitartuner.php

SunMicrosystems (2006) *CLDC Library API Specification 1.0*. Retrieved 07 30, 2007, from Java Technology: http://java.sun.com/javame/reference/apis/jsr030/

SunMicrosystems (1994 - 2006) *Develop a Musical Tuner*. Retrieved 07 30, 2007, from Developer Forums Java Technology Forums:http://forum.java.sun.com/thread.jspa?forumID=76&threadID=613127

SunMicrosystems. (2006) *Overview (CLDC 1.1)*. Retrieved 07 30, 2007, from Java Technology: http://java.sun.com/javame/reference/apis/jsr139/

Wilson, S. (2003, 01 20) *WAVE PCM soundfile format*. Retrieved 07 30, 2007, from Center for Computer Research in Music and Acoustics:http://ccrma.stanford.edu/courses/422/projects/WaveFormat.

Zaykovskiy, D., and Schmitt, A. (2007) Java (J2ME) Front-End for distributed speech recognition, *21st International Conference on Advanced Information*, Washington 2007. 353-357.