IEICE Electronics Express, Vol.9, No.6, 509–514

An enhanced touch event processing on Android

Y.K. Lim[1,3], C.G. Kim[2a)], and S.D. Kim[1]

[1] *Department of Computer Science, Yonsei University*
134 Shinchon-dong, Seodaemoon-ku, Seoul, Korea

[2] *Department of Computer Science, Namseoul University*
21 MaejuRi, SeongwhanEub, CheonAn, ChoongNam, Korea

[3] *LG Electronics*
327–23 Gasan-dong, Geumcheon-gu, Seoul 153–802, Korea

a) *cgkim@nsu.ac.kr*

Abstract: The evolution of Android smartphone has increased the importance of touchscreen capabilities as user interaction as well as user input method. However, due to the limitation of Android architecture, every touch event may not create its corresponding display update all the times. To overcome this limitation, this paper proposes a novel technique of Touch Event Handling Block (TEHB) with capability of virtual touch event generation and low pass filters. TEHB is implemented on Android 2.2 and its performance simulations show that TEHB can achieve more performance gain than conventional triggering method.

Keywords: Android, touchscreen, touch event, low pass filter
Classification: Wireless communication hardware

References

[1] D. Ehringer, The dalvik virtual machine architecture, March 2010 [Online] http://davidehringer.com
[2] N. FitzRoy-Dale, I. Kuz, and G. Heiser, "Architecture Optimisation with Currawong," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 115–119, Jan. 2011.
[3] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security," *Proc. 20th USENIX Conference on Security (SEC'11)*, Aug. 2011.
[4] [Online] http://developer.android.com/sdk/android-2.2.html.
[5] R. Isermann, *Mechatronic Systems : fundamentals*, 2nd Edition, Springer-Verlag, London, 2005.
[6] R. Schaumann and M.E. Van Valkenburg, *Design of Analog Filters*, 2nd Edition, Oxford University Press, 2001.
[7] C. Pozrikidis, *Numerical Computation in Science and Engineering*, Second Edition, Oxford University Press, 2008.
[8] [Online] http://www.optofidelity.com/en/test-automation/touch-screen-tester

DOI: 10.1587/elex.9.509
Received January 19, 2012
Accepted February 13, 2012
Published March 25, 2012

ELectronics EXpress

IEICE Electronics Express, Vol.9, No.6, 509–514

1 Introduction

One important change being lead by Smartphone evolution is touchscreen based user interaction. Touchscreens allow Smartphone to have more functionality and enhance users' interaction by placing buttons of any shape anywhere on the screen. In Android, a customized embedded Linux system for Smartphone, applications are written in Java and executed within Dalvik Virtual Machine (DVM) [1]. They are reliant on a system framework for services, some of which run in a separate process. They communicate across processes using a custom IPC mechanism. Moreover, the complex layout of Android application programs gives rise to complications such as long view system processing time, in which the next scene cannot be displayed quickly enough [2, 3]. For example, a touch event is processed by multiple layers of Android, in which events are exchanged between many threads and processes, having many factors that slow down the process. Therefore, this is closely related with the reason why a certain touch event is lost and provides poor quality support for touch-based user interfaces. This paper proposes Touch Event Handling Block (TEHB) to compensate touch event losses and deliver touch event to display block seamlessly.

The conventional View System in Android might have problems with processing rushing touch events like dragging because of the following reasons. First, a screen update is being triggered by a touch event so that consecutive touch events could be piled up especially when touch events happen too many times to be handled by a system. Second, each touch event might have different arriving time to the relevant View; therefore, there might be a touch event that does not create display update in case of successive events. Finally, the case that touch event occurrence cycle and display refresh cycle might be different may cause the situation that a touch event cannot create display update.

2 Touch Event Handling Block (TEHB)

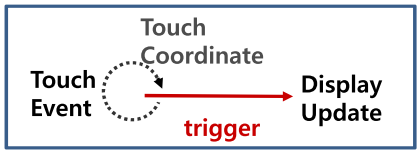
The proposed TEHB, a novel function block shown in Fig. 1, is not synchronized to touch event inputs but creates a new display thread internally by generating new coordinates upon every display update independent from the actual touch coordinates. It compensates touch event delivery losses by generating virtual touch events and also calibrates irregularity of touch event delivery time using combination of low pass filters.

The algorithm to generate new coordinates for the proposed method consists of four steps: Noise Filtering, Up-sampling, Low-pass Filtering, and Delay Compensation. For the performance evaluation, this method is implemented on Listview, the mostly used application framework component in Android.

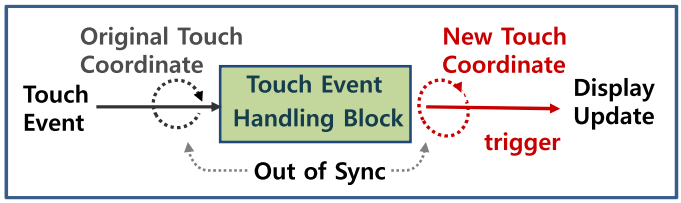
2.1 Noise filtering

In touchscreen, touch noises can be defined as touch coordinates value regardless of actual touch events made by users. In order to mathematically

DOI: 10.1587/elex.9.509
Received January 19, 2012
Accepted February 13, 2012
Published March 25, 2012



(a) Conventional dragging scheme



(b) Proposed dragging scheme

Fig. 1. Conventional vs. proposed method

model the noise reduction, Spring-damper system [5] with 1 degree of freedom is created, shown in Fig. 2 (a). Mass-spring-damper model [5] is widely used as a basic vibration modeling. Spring-damper model is a slight variation from Mass-spring-damper model in which the mass value is set to zero. In a physical vibration model, Spring-damper model may be inapplicable since mass can never be zero. However, it is applicable for its use in eliminating the fast vibration of Listview.

When touch coordinates are defined as $y(t_n) = y[n]$ and the equation of motion is assigned with discrete digits via limited differentiation, Noise Filtering equation can be derived as follows:

$$y[n] = \frac{2k}{2k+3c}x[n] + \frac{4c}{2k+3c}y[n-1] - \frac{c}{2k+3c}y[n-2] \tag{1}$$

Since constant k and c are located in the numerator and denominator, respectively, k is set as 1 and c is adjusted to find the most adequate solution for the device.

2.2 Up-sampling and low-pass filtering

Up-sampling is realized by padding the touch coordinates used recently. Suppose there are 30 touch coordinates per second and the display refresh rate is 60 samples per second, recent touch coordinates begin to be padded. If touch coordinates, at this point, are upsampled just as they are, the screen dragging does not become seamless. For seamless connection, upsampled touch coordinate stream should be passed through low-pass filter function. When input signals are applied to touch coordinates at 60 Hz sampling rate, they can be expressed as $x[n]$ because touch coordinates are defined as $x[0]$, $x[1], \ldots, x[n-1]$, $x[n]$, and so on. At this time, they are defined every 16.7 ms and the time instances t_n are equal to the time period T ($t_n = nT$). Low-pass filter can be made from FIR and IIR; however, for better bandwidth limitation performance in the identical degree, biquad IIR filter [6] is used to construct low-pass filter. The 2^{nd} order biquad filter [6] is shown in Fig. 2 (b). Here, the block Z^{-1} signifies the delay factor, $\oplus$ means addition,

DOI: 10.1587/elex.9.509
Received January 19, 2012
Accepted February 13, 2012
Published March 25, 2012

and other variables written on the top of the signal route arrow indicate multiplication. The specifications of the filter used in this paper are as follows: Sampling frequency is 60 Hz, cut-off frequency and damping constants are decided as 10 Hz and $\sqrt{2}/2$ (approximately 0.707), respectively, through our experiments. Each variable, $\{a_0, a_1, a_2, b_0, b_1, b_2\}$, is matched to the results, $\{1, -1.60826447, 0.73724435, 0.03224497, 0.06448994, 0.03224497\}$, respectively.

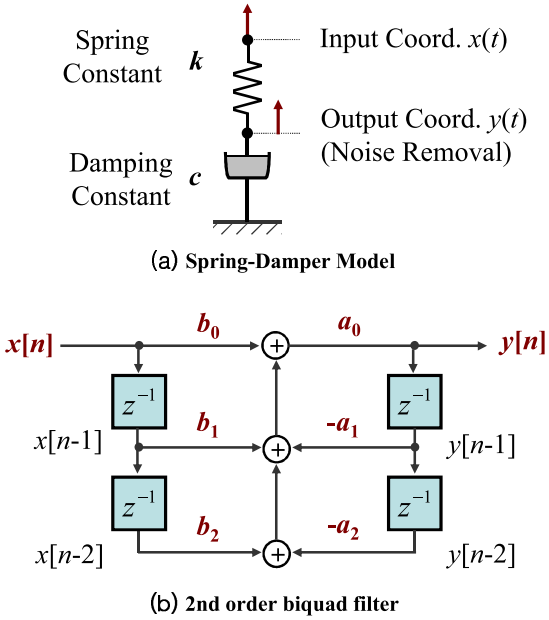


Fig. 2. Filtering methods

2.3 Delay compensation

Although missing frame correction and consistent movement can be achieved through Low-pass Filtering, movement delay and bounce problems still remain. Therefore, we have to reduce the delay and bounce by minimizing the input and output difference via Proportional-Integral control. The Proportional-Integral control algorithm can be expressed as Eq. (2).

$$MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau \tag{2}$$

where $MV(t)$ is the control value; Kp is ratio coefficient; $e(t)$ is error. Proportional-Integral control algorithm for cellular devices is altered as follows:

$\overline{x} = \{x[0], x[1], \ldots, x[n]\}$: up-sampled touch coordinate,

$\overline{y} = \{y[0], y[1], \ldots, y[n]\}$: low-pass filtered touch coordinate,

$\overline{z} = \{z[0], z[1], \ldots, z[n]\}$: touch coordinate with delay and bounce correction,

$e(t) \approx e[n] = y[n] - x[n]$: error definition, and

DOI: 10.1587/elex.9.509
Received January 19, 2012
Accepted February 13, 2012
Published March 25, 2012

$$\int_0^t e(\tau)d\tau \approx \frac{1}{3}(e[n] + 4e[n-1] + e[n-2]) \quad : \text{Simpson's } 1/3^{\text{rd}} \text{ rule [7].}$$

Suitable Kp and Ki values also need to be set through a test because each device has different touch performance. From the experiment, Kp of 0.1 and Ki of 0.2 were selected for having the least amount of delay.

3 Experimental results

For the experiment, we implemented the proposed system on a real phone with Android 2.2 on ARM Cortex A8 with 1 GHz clock and constructed Touch UI measurement device with high-speed camera for more accurate and quantitative measurement of touch performance at UI. The device, shown in Fig. 3, is equipped with measuring JIG and sensor to emulate users' touch pattern and analyze uniform touch performance with a graphical analysis software tool on PC. This device can provide more accurate and useful measuring items than commercial one [8]. We evaluated the performance by measuring TRT (Touch Response Time), FPS (Frame per Second), and FMU (Frame Movement Uniformity) using this device.

Fig. 3. Touch UI performance measurement device

3.1 TRT and FPS

TRT in Touch UI can be defined as the time required for responding to dragging. Dragging can be either moving an object such as icon from point A to B or a screen display from List/Idle point A to B. We take the average of ten TRT measurements without/with TEHB and the results are 255 ms and 258 ms, respectively. The proposed method does not have any delay even tough having additional processing block and shows similar measured values within the margin of error.

FPS is the frequency of frame updates in screen display. LCD is able to display 60 pages per second; therefore, screen display update can be smoother as FPS becomes closer to this frequency. However, when there are heavy loads of work for CPU to process such as touch dragging, the actual FPS may decrease. FPS measurement results without/with TEHB are 50.46 and 59.26 frame/sec. FPS with TEHB is closer to LCD display update frequency.

DOI: 10.1587/elex.9.509
Received January 19, 2012
Accepted February 13, 2012
Published March 25, 2012

3.2 Uniformity evaluation

Also, we examine the distance between frame movements on dragging. If each frame moves at regular distance interval, then we can say that the frame movement is uniform and screens may be updated regularly. We define such measured values as FMU and expressed as the below equation.

$$FMU = \sqrt{\frac{\sum \left(V_i^1 - \mu^1\right)^2}{N}}^{-1} \quad \text{where} \quad V_i^1 = \frac{NV_i}{\sum Vi}$$

Here, N is the total number of frames required for dragging; Vn is the distance between two consecutive frames when a start frame is called $n = 0$ and the last frame is called $n = N - 1$. FMU becomes the reverse value of a standard deviation after normalizing the average speed of frame movement as 1 while dragging. The large value means that frame movement uniformity is good. Results without/with TEHB are 2.63 and 5.01, respectively. The proposed method gains larger FMU than the conventional one. As for iPhone, it is 4.35. This means that when TEHB is applied, the screen will be updated within regular interval and this makes users feel smooth even with high frame rates on Touch UI.

4 Conclusion

This paper proposes a novel touch event processing mechanism to overcome the defect of processing touch event caused by the architectural limit of Android. For this purpose, the proposed method works on the basis of display thread with enhanced filtering feature. The new method is implemented on the Android 2.2 for performance evaluation using our test environment. Experimental results show that the proposed method achieves more performance gain than conventional triggering methods.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (KRF 2011-0027264).

DOI: 10.1587/elex.9.509
Received January 19, 2012
Accepted February 13, 2012
Published March 25, 2012