IEICE TRANS. INF. & SYST., VOL.E94–D, NO.8 AUGUST 2011

PAPER

Sound Specific Vibration Interface for Enhancing Reality in Computer Games

Kyungkoo JUN[†a)], ***Member***

SUMMARY This paper presents the development of a sound–specific vibration interface and its evaluation results by playing three commercial games with the interface. The proposed interface complements the pitfalls of existing frequency–based vibration interfaces such as vibrating headsets, mouses, and joysticks. Those interfaces may bring negative user experiences by generating incessant vibrations because they vibrate in response to certain sound frequencies. But the proposed interface which responds to only target sounds can improve user experiences effectively. The hardware and software parts of the interface are described; the structure and the implementation of a wrist pad that delivers vibration are discussed. Furthermore, we explain a sound-matching algorithm that extracts sound characteristics and a GUI-based pattern editor that helps users to design vibration patterns. The results from evaluating the performance show that the success ratio of the sound matching is over 90% at the volume of 20 dB and the delay time is around 400 msec. In the survey about user experiences, the users evaluates that the interface is more than four times effective in improving the reality of game playing than without using the vibration interfaces, and two times than the frequency–based ones.

key words: *vibration interface, sound, frequency, haptics, reality, games*

1. Introduction

Reality in games and virtual reality applications has been sought by various research efforts. Graphics and sound areas have been the central focus of such efforts. Recently, haptics, i.e. sense of touch, has emerged as a next step to move on for further enhanced user experience. Such moves share the motivation discussed in [1], insisting that haptic effects accompanying visual and audio effects can enrich user experience.

A simplest way to apply haptic effects to applications is via vibration interfaces such as vibrating mouses [2], headsets [3], joysticks [4]. There are two types of such interfaces depending on how the vibration is triggered. The first type is program-based. Applications themselves embed codes to control the vibration interfaces. The second type is frequency-based. The interfaces of this type are able to analyze the frequency distribution of sound. On detecting the presence of certain frequency bands, the interfaces generate vibration.

However, since the program-based ones are tightly coupled with applications, the compatibility issues between interfaces and applications pose an obstacle to its widespread adoption. The frequency-based interfaces have the problems to trigger vibrations unnecessarily often and improperly. According to the survey undertaken in [5], a group of people who used vibrating mouses and headsets for game play complained that the vibration occurs too often and sometimes at unwanted moments, having them feel haptically-numb soon and even being annoyed. It was because the frequency bands to which the headset and the mouse vibrates exist in many sound effects of the game including background music.

Therefore, the frequency-based interfaces need to be able to generate vibration only at certain types of sounds, referred to hereafter as *target sounds*, while not responding to frequency bands. To do so, the sound detection and recognition capability is crucial. Past research efforts related with the sound matching are as follows. Speech recognition by Hidden Markov model (HMM) [6] uses the fast Fourier transformation (FFT) and the learning process to search for the match of a target sound over input sound. However, the computing requirements of HMM can easily exceed the capacity that gaming interfaces provide. SOLAR [7] compares the frequency distribution with less computing overhead to recognize occurrences of target sounds. However, it is a non realtime solution that cannot be used for game play. [8] tries to detect and recognize impulsive sounds such as glass breaks, human screams. It uses the non-linear median filter for the detection and two statistical classifiers based on Gaussian Mixture Models and HMM for the recognition. However, it is too complex to put into gaming interfaces and it is not designed for real time use. [9] combines statistical pattern recognition methods with statistical signal processing and [10] proposes a detection method using dyadic trees of wavelet coefficients and a recognition method using GMM. However, neither of them is designed for realtime use and light enough to be embedded in gaming interfaces.

This paper proposes a sound specific interface that vibrates responding to only a set of pre-specified target sounds. It can provide real-time sound matching with low computing overhead. In addition, a GUI-based software tool by which users design various vibration patterns for target sounds is introduced.

This paper is organized as follows. Section 2 introduces the proposed vibration interface by explaining its hardware and software aspects. Section 3 presents the test results of the proposed interface. The results from the performance measurement of the sound matching and the survey about user experiences are discussed. Section 4 concludes this paper.

Manuscript received September 27, 2010.
Manuscript revised March 21, 2011.

[†]The author is with the Dept. of Embedded Systems Engineering, University of Incheon, Korea.
a) E-mail: kjun@incheon.ac.kr

DOI: 10.1587/transinf.E94.D.1628

2. Sound–Specific Vibration Interface

The proposed interface consists of hardware and software parts. The hardware parts are a device that delivers vibration to human users and a board that runs a sound matching algorithm. Regarding the software parts, there is a vibration pattern editor along with the sound matching algorithm.

2.1 Hardware

The proposed interface consists of a sound analyzer and a wrist pad as shown in Fig. 1 and its implementation is in Fig. 2. The sound analyzer receives sounds from game play as input and performs a sound matching algorithm on them to check whether it is the sound to trigger vibration. It then requests the wrist pad to generate vibration. The vibration varies depending on the matched sounds. Since the sound input is given in realtime, the algorithm should also run in the same way.

The sound analyzer is implemented by using a Davinci evaluation board [11] equipped with a TMS320DM6437 digital media processor of Texas Instruments. It runs at the speed of 400 MHz. Among the peripherals on the board, only UART port is required to communicate with the wrist pad.

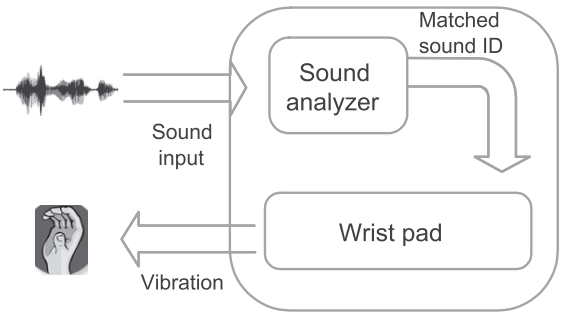

Fig. 1 The proposed interface consists of a sound analyzer and a wrist pad.

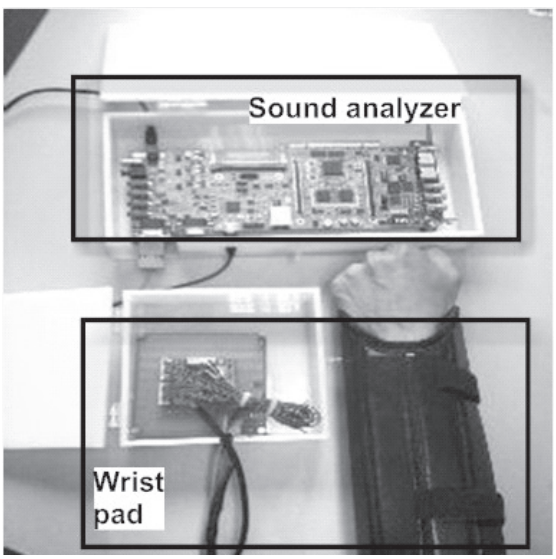

Fig. 2 The implementation of the proposed interface.

The wrist pad consists of an array of vibration motors and a controller. The vibration motors which are coin type require 3 V as input voltage and can produce 9,000 RPM (rotations per minute) vibration. These motors are widely used in most mobile phones that generate haptic output. The array of the motors are attached to a pad, which human users wrap it around their arms or wrists when they play games. The array is composed of sixteen motors. There are two types of the pad. One is a protector type as shown in Fig. 2 and the other is a bandage type as shown in Fig. 7. The protector type is more durable and can deliver the vibration more sensitively than the bandage type.

The controller that steers the vibration of the motors is connected with them as shown in Fig. 3. The controller is operated by an ATMEGA128 MCU [12] which is 8-bit microprocessor. It connects with the motors through general purpose input/output (GPIO) ports. Since one GPIO port can manage up to eight motors, two GPIO ports are used to control a total of sixteen motors. The motors are controlled by writing one byte value into the port. Each bit corresponds to one of the eight motors. Setting 1 on a bit causes a corresponding motor to vibrate, 0 is to stop the vibration. The duration of the vibration is controlled by the interval between the bit value changes. However, the strength is not manageable.

The controller stores different vibration patterns depending on target sounds. The pattern is a description about which motors start vibrating, from when, for how long. It is created by a pattern editor which will be discussed shortly. The controller is connected with the sound analyzer through a serial port. Once one of target sounds are detected, the sound analyzer sends to the controller a unique identifier of the target sound. The identifier is predefined by the pattern editor.

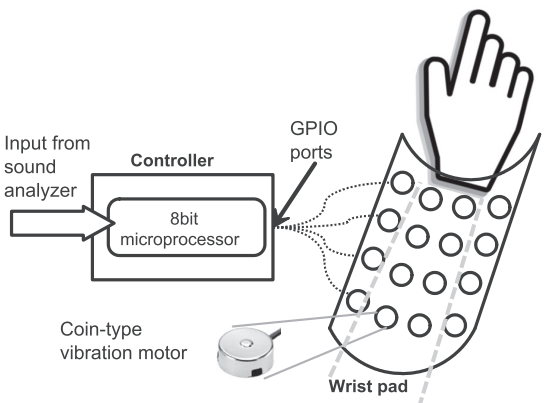

Fig. 3 The controller steers the vibrations of the motors.

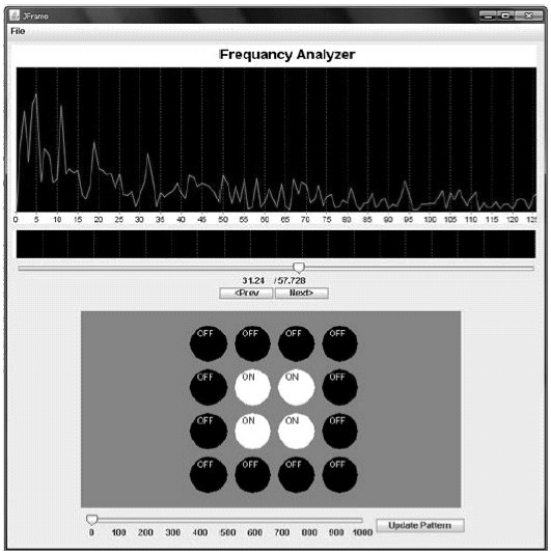

Fig. 4 The pattern editor extracts the characteristics from target sounds and allows users to design vibration patterns.

2.2 Software

This section discusses the pattern editor and the sound matching algorithm. The pattern editor is a GUI-based tool. Given a target sound, it generates a sound vector that represents the characteristics of the sound. Technically, the sound vector of a sound is a set of pairs of selected frequencies and their magnitudes. In addition, the editor allows users to design the vibration pattern for the sound. The sound matching algorithm is to detect and recognize target sounds from sound input in realtime. For matching, it uses the sound vectors from the pattern editor.

Figure 4 shows the pattern editor. Given a target sound, the top half shows that the editor analyzes the frequency distribution of the sound to extract the characteristics. The bottom half presents an interface that allow users to design a vibration pattern for the sound. The circles correspond to the sixteen motors and the time bar at the bottom represents the duration of the vibration. Users can create a pattern by mouse clicking and dragging.

The sound vector of a target sound is made from the results of applying FFT on the sound. It is a set of pairs whose elements are a frequency and its magnitude. From the FFT results, n frequencies with highest magnitudes are selected. The vectors have their own unique identifiers to denote matched sounds. The details about producing the sound vectors are discussed when presenting the sound matching algorithm.

Figure 5 shows the format of the vibration pattern created by the pattern editor. The first field is the pattern length in bytes since the patterns have variable lengths. The second field denotes the identifier of the sound vector with which this vibration pattern associates. The rest is a list of pairs having the duration and the pattern. The duration in milliseconds specifies the lasting time of the pattern. The pattern is 2 byte-long and each bit matches with one of the sixteen motors. If the bits are set 1, the corresponding motors are on to vibrate. Using a total of 16 motors that are placed in 4 rows each row with 4 motors, for example, the vibration pattern for the gunshot of rifle is to vibrate only 8 motors, two each from the four rows, for 100 msec. For a sniper rifle, the pattern of 8 motors with a longer duration of 300 msec. is used because the sniper rifle has a more lasting resonance than a rifle. Another pattern that vibrates all the 16 motors for 300 msec. is for the gunshot of a long range rifle that has bigger reactionary shock than a sniper rifle. For a machine gun, each row of the motors vibrates for 100 msec. row by row sequentially and repeatedly, producing a wave-like form of vibration.

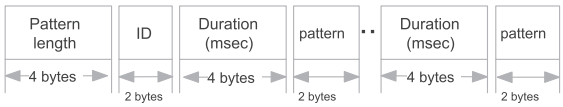

Fig. 5 The format of the vibration pattern.

The sound vectors are stored in the sound analyzer to find matched sounds, while the vibration patterns are loaded into the wrist pad. Once matching a target sound, the sound analyzer delivers the identifier of the matched sound vector to the wrist pad, which then controls the vibration according to the corresponding pattern.

Challenges in developing a sound matching algorithm for the proposed interface are as follows. Firstly, the comparison of the frequency distributions is a time-consuming operation, if done in a sequential way it is not able to meet the real-time requirements. Secondly, the FFT results change depending on sound volume, producing different magnitude values as the volume changes. Thirdly, it is unknown exactly at what moments target sounds start within continuous sound signal, thus easily missing the right moment to apply the FFT.

To face these challenges, the proposed algorithm introduces the following techniques. Firstly, *selective comparison* and *silence filtering* are used to reduce computation overhead. The selective comparison is to compare frequencies only over a small set of frequency bands by using the sound vector. The silence filtering is to skip the computation over the silence periods. Secondly, *normalization* is for volume independency. It exploits the magnitude ratio which is not affected by the volume change. The magnitude values are normalized by dividing by the maximum magnitude. Thirdly, a *sliding FFT* reduces the probability to miss the target sounds by moving a FFT window along the input sound signal overlapping with the previous window.

The sound vectors used by the selective comparison are created as follows. From the FFT results of a target sound S, n frequencies, $f_0, f_1, \cdots f_{n-1}$, are selected in descending magnitudes where f_0 is the frequency with the biggest magnitude. One restriction imposing on the selection is that any two f_i, $f_j, i \neq j$ should be apart at least a pre-specified δ

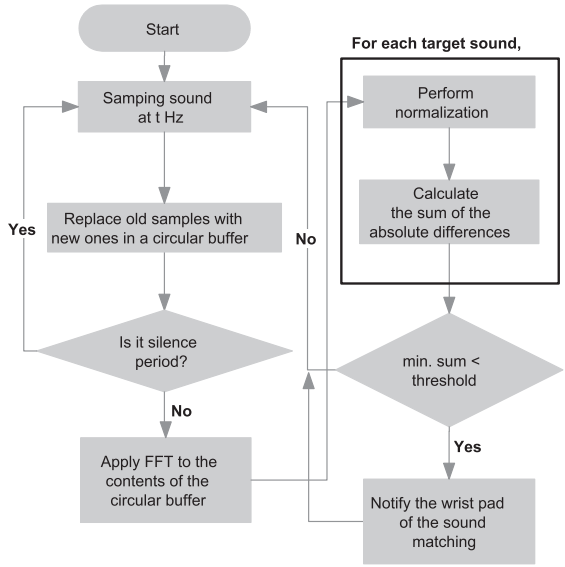

Fig. 6 The flow diagram of the sound matching algorithm.

guard band, i.e.

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

δ has the effect to spread f_is over B_{in}, the bandwidth of interest, particularly in this paper between 150 Hz to 500 Hz because most of sound effects have their characteristic frequencies over that range. Otherwise, f_is are cluttered over narrow band and comparison becomes less effective. Thus, δ is determined by dividing B_{in} by n, then it becomes inversely proportional to n as follows.

$$\delta = \frac{B_{in}}{n} \tag{2}$$

The normalization is the next step. The magnitude of f_0 which has the biggest value divides the magnitudes of f_is, producing the normalized magnitudes m_is $i = 0, 1, \cdots, n-1$ where $0 \leq m_i \leq 1$. Comparing m_is at only f_is can reduce the computation amount to determine the sound matching.

Figure 6 shows the flow diagram of the proposed sound matching algorithm. It is assumed that there are l target sounds to match and the sound vector of a target sound snd_j has n frequencies f_i^js and corresponding normalized magnitudes m_i^j, $i = 0, 1, \cdots, n-1$. The steps of the algorithm are as follows.

1. Input sound is sampled at t Hz. When determining t, it should be considered whether its Nyquist frequency is able to cover the whole range of target sounds. The resulting samples are stored in a circular buffer. The buffer capacity is equal to the FFT window size, which is in turn equal to the number of FFT points. And the number of FFT points should be determined by considering whether its resulting frequency resolution is fine enough to capture variations of target sounds. Once the buffer becomes full, it drops off oldest samples first as new samples arrive. And, the number of samples to be dropped at a time is s, which is the sliding size of the FFT window
2. Test whether the buffered samples represent a non-silence period by comparing the sum of all the samples with a silence-filter threshold γ. If the sum is smaller than γ, go back to the step 1 to avoid processing the silence period. Otherwise it proceeds to the next step. To determine γ, it is necessary to perform a set of tests to actually obtain the sum values during silence periods.
3. Apply the FFT on the buffered samples. For the sound vector of each target sound, snd_j, $0 \leq j \leq l-1$, calculate the sum of absolute differences as follows.
$$sum_j = \sum_{i=0}^{n-1} |m_i^j - \mu_i^j| \tag{3}$$
where m_i^j is the magnitude at the frequency f_i^j of a target sound snd_j, μ_i^j is the magnitude at the frequency f_i^j of the sample. μ_i^js are normalized by $max(\mu_i^j)$ $\forall i = 0, 1, \cdots, n-1$ before the calculation.
4. Decide the matched sound by finding a target sound snd_k of which sum_k is as follows.
$$sum_k \leq min(sum_j, sum_{max}) \forall j = 0, 1, \cdots, l-1 \tag{4}$$
where sum_{max} is a difference threshold to impose the required similarity level. If no sum_j satisfies the condition, the matching fails. If so, go back to the step 1, otherwise move to the next step.
5. Command the wrist pad to vibrate by sending the ID that corresponds to the matched target sound. The ID is the one defined by the pattern editor earlier.
6. Go back to the step 1.

To analyze the complexity of the sound matching algorithm, let q, l, and n denote the number of FFT points, the number of target sounds, and the number of frequencies that compose a sound vector of a target sound, respectively. Firstly, the step 2 performs $q-1$ additions and 1 comparison to determine whether the samples represent non-silence period, resulting in q operations. Secondly, only in the case of non-silence period, the FFT is applied in the step 3 which is known to have the time complexity of $O(q \log_2 q)$. Thirdly, the step 3 also calculates the sum of absolute differences on n frequencies of a sound vector for each of l target sounds. It involves n subtractions and $n-1$ additions for each target sound and additional $n-1$ comparisons and n division that normalize the magnitude values, thus resulting in $l(4n-2)$ operations. Finally, the step 4 performs $l-1$ comparisons to find matching sounds. The sum of all the numbers of operations becomes $q + O(q \log_2 q) + 4ln - l - 1$, and hence the complexity is $O(q \log_2 q + ln)$. In game-playing environments, q is significantly larger than l or n, e.g. 2048 points of FFT is used while there are less than 10 target sounds and less than 50 frequency bands to be compared. Thus the performance largely depends on the amount time that the FFT consumes rather than the number of target sounds or the sound vectors.

Regarding the amount of computation, the average or expected value EC can be obtained because the step 3 and 4 are performed only in the case of non-silence period. Let p_s denote the probability that the samples of the step 2 represent a silence period $p_s = Pr\{\sum samples < \gamma\}$. Then,

$$EC = q + (1 - p_s)(q \log_2 q + 4ln - l - 1) \quad (5)$$

Since γ determines p_s, it can influence the amount of the computation, although it does not change the complexity itself. However, another parameter δ introduced in the sound matching algorithm does not influence either the complexity or the computation amount. It is because the preparation of sound vectors in which δ involves is performed only once and before starting real–time processing. Eq. (5) also implies that the FFT computation is a dominant factor that influences on delays. Also, since the number of target sounds in games is not expected to be large, in most cases less than 10, the delays by the increasing number of target sounds would be insignificant, typically compared with that of the FFT.

3. Performance Evaluation

The evaluation of the proposed interface is carried out in the aspects of performance and user experience. For that, the interface is applied to the play of commercial games. Three different genres of the games are selected to test the interface under various environment. As shown in Fig. 7, 8, 9, the games are a First Person Shooting (FPS) game, a car driving game, and a role–playing action game. The FPS game has four different types of gun shots as the target sounds. The car driving game has three target sounds: acceleration, brake, and crash. The role–playing game has three target sounds: sword swing, sword clang, and explosion.

As the algorithm parameters, the sampling rate is 8 KHz and each sample is 16-bit long. 2048-point FFT is applied and old 36 samples slides out at every FFT window moving, thus 2012 samples being overlapped between windows. γ and sum_{max} are set to 5 and 1.0 respectively.

For the performance evaluation, the ratio of success in the target sound matching and the delay for the computation are measured. While playing the games, the success ratio is calculated for each target sound. For this, the sound analyzer is modified to output the names of matched sounds through its debug port whenever it matches target sounds. Checking whether the names are really those of the actual sounds can determine the success. Regarding the volume, the measurements are performed under five different levels. At least over 100 attempts for one target sound at each volume are tried. For the delay, the times only from the correct matching attempts are averaged.

The audio output jack of a PC on which games run is connected through a wire to the audio input jack of the sound analyzer. Because such wiring is susceptible to noise, sound volume relative to noise is a major factor that influences the success probability of sound matching. Figure 10 shows the success ratios of the three games under different noise levels. Figure 10 (a), Fig. 10 (b) and Fig. 10 (c) are the results from the FPS game, the car driving game and the role-playing game, respectively. The success ratios escalate rapidly as the SNR level increases from 3.9 to 6.9 dB of SNR, but since then grow smoothly until the success ra-

Fig. 7 The first person shooting game is played with the proposed interface.

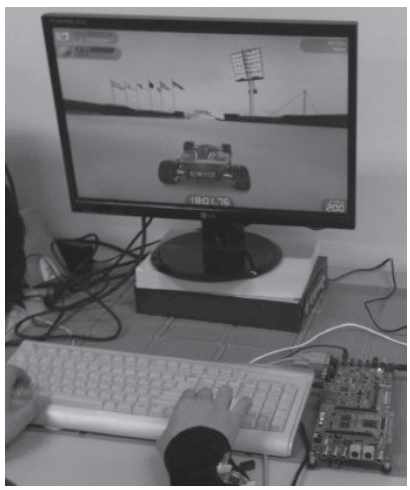

Fig. 8 The car driving game is played with the proposed interface.

Fig. 9 The role–playing game is played with the proposed interface.

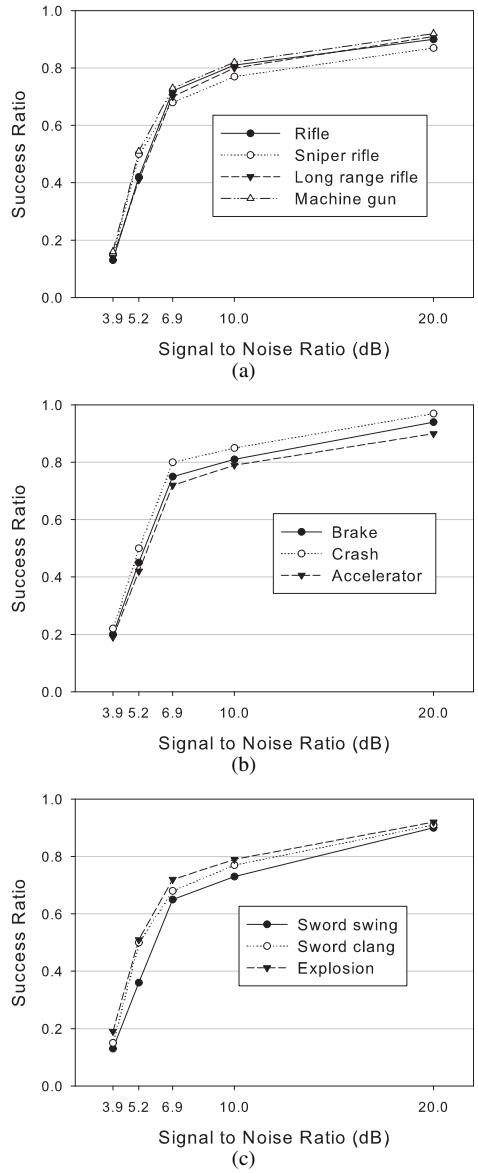

Fig. 10 The success ratio in matching target sounds in three games; (a) the results from the FPS game (b) the results from the car driving game, (c) the results from the role–playing game.

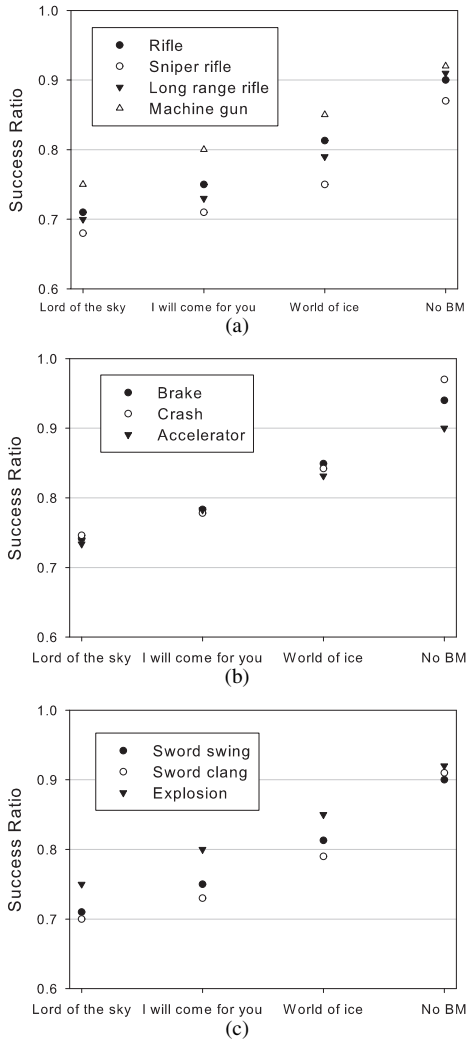

Fig. 11 The success ratio when background music is on; (a) the results from the FPS game (b) the results from the car driving game, (c) the results from the role–playing game.

tios in all the three games reach over 90% at 20 dB, which is around 70% to 80% of the maximum audio output of the PC that was used in the experiment. The rapid increase in the ratios between 3.9 dB and 6.9 dB shows that the matching success in this interval is heavily dependent on signal attenuation due to noise because small changes in SNR levels bring about large improvement in the ratios. Meanwhile, the smooth increase beyond 6.9 dB indicates that the algorithm itself is a major determinant of the matching success.

The performance of playing games with background music on is also investigated. Three sets of background music are used, lord of the sky [14], I will come for you [15], and world of ice [16], which are freely available in the internet. While the volume level of game sounds is set to the maximum SNR, the background music has only 40% of the volume level of the game sounds, which is considered as a casual setting in game play. The experiments use a separate media player running in background to play the music and are repeated at least five times for each target sound to average the success ratios.

Figure 11 shows the success ratios from each of three games along with the titles of the three sets of background music and 'No BM' which means the case without background music. The success ratios with background music are least 10% to 25% lower than without it. It is because background music is a major interfering factor in sound detection. However, it was observed that the performance varies depending on the style of background music and also what part of music was in play when target sounds occurred. For example, since 'world of ice' is softer and more delicate than 'lord of the sky' which is rather grandeur, it less interferes with target sounds. But, even 'lord of the sky' still has partially softer duration during which sound detection is easier than other parts.

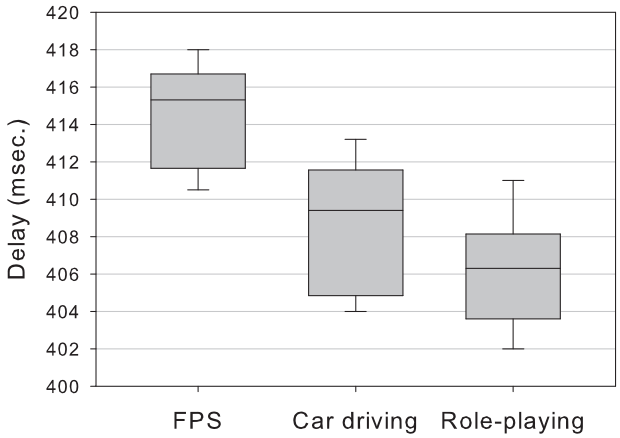

Fig. 12 The computing delays for the sound matching in the three games.

Figure 12 shows the box plots of the delays in the sound matching of the three games. The delays were measured in milliseconds each for the three games with the volume set to 20 dB. Since the amount of computation remains the same regardless of the volume, the delay would not be affected by the volume. In the box plots, the middle line inside a box indicates the median value, the top line of a box is the 75 percentile and the bottom line is 25 percentile. The top–most and bottom-most bars are 90 and 10 percentiles, respectively. The median delay of the FPS game is at 415 msec., while those of the other two games are around 409 msec. and 406 msec. Users barely recognized 415 to 416 delays according to [5].

A survey was carried out to evaluate how much the proposed interface contributes to the improvements of user experiences. The surveyees of twenty two university students, randomly chosen 17 males and 5 females, were asked to play the three games in three different settings: without vibration interface, with vibrating headset and mouse, and with the proposed interface. They played all the combinations of the games and the interfaces for one and a half minute and answered for each game the following two questions by the integer scale of 0 to 5; "how much reality improvement do you feel comparing with the case without vibration?", and "how much inconvenience do you feel comparing with the case without vibration?". In addition, they were also asked about areas of improvements regarding the proposed interface.

Figure 13 shows the survey results about the question, "how much reality improvement do you feel comparing with the case without vibration?". The results show that the proposed interface is two times more effective in improving reality than the headset and mouse; the scores of the proposed interface are evenly over 4 in all the three games, while those of the headset and mouse vary depending on the games. According to the surveyees, the ability to produce different vibration patterns for different target sounds was the main reason that the proposed interface was highly evaluated. On the contrary, the monotonic vibration of the headset and mouse worked well only with the gunshots of the FPS game. But

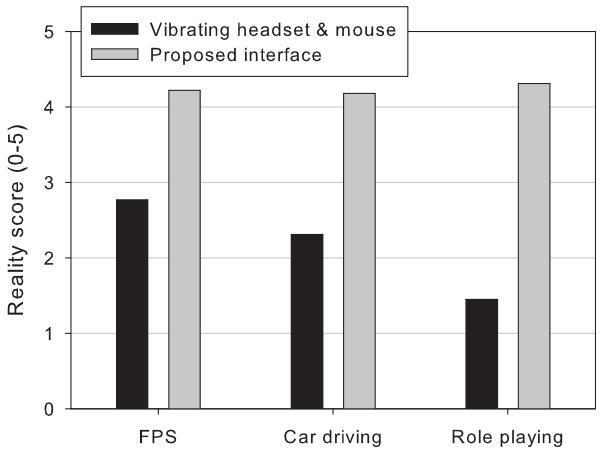

Fig. 13 The results about the question, "How much reality improvement do you feel comparing with the case without vibration?" (5: the highest level of reality - 0: no reality at all).

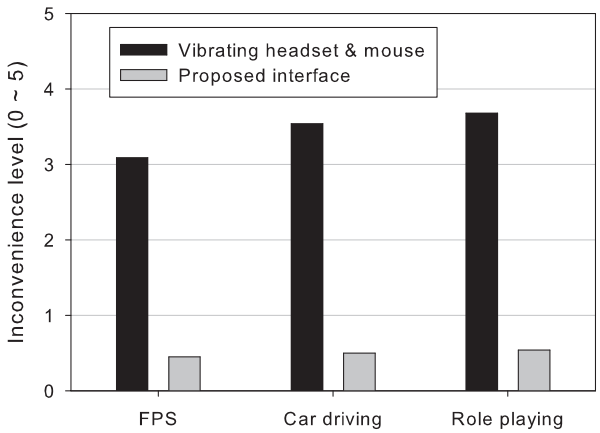

Fig. 14 The results about the question, "How much inconvenience do you feel comparing with the case without vibration?" (5: very inconvenient - 0: no inconvenience at all).

it was not impressive in the role playing game and the car driving game.

Figure 14 shows the results of "how much inconvenience do you feel comparing with the case without vibration?". The scores of the proposed interface were uniformly under 1 meaning little inconvenience, while those of the headset and mouse were above 3 that indicates a notable level of inconvenience. The surveyees complained that the headset and mouse vibrated almost incessantly regardless of the occurrences of target sounds, thus disturbed their game play. The surveyees felt awkwardness about the proposed interface at first because they should wear the wrist pad. But they soon became familiar with the proposed interface, feeling very low level of inconvenience.

Regarding the areas of improvements, 14 of the surveyees pointed out its design issues; particularly recommended to arrange neatly the wires that connect the wrist pad with the sound analyzer and mentioned about the comfortableness of the pad itself. 10 of them commented about the need to diversify the vibration patterns by increasing the number

of either the patterns or the vibration motors. Besides, there were opinions that suggested to modify the pad to be attachable to other parts of the body such as legs and upper arms.

Another experiment was conducted to investigate whether the playing order between vibration and non-vibration versions of a same game influences the human perception about reality improvement. Firstly, half of the twenty two surveyees were asked to play the FPS game with the proposed interface first and the same game without vibration later, while the other half played the same game in the opposite order. Then they were asked to score the reality improvement by the integer scale of 0 to 5. Secondly, they played the same game again but in the opposite order to the previous try and repeated the scoring. The average scores were different depending on the playing order such as 4.36 for the vibration version first and 3.87 for the non-vibration version first. The conjecture about the difference is that humans can perceive changes from the loss of sensory input more significantly than from the addition of sensory input.

4. Conclusions and Future Works

This paper presents the development of the sound specific vibration interface and its evaluation results. The proposed interface complements the pitfalls of the frequency–based vibration interfaces; those interfaces may bring negative user experiences by generating improper vibrations. But the proposed interface which responds to specific sounds is able to improve user experiences effectively. The hardware and software parts of the interface are described. The structure and the implementation of the wrist pad that delivers vibration are discussed. The sound matching algorithm and the pattern editor that extracts the sound vector and help users to design the vibration patterns are explained. The results from evaluating the performance of the interface show that the success ratio of the sound matching is over 90% at the volume of 20 dB and the delay time is around 400 msec. In the survey about user experiences, the users evaluated that the interface was more than four times effective in improving the reality of the game playing than without using the vibration interfaces, and two times than the frequency–based ones. The inconvenience from using the interfaces is one fourth of the frequency–based ones.

The interface has, however, the following limitations that require further works. First of all, the delay increases as the number of target sounds grows in spite of the selective comparison. To meet this challenge, the longest prefix matching technique [17] which is widely used in the routing table matching is under consideration. Secondly, the matching success ratio should be improved more by providing a new method to extract the sound vectors that can differentiate a target sound from others more effectively. In the third place, the FPS game has the cases that the interface responds to the gunshot of other users. The fact that the gunshot of user's own is louder than the others' should be considered by the sound matching algorithm. The fourth one is that the proposed algorithm may entail a probability of false detection because there exists the case that the sum of differences are the same for more than two different target sounds, thus matching with multiple target sounds. In a future work, an additional step should be provided to deal with the multiple matching; for example, calculating the sum of differences again only for the matched sounds, but with different weights depending on frequency bands.

References

[1] A. Chang and C. O'Sullivan, "Audio haptic feedback in mobile phones," Proc. Conference on Human Factors in Computing, 2005.
[2] http://www.sound-scape.com
[3] http://www.xfxforce.com
[4] http://www.logitech.com
[5] H. Lee, Y. You, C. Song, J. Jeong, M. Sung, K. Jun, and S. Lee, "Analysis of tactitle effects on the different body parts by the various vibration patterns," Proc. HCI 2008, Korea, 2008.
[6] L.R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," Proc. IEE, vol.77, no.2, pp.257–286, Feb. 1989.
[7] D. Hoiem, Y. Ke, and R. Sukthankar, "SOLAR: Sound object localization and retrieval in complex audio environments," Proc. IEEE Int. Conf. Acoust., Speech Signal Process., 2005.
[8] A. Dufaux, L. Besacier, M. Ansorge, and F. Pellandini, "Automatic sound detection and recognition for noisy environment," Proc. European Signal Processing Conference 2000, 2000.
[9] C. Couvreur, Environmental Sound Recognition: A Statistical Approach, Ph.D. Thesis, Faculte Polytechnique de, 1997.
[10] M. Vacher, D. Istrate, and J. Serignat, "Sound detection and classification through transient models using wavelet coefficient trees," Proc. European Signal Processing Conference 2004, 2004.
[11] http://www.ti.com
[12] http://www.atmel.com
[13] J. Oh, D. Cho, Y. You, M. Sung, and K. Jun, "Wrist strip and pattern editor for sound specific vibration interface," Proc. HCI 2008, Korea, 2008.
[14] http://www.mikseri.net/music/play.php?id=223080&type=dl
[15] http://www.mikseri.net/music/play.php?id=222840&type=dl
[16] http://www.mikseri.net/music/play.php?id=222841&type=dl
[17] K. Sklower, "A tree-based routing table for Berkeley Unix," Technical report, Univ. California, Berkeley, 1993.

Kyungkoo Jun received his B.S. degree in computer science from Sogang University, Korea in 1996 and his M.S. and Ph.D. degree in computer science from Purdue University, U.S. in 1998 and 2001. Then he joined Samsung Electronics, Korea as a Senior Research Engineer. Since 2004, he has been with the Department of Embedded Systems Engineering, University of Incheon where he is currently an associate professor. His research interests include human computer interfaction, mobile solutions, and wireless networks.