Smoothed Graphic User Interaction on Smartphones With Motion Prediction

Ying-Dar Lin, *Fellow, IEEE*, Edward T.-H. Chu, *Member, IEEE*, Evan Chang, and Yuan-Cheng Lai

***Abstract*—The smoothness of human–smartphone interaction directly influences users experience and affects their purchase decisions. A commonly used method to improve user interaction of smartphones is to optimize the CPU scheduler. However, optimizing the CPU scheduler requires a modification of operating system. In addition, the improvement of the smoothness of human–smartphone interaction may be limited because the display subsystem is not optimized. Therefore, in this paper, we design a motion prediction queuing system, named MPQS, to improve the smoothness of human–smartphone interaction. For this, we use the information of vector, speed, movement, provided by the queuing mechanism of Android, to predict the movement of user-smartphone interaction. Based on the prediction, we then utilize available execution time between frames to perform image processing. We conducted a set of experiments on beagleboard-xM to evaluate the performance of MPQS. Our experiment results show that the proposed method can reduce the number of jank by up to 21.75%.**

***Index Terms*—Frame interval, graphic user interaction, motion prediction, smartphones, smoothness.**

I. Introduction

SMARTPHONES have played an important role in our daily life because they provide many valuable applications, such as shopping, news browsing, map navigation, e-mail, and video playing. The smoothness of human–smartphone interaction directly influences users experience and affects their purchase decisions. According to Nielsen [1] and Miller [2] investigation, 0.1 s is the minimum delay that human can feel. A delay is noticeable if the interval of the delay is longer than 1 s. Further, if the delay is longer than 10 s, users will switch to other tasks. Similar results can be found in [3], in which 0.2 s is the minimum threshold for human to perceive a delay of an application. As a result, it becomes an important issue for smartphone manufactures to improve the smoothness of human–smartphone interaction.

A simple and intuitive method to improve user interaction of smartphones is to use graphics acceleration [4]. Although graphics acceleration can boost performance levels, it increases hardware cost. Central processing unit (CPU) schedulers are another targets for optimization because they also affect the smoothness of human–smartphone interaction. Wang *et al.* [5] and Wong *et al.* [6] systematically analyzed and measured the performance of three schedulers: 1) $O(1)$; 2) rotating staircase deadline scheduler (RSDL); and 3) completely fair scheduler (CFS). The $O(1)$ scheduler provided constant time scheduling services so as to minimize the amount of jitter incurred by the invocation of the scheduler. However, the $O(1)$ algorithm utilized a complicated method to classify tasks. The calculation of determining the interactivity of tasks may also incorrect and results in a performance degradation. RSDL aimed to deliver a better fairness among tasks. However, it may induce a long response time of starved tasks. The design goal of CFS is to provide a fair CPU resource allocation among executing tasks. However, it may sacrifice performance of interactivity tasks. Although the above scheduling policies may improve the performance of interactive tasks, it requires a modification of operating system (OS). In addition, the improvement of the smoothness of human–smartphone interaction may be limited because the display subsystem is not optimized. Therefore, the research problem of this paper is to improve the smoothness of human–smartphone interaction.

Average frame rate is the most commonly used index to measure the smoothness of a video. The higher the frame rate is, the better the quality of played back video becomes. However, two videos with the same average frame rate can provide very different user experiences, because one may abruptly drop a large number of frames while another may maintain a uniform frame rate. Li [7] claimed that the smoothness of user interactions can be measured by five key indexes: 1) response delay; 2) maximal frame time; 3) frame time variance; 4) frame per second; and 5) frame drop rate. In addition, Wen [8] provided six different indexes to evaluate the smoothness of smartphones. They are the mean of frame intervals (MFIs), variance of frame intervals (VFIs), maximal frame interval (MaxFI), frame no response, times of MaxFI, and number of frame intervals (NFIs). According to the analysis results of the above researches, VFI is the most representative index to evaluate the smoothness of human–smartphone interaction. Their research results also showed that if intervals

Manuscript received October 3, 2015; revised December 27, 2016; accepted March 5, 2017. This work was supported in part by the Ministry of Science and Technology, Taiwan, and in part by the Institute for Information Industry. This paper was recommended by Associate Editor S. Nahavandi.

Y.-D. Lin and E. Chang are with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: ydlin@cs.nctu.edu.tw; evanchang.pcs00g@g2.nctu.edu.tw).

E. T.-H. Chu is with the Department of Computer Science and Information Engineering, National Yunlin University of Science and Technology, Douliu 64002, Taiwan (e-mail: edwardchu@yuntech.edu.tw).

Y.-C. Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan (e-mail: laiyc@cs.ntust.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2685243

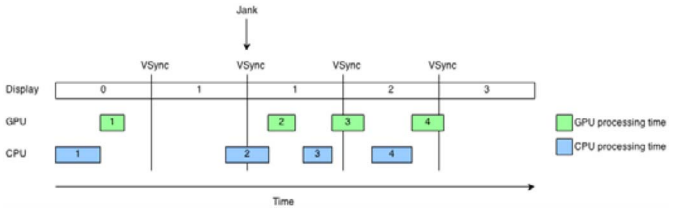

Fig. 1. Illustrated example of jank.

between frames vary significantly, the delay becomes noticeable. In this paper, we use jank as performance index, which is defined as the number of nonupdated screens in a given time slot. The more the number of janks, the poorer the performance, and the higher VFI. As Fig. 1 shows, the display (i.e., the screen of a smartphone) is updated periodically according to the vertical synchronization (VSnyc) signal. Displaying a frame requires the computation of CPU and graphics processing unit (GPU). The display subsystem first uses CPU to calculate the regions that should be refreshed and then marks the regions as dirty regions. Next, the display subsystem adopts GPU to draw the display by updating the video buffer. The computation workload of CPU and GPU should be finished before the VSnyc signal is triggered. Otherwise, the screen will not be updated in time. As shown in Fig. 1, a jank happens at the time when the second VSnyc is triggered and the computation of CPU and GPU is not completed.

In this paper, we aim to reduce the number of janks so as to improve smoothness of human–smartphone interaction. For this aim, we use the information of vector, speed, and movement, provided by the queuing mechanism of Android, to predict the movement of user-smartphone interaction. Based on the prediction, we utilize the slack time between tasks to preprocess portions of raw data in the video buffer. Two methods are proposed to improve smoothness. First, we condense the computation load of both CPU and GPU so as to minimize the number of janks. All frames need to be processed before they can be rendered. In an event-driven application, the rendered frame will be displayed in next frame slot. The major advantage of the proposed method is that we can obtain the events in advance, and preprocess those frames to reduce computation of both CPU and GPU. Second, we use the same prediction mechanism to get future events to reduce computation. The main idea is to reduce the number of output frames. We ensure that the minimum frames per second is larger or equal to 30 so that the reduction of frames will not affect user experience.

To our best knowledge, this is the first work that utilizes motion prediction and slack time to reduce the number of janks and the computation load of CPU and GPU so as to improve the smoothness of human–smartphone interaction. The major difference between our method and existing works is that we focus on how to reduce the computation load rather than how to properly manage resource content among concurrent running processes. The major contributions of this paper include the following.

1) Adopting motion prediction to get future events to reduce computation of both CPU and GPU.
2) Utilizing slack time to preprocess following frames to reduce the number of janks.
3) Providing a thorough review on the Android display subsystem (ADSS).
4) Conducting a series of experiments in a real environment to critically evaluate the performance and feasibility of the proposed method.

In short, we provide a new system that reveal new opportunities to improve the smoothness of human–smartphone interaction.

The remainder of this paper is structured as follows. Section II presents the background. Section III describes the assumption and problem statements, and Section IV describes the architecture of motion prediction queuing system (MPQS). Section V describes implementation issues. The experimental results and case study are presented in Section VI. Finally, we conclude this paper and discuss the future works in Section VII.

II. Background and Related Work

In this section, we first provide background information of the ADSS. We then discuss related work that aimed to improve interactivity.

A. Android Display Subsystem Architecture

Fig. 2 shows the architecture of ADSS. The first layer, also the top layer, is application layer. Based on the data types, we classify applications into three different categories. Applications belong to the first category are general applications which display data in red, green, and blue (RGB) color model. Most applications belong to the first category. Applications belong to the second category display information in luminance, chrominance, and chroma (YUV) color space. Example applications include camera preview and video playback. These applications sent YUV data directly to the kernel and display data through overlay interface. The third category is similar to the first category, but the displayed data need to be processed by Open Graphics Library (OpenGL), Open Vector Graphics, Scalable Vector Graphics, and Skia. Example applications include games, navigation map, and flash software. The second layer is the framework layer, and the core module is named SurfaceFlinger which provides services to all processes who needs rendering. Because the overlay interface is packaged in SurfaceFlinger, the processes that need to use the overlay interface have to communicate with SurfaceFlinger. In addition, SurfaceFlinger will use OpenGL to compose different surfaces. Graphics hardware abstract layer (HAL), is invoked by 2-D/3-D applications and SurfaceFlinger to perform graphics processing. The third layer is HAL, which provides control channels and data channels to upper layers, and interacts with image processing unit (IPU) driver. The IPU driver handles image data processing such as transforming YUV to RGB or performing image rotation. Gralloc is divided into two parts. The first part is used by persistent memory (PMEM) to communicate upper layer. The second part is used to refresh frame buffer, which stores the

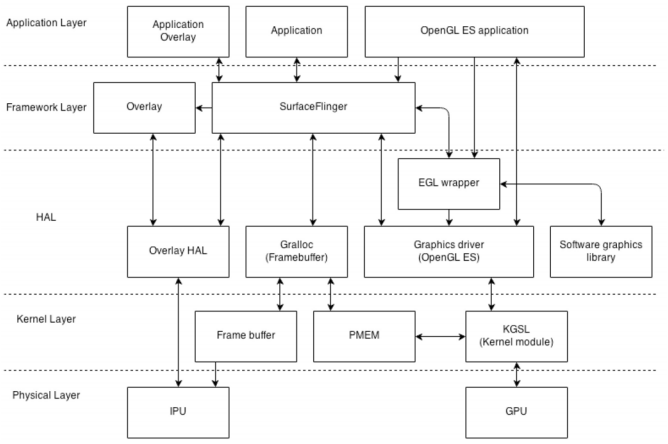

Fig. 2. Architecture of ADSS.

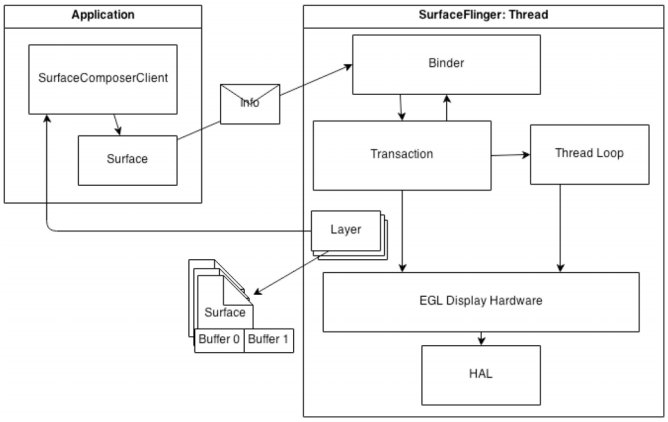

Fig. 3. Relationship between application surface and SurfaceFlinger.

displayed frames. Upper layer sends frame data to OS kernel through the overlay or the frame buffer. If the hardware OpenGL for embedded systems (OpenGL ES) does not support the corresponding functions, image data will be processed by software. The fourth layer is the kernel device drivers, which includes frame buffer driver and IPU Overlay driver. PMEM and Kernel GNU Scientific Library are the encapsulation of PMEM driver and GPU. The last layer is physical layer which is composed by different hardware devices.

SurfaceFlinger is the core services of ADSS. The main function of SurfaceFlinger is surface composer. It constructs 2-D or 3-D surface for all kinds of applications, and sends the final surface data to the display memory. Each application has several z-order layers, and each surface in a layer is a canvas that allows applications to paint on it. Surfaces in application are obtained from SurfaceComposerClient which is an interface to communicate with SurfaceFlinger. The communication channel between surface and SurfaceFlinger is binder. Binder is an interprocess communication mechanism provided in Android. When application modified content in surface then the transaction is passed to SurfaceFlinger and informs the loop thread to update frame. SurfaceFlinger then composes each surface by position, size, and z-order. The relationship between application surface and SurfaceFlinger is shown in Fig. 3.

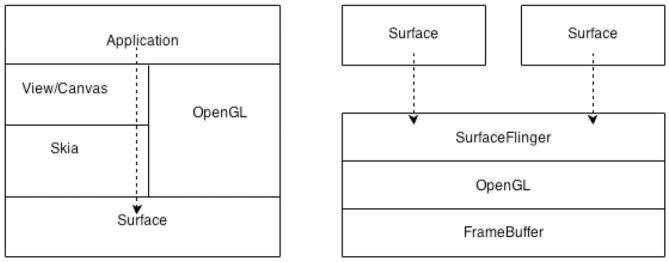

Fig. 4. Client and server architecture of ADSS.

ADSS uses client and server architecture, shown in Fig. 4, for applications and surface to communicate with each other. The right-hand side of Fig. 4 is server and the left-hand side of Fig. 4 is client. The responsibility of the server is to create surfaces. On the other hand, a client provides an interface for upper layer to control surface, and sends a message to the server to complete the processing. The architecture is shown in Fig. 4.

B. Related Work

Several research efforts have been made to improve the interactivity of an Android system. We divide them into two categories: 1) kernel level optimization and 2) framework level optimization. Table I makes a comparison among these works. The graphics acceleration is a hardware mechanism which can improve graphical computation. It uses specially instruction pipeline to reduce computation latency. However, the specially designed hardware is not supported by all smartphones. VT-CFS [9] and MLFQ [10] are OS-level solution to improve the user interaction. Both methods required a modification of OS. Nguyen *et al.* [21] designed a system prototype called SmartIO that reduced the application delay by prioritizing read operations over write operations, and grouping them based on assigned priorities. Android develop team adopted control group, SurfaceFlinger and project butter to improve the smoonthness. Unlike existing works, we developed a prediction mechanism to further improve the smoonthness of smartphones.

Linux kernel communities have made significant efforts to improve the interactivity of Linux by developing sophisticated task scheduling algorithms. For example, in Linux 2.6, an $O(1)$ scheduler was proposed to replace original $O(n)$ scheduler in order to minimize the amount of jitter, OS service overhead, and influence to programmers. This $O(1)$ algorithm intended to address the performance issue of interactivity by using a number of heuristics to determine if tasks were I/O-bound or CPU-bound. Once it characterized tasks, it promoted the priorities of I/O-bound tasks to improve user experiences. Generally speaking, the $O(1)$ scheduler can deliver better interactivity performance than CFS [5], [6]. However, the heuristics may misclassify characteristics of tasks and lead to starvation and unfair allocation of CPU resource.

Some research efforts have focused on resource management in OS, middleware level and big-data infrastructures. Wang *et al.* [22] proposed an optimal priority-free real-time scheduling to process tasks running in a real-time

TABLE I
COMPARISONS OF DIFFERENT METHODOLOGIES

Topics	H/W	Scheduler	CPU Allocation Allocation	Dirty Region Computation	Display Mechanism	Predict	Frame Computation Reduction	Method
Graphics Acceleration [4]	√	×	×	×	×	×	×	Instruction pipeline
Improve interactivity via VT-CFS [9]	×	√	×	×	×	×	×	VT-CFS
Fairness and interactive performance [10]	×	√	×	×	×	×	×	MLFQ Scheduler
Android	×	×	√	√	×	×	×	Control Group SurfaceFlinger
Project Butter	×	×	×	×	√	×	×	VSync 60Hz Triple buffer Touch response
MPQS	×	×	√	√	×	√	√	Motion prediction

system. Li *et al.* [23] designed scheduling algorithms that ensure timing correctness and optimize energy consumption on a processor with variable speeds. Valls and Val [24] gave a comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems. Basanta-Val and García-Valls [25] explored the integration of an efficient programming language and high-level real-time programming abstractions so as to improve the responsiveness of the system. A library was proposed for programmers to develop real-time and embedded applications in *C* language. Basanta-Val and García-Valls [26] also developed an integrated technique to manage real-time remote invocations in Java's Remote Method Invocation. They further considered time-critical big-data applications. An architecture for time-critical big-data system was designed to run time critical analytics and and applications [27]. The major difference between our method and these quality of service techniques is that we focus on how to reduce the computation load rather than how to properly manage resource content among concurrent running processes. MPQS and the quality of service techniques are complementary to each other and not mutually exclusive.

C. Improved Android Display Subsystem

Android also uses various techniques to enhance the performance of interactivity. For example, in order to provide better user experience, Android uses control group mechanism supported by Linux to maintain priority of background and foreground tasks [11]. Background groups are restricted to use no more than 10% of processor utilities so that they will have limited impact on foreground applications. In addition, Android performs rendering work separately in order to reduce the time used to draw a screen [12]. Inside Android, a screen is composed of separate pieces of region called surfaces. Android draws each surface independently so that only the updated surface is rendered. Each individual surface is then composed with others by a dedicated system server

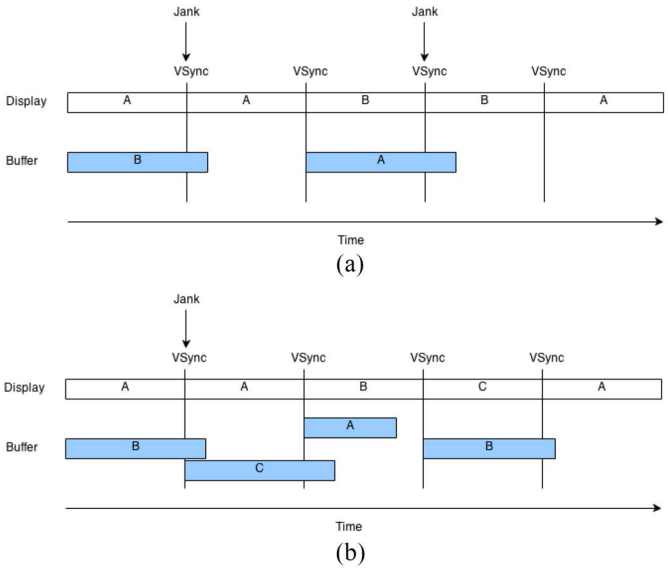

Fig. 5. Display behavior (a) without Vsync and (b) with Vsync.

called SurfaceFlinger. By doing so, total rendering time is reduced and users can perceive improved interactivity. Further, a famous Android project, called Butter, was proposed to improve the performance of rendering [13]. The Butter reconstructs the ADSS by introducing three effective functions: 1) VSync; 2) triple buffer; and 3) choreographer. As Fig. 5, the upper part is the case without VSync. Jank happens at the time when the second VSnyc is triggered and the second frame has not been rendered. This is because there is no mechanism to inform GPU to perform rendering for the second frame after the first frame is processed. In order to solve this problem, Buffer use VSync to interrupt CPU and GPU to handle upcoming frames. The interval between VSyncs is 16 ms because the display refresh rate is 60 frames/s.

In order to improve the performance in processing frames, Butter adopts a multiple buffer mechanism. In Fig. 6(a), the

TABLE II
NOTATION TABLE

Categories	Notations	Description
Given	i	The level of processing time
	$T_{x,i}$	The processing time of frame function in the i^{th} level
	w_i	The processing time of frame function in the i^{th} level
	T_F	Frame time
	l_v	The *VSync* interval
	$T_{idle,j}$	The idle time after j^{th} frame is processed
	FPT_j	The processing time of the j^{th} frame
Input	I_s	Input Speed
	I_r	Input region
	I_d	Input direction
	T_c	Wall clock time
	MT	Movement time
	ID	The measurement of the theoretical difficulty of performing an aiming movement
Design	p	Prediction function
Output	n_{ff}	Frame function numbers
	n_F	The number of frames

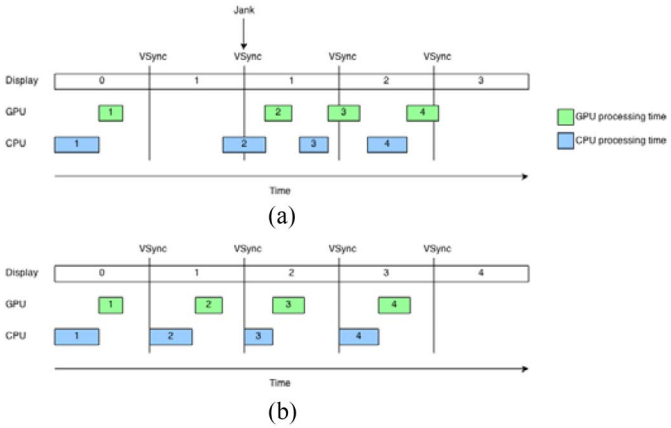

Fig. 6. (a) Two buffers. (b) Triple buffers.

frame in buffer *A* is being rendered while the frame in buffer *B* is being processed so that both CPU and GPU are waiting for each other at the time instant when the second VSync is trigger. Butter solves this problem by introducing the third buffer *C*. As Fig. 6(b) shows, the third buffer *C* can be used to process the third frame so that CPU and GPU will not turn into idle mode in the second time slot. As a result, there is no jank in the latter case. However, the jank issue still exists if processing time is longer than 16 ms. Thus, we need a new mechanism to further reduce computation or increase availability processing time.

In our work, in order to further reduce the number of janks, we use the information of vector, speed, and movement, provided by the queuing mechanism of Android, to predict the movement of user-smartphone interaction. Based on the prediction, we utilize the slack time between tasks to preprocess portions of raw data in the video buffer. The preprocessing helps to reduce the computation load of both CPU and GPU so as to minimize the number of janks.

III. PROBLEM STATEMENT

This section first explains the terminology and assumption of this paper. It then formally describes the problem statement.

A. Terminology and Assumptions

In this paper, the frame function is defined as a graphic instruction, which can be invoked by CPU or GPU. In the implementation of OpenGL ES [14], a graphic instruction is executed by CPU if GPU hardware does not support the instruction. Table II lists the definition of notations used in this paper. Notation i stands for the level of processing time. For example, if one frame function takes 100–200 cycles to complete the operation while another frame function takes 200–300 cycles to completes the operation, then they will be in different level. We define $T_{x,i}$ as the processing time of frame function which includes CPU or GPU processing time in the ith level. According to the different processing time level, the weighted value w_i is defined as the ratio of frame functions to one frame. For example, if the current frame contains more strait line functions than circle functions, we will give strait line functions a higher weight. In addition, l_v is 1/refresh rate, which depends on system design. Our prediction function p takes the speed I_s, region I_r, and direction I_d of user gesture as input. These three parameters are obtained from Android system.

B. Problem Description

Fig. 7 demonstrates the processing time frame processing time (FPT) and idle time $T_{\text{idle},j}$ in a sequence of time slots. The time line from left to right is mapped to wall clock time T_c in real world. When display system handles with a sequence of frames, it adopts VSync signal to prevent the system doing anything visible to the display memory buffer until the current refresh cycle finishes. The time interval between two VSync is composed of FPT and $T_{\text{idle},j}$. According to that, we can summarize the processing into equation form as follows.

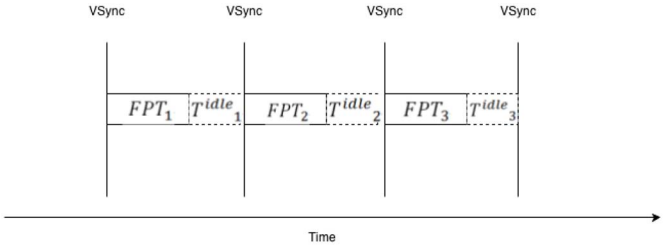

Fig. 7. FPT.

Equation (1) is used to calculate the computation time in each frame. Because a frame is constructed by several frame functions, we level them from low to high according to the computation time. The w_i is related to the ratio of leveled frame functions used in this frame. In this paper, we regard w_i as the ratio of frame functions

$$\mathrm{FPT} = \sum_{i=1}^{k} \left(T_{x,i} w_i\right). \tag{1}$$

Equation (2) is used to calculate the total processing time of a sequence of frames. The idle time $T^{(\mathrm{idle})}$ is the interval where CPU and GPU are idle. We calculate the total processing time T_c by

$$T_c = \sum_{j=1}^{n_F} \left(\mathrm{FPT}_j + T_{\mathrm{idle},j}\right). \tag{2}$$

Assume that the fresh rate of the system is 60 frames/s and l_v is equal to 16 ms. A jank happens if $T_F > l_v$. In other words, a jank happens if FPT is larger than the interval between two VSync signals. We use (3) to represent the condition that a jank will happen; that is

$$\frac{T_c}{n_F} = T_F \leq l_v. \tag{3}$$

The formal problem description is given as follows. Given I_s, I_r, and I_d as inputs, we aim to design a set of prediction function p so as to minimize FPT_j and n_F, and ensure that $T_F \leq l_v$.

IV. Motion Prediction Queuing System

This section first gives an overview of MPQS. It then describes motion prediction, linear fitting, data training, prediction module, and queuing system, respectively. Finally, an example is used to explain how MPQS works.

A. Overview of MPQS

MPQS is implemented in application layer. As Fig. 8(a) shows, the architectures of MPQS are composed of several modules, and each of them performs different functionalities. The prediction module is used to predict user gestures based on the historical data stored in the training database. The prediction results are sent to the reallocation module to process frames. The processed frames are finally sent to the frame queue and displayed by the ADSS. The flowchart of MPQS is shown in Fig. 8(b).

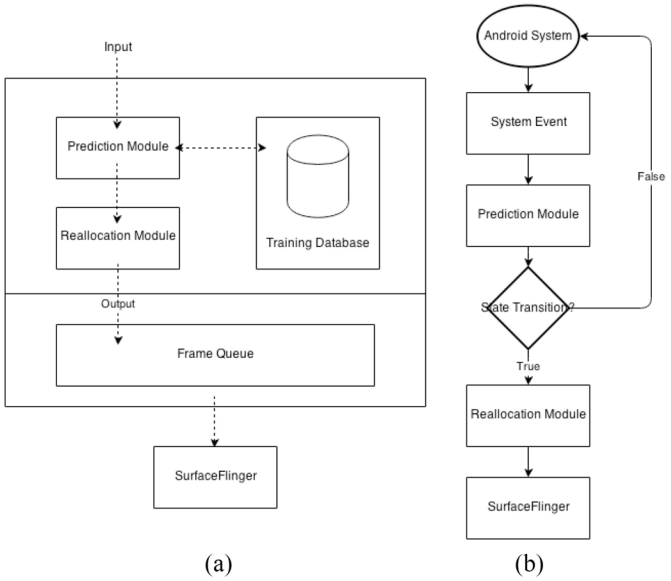

Fig. 8. MPQS (a) architecture overview and (b) algorithm.

We extract system events from the even hander inside Android kernel. In event handler, we can obtain four important information of a user gesture. They are I_s, I_r, I_d, and T_c, which are sent to the prediction module for further analysis. Depending on the system state transition, MPQS decides whether or not to switch the reallocation module to compute frame and send the results to the frame queue. Finally, ADSS gets the frames from the frame queue and renders them on the output device.

B. Motion Prediction

Motion prediction is used to predict the final position of a user gesture. This section first introduces Fitts's law and then discusses linear fitting and prediction module, respectively.

1) Fitts's Law: Fitts's law [15] is a widely used human movement model. Fitts's law has been formulated in several different ways. A common form, proposed by MacKenzie *et al.* [16], [18] and Douglas *et al.* [17] for 1-D movement is

$$\mathrm{MT} = a + b \times \log_2\left(1 + \frac{D}{W}\right) \tag{4}$$

where MT is the move time of the movement, a stands for the start or stop time of the device, and b stands for the speed of device. Both a and b are constants, which can be determined experimentally by fitting a straight line. The D is the distance between the starting point and the center of the target position. W is the width of the target measured along the axis of motion. The binary logarithm in Fitts's law is called the index of difficulty (ID) for the target, and has units of bits. ID is a measurement of the theoretical difficulty of performing an aiming movement. For easy of computation, we rewrite the Fitts's law as

$$\mathrm{MT} = a + b \times \mathrm{ID} \tag{5}$$

$$\mathrm{ID} = \log_2\left(1 + \frac{D}{W}\right). \tag{6}$$

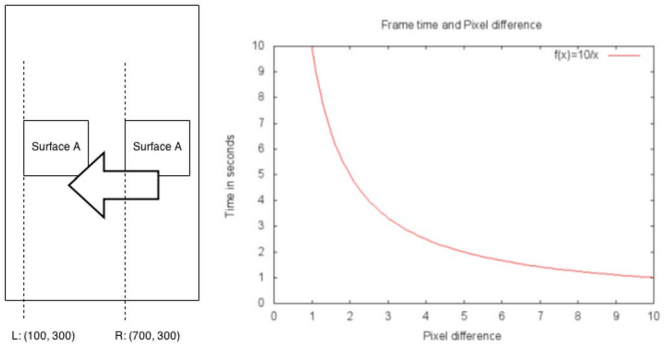

Fig. 9. Concept of reallocation.

Equations (5) and (6) show that a movement will take longer to complete if the area of the target region is small or the location of the target region is far away.

2) Linear Fitting and Data Training: Before performing the prediction on a smartphone, we have to construct a fitting linear equation from training data. This is because the constant coefficients a and b are hardware dependent.

3) Prediction Module: The prediction module is used to predict the final position of a user triggered movement. The Fitts's law and our prediction module differ in the target of prediction. The Fitts's law is used to predict the movement time while our prediction module is used to predict the final position of a movement. Therefore, based on (5) and (6), we obtain the MT by solving

$$D = \left(2^{\frac{\mathrm{MT}-a}{b}} - 1\right) \times W. \tag{7}$$

C. Queuing System

The reallocation module is used to maintain the frame queue, which stores a sequence of frame according to the prediction result. Reallocation module is designed to balance frame time variance by reducing the number of frames or rearranging the frames.

We use an example, shown in Fig. 9, to explain the main concept of the reallocation module. Assume that a user moves a surface A from point R (700, 300) to point L (100, 300) and the movement takes 10 s to complete. It implies that the system needs to render 600 frames if the frequency of VSync is 60 Hz. When user's finger moves one pixel, it needs to update one frame. The reallocation module will reduce the frame number so as to average frame time variance. For example, if the distance between two continuous frames is enlarged to 30 dots per frame, only 20 frames need to be rendered and the computation load is reduced to 1/3 its original execution time.

The algorithm of the prediction module is shown in Fig. 10(a). First, we use the information MT and ID stored in training database to estimate regression model and obtain the parameters a and b. Second, we retrieve I_s, I_r, and I_d to measure D by (7). Finally, we send the output information MT, l_v, I_s, I_r, and I_d to the reallocation module. In addition, in this paper, we build a tracer to ensure the tracking accuracy of prediction results. For this, the tracer will record every single interaction operations from user, and compare it with actual

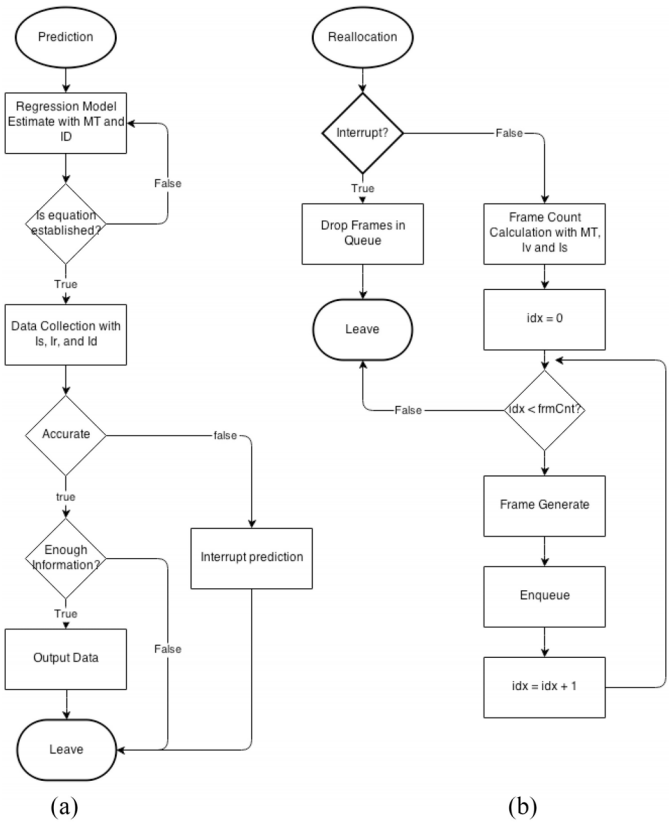

Fig. 10. (a) Prediction module. (b) Reallocation module.

results. When the predicted result does not match the actual result, the tracer will issue an interrupt signal to inform reallocation module to drop those completed frames, and notice the prediction module to predict results by new interaction operations. The algorithm of the reallocation module is shown in Fig. 10(b). Reallocation module will drop all frames in the frame queue if an interrupt is received from the prediction module. Otherwise, it will generate frames according to the prediction results. In this paper, we use D, MT, and I_s to calculate frmCnt. Variable idx is used to count the index of processing frame. The maximum value of idx is frmCnt. For example, if the input speed I_s is 10 pixels per millisecond, the movement time MT, based on the training database, is 90 ms. The movement distance D will be determined. Finally, we use D and l_v to determine frmCnt.

D. Illustrated Example

We use an example to explain how we get the parameters a and b of (7) from the training data module. As Table III shows, we assume that the target width is 50 pixels. In the first round, the movement distance is 330 pixels and the movement time is 458 ms. In the second round, the movement distance is 708 pixels and the movement time is 871 ms. Therefore, the IDs are $\log_2((330/50) + 1) = 2.9259$ and $\log_2((708/50) + 1) = 3.9221$. Based on the IDs, we construct a linear fitting equation with constant

$$a = \left(\frac{458 + 871}{2} - 414.575 \times \frac{2.9259 + 3.9221}{2}\right) = -755.005$$

TABLE III
EXAMPLE INPUT INFORMATION

Round	D (pixels)	MT (milliseconds)	W (pixels)
1^{st}	330	458	50
2^{nd}	708	871	

and

$$b = \left(\frac{\left((2.9259 \times 458 + 3.9221 \times 871) - (2.9259 + 3.9221) \times \frac{458+871}{2} \right)}{\left((2.9259^2 + 3.9221^2) - (2.9259 + 3.9221) \times \frac{2.9259+3.9221}{2} \right)} \right) = 414.575.$$

Based on (5), we have $\mathrm{MT} = -755.006 + 414.575 \times \mathrm{ID}$. Finally, we use the obtained prediction function to process input events. For example, if the movement time is 900 ms, the prediction distance will be

$$D = \left(2^{\frac{(900-(-755.006))}{414.575}} - 1 \right) \times 50 = 745.5761$$

according to the prediction function.

MQRS is also feasible in supporting small computing infrastructure because MPQS requires a very little computation. The core equations of MPQS are (4)–(6), in which (4) and (5) can be executed offline before MPQS starts to run. On the other hand, (6) is a very simple online exponentiation operation and can be done in few instructions by a CPU with floating-point unit. For CPU that does not support floating-point computation, (6) can be done by an approximation function efficiently. MPQS is independent on OS, library, middleware, and hardware. MPQS can be integrated with smartphones or handheld devices as long as the system parameters and input variables listed on Table II are available. Based on the information, MPQS utilizes motion prediction and slack time to reduce the number of janks and the computation load of CPU and GPU so as to improve the smoothness of human–smartphone interaction.

V. IMPLEMENTATION

This section introduces the implementation of the MPQS system. It first describes the architecture and data flow of Android display mechanism. It then introduces the detailed implementation of the MPQS.

A. Android Display Mechanism Architecture

In order to process upcoming frames, we perform a series of operations, including adopting the ADSS to perform frame computation and composition and requiring the WindowManager to intercept and pass input event. WindowManager is a system service used to receive signals from monitor. As shown in Fig. 11, the lowest layer is physical layer that contains input devices, which generates gestures events. The responsibility of the device drivers is to parse gesture events and transform the events into standard Linux I/O event formats. The EventHub then reads these events from Linux kernel and passes the event information to InputReader. The responsibility of InputReader is to decode input gesture events and generate a sequence of Android input events.

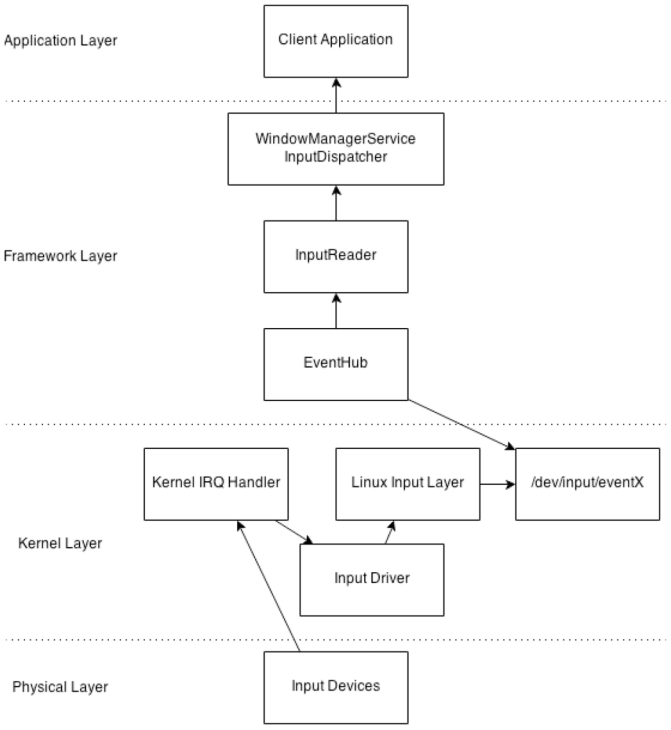

Fig. 11. Input pipeline.

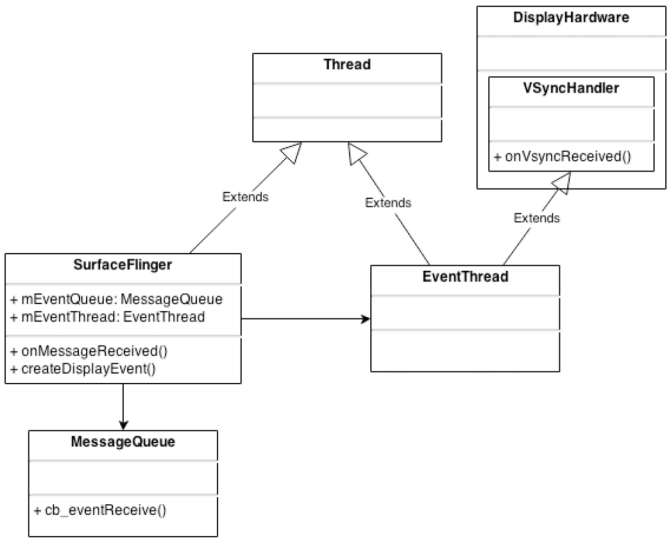

Fig. 12. Related class diagram.

The input events then are dispatched to associated applications through InputDispatcher.

As Fig. 12 shows, SurfaceFlinger uses EventThread to monitor the VSync signals. EventThread communicates with VsyncHandler to monitor VSync signals. Whenever receives a VSync signal in onVSyncReceived, EventThread will informs the MessageQueue by calling cb_eventReceive to generates a refresh event and inform onMessageReceived in SurfaceFlinger to process the upcoming events in createDisplayEvent.

Fig. 13 shows the architecture and data flow of display mechanism. Threads 1 and 2 are system threads created by

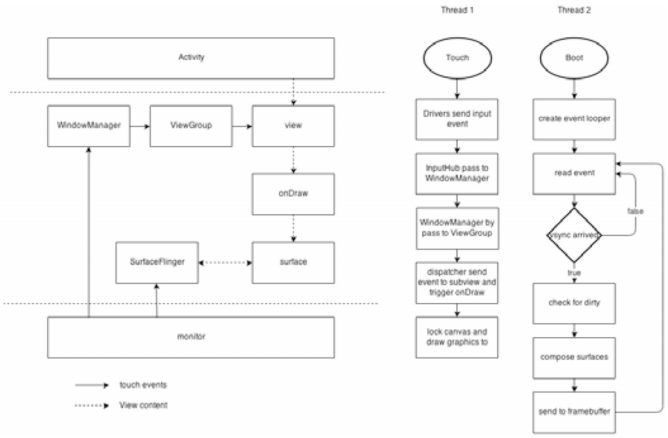

Fig. 13. Display mechanism and flow.

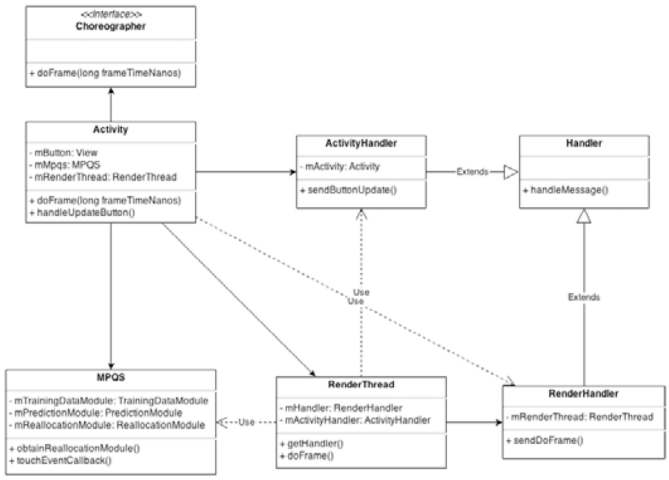

Fig. 14. Testbed class diagram.

Android. A user application is called activity in Android. Whenever a user generates a gesture event, such as touch, tap, scroll, and so on, the event will be sent to Activity by the InputDispatcher in WindowManager, and passed to ViewGroup. ViewGroup determines which view can handle this event. For general applications, touch event may trigger onDraw method to refresh the content in a view. As mentioned in Fig. 3, SurfaceComposerClient uses a surface to possess a canvas to draw. When onDraw method is invoked, a canvas should be locked by surface. A canvas holds three buffers: 1) front; 2) back; and 3) temporary buffer. A concurrent running system will use SurfaceFlinger to check the surface. If surface is updated, SurfaceFlinger will compose all surfaces in each layer from top to bottom and render them to the output device.

B. MPQS Mechanism Architecture

MPQS is an application layer motion prediction system. The implementation of MPQS is based on Android 4.1.2 JellyBean [19]. MPQS first intercepts and reproduces drag events. It then reallocates frame rendering. In order to simulate a similar operation of Android display mechanism in the application layer, we wrote a call back function name doFrame based on Choreographer which is triggered by VSync signals. A call back function will invoke sendDoFrame method in RenderHandler to inform RenderThread. As Fig. 14 shows, we provide two handlers and one render thread. The ActivityHandler maintains UI message while RenderHandler is responsible for the callback function triggered by VSync signals. After the RenderHandler passes input events, the RenderThread takes out the frame from ReallocationModule and informs ActivityHandler to trigger activity UI to update screen. We also implement a TouchEventListener in activity in order to handle the drag event from specific view, for example, a button in activity. Event listener is an interface provided by Android API that contains a single callback method triggered by user interaction. In this listener, it calls prediction module in MPQS to calculate the distance of movement. For example, if the predicted distance is 600 pixels and we will display those frames in 1 s under 30 frames/s. The distance between each frame is 20 pixels, which is transformed to relative polar coordinate by RenderThread. The RenderThread is a thread invoked by VSync callback function doFrame in choreographer. In our implementation, we handle the UI update frequency in doFrame, and the update frequency is 30 frames/s.

For example, when we drag the button in an activity, the drag event will be caught by the event listener. The input information I_s, I_r, and I_d then is passed to MPQS to generate frames. These frames will be processed and displayed by render thread when VSync signal is arrived.

SurfaceFlinger provides surface to an activity. There is a canvas in a surface. Canvas can be regarded as the actual memory space used to store the content of a view. Memory space will be sent to framebuffer later by SurfaceFlinger. In our implementation, in order to reduce computation load, we will reproduce output frames by different display refresh rate. For example, if the number of output frames is 60 frames and the refresh rate is 60 frames/s, the number of output frame will not be changed. On the other hand, if the refresh rate is 30 frames/s, only 30 frames will be output in order to reduce computation load.

VI. Experiments

The experiment environment and results will be discussed in this section. We first introduce the testbed used in the experiment. We then discuss the experimental results.

A. Experiment Testbed

The testbed is built on beagleboard-xM [20]. On this platform, we use Android API getRefreshRate to get the display refresh rate 77.85 frames/s and l_v 12 ms. The tolerance drawing time is set to 2 ms, which means that available drawing time is $12 - 2 = 10$ (ms). In order to collect drag information, we have to know the coordinates of the starting point and stoping point of drag operation and the movement time MT. The collected data are used to construct our prediction module We repeat the experiment of dragging gesture 200 times and show the results in Fig. 15. The start time a is equal

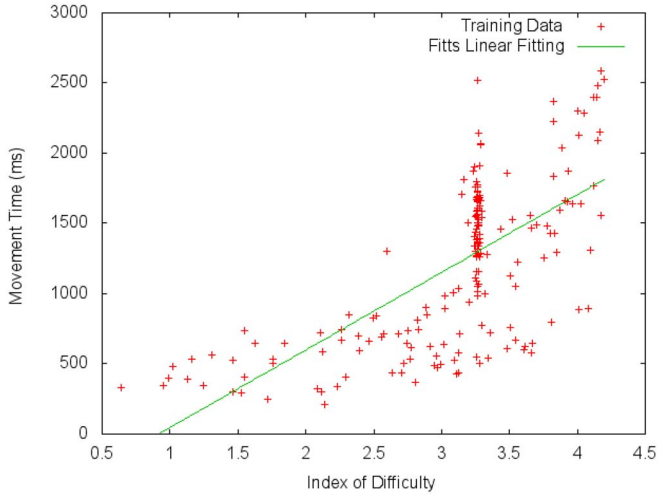

Fig. 15. Linear fitting result.

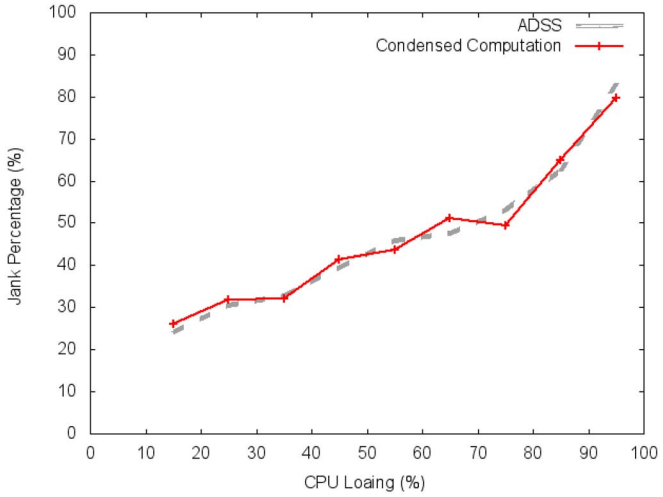

Fig. 16. Jank percentage in ADSS and condensed computation.

to -506.231 with $\pm 133.6(26.38\%)$ asymptotic standard error and the speed b is 552.716 with $\pm 42.04(7.605\%)$ asymptotic standard error. These values were calculated by gnuplot with linear fitting function. Gnuplot is an utility that can generate 2-D and 3-D plots of data and data fits. The experiment result of MT between ID 3 and 3.5 is dispersed. It means although movement distance D are similar, the movement time MT are quite different. Hence, it is desirable to have a tracer in prediction module. The detailed results are discussed in the following sections.

B. Experiment Results

In this section, we first discuss the accuracy of motion prediction. We then investigate the effectiveness of the proposed method in reducing the number of janks.

1) Accuracy of Motion Prediction and Impacts of Prediction Misestimate: Accuracy is an important factor that can affect the performance of motion prediction of MPQS. The more accurate the prediction is, the more CPU load can be reduced. On the other hands, prediction errors will induce extra CPU overhead to discard all frames and repredict new movement. In this paper, we built a tracer to improve the accuracy in prediction results. We conducted the operation more than 200 times and record the results in each operation, we take the first ten records as prediction vector, and predict coordinate of destination by vector. Each predicted coordinate and actual coordinate will be calculated in standard error, and the result is $(\alpha_x, \alpha_y) = \pm(26.6910, 45.0678)$ pixels. Based on this result, we understand there is a certain error in user interactions on the screen. This error will lead user notice that the predicted pointing position is different with user's action. So, we add a correction function in prediction module, and record operation results again. Amended conditions is set to comparison in every 100 pixels to the actual coordinate. When the coordinate difference is larger than 10 pixels, the tracer triggers interrupts, and computes standard error again in the last step of interaction. The result shows a correction will be generated every 233.74 pixels. In this paper, the average move distance is 745.5761 pixels and the average move time is 900 ms. It means that only three interrupts in average will be generated in one interaction operation. Therefore, the overhead of the proposed method is negligible. In addition, with the correction function, the standard error is reduced to $(\alpha_x, alpha_y) = \pm(7.3426, 9.5713)$, and the accuracy of prediction is increased from 75% to 80%.

2) Computation (Condensed Versus Dispersed): Fig. 16 shows the effect of CPU loading on the jank percentage. The jank percentage represents the percentage of jank when displaying a series of frames. We compare the ADSS mechanism with condensed computation mechanism. The reallocation module will tag a timestamp for each frame. The timestamps indicates the time the frame should be displayed. The frame will be dropped when system time has exceeded the expected display time, and therefore a jank occurs. If the frame application cannot obtain the canvas in surface before drawing, then this frame will not be drawn into the buffer. We called this situation lock failed frame, which will also generates a jank. Because of prediction module will induce extra CPU loading to operate condensed computation, jank percentage under condensed computation mechanism is similar to that of ADSS mechanism.

3) Computation (Regular Versus Reduction): Fig. 17 shows the effect of CPU loading on the jank percentage. The result shows that the average jank percentage is reduced from 48.63% to 34.37% after we added the motion prediction mechanism. In particularly, when the CPU loading is in the range of 90%–100%, the jank percentage is reduced to 29.87%. In application layer, the touch event is triggered by event dispatcher. Although the hardware sampling frequency is around 250 Hz, the touch event we received is much fewer than that in application. This unstable touch sampling rate will lead to inaccurate VFI. Our motion prediction, however, can avoid drawing frames based on the touch event. Because, users will not notice stutter when jank percentage is lower than 30%, our method is proved to be a practical solution to smooth graphic user interaction.

According to our results, MPQS cannot eliminate all janks because both dropped frames and lock failed frame will

TABLE IV
JANK PERCENTAGE AND FPT

		CPU loading(%)								
		10	20	30	40	50	60	70	80	90
Jank	ADSS	24.42	30.61	32.8	39.64	45.78	47.61	53.24	62.45	83.14
	Condensed Computation	26.13	31.72	32.12	41.29	43.53	51.14	49.36	64.92	79.81
	Reduced Computation	21.29	28.66	28.53	30.31	31.37	32.14	38.66	46.04	53.27
Processing Time (ms)		4.17	4.23	4.16	4.51	6.72	7.11	7.9	13.65	15.74

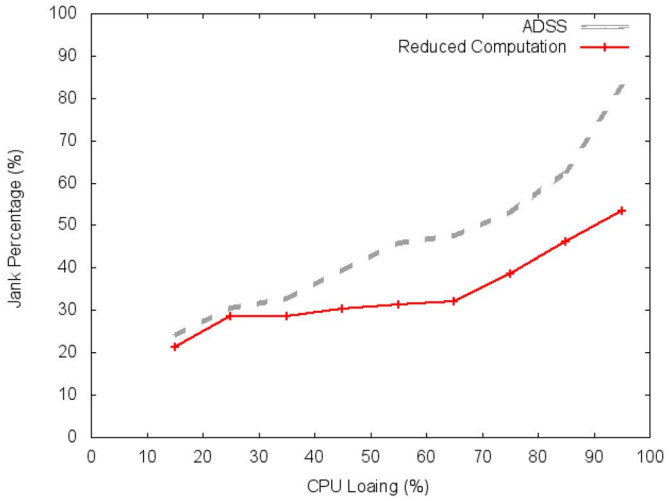

Fig. 17. Jank percentage in ADSS and reduced computation.

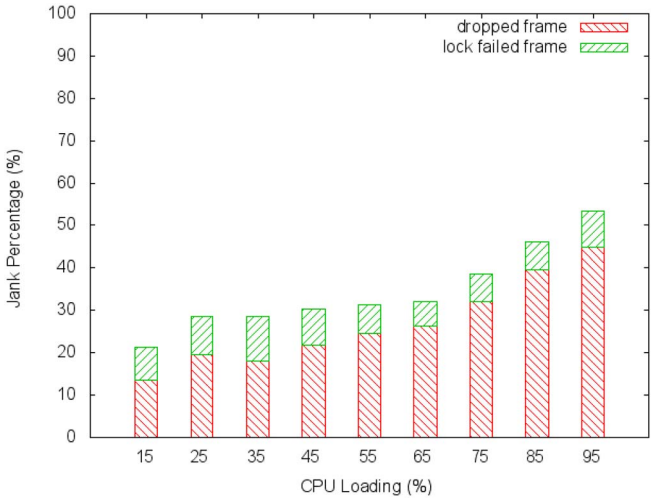

Fig. 18. Dropped frame ratio and lock failed frame ratio.

induce janks. Fig. 18 shows the average ratio of dropped frame and lock failed frame. When CPU loading is heavy, it will decrease the performance of activity. In addition, choreographer's callback time is beyond tolerated drawing time.

To further demonstrate that the proposed mechanism can reduce jank percentage, we record the drawing time in different CPU loading. As Table IV shows, when CPU loading is less than 70%, the drawing time does not exceed one VSync interval, which is 12.98 ms. Therefore the effect of rescued computation is not significant when the refresh rate

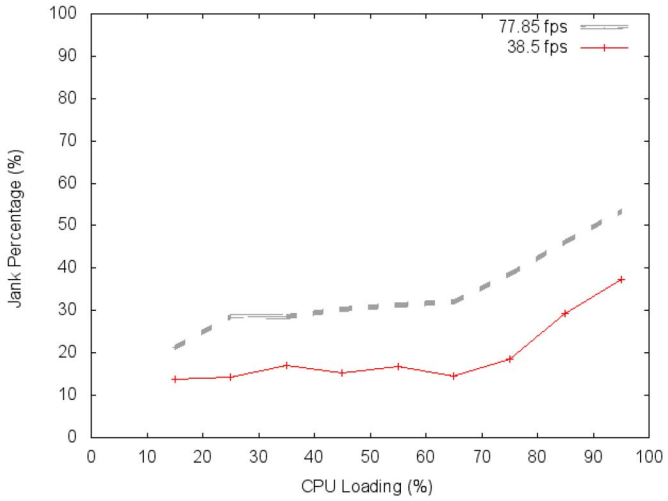

Fig. 19. Jank percentage in different FPS.

is 77.85 frames/s. However, when the CPU loading is larger than 80%, the time needed to draw each frame becomes 13.65 ms. It means an increase in jank percentage is inevitable phenomenon.

In the same case, when the display refresh rate is set to 38.92 frames/s in the reduced computation mechanism, the display interval becomes 25.97 ms. It means that the processing time of drawing becomes much longer than the original processing time. Therefore, the effect of reduced computation mechanism is more significant in a higher CPU loading. VSync interval remains unchanged in 12.98 ms, but reallocation module displays a frame every two VSync. When the CPU loading exceeds 80%, the processing time of a frame becomes more longer. Fig. 19 shows the jank percentage in different setting of display refresh rate. The jank percentage in 38.92 frames/s is much lower than that in 77.85 frames/s. But, the relative MFI and MaxFI are also increased, especially when the refresh rate is lower than 30 frames/s. When the refresh rate is less than 30 frames/s, the delay becomes noticeable. Hence, the minimum refresh rate is set to 30 frames/s, and therefore, we have two refresh rate 77.85 and 38.92 frames/s in this paper.

4) Condensed Versus Reduced: According to the result of Fig. 20, we find that reduced computation mechanism can reduce the jank percentage significantly. The reason is that the reduced computation can increase available processing time in each frame. It prevent FPT from exceeding a restricted processing time inside a frame. If display refresh rate is 77.85 frames/s, the available processing time is only 12.98 ms

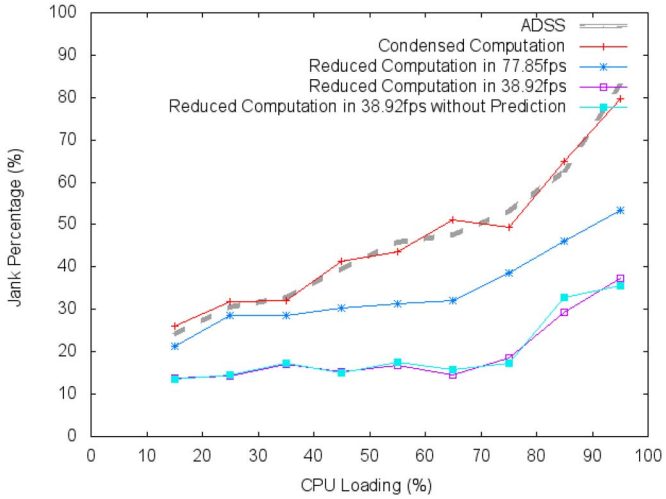

Fig. 20. Jank comparison.

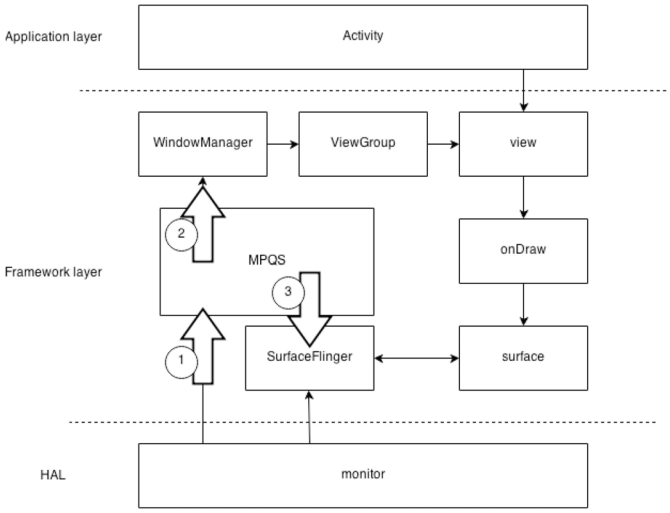

Fig. 21. Improved display mechanism.

for each frame. Reduced computation mechanism can increase the available processing time up to 26.97 ms for each frame. Therefore, it can reduce jank percentage effectively. Although reducing FPS directly without prediction module can also lead the same result in jank percentage, the touch sampling issue is still exist. This is because unexpected VFI signals will cause a noticeable delay.

VII. Conclusion

In this paper, we propose the MPQS mechanism to reduce the number of janks. The proposed MPQS mechanism collects real touch inputs on the target platform and reallocates display frames. With prediction module, the jank reduction can be up to 21.75%. Our results also show that the accuracy of motion prediction can reach up to 80% and induces only 3% CPU overhead to recover a misprediction. The comparison between condensed and reduced shows that the best method to reduce jank percentage is computation reduction. It implies jank percentage can be reduced by advanced graphic hardware. In addition, if FPT can be fewer than 16 ms and the refresh rate is 60 frames/s, the jank percentage can be minimized.

In the future, we plan to implement the MPQS in the framework layer of Android system so that we can reduce jank percentage without a modification of applications. We also plan to adjust drawing tolerance time by using statistic leveled drawing time. By doing so, we can remove the limitation of buffer count and solve the buffer issue in condensed computation and adjust l_v in SurfaceFlinger. As Fig. 21 shows, MPQS will be built on the top of the SurfaceFlinger. MPQS will listen and intercept input event by using WindowManager. In addition, the prediction module will be used to perform linear fitting estimation based on the collected events. After the regression module is constructed, we intercept input events in order to avoid triggering the onDraw method. Based on the prediction results, the reallocation module will send a virtual input event to views in activity to call the onDraw method. Next, an tracer in prediction module will be used to examine the correctness of the prediction results. If the prediction is different from users' gestures, it will drop the preprocessed results from the frame queue. In the future, we also plan to modify the onMessageReceived method of SurfaceFlinger to control the timing of surface examination. It means that SurfaceFlinger and surface will be in the asynchronous state at the same time. In ADSS mechanism, the frame drawn by surface will be composed and rendered by SurfaceFlinger at next VSync signal received. After adding MPQS mechanism, surface can produce preprocessed frames to buffer in advance. Then SurfaceFlinger can compose and render these frame at proper time which is advised by prediction module. In another word, SurfaceFlinger does not necessarily process frames output by surface. SurfaceFlinger can compose and render frames according to the timestamp which is marked by the reallocation module. The timing of render is decided by reallocation module. In order to further increase the accuracy of the motion prediction, we also plan to collect more training data from different applications.

References

[1] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann, 1993, p. 362.
[2] R. B. Miller, "Response time in man-computer conversational transactions," in *Proc. Fall Joint Comput. Conf.*, San Francisco, CA, USA, Dec. 1968.
[3] Anroid Development. *Keeping Your App Responsive*. Accessed on Sep. 2015. [Online]. Available: http://developer.android.com/training/articles/perf-anr.html
[4] F. Guo, W. Wan, W. Zhang, and X. Feng, "Research of graphics acceleration based on embedded system," in *Proc. IEEE Int. Conf. Audio Lang. Image Process. (ICALIP)*, Shanghai, China, 2012, pp. 1120–1124.
[5] S. Wang *et al.*, "Fairness and interactivity of three CPU schedulers in Linux," in *Proc. 15th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Beijing, China, 2009, pp. 172–177.
[6] C. S. Wong, I. K. T. Tan, R. D. Kumari, J. W. Lam, and W. Fun, "Fairness and interactive performance of O(1) and CFS Linux kernel schedulers," in *Proc. Int. Symp. Inf. Technol.*, Kuala Lumpur, Malaysia, 2008, pp. 1–8.
[7] X. F. Li. (Dec. 12, 2011). *Quantify and Optimize the User Interactions With Android Devices*. [Online]. Available: http://software.intel.com/en-us/articles/quantify-and-optimize-the-user-interactions-with-android-devices
[8] C. L. Wen, "Benchmarking handheld GUI: Smoothness QoE," M.S. thesis, Dept. CSIE, Nat. Chiao Tung Univ., Hsinchu, Taiwan, 2013.

[9] S. Huh, J. Yoo, and S. Hong, "Improving interactivity via VT-CFS and framework-assisted task characterization for Linux/Android smartphones," in *Proc. IEEE 18th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Seoul, South Korea, 2012, pp. 250–259.

[10] L. A. Torrey, J. Coleman, and B. P. Miller, "A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler," *Softw. Pract. Experience*, vol. 37, no. 4, pp. 347–364, 2007.

[11] *Android Development Process*. Accessed on Sep. 2015. [Online]. Available: http://developer.android.com/reference/android/os/Process.html

[12] Android Development. *How Android Renders Graphics*. Accessed on Sep. 2015. [Online]. Available: https://source.android.com/devices/graphics.html. sec.2

[13] Wikipedia. *Android Version History*. Accessed on Sep. 2015. [Online]. Available: http://en.wikipedia.org/wiki/Android_version_history.sec.Android_4.1_Jelly_Bean_.28API_level_16.29

[14] OpenGL ES. *The Standard for Embedded Accelerated 3D Graphics*. Accessed on Sep. 2015. [Online]. Available: http://www.khronos.org/opengles/

[15] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *J. Exp. Psychol.*, vol. 47, no. 6, pp. 381–391, 1954.

[16] I. S. MacKenzie, "Fitts' law as a research and design tool in human-computer interaction," *Human Comput. Interact.*, vol. 7, no. 1, pp. 91–139, 1992.

[17] S. A. Douglas, A. E. Kirkpatrick, and I. S. MacKenzie, "Testing pointing device performance and user assessment with the ISO 9241, part 9 standard," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, Pittsburgh, PA, USA, 1999, pp. 215–222.

[18] I. S. MacKenzie, T. Kauppinen, and M. Silfverberg, "Accuracy measures for evaluating computer pointing devices," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, Seattle, WA, USA, 2001, pp. 9–16.

[19] *Android 4.3, Jelly Bean*. Accessed on Sep. 2015. [Online]. Available: http://www.android.com/about/jelly-bean/

[20] Wikipedia. *System Time*. Accessed on Sep. 2015. [Online]. Available: http://en.wikipedia.org/wiki/System_time

[21] D. T. Nguyen *et al.*, "Reducing smartphone application delay through read/write isolation," in *Proc. 13th Annu. Int. Conf. Mobile Syst. Appl. Serv. (MobiSys)*, Florence, Italy, 2015, pp. 287–300.

[22] X. Wang, Z. Li, and W. M. Wonham, "Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.

[23] J. Li, L. Shu, J.-J. Chen, and G. Li, "Energy-efficient scheduling in nonpreemptive systems with real-time constraints," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 2, pp. 332–344, Mar. 2013.

[24] M. G. Valls and P. B. Val, "Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems," *J. Syst. Architect.* vol. 60, no. 2, pp. 221–233, 2014.

[25] P. Basanta-Val and M. García-Valls, "A library for developing real-time and embedded applications in C," *J. Syst. Architect.*, vol. 61, nos. 5–6, pp. 239–255, 2015.

[26] P. Basanta-Val and M. García-Valls, "Resource management policies for real-time Java remote invocations," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1930–1944, 2014.

[27] P. Basanta-Val, N. C. Audsley, A. J. Wellings, I. Gray, and N. Fernandez-Garcia, "Architecting time-critical big-data systems," *IEEE Trans. Big Data*, vol. 2, no. 4, pp. 310–324, Dec. 2016.

Ying-Dar Lin (F'13) received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 1993.

He is a Distinguished Professor of Computer Science with National Chiao Tung University, Hsinchu, Taiwan. He was a Visiting Scholar with Cisco Systems, San Jose, CA, USA, from 2007 to 2008. Since 2002, he has been the Founder and the Director of the Network Benchmarking Laboratory, National Chiao Tung University, which reviews network products with real traffic and has been with the Certified Test Laboratory of the Open Networking Foundation since 2014. He also cofounded L7 Networks Inc., Hsinchu, in 2002, which was later acquired by D-Link Corporation. His current research interests include network security, wireless communications, and network cloudification. His research on multihop cellular was the first along this line, and has been cited over 750 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced.

Dr. Lin is the Editor-in-Chief of the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He serves on the editorial boards of several IEEE journals and magazines. He is an IEEE Distinguished Lecturer for the period 2014–2017, and an ONF Research Associate.

Edward T.-H. Chu (M'10) received the Ph.D. degree from National Tsing Hua University, Hsinchu, Taiwan, in 2010.

He has over four years of work experience in industry, where he researched on embedded software. He was a Visiting Scholar with Purdue University, West Lafayette, IN, USA, in 2009. He joined the Department of Electronic and Computer Science Information Engineering, National Yunlin University of Science and Technology, Douliu, Taiwan, as an Assistant Professor in 2010, and has become an Associate Professor in 2015. He holds a Chinese patent. His current research interests include embedded systems and real-time operating systems.

Evan Chang received the M.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan.

He is a Software Engineer with Sonix Technology Co., Ltd. His current research interests include embedded systems and human–machine interaction.

Yuan-Cheng Lai received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 1997.

He joined the faculty of the Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, in 2001, and has been a Professor since 2008. His current research interests include performance analysis, protocol design, wireless networks, and Internet of Things applications.