Sound-Specific Vibration Interface using Digital Signal Processing *

Dongju Chi, Donghyun Cho,
Sungjin Oh, Kyungkoo Jun
Dept. of Multimedia Systems Engineering
University of Incheon, Korea
chidj10, dhcho, oro, kjun@incheon.ac.kr

Yonghee You, Hwanmun Lee, Meeyoung Sung
Dept. of Computer Engineering
University of Incheon, Korea
yhinfuture, leehwanmun, mysung@incheon.ac.kr

Abstract

The adoption of vibration interfaces in the games and the virtual reality is increasing since they improve the user experience. The principle of existing vibrating devices is to vibrate when detecting the presence of particular frequency bands. However such frequency-based scheme, according to our survey, cannot meet the user expectation; they vibrate too often and even at unwanted moments, having users feel haptically-numb soon and being annoyed. We develop a sound-specific vibration interface. It is wrist-wearable and vibrates at only designated target sounds. It uses a real time sound matching algorithm. Using the FFT, the algorithm compares the similarity of the frequency distribution between the input sound and the target sounds. In the performance test applying the interface to a commercial game, the matching success rate was 80% while the computation delay was 415 ms. Such results confirm the applicability of the proposed interface to the game play.

Keywords: FFT, Vibration, Sound, Interface

1 Introduction

The reality improvement in games and virtual reality are being sought by various research efforts. The graphics and sound effects are the main focus of such efforts. Recently, haptics, i.e. sense of touch emerges as a next move for further enhanced reality. Such moves share the motivation discussed in [1] saying that haptic effects accompanying visual and sound elements can enrich the user experiences. The simplest way to integrate the haptics is by means of vibration-enabled interfaces such as joystick, mouse, and headset.

The ways to use the vibration interfaces can be categorized into two types. The first one is that applications embed the codes to control the vibration. However this type has very limited use because only 10% of games consider the vibration at the development stage. The second one is more applicable in the sense that the interfaces themselves are equipped with the frequency detection capability; on detecting the presence of certain frequency bands, they vibrate accordingly.

Although such frequency-based vibration supported by the interface is useful, it has critical drawbacks. According to [2], such frequency-driven way is less effective than expected. For instance, they vibrate too often and even at unwanted moments, having users feel haptically-numb and being annoyed. It is because those interfaces are configured to vibrate at a low tone of 100 $\sim$ 150 Hz, and too many sound effects including the background music contain the frequency elements belonging to that band.

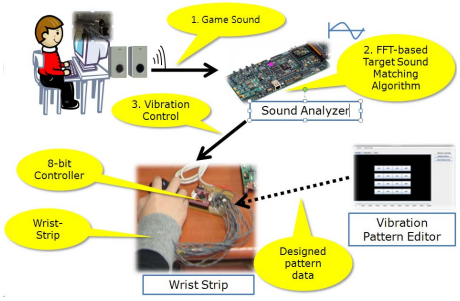


Figure 1. Sound-specific vibration interface: it vibrates responding to the sounds of games and virtual reality.

We propose a sound specific vibrating interface as shown in Figure 1. Being worn around the wrist, the interface vibrates only when user-specified sounds occur. It is different from the existing interfaces in that it matches sounds, not frequency. For example, in first person shooting (FPS) games, the interface can be configured to vibrate only when certain gunfire occur. The digital signal processor is used

*This work was supported by the grant from the Regional Technology Innovation Program of the Ministry of Knowledge Economy and by the Brain Korea 21 Project in 2008

978-0-7695-3336-0/08 $25.00 © 2008 IEEE
DOI 10.1109/CSSE.2008.931

for such sound matching. Another feature of our system is that users are able to customize their own vibrating patterns by using a tool provided by our system, i.e. *vibration pattern editor*.

For the sound matching, past research efforts can be summarized as follows. Speech recognition by the hidden Markov model (HMM) [3] is one example. The HMM uses the fast Fourier transformation (FFT) to match input sound with prepared samples. However, the HMM is far more complicated than needed for our case. [4] is another example. It is to match particular sound playback by comparing the frequency distribution of the input sound with stored ones. However, it lacks the real time processing capability, which is essential for our case.

This paper is organized as follows. Section 2 introduces the sound-specific vibration interface and discusses the sound matching algorithm in detail. Section 3 presents the performance results of the algorithm and Section 4 concludes the paper.

2 Sound Specific Vibration Interface

The sound specific vibration interface receives the PC sound as input and, at the moments when certain target sounds occur, vibrates a wrist wearable interface. To detect the occurrences of the target sounds, we compare the frequency distribution of the input sound with that of the target sounds. It can distinguish between multiple target sounds, and vibrate differently for each target sound.

Our interface consists of three components as shown in Figure 1: a sound analyzer, a wrist strip, and a vibration pattern editor. The sound analyzer is to detect the occurrences of the target sounds by analyzing the frequency distribution. For efficient computation, the sound analyzer is built with a digital signal processor (DSP) on an external board. More details about the sound analyzer are discussed shortly. The wrist strip is an interface to convey the vibration to users. It is worn around the wrist and a set of coin-type oscillators in it generate the vibration. The vibration pattern editor is a Java application to program the wrist strip with various vibration patterns. It programs the firmware of an 8-bit micro-controller inside the wrist strip. It provides a graphical user interface for users to edit the patterns visually. Due to the space limitation, the details about the wrist strip and the editor are omitted, but can be referred to [5].

Here, we discuss the details about the algorithm of the sound analyzer. The sound analyzer is to find the match of an input sound among one of the target sounds. For this, it pre-calculates and stores the frequency distribution of the target sounds. Then, in real time, it computes the frequency distribution of the input sound and finds the match among the target sounds. The FFT is used to compute the frequency distribution.

However, the application of the FFT faces several challenges. Firstly, volume dependency should be handled; depending on the volume level, the results of the FFT vary. Secondly, picking up the time segment containing the target sounds from continuous input sound is not trivial. Silence period should be filtered out for efficiency, but be careful not to miss the target sound. Thirdly, the comparison between the frequency distributions should be computation-efficient. Comparison at all frequency bands is not only time consuming but also obviously harmful to satisfying the real time requirement.

Our algorithm proposes efficient solutions for these challenges. We here summarizes them briefly and discusses the details later. For the volume independency, we use the ratio among the magnitudes of the FFT results because the ratio is preserved regardless of the volume. For this, we perform *normalization* on the FFT results, i.e. dividing the magnitudes of all the bands by the maximum magnitude. For efficiently filtering out the silence period, we perform the computation only when the input signal level is higher than a threshold. In addition, we overlap the FFT windows in order not to miss the target sounds. Finally we compare only a set of representative frequency bands which characterize the target sound; we call it *selective comparison*, for efficient comparison.

Section 2.1 explains how to extract from the target sounds a set of the frequency bands that distinguish the sounds. Section 2.2 describes an algorithm to match the input sound with the target sounds. The algorithm checks whether the frequency characteristics of the target sounds are found in the input sound.

2.1 Extraction of the Characteristics of Target Sounds

In this section, we explain how to choose a set of frequency bands that characterize the target sounds. The similarity on those frequency bands is the criterion for the sound matching. More specifically, when matching the sound, not only the bands but also the magnitude values on those bands are compared.

For a target sound to which the FFT is applied, we select n frequency bands with the biggest magnitude values in descending order, i.e. $f_0, f_1, \cdots f_{n-1}$ where f_0 is the band with the maximum magnitude. Here, we enforce that any two selected bands f_i, f_j should be apart at least a designated value δ, i.e.

$$|f_i - f_j| > \delta, \quad i \neq j \tag{1}$$

It helps capture the sound characteristic which occurs on multiple discrete parts of the frequency bands because it prevents the selected bands f_is from being crowded within a short range of frequencies.

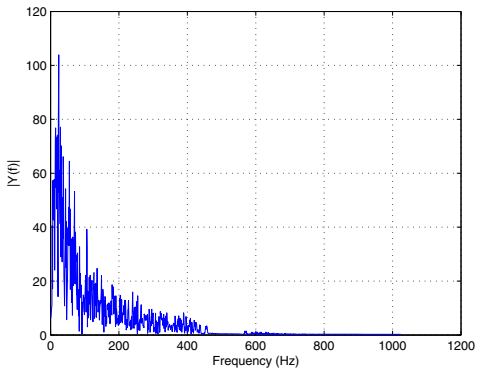


Figure 2. The FFT result of a target sound: a gunfire of a rifle

Then, we perform the normalization for the volume independency. It divides the magnitude value $m_{i,\,0\leq i\leq n-1}$ of the selected frequency band f_i by m_0 which is the maximum magnitude.

Figure 2 shows the FFT result of a gunfire example as a target sound. The 2048-point FFT is performed on a sound clip sampled at 8 KHz and 0.2 second long. With $\delta = 19.5Hz$ for Eq. 1 and $n = 10$, the following 10 frequency bands are chosen; 93.7, 58.5, 121.0, 144.5, 214.8, 27.3, 171.8, 273.4, 414.0, 238.2 Hz, and the corresponding normalized magnitudes are 1, 0.74, 0.67, 0.64, 0.61, 0.55, 0.52, 0.51, 0.37, 0.32, respectively.

2.2 Real-time Sound Matching Algorithm

We present an algorithm to match an input sound with the target sounds in real time. With the continuous input sound, it regularly performs the FFT, and then compares the normalized magnitudes with those of the target sounds. Real time processing is one of the critical requirements of this algorithm. The flow of the algorithm is as follows.

1. Firstly, the input sound is sampled at K Hz, and the samples are stored in a circular buffer, replacing old samples with new ones when the buffer is full.

2. The second step ensures that the sound fragment in the buffer is more than just noise by ensuring the sum of the buffered sample values bigger than a threshold A. Otherwise it goes back to the step 1.

3. The third step applies the FFT on the buffered samples to analyze the frequency distribution.

4. The fourth step normalizes the FFT results for the volume independency. The normalization performs only on the magnitudes of the frequencies. saving unnecessary computation.

5. The fifth step, for each target sound snd_k, computes the summation sum_k of the absolute differences between the magnitudes $m_{i,\,0\leq i\leq n-1}$ of snd_k and those of the input sound as follows

$$sum_k = \sum_{i=0}^{n-1} |m_i^k - m_i| \tag{2}$$

where m_i^k is the normalized magnitude of the frequency f_i of the target sound snd_k, m_i is the magnitude of the frequency f_i of the input sound.

6. The sixth step chooses the target sound snd_k with the minimum sum_k and at the same time that satisfies

$$min(sum_k) < sum_{max} \tag{3}$$

where sum_{max} is a threshold which limits the maximum differences between the input sound and its matching sound. Without the threshold sum_{max}, since the minimum sum_k always exists as a result of Eq. 2, incorrect matching decision would be made.

7. Finally, on matching a target sound snd_k, it vibrates the wrist strip.

The sound matching algorithm runs on a DSP board for performance. Since the main processor is busy in running games, if the additional FFT computation was performed on the processor too, it would degrade the game experience.

3 Performance Evaluation

This section discusses the performance of the proposed sound matching algorithm. We measured the matching success rate and the delay for the computation. Such measurement was carried out by playing a commercial FPS game, *Call of Duty I* as shown in Figure 3. Using the commercial game in the measurement serves an additional purpose: i.e. test the applicability of our interface to commercial games. As the target sounds, two types of gunfires were used; a rifle and a machine gun. We used digital signal processor with 225 MHz clock speed to run the algorithm. The parameters used for the measurement are as follows. Sampling rate of the PC sound was 8 KHz, each sample data is 16-bit long. 2048-point FFT was applied on the sampled data, and 36 samples were shifted out at each iteration, resulting in 2012 samples overlapped. Sum_{max} and δ are set 1.0 and 5 Hz, respectively.

Figure 4(a) shows the matching success rate. The background noise level was increased gradually in order to observe the changes in the success rate. The x axis is the

Figure 3. Performance Measurement: we measured the performance by playing a commercial FPS game, *Call of Duty I*

signal-to-noise (SNR) in dB, in other words, the ratio of the input sound volume to the noise. The y axis is the success rate. At SNR 20 dB, the matching rates of both gunfires are over 85% but, ss the noise increased, the success rate started to drop. At SNR 10 db which is similar to the game play environment, the success rates were around 80%. Such results are not quite satisfactory, but show that the proposed interface is still useful for the game play.

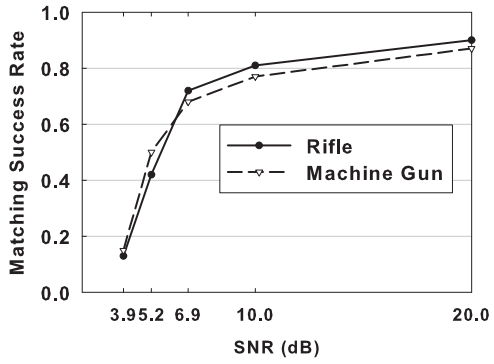
Figure 4(b) shows the delay for the algorithm computation including the FFT and the matching process. Two more gunfires were used to increase the matching overhead. The x axis is the number of the target sounds to be compared and the y axis is the delay in milliseconds. It took around average 415 ms, which did not cause critical mismatching delay from the moment of a sound occurrence to the corresponding vibration. The delay increase due to the number of the target sounds was insignificant because the most delay is caused by the FFT, thus the number of the target sounds imposes relatively little overhead.

4 Conclusions

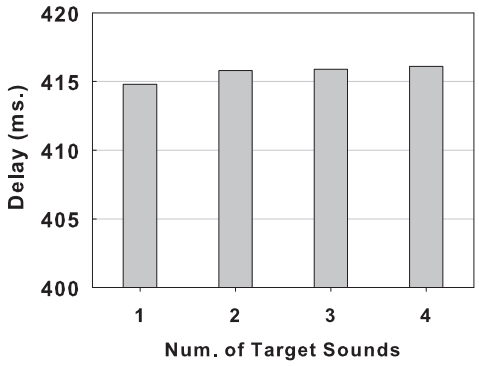
This paper discussed the development of the sound-specific vibration interface including the sound matching algorithm. We applied the interface to the commercial FPS game by configuring it to vibrate at the gunfires and confirmed its applicability to the game play. The performance evaluation showed that the matching rate is about 80% and the delay is around 415 ms.

References

[1] A. Chang, C. O'Sullivan, "Audio Haptic Feedback in Mobile Phones," in Proceedings of Conference on Human Factors in Computing, 2005.



(a)



(b)

Figure 4. The matching success rate according to SNR 4(a) and the delay to compute the matchings depending on the number of the target sounds 4(b)

[2] H. Lee, Y. You, C. Song, J. Jeong, M. Sung, K. Jun, S. Lee, "Analysis of Tactitle Effects on the Different Body Parts by the Various Vibration Patterns," in Proceedings of HCI 2008, Korea.

[3] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," in the Proceedings of the IEE, 77(2), p.257-286, February 1989.

[4] D. Hoiem, Y. Ke, R. Sukthankar, "SOLAR: Sound Object Localization and Retrieval in Complex Audio Environments," in the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.

[5] J. Oh, D. Cho, Y. You, M. Sung, K. Jun, "Wrist Strip and Pattern Editor for Sound Specific Vibration Interface," in Proceedings of HCI 2008, Korea.