Reducing Power Consumption in Android Applications

Amit Singhai, Joy Bose, Nagaraju Yendeti
Web Solutions
Samsung R&D Institute India- Bangalore
Bangalore, India
{a.singhai, joy.bose, yn.raj}@samsung.com

Abstract— **Battery consumption in mobile hand held devices is an important issue, given the recent slew of such devices with advanced computing capabilities. In this paper we propose some software based approaches to reduce the power consumption of a mobile device running an Android application. In the first approach we consider a typical Android application. We try to minimize the power consumption by reducing the main window surface size. In the second approach we consider the applications having higher graphics or rendering requirements and utilize surface view for rending the content, such as games, internet browser and video player, which contribute most in draining the battery. Our approach involves removing the surface view and directly utilizing the application's main window surface. We present various experiments to measure the power consumption for every scenario. We also discuss about advantages and problems of the proposed methods along with some possible solutions.**

Keywords—android; power consumption; current measurement; surface view; window manager; surface flinger

I. INTRODUCTION

Mobile computing devices such as smart phones and tablets are embedded systems mostly powered by battery. Power consumption in such devices is an important issue, with the advances in processing capability of such devices and restrictions on the battery size and weight.

Certain applications such as network I/O intensive applications consume lot of power due to transmitting and receiving signals, whereas graphics intensive applications like games consume lot of power due to large GPU and CPU utilization. Since power consumption directly has an effect on battery time in mobile devices, so while designing an application for such devices it is crucial to have this consideration as well. In this paper we propose a few methods which result in lesser power consumption for applications on Android operating system powered devices.

Our aim is to reduce the CPU/GPU utilization in maintaining and merging various drawing surfaces in android application. Although in this paper we consider only Android, the same principle can be theoretically applicable for other mobile operating systems.

The rest of the paper is organized as follows: Section 2 looks at related work in the area of power consumption in mobile devices. Section 3 discusses an experimental setup to measure power consumption in a mobile device. Section 4 talks about our first approach of reducing main application window surface size for reducing power consumption, along with possible cons with the approach and way to overcome the problems. Section 5 looks at Android provided Surface View in detail. Section 6 talks about our second approach to reduce power consumption by utilizing window surface instead of a surface view, along with possible cons with the approach and way to overcome the problems. Section 7 concludes the paper.

II. RELATED WORK

In order to minimize the power consumption, one way is to minimize the utilization of hardware components. Turning off sensors which are not being used at the user level is one method to optimize the power resulting in longer battery hours. Hardware manufacturers especially in embedded systems domain are coming up with the latest processors and DSPs which consume less battery and can also operate on low voltages. To understand utilization of power in various hardware components, the work by Aaron Carroll [6] is very useful.

Another approach to optimize the power consumption is software based. Programmers can optimize the power consumed by using software techniques, efficient design and algorithms. Mian Dong et al [14] have experimented with different color schemes to analyze their power consumption for OLED displays, and found that choosing darker color schemes can also result in large power saving. This approach is applicable on high end mobile devices as the display is mainly composed of OLEDs.

There have been a number of studies undertaken to analyze the power consumption in smartphones. Warty et al [8] have presented some ways to measure the power consumption in a mobile device. We have used a similar setup and equipment [5] for our readings too. Perucci et al [12] survey the energy consumption of smartphones to determine the energy intensive components of a mobile device. Bo Zhao, Byung Chul and Guohong [7] suggest a way to reduce power consumption in

978-1-4799-2572-8/14/$31.00 ©2014 IEEE

Web Browsers by using a proxy mechanism. Mandyam et al [10] study the power consumption related to Web Sockets. Thiagarajan et al [11] study energy consumption in mobile browsers. All these papers focus on the network side, aiming to optimize the power consumed in network connectivity. But they do not focus on the Graphics side.

An NVidia whitepaper [13] details techniques to enable high end graphics in mobile devices. However, it is also concentrated on the hardware side and do not talk of how power consumption can be optimized by using programming techniques. Generally speaking, the software aspect of power optimization in graphics and application side is not well studied. In this paper, we focus on software techniques to reduce the power consumption for Android devices on the graphics, rendering and application UI side.

III. POWER CONSUMPTION MEASUREMENT SETUP

The setup for power measurement in this paper is shown in figure 1. The following equipment has been used for the measurement

- Samsung Galaxy Note3 device
- PC for Android application development
- USB Cable
- Power Monitor from Monsoon Solutions Inc. [4]

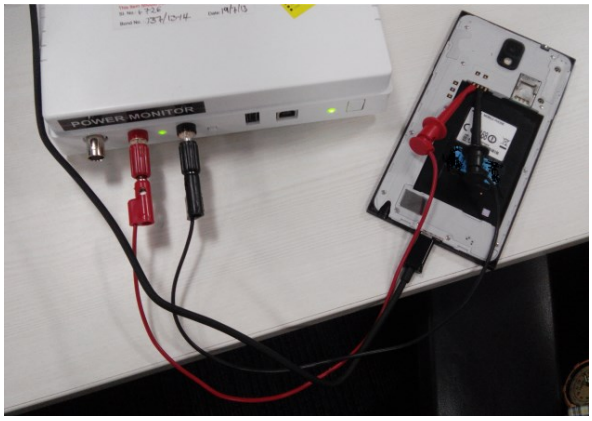

Fig. 1. Setup of the equipment to monitor power consumption

The power monitor comes with a software suite which the user can use to start and stop the measurement for the use case duration. We have fixed the output voltage to 4.0 volts so the current consumption is directly proportional to consumed power. Fig. 2 shows the graphical user interface provided by the power monitor software. When the user wishes to measure power consumed in use case under interest he/she presses run and start performing the use case. Once the use case is finished, stop button is pressed. Average current/power consumed during the experiment is shown along with the graphical power consumption over the time.

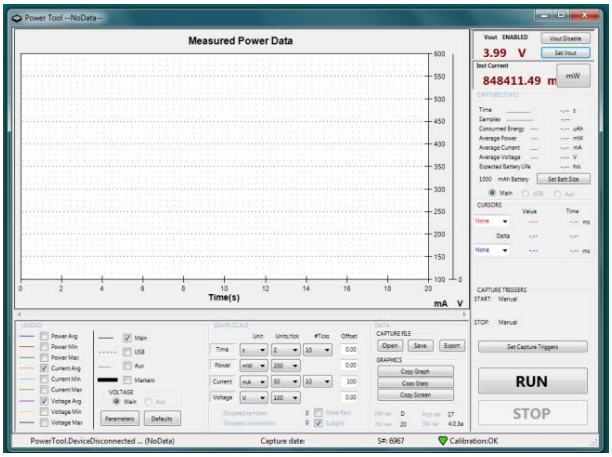

Fig. 2. GUI for executing a use case for power consumption measurements. The software controls for power monitoring and data capture for a particular use case are displayed.

IV. REDUCING POWER CONSUMPTION BY REDUCING MAIN WINDOW SIZE

Every Android application consists of one or more activities. An activity can have one or more windows associated with it. At a minimum an activity has one main window associated with it. If activity shows a dialog then the dialog has another window. A window is inherently of type android.view.Window class and contains one or more surfaces.

At a minimum one window has one surface, but if it includes a surface view then the surface view has its own surface. The details of surface view are discussed in next section.

The activity's window is added to window manager as top view. As we mentioned earlier every window has its own surface where all the UI contents are actually drawn by android graphics. The android view hierarchy is tightly coupled to this window surface so that various user touch events etc. can be passed to proper view element in the view hierarchy as shown in Fig. 3.

We can say that the view root which is the inner most element in view hierarchy is attached to window surface.

Many applications require full screen area to show their user interface or contents while some applications only require a partial screen area. One such typical example is chromium browser in which only the address bar is the android view in entire window, which is less than 10% of overall window height.

As we mentioned earlier that a drawing surface is associated with every window, reducing the size of the window results in reducing the same size of the surface. Thus lesser area requires drawing by android's graphics system which results in saving some power.

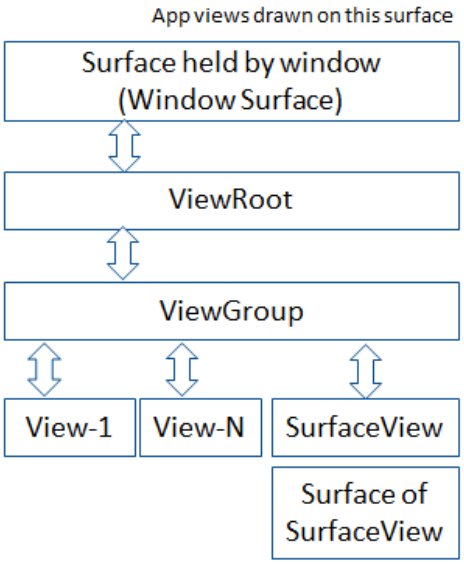

Fig. 3. The view hierarchy in a typical Android application.

If an application has limited UI and does not utilize full screen area, it is recommended to use reduced height of main window. Similarly if an application consists of many surfaces or layers such as which involve a surface view can also reduce the size of main window at the same time stretching the parent layout in which surface view is added so that it can show in reduced window size. Detailed mechanism of surface view is explored in section 5.

Fig. 4 shows a typical such implementation. In above such example activity the window height can be reduced to layout-1 height if remaining area is not utilized.

In case the remaining area is filled with surface view we can still reduce the height of window if we just stretch the layout-2 height to full window height, it is due to the fact that surface view has its own surface which is separate from main window surface.

As we have stated earlier that view root is associated with window surface. Reducing the window height will result in smaller attachment with view root, so in remaining area user events will not respond. To overcome this problem the developer has to delegate the touch events from Activity to the corresponding view group which is now out of modified window rectangle. Table 1 shows some measurements that we made on a chromium based browser. In the measurement we reduced the main window size by 90%.

As we can see from the table 1, the power consumption has been reduced in some cases. At the same time in forth case power consumption actually increased by a marginal amount, it is due to the fact that we had to delegate touch events to the text edit fields which actually increased a little power consumption. Thus, depending upon application requirements and careful programming this approach results in reducing the power consumption.

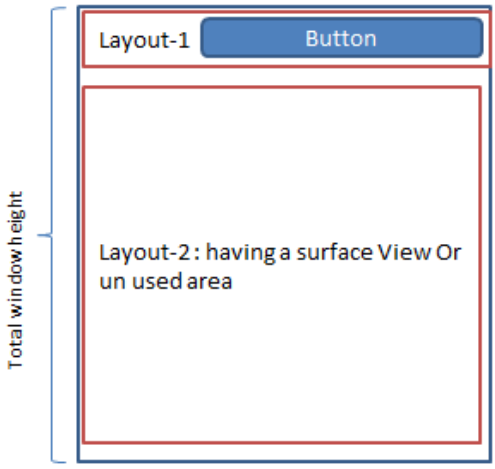

Fig. 4. An example activity with or without a surface view.

TABLE I. MEASUREMENT OF THE AVERAGE CURRENT WITH FULL WINDOW SIZE AND REDUCED WINDOW SIZE ON A CHROMIUM BASED BROWSER

Use Case	Average current in Full window size(mA)	Average current in Reduced window size(mA)	Improvement (mA)
Browsing same web site over wi-fi	877	834	43
Pinch zoom	922	868	54
Device rotation	697	690	9
Text Input on Facebook	415	427	-12

V. APPLICATIONS USING SURFACE VIEW

In the Android OS, only the UI thread can update the user interface. This means that worker threads cannot update the views. For graphics related application like games, updating the view in a separate thread is desirable. To solve this problem, Android provides a special view called Surface View which has its own surface. In this section we explain how surface view works. In the next section, we present an optimization technique to reduce power consumption.

An example activity in Android is shown in Fig. 5. It has one surface view along with other standard android widgets e.g. “Text View”. The surface view requests a transparent hole in the window in which it has been added [1]. This transparent hole allows user to see the surface (which belongs to surface view) behind the window. Since surface view has its own surface, a separate thread can update it. It is particularly useful in games and other applications where rendering is done by native code.

One drawback of this mechanism used by surface view is that it consumes more power as two different surfaces are maintained.

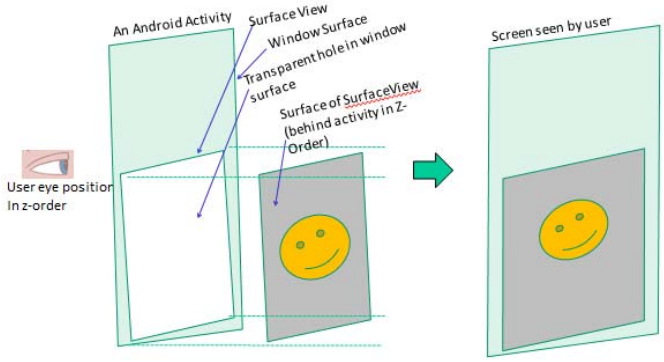

Fig. 5. Mechanism of the Surface View in Android OS.

Also, the upper surface contributes in more processing as the area acquired by surface view has to be made transparent so that the below layer which is Surface View's own layer can be visible. These surfaces have to be painted separately and merged at flinger level. However using surface view has its own benefit. It is part of the view hierarchy, so any kind of manipulation and touch event processing are easily possible.

In the following section, we explain how power consumption can be optimized in Android when using Surface View.

VI. Proposed Approach for Optimizing the Power Consumption

In the previous section we introduced Surface View and explained how it consumed more power since two surfaces are maintained and merged. We propose to save power consumption by directly drawing on the application's main window surface.

In certain applications where Android specific widgets such as buttons or text edit areas are minimal, it is possible to directly use the surface of the window instead of surface view for rendering or drawing.

This approach has some limitations. If the User Interface has a lot of views which respond to user touch, using the window's surface directly is not recommended. So the proposed approach can be utilized mainly if the application is graphics intensive, and where the number of android widgets shown on main window is negligible

Once the window's surface is taken for showing contents, other views cannot be visible until they are directly added via the window manager. Adding a view via 'WindowManager' has some disadvantages, which are discussed in a later section of this paper.

As mentioned, in our proposed approach, the application main window's surface is used to render the content [2]. In this section, we test the effectiveness of our approach in reducing power consumption. We do this by measuring the power consumption in the two mentioned cases (with and without surface view) for the same graphical rendering requirement on the same device. In this way, we find out the improvement by removing the surface view and drawing directly on window's surface. Later we verify the number of surfaces visible at flinger level in our test applications.

A. Experiment to measure the power consumption for content rendering on surface in both the cases (with and without surface view)

We conducted an experiment to find out the consumption of power when only drawing is being performed. For this two Android applications have been developed using the publicly available Android SDK. In the first application, random bubbles have been drawn on the surface of surface view in a thread.

In the second application random bubbles have been drawn with exact size and duration as in first application, but here they are drawn directly on the surface of main window.

TABLE II. Measurement of the Average Current With and Without Surface View

Iteration number	Application running time (in seconds)	Average current with Surface View (mA)	Average current without Surface View (mA)
1	10	189	182.7
2	10	189	182.12
3	10	188.5	182.3

For precise measurements, all sensors such as accelerometer, Wi-Fi, Bluetooth, screen rotation, GPS were disabled on the Android device.

For drawing directly on the surface of window, an API provided by Android called Window.takeSurface (SurfaceHolder.Callback2) has been used.

For the experiment, the power monitor instrument replaces the battery and logs the power consumed. The instrument also provides the software user interface (UI) to record the measurement of the current (and hence, power) consumed when the experiment is run for a time period. The user has a choice to start and stop the measurement as they wish.

We recorded the current consumption for 10 seconds of application run time with stable current output from the power monitor equipment. Measurement for longer durations shell have same results as current output is stabilized.

The results are displayed in table 2 for three iterations. As we can see, we get an approximately 7 mA advantage in the power consumption in our approach, as compared to the power consumption of the application with the surface view approach.

A second set of measurements has been taken on a Chromium based browser on the same Android device. The browser uses heavy graphics and rendering capabilities to draw html content on the screen. The Chromium based browser uses surface view for drawing the content.

TABLE III. MEASUREMENT OF THE AVERAGE CURRENT WITH AND WITHOUT SURFACE VIEW IN CHROMIUM BASED BROWSER

Scenario for power measurement	Average current with Surface View (mA)	Average current without Surface View (mA)	Improvement with our approach
Pinch & zoom of content	924.89	922.02	2.8 mA
Browsing same web site over wi-fi	604.73	543.84	60.0 mA

We modified the code so that it utilizes widow surface for drawing the content instead of surface view. With two Application packages (.apk) prepared with and without surface for same browser code base we were able to take power readings and thus compare the results in both the user scenarios. The readings have been taken 3 times for each use case and the average results have been taken into consideration. The results are shown in table 3. As we can see, our approach results in an improvement in the power consumption while using heavier applications such as a web browser.

B. Experiment to find the visible surfaces for the two cases (with and without surface view)

To find out various visible surfaces or layers in our application, we use an Android SDK provided tool called "dumpsys". We use this tool for capturing surface flinger dump logs using shell command "adb shell dumpsys SurfaceFlinger"[3].

Following are dumpsys output for Surface Flinger in both cases.

- *Extract of Surface flinger output with surface view*

```
Visible layers (count = 6)
.
.
+ Layer 0xb8057b90 (SurfaceView) id=64
    Region transparentRegion (this=0xb8057d80, count=1)
    [  0,   0,   0,   0]
    Region visibleRegion (this=0xb8057b98, count=1)
    [  0,  75, 1080, 1920]
    .
    .
+Layer0xb802ac68
(com.example.withsurfaceview/com.example.withsurfaceview.MainActivity)
id=63
    Region transparentRegion (this=0xb802ae58, count=1)
    [  0, 121, 1080, 1920]
    Region visibleRegion (this=0xb802ac70, count=1)
    [  0,  75, 1080, 1920]
```

The output clearly shows two layers. First is Surface View as it has its own surface, second is application's main Activity which can be identified by the package name provided in AndroidManifest.xml file. The transparent area in Main Activity corresponds to size of the Surface View. These readings are in line with the fact that Surface View requests transparency.

- *Extract of Surface flinger output without surface view*

```
Visible layers (count = 5)
.
.
+Layer0xb805a398
(com.example.withoutsurfaceview/com.example.withoutsurfaceview.MainActi
vity) id=69
    Region transparentRegion (this=0xb805a588, count=1)
     [  0,   0,   0,   0]
    Region visibleRegion (this=0xb805a3a0, count=1)
     [  0,  75, 1080, 1920]
```

The surface flinger output shown above clearly states that there is only one surface present, which corresponds to the main window's surface.

However, there are certain disadvantages to using our approach. As taking window's surface bypasses the view hierarchy, it is difficult for the application to add some Android widgets such as buttons, text edit or other touch responding widgets.

As mentioned earlier in section 4, 'ViewRoot' is parent of all other views in Android and one of the important roles of ViewRoot is to delegate the touch events to underlying child views properly so that only relevant child responds to the corresponding touch events. Though surface view has its own surface but it is still a child of 'ViewRoot' so the view hierarchy works perfectly.

Fig. 6 shows the view hierarchy after capturing the window's surface for drawing. As shown in the figure, the views have to be drawn directly via window manager this is one of the disadvantages of our approach.

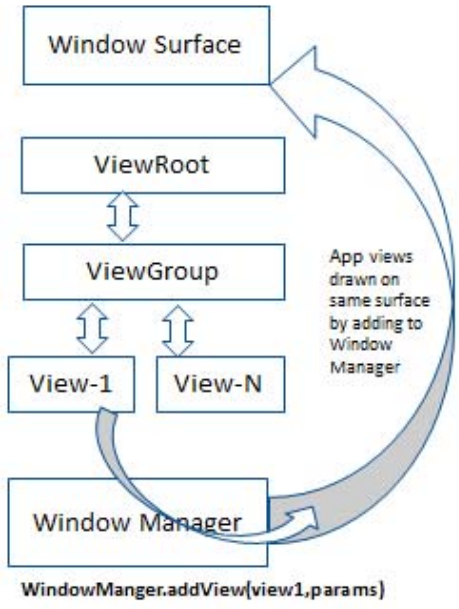

Fig. 6. Showing a view via window manager.

Window Manager is responsible for maintaining all the surfaces. Since only one surface is being utilized for both rendering and Android views, the management of these views would be difficult, in comparison to the surface view approach. This is another disadvantage of our approach.

Now we will find a few ways to receive user input as well as manage Android views effectively, thus overcoming some of the disadvantages of our approach as mentioned in the previous section.

Since view management is difficult in our approach, at the same time the number of views is limited, we provide some methods to overcome the limitation of view management.

Once the window's surface is taken, the following actions can be done to add the remaining widgets:

- Add the widgets in separate window such as dialogs. In case the application has very few standard Android widgets, then it is good to create a separate layout and inflate in a separate dialog. Such a dialog can be shown and hidden as per the need.
- Add the widgets directly via window manager. If views/layouts are added directly via window manager the touch events should be delegated from main activity of application to such views for responding to user actions. It is due to the fact that the view hierarchy is bypassed and programmer has to take care of these events.

  This delegation of event will require more processing and thus results in slightly more power consumption especially for faster gestures such as pinch zoom-in, pinch zoom-out, scrolling the content. So before adapting a particular solution, it is recommended that the application requirements are analyzed thoroughly along with its user interface.

In this section, we looked at a few possible ways to overcome the limitations of our approach. In the next section, we conclude the paper and explore some avenues for future work.

VII. Conclusion and Future Work

We have proposed few approaches to reduce the power consumption. In the first approach, we tried to reduce the window height thus reducing the drawing surface area which is not utilized by the application. In second approach we proposed to use window surface instead of surface view in selective android applications. We performed an analysis of our approaches and measured the current consumption on fixed voltage on various user scenarios. Our motive was to enable researchers to find out ways using only software to reduce the power consumption by an application. We have analyzed the scenarios and requirements when the approaches fit well. Our solutions were limited to the applications which either do not utilize full screen area or which have higher graphics rendering requirements in comparison with the applications which have more static standard Android views and widgets.

The power is consumed in hardware components which are actuated and utilized by software. In future, we will study ways to reduce the power consumption by optimization of hardware components as well. This can be done by writing sophisticated algorithms and designing applications properly, thus saving a considerable amount of power that increases the battery life.

There are many such scenarios in which power can actually be saved. In future we will analyze some of these scenarios.

Acknowledgment

The authors would like to thank Harshala, Darshan Venu and Prakash Reddy from testing team, of Samsung Research Institute Bangalore, India for providing support in this research.

References

[1] Android Surface View documentation http://developer.android.com/reference/android/view/SurfaceView.html

[2] Android application window and takeSurface API documentation http://developer.android.com/reference/android/view/Window.html

[3] ADB and Android tools documentation http://developer.android.com/tools/index.html

[4] Android developer documentation http://developer.android.com

[5] Power Monitor equipment manual http://www.msoon.com

[6] A. Carroll, G. Heiser, "An analysis of power consumption in a smart phone", Proc. USENIX Annual Technical Conference, 2010

[7] B. Zhao, B. C. Tak, G. Cao, "Reducing the Delay and Power Consumption of Web Browsing on Smartphones in 3G Networks", Tech Report, Department of Computer Science & Engineering, The Pennsylvania State University, 2010.

[8] N. Warty, R. K. Sheshadri, W. Zheng, D. K. Outsonikolas, "A First Look at 802.11n Power Consumption in Smartphones", Proc. ACM PINGEN 2012

[9] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. Morley Mao, L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones", Proc. CODES/ISSS '10 Eighth IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis, 2010

[10] G. D. Mandyam, N. Ehsan, "HTML5 Connectivity Methods and Mobile Power Consumption", W3C Proposed Recommendation, 2012

[11] N. Thiagarajan, G. Aggarwal et al, "Who killed my battery?: analyzing mobile browser energy consumption", Proc. WWW' 12, 21st International Conference on World Wide Web, 2012

[12] G. B. Perrucci, F. H. P. Firtzek, J. Widmer, "Survey on Energy Consumption Entities on Smartphone Platform", Proc. IEEE 73rd Vehicular Technology Conference, 2011

[13] "Bringing High-End Graphics to Handheld Devices", NVIDIA Whitepaper, 2011

[14] Mian Dong Yung-Seok Kevin Choi Lin Zhong, "Power-Saving Color Transformation of Mobile Graphical User Interfaces on OLED-based Displays", *IEEE Transactions on Mobile Computing*, September 2012