2014

Using User Saliency For Effective OLED Display Power Management

Kiat Wee TAN

Singapore Management University, kwtan.2010@phdis.smu.edu.sg

Using User Saliency For Effective OLED Display Power Management

by

Tan Kiat Wee

Submitted to School of Information Systems in partial fulfilment of the
requirements for the Degree of Master of Science in Information Systems

Dissertation Committee:

Rajesh Krishna Balan (Supervisor / Chair)
Associate Professor of Information Systems
Singapore Management University

Archan Misra (Co-supervisor)
Associate Professor of Information Systems
Singapore Management University

Jiang LingXiao
Assistant Professor of Information Systems
Singapore Management University

Singapore Management University
2014

Using User Saliency For Effective OLED Display Power Management

Tan Kiat Wee

Abstract

The display component on mobile device has always been a main power drain compared to networking and processing components. Even with advancement in display technology such as Organic Light Emitting Diodes (OLED) displays to help reduce this cost, the increasing display size and new mobile usage paradigm (App stores and Social networking), the display still remains as the largest power drain component.

In this thesis, we describe an approach to reduce power consumption on OLED display by dimming areas of the screen that the user might not be interested in. We determine these uninteresting areas by reviewing large number applications to determine the Region-Of-Interest; this ROI is used to develop a simple dimming model that is deployed to save power. The system is designed such that power can be save without impacting the user usability, task time, as well as a small system overhead to reduce power cost and not impact over system performance. This thesis show power savings of between 23 to 34%. Our work complements existing OLED power management literature such as colour mapping approach and improves over default constant dimming approaches.

Table of Contents

List of Figures

ix

List of Tables

Acknowledgments

I would like to express my gratitude to my advisor Prof. Rajesh Krishna Balan for the guidance and support of my study and research, for his patience, encouragement.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Archan Misra and Prof. Jiang LingXiao for their encouragement, insightful comments, and hard questions.

I thank fellow students, Kartik Muralidharan, Sougata Sen and Joseph Chan for the stimulating discussions. In addition, I like to thank Tadashi Okoshi of Keio University for his help in this thesis. Lastly, I like to thank Ms Ong Chew Hong and Ms Seow Pei Huan for their admin support.

Dedication

I dedicate my dissertation work to my parents for their unwavering support.

List of Publications

Conference Papers

Kiat Wee Tan, Tadashi Okoshi, Archan Misra, and Rajesh Krishna Balan. 2013. FO-CUS: a usable & effective approach to OLED display power management. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, (UbiComp '13).

Kiat Wee Tan and Rajesh Krishna Balan. 2012. Adaptive display power management for OLED displays. In *SIGCOMM Comput. Commun. Rev. 42*, 4 (September 2012).

Kiat Wee Tan and Rajesh Krishna Balan. 2012. Adaptive display power management for OLED displays. In Proceedings of the first ACM international workshop on Mobile gaming (MobileGames '12). ACM, New York, NY, USA, 25-30.

Chapter 1

Introduction

The boom of mobile devices in the last decade has seen these devices becoming more powerful and larger (in terms of screen size). With these devices becoming powerful which leads to a higher power drain on their batteries. Although battery technologies have tried to keep up, with the advancement in microelectronics, power consumption has never stay stagnant. In fact, this power consumption has been made worse due to the changing nature of smartphones such as increase popularity of App stores, improved network condition (faster bandwidth, reduced latency), new application paradigm such as social networking and even mobile gaming.

In general, there are two approaches to reduce power consumption on the devices: Hardware or software. The hardware approach targets hardware components to reduce their power consumption. This can include using power efficient semiconductors such as processors, chipsets, wireless communication devices, etc. The software approach targets the efficient utilisation of hardware through clever adaption to specific scenarios such as context or content. For example, adaptively adjusting processor frequency to reduce power consumption or putting the device to sleep when the user is not using the device.

In this thesis, we focuses on the software approach on displays of the mobile device, in particular OLED (Organic Light-Emitting Diode) and/or AMOLED (Active-Matrix Organic Light-Emitting Diode) display technology. This is because

these displays compared to other components such as processor or network consumes the largest amount of power. This is due to display being one of the primary interface component on the device that users put their attention on (always on). Hence, even with advanced OLED technology that was designed to save power (and space), displays still dominate the power consumption on modern mobile devices.

1.1   Motivation

Modern handheld mobile devices such as Smartphones and tablets uses two main display technology, LCD (Liquid Crystal Displays) and OLED to support high resolution content with low latency response. The main power drain difference between these two technology is that LCDs utilise the backlight component to illuminate the screen that draws the most power whereas OLED turns on individual pixel based on what is required to be displayed. In this work, we focuses on OLED due to their increasing popularity that are being used in devices such as the Samsung Galaxy Series (from 1 to 4), HTC One, Nokia Lumias to tablets.

Although OLED displays are very power efficient compared to their LCD cousins, they still consumes large amount of energy when compared to other phone components, for example, as shown in Figure 1.1, OLED display on the Samsung Galaxy S3 still dominates about 44% alone in power consumption compare to other phone modules using the Facebook App. Even with 3 network refresh to fetch new feeds from the server within a test period of 10 minutes, the network transaction cost consumes much less than the display when compared to display given the use-case where the user is browsing news feed on Facebook.

Thus one of the main goals of this thesis is to reduce the power consumption of mobile devices that utilised OLED display. OLED display power management via software is not a new approach, two solutions exists: colour mapping [13, 14, 36] and screen dimming [36, 40]. Both approaches save power by manipulating the displayed content, however it comes with usability trade off (unnatural colour,
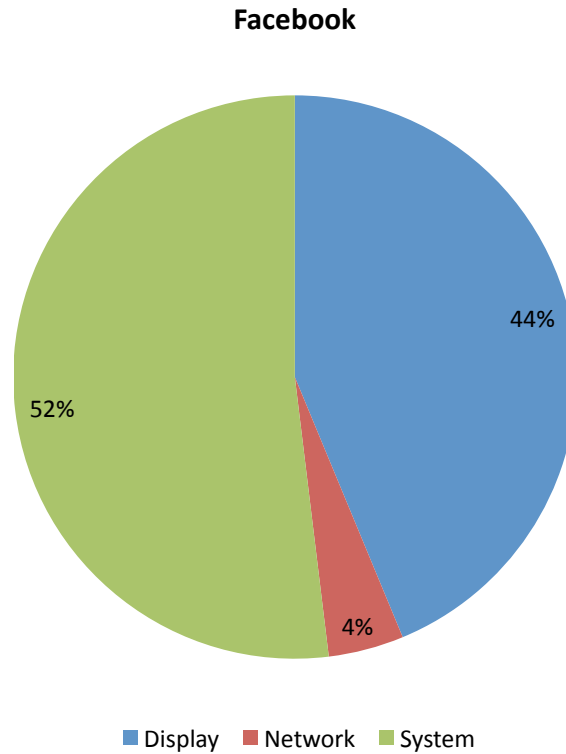
Facebook

Figure 1.1: Power consumption of Facebook on Samsung Galaxy S3. The display consumes 44% of the total power compare to 4% network transaction (3 Facebook newsfeed refresh) with the rest 52% consume by the system (such as Facebook process, Android OS and other process)

unable to see content, etc). Hence, the main research motivation for this thesis is to effectively save power on these OLED mobile devices without affecting usability cost ddm.

1.2   Research Objectives And Contributions

The research objectives of this thesis are defined by the following goals:

1. The use of human saliency to aid in OLED display power management.

2. The effective use of saliency to balance the trade off between power savings and usability of the system.

3. The implementation, deployment, and testing of the system using both modern mobile operating systems and devices and a large range of commonly

used applications.

4. The implementation must be of low overhead and capable to scale across different App category.

This system will have the following contributions:

1. The system will show how OLED based power management can be effectively and efficiently applied to modern mobile device and usage scenario. OLED-based power management is not a new area but our research will show how these techniques can be effectively applied to modern setting.

2. This research will also show how the proposed technique will minimise the user impact.

3. Our implementation will take into account the efficiency and scale such that the overhead will be kept to a minimum.

4. This technique presents a viable option for low battery constraint scenarios to extend the usage mileage as well as for other processes to utilise as a feature/option to offset the power cost if the processes are unable to reduce power cost within its own processing space.

1.3 Dissertation Roadmap

This dissertation is organised into the following 9 chapters. Chapter 2 reviews the literature background of the related work that is needed in this work. We then formally define research problem and solution in Chapter 3. Chapter 4 describes the methodology where we review the design and architecture. Chapter 5 describes the implementation details in Android. The evaluation is done in both Chapter 6 and Chapter 7 for power performance and user study respectively. Chapter 8 looks at the issue of implementation of the solution in games. Lastly, we conclude and review possible future work direction in Chapter 9.

Chapter 2

Literature Review

Power management is not a new concept on mobile devices, for example, modern mobile operating systems incorporates power management functionality in the devices. These may includes (this list is not exhaustive):

1. **Screen Timeout** This time out setting forces the system to switch off the screen and goes into idle mode when it detects no interaction by the user. By switching off the screen and going into idle mode, this allows the system to save power.

2. **Automatic Brightness Adjustment** This setting allows the OS to automatically adjust the screen brightness to different lighting condition. By adaptively reducing the brightness allows power saving to occur.

3. **Idle/Sleep mode** Idle or Sleep mode allows the operating system to put either the entire phone into low power mode and/or specific components such as cellular/Wi-Fi communication or sensors to low power mode to conserve power. This employs techniques such as processor frequency scaling or intelligent scheduling in the communication protocol, which are usually transparent to the user or developers.

The functionality list presented is not exhaustive but it can be broadly categorised into four main power management areas: Display, communication, process-

ing(CPU) and sensors. We will review some key ideas drawn from each of these areas.

1. **Communication** Communication refers to the available communication components on the device, this may include cellular (2G/3G/LTE), Wi-Fi, Infrared and Bluetooth. The well-known technique to save power on these network interfaces is to put them to sleep. Past and recent work has shown that by putting network interfaces to sleep allows considerable power savings [7, 39, 22].

   However, the key idea is when and how to put the interfaces to sleep without trading off aspect of usability, efficiency and in some case utility. For example, one such technique utilised in-game context within a gaming application on the mobile device to put network interface to sleep achieving power saving without impacting game play [1].

2. **Processing** In processing, power can be save in a number of ways. In addition to switching between standard sleep/idle modes, one simple approach is to change the processor frequency to save power [5]. However, to effectively manage power savings, software approaches are needed in conjunction with hardware methods. For example, one such approach, application-aware, based on the constraints of the system, applications adaptively changes their fidelity to match power requirements hence prolonging the power [33, 16, 3]

3. **Sensors** Sensors are relatively new additions to consumer handheld mobile devices. These sensors can include from accelerometer, GPS, barometer, light and even fingerprint. As most sensors require their data streams to be processed continuously, this posed a problem to their deployment due to the fact that operating continuously meant a constant draw on power.

   Hence, methods such as intelligent duty cycling and effectively switching on and off sensors set by detecting which context they entered into provides a good means of power savings [43]. Similarly, there are also works that

optimise the sensing pipelines to manage data such as piggyback allow power savings to be achieved at application-agnostic level [45].

4. **Display** The display on mobile devices consumes the most power regardless if they are LCD or OLED based. Although newer OLED technology has vast improvements on power usage, the nature of the device still dominates power consumption. In this paper, we specifically address the issue of power management on mobile display and demonstrate the solutions.

2.1 The Displays

Displays on mobile devices have gone through several technology through the decades. Prior to the launch of Android and iPhone Smartphones, majority of handheld mobile devices (feature phones) usually utilised passive-matrix Liquid-Crystal-Displays (LCD) that usually requires an external light source (sunlight or an extra light bulb in the device) for the user to read off the screen. Although power efficient, these devices generally do not offer high quality resolution or colour.

Current modern mobile devices, specifically Smartphones and tablets in current consumer market uses generally two types of display technology: Active-Matrix LCD types and OLED. Although providing high resolution, low latency and colour to modern devices to support modern demands and usage, these devices compared to their predecessors trade off power efficiency.

The power problem is further compounded with ever increasing large handheld devices such as 5 to 7 inches screen size and large tablets (7 inches and above). In addition, with the change of usage behaviour (social networking, etc) on modern handheld devices and the popularity of Application (App) stores (Google Play, Apple App Stores, etc), users now spent considerable amount of time on these devices than before further aggravating the power issue.

7

2.1.1 LCD

Modern Active-Matrix LCD (Liquid-Crystal Display), can be generally, viewed as being make up of two main components that draws power: display and backlight (see Figure 2.1). The backlight sits behind the display and controls the amount of light (illumination) projected through to the display whereas the display holds the displayed content. Hence, by varying the backlight, the LCD controls how bright the display but at the same time the backlight dominates the power consumption (the brighter screen, the higher the backlight power consumption).
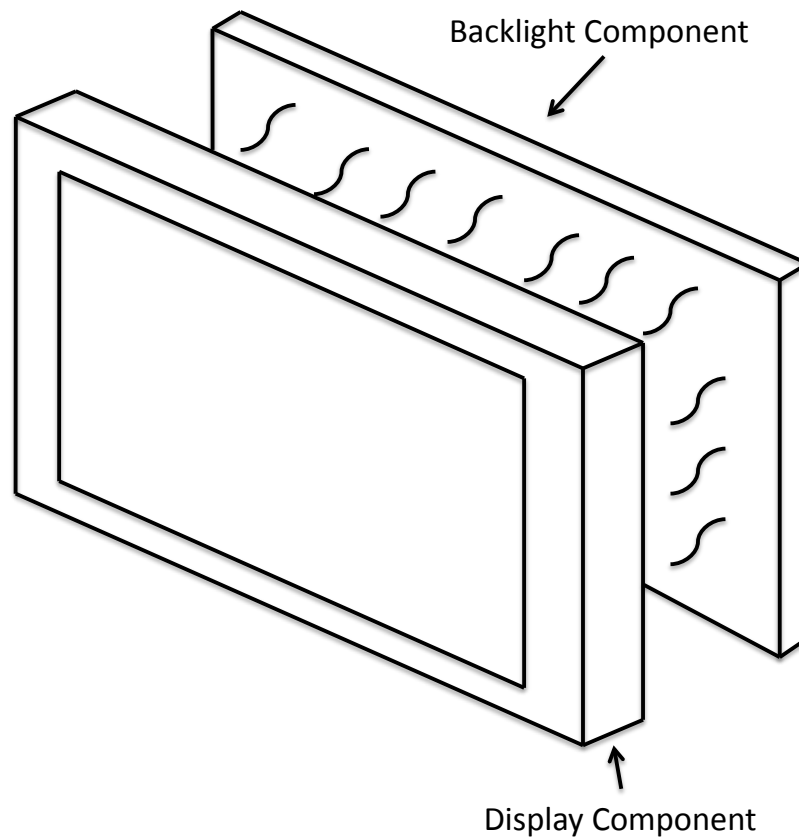
Figure 2.1: Simplified exploded view of a Liquid-Crystal Display (LCD). The *Display Component* contains the glass substrate, liquid crystal and polarising filter. The *Backlight Component* provides the light source that illuminates the display content

Power management has predominately focus on the backlight to reduce power consumption from better hardware designs to system-level optimisations [19, 2, 8, 10, 34] that utilised component power model or application context. The technique

mainly focuses on reducing the backlight and compensating the image (Gamma correction, etc). This approach tries to reduce power consumption without impacting on usability on the user such that the compensated image resembles as closely to the original.

2.1.2 OLED

In contrast, OLED or Organic Light Emitting Diodes displays do not have backlight component. Instead, individual pixel on OLED displays consists organic semiconducting compound. Depending on the manufacturing process, each pixel can consists of the red, green and blue (RGB) diodes [38] or more see Figure 2.2. Hence, for any image content, each pixel will light up and vary each RGB diodes to achieve the desired colour and intensity for the content.
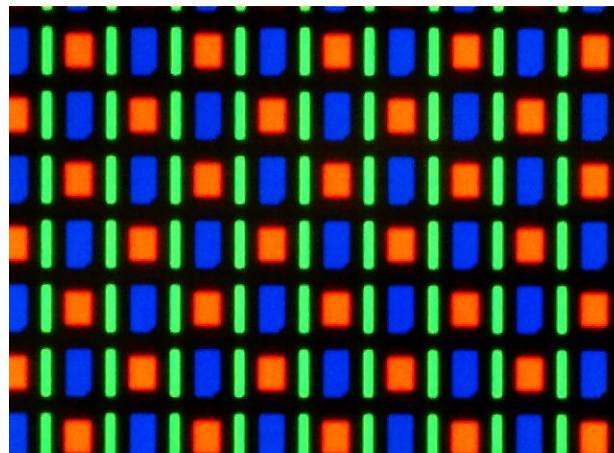
Figure 2.2: Magnified image of the AMOLED screen on the Google Nexus One smartphone using the RGBG system. [30]

Hence, this meant that content dictates the power consumption for day-to-day usage compare to backlight component in LCD displays. As a result, LCD-based approach to power saving will not work on OLED displays such as reducing the backlight and compensating it by adjusting the displayed content's gamma to compensate for the reduce in brightness [2, 8, 10, 34].

2.2 Power Management On The OLED Display

OLED and Active Matrix OLED (AMOLED) displays have the characteristic that each pixel is emissive (no backlight). This meant that each pixel contains red, green and blue component when light up generates the required colour for the displayed content thus no backlight required. However, each organic compound differs from each other in terms of their organic compound, lifetime and power consumption. Hence, to achieve a constant lifetime for the OLED display, each RGB component differ in size during manufacturing seen in Figure 2.2.

This results in different power consumption curves for different devices, shown in the Figure 2.3 is our measurement of the Samsung Galaxy S3 OLED Dim-Response curve, each colour consumes different power at different brightness with the white colour consuming the most power. Thus, the two main approaches for power management on OLED devices are the colour mapping and/or display darkening which will be discussed in the next section.

2.2.1 Colour Mapping

The colour mapping technique essentially maps a high power consuming displayed colour to a lower power-consuming colour. This approach uses display characteristic of the OLED display. Seen in Figure 2.3 shows the power vs. colour response for Samsung Galaxy S3, each colour consumes different power due to the different properties of the organic semiconductor material. Thus by mapping one colour to another power saving colour provides an advantage to save power [23, 36], however, the problem to this approach is two fold.

First, each OLED manufacturer uses different manufacturing process, hence each power curve seen in Figure 2.3 would differ between manufacturers, this would be hard to derive a device agnostic solution. Seen in works from Mian Dong et. al [13, 12, 14], power models derived from each phones differs hence, to effectively apply this solution, a series of models has to be build prior to accommodate different

Dim-power curve.pdf

Samsung Galaxy S3 OLED Brightness-Power Curve
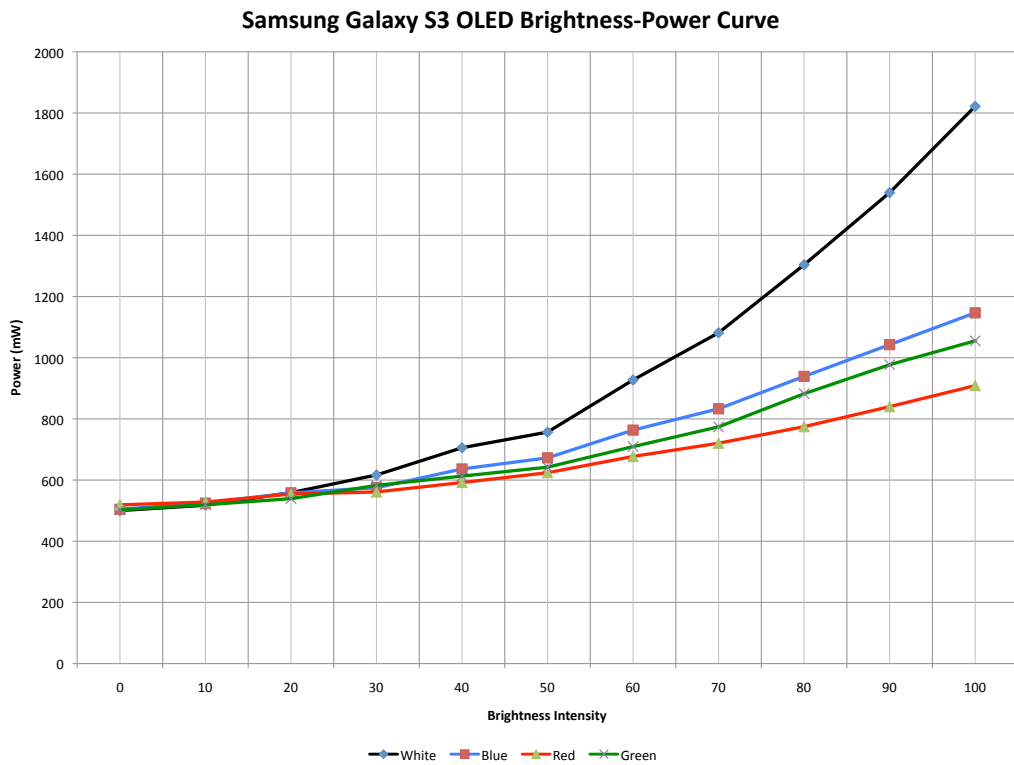
Figure 2.3: OLED Brightness-Power curve of Samsung Galaxy S3. Power consumption of the primary RGB + White colour with increasing brightness. Zero brightness indicate black screen

phone types.

In additional, different screen sizes as well as resolution density or DPI (dots per inch) affects how the content is going to be displayed. Smaller DPI would have a poorer power savings when compared to a larger DPI. Hence, colour mapping has to be adapted carefully in order to maximise power savings.

Second, colour mapping usually comes a cost of usability, as an example, one efficient colour mapping solution is to do colour inversion (white to black or vice versa) seen in Figure 2.4. Although achieving power savings, it would be not usable since the image may have has lost its semantic information during the inversion as user now no longer able to identify what the picture is about. This lost in semantic information can be resulting from a change in colour can include lost of fidelity or usability [14].

Original   Colour Inversion
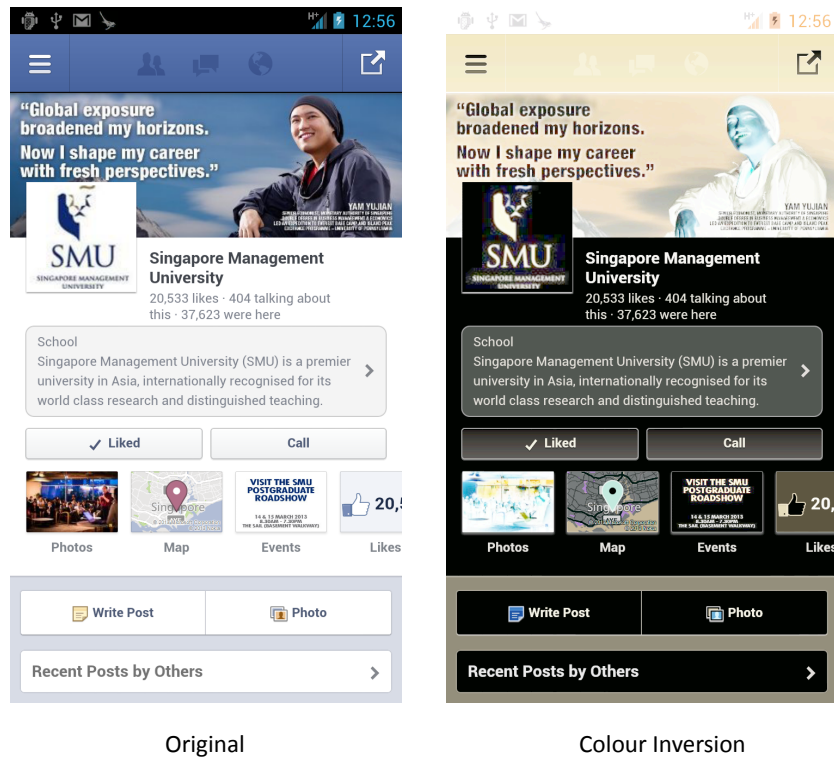
Figure 2.4: Colour Mapping: Inverting colours impact usability

2.2.2   Screen Dimming

Observing Figure 2.3, the other alternative is to reduce power consumption is to reduce the intensity of the pixels or screen dimming to save power. Earlier works by Kamijoh et. al [27] as well as Ranganathan et. al [36] showed that saving OLED display power was possible by reducing the image brightness and reducing the number of pixels to be turn on.

Kamijoh showed that by choosing a better UI design for the Smart watch, the system is capable of saving power and does not impact on usability at the same time. Seen in Figure 2.5, although both clock face shows the same time. But by choosing the analogue clock face, it saves more energy than the numeric clock face due to the fact that less pixels are turn on compared to the numeric face[27].

Modern OLED based Smartphone also utilised this technique to save power such as dimming the screen after a certain period of in activity. Recent work has also tried to similar approaches in the hardware level using dynamic voltage scaling

Clock Face: Numeric

Clock Face: Analogue
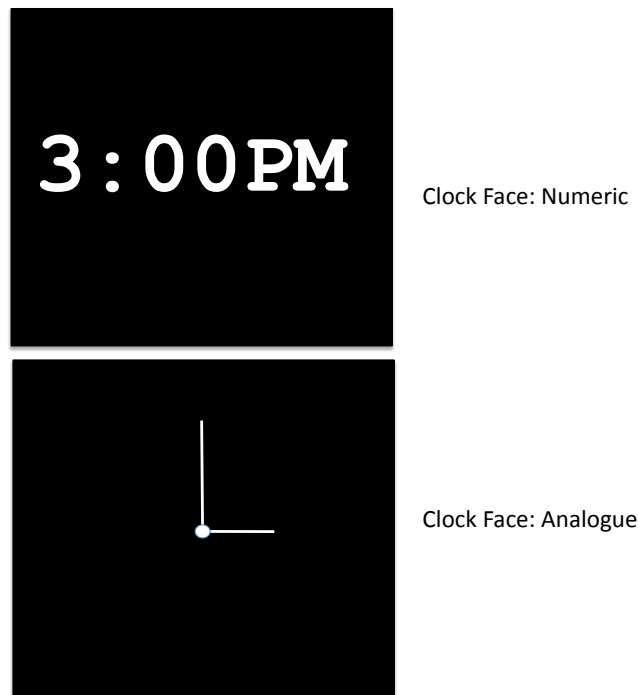
Figure 2.5: OLED clock-face on Smart-watch. The analogue clock face saves more power than the numeric due to lesser pixels gets turn on

(DVS) to change the power consumed by the diodes in an OLED pixel [9, 37]. However this approach degrades the luminance of the image, which can affect the end user experience. Our current solution is complementary to and can be used in tandem with both colour-remapping and hardware-based approaches.

2.3 Saliency

User saliency in this work is defined as the area on the screen where user's Region-Of-Interest (ROI) lies or the area where the user places attention on. Saliency arises from the study of human visual attention from cognitive science [17] through a comprehensive study of where humans places their visual attention given a scene. For our purpose of this work, we will only require to use the following scope, in particular, two main concepts: The bottom-up and top-down approach to visual attention.

Figure 2.6: Saliency Examples. Bottom-Up Approach uses the content features such as colour, contrast, etc to determine the salient region for image segmentation [4]. Whereas Top-Down usually relies on cognitive process of humans such as reading a page would require the user to have certain patterns and directions on where the visual attention would be [29].

2.3.1 Bottom-Up

Saliency has been widely used in the field of image and video processing [25], this approach is generally known as the bottom-up saliency where interesting visual features (colour, contrast, etc) are used to determine the salient or interesting bits of the displayed contents (see Figure 2.6. Such approaches have also been used on mobile platforms [44], where it bridge the disconnect between users viewing attention with the salient image features to build a better Region-of-Interest (ROI) model that aid in the user viewing process. However, although such approaches are good in detecting ROI in still images, it is not useful when the system is interactive such as in gaming and it consumes battery resources for processing.

2.3.2 Top-Down

The top-down approach uses the cognitive approach to determine where the visual attention is being placed. For example, when the user is reading a book where user places the attention on the start of the paragraph and follows the sentences across the paragraphs on a page, see Figure 2.6.

Hence, to determine the focus of the user, we can either study the content and use cognitive heuristic to locate the region of interest(where the user might start

reading the paragraphs, etc) or use eye tracking equipment to determine the user gaze [15, 17, 26, 44]. The drawbacks of using this approach is such that additional specialised equipment needs to be use and results might differ due to different users or heuristic (E.g. right to left reading style in Arabic script).

2.4 Behaviour

Another factor that influence this work in power management is user behaviour. Behaviour affects power consumption, specifically, when battery levels influence the usage behaviour in order to extend usage mileage. Truong et. al [41] shown that users change their usage patterns from altering the screen brightness, turning off communication radios to reducing app usage or even limiting themselves to phone usage to reduce their consumption on the battery. This shows users are willingly to trade usability aspects on their mobile devices for extended battery usage.

In addition, earlier works by Rahmati et. al has shown that mobile users do not really understand power management features. Hence users are not taking full advantage of power management features [35]. This also indicates the system will have to do manage power on the user behalf in order to maximise the power management features on the phone.

Chapter 3

Research Problem and Proposed Solution

3.1 The Research Problem

Organic Light Emitting Diodes (OLED) displays that is use on modern mobile devices such as Smartphone or tablet continue to be the dominant power consumption component on these devices. Although OLED offers better power savings compared to their LCD cousin, their power consumption still depends very much on their display content in terms of colour and intensity. Hence, software approaches to manage power via colour mapping and darkening is insufficient. These techniques require not only the use of application context but also how the context is being utilised by the user to be truly effective in managing the power consumption.

3.2 The Proposed Solution

We proposed our OLED power management technique by utilising existing OLED power saving techniques with the addition of usability consideration. This involves the concept of human saliency and behaviour to provide an effective and efficient way of power management on OLED displays minimising the system overhead and

user usability while achieving power savings. The following motivating scenario highlights how we expect our system to work.

3.3 Motivating Scenario

Lisa, a third year university student, own a Samsung Galaxy S 3 smartphone that has an OLED display.Like many of her peers, she uses her Smartphone through her school day for chatting with friends, checking status on social networking applications such as Facebook, Google Plus, WhatsApp, short gaming sessions, YouTube watching and traditional usage such as email, texting, calendar, etc. She charges her phone if opportunity allows but usually, her Smartphone runs out of battery before the end of her school day.

Using our solution, Lisa prepares a set of dimming profiles for applications that she commonly use (e.g. Facebook, WhatsApp, Gmail). As soon as her smartphone's battery reaches 20%, our solution will activate to prolong the usage time. As soon as Lisa activates an application with a dimming profile, the dimming will be applied to the application.

Each dimming profile contains the areas of dimming as well as the amount of dimming needed for that area. Each dimming profile is configured by Lisa with the aim of saving as much power as possible while maintaining the usability Lisa is comfortable with. With the dimming applied, the power consumption of the display is reduced thus allowing her extended usage time.

This ability to significantly extend the smartphone's battery life by losing a little bit of display usability is the key benefit of this thesis.

Chapter 4

Methodology

In this chapter, we first review the popularity of OLED based handheld mobile devices to verify the merit of the problem and solution. We then review the design principles and requirements. We first determine the value of dimming by understanding the power savings it can achieved. Second, we looked at the saliency of mobile applications to determine the salient regions and formulate a simple Region Of Interest (ROI) model. Lastly, we look review the design architecture based on the design principles, requirements and the ROI model.

4.1   Popularity of OLED Devices

Although LCD based smartphones continues to be in the market place (e.g. Apple iPhone). OLED based smartphones have become extremely popular due to their low power and slim form factor. We discovered that OLED-based smartphones are extremely popular, with Samsung leading the way and other manufacturers soon to follow [31]. Samsung, according to Gartner [18], has become the world's most popular smartphone manufacturer — with their OLED-based Galaxy Series smartphones (S II, III & IV, Note I & II, etc.).

In addition, due to market and technology issues, larger OLED displays (e.g., tablets, laptops, or TVs) [31] will become common anytime soon. Indeed, searching

the tablet market revealed only 2 OLED-based tablets (the Samsung Galaxy Tab 7.7 and the Toshiba Excite 7.7) with the larger 10 inch variant only available after 1st quarter of 2014. Hence, we focused our research effort on reducing the power consumption of smartphone OLED displays.

4.2 Design

The proposed solution aims to save power on OLED-based mobile devices using the dimming (darkening/gradient) approach by utilising the saliency properties of the user in such a way that we can effectively managed power with minimising the impact to usability. The system design can be broken down into three areas, the display, saliency and user/usage behaviour as seen in Figure 4.1. These principles are derived from our related work in Chapter 2.
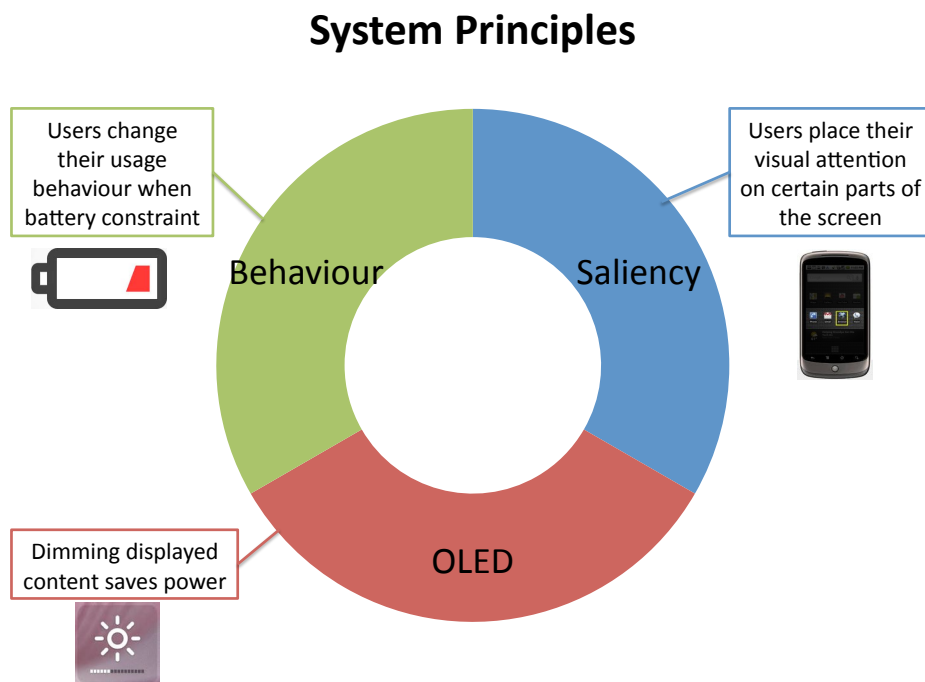
Figure 4.1: System design is influenced by 3 main areas: The OLED Display, User Saliency and User Behaviour

Seen in previous section, to save power on the OLED device, our work focuses on utilising a combination of reducing the pixel intensity or switching off the pixel

to save power. To do so we dimmed areas that area that are not interesting to the user. To determine which areas are interesting we used the concept of user saliency.

User saliency helps to determine which area on the screen that the user is putting their attention to, as discussed in Section 2.3), this area is known to be the Region-Of-Interest (ROI). To determine this ROI on the applications, we intensively review a large number of applications and build a simple ROI model of which areas on the screen the users are looking at and using these models to dimmed the uninteresting areas. However, these ROI will inadvertently cause usability problems to the user since other areas of the screen is dimmed, hence to mitigate the problem, we looked at user behaviour.

User behaviour influence our design such that we know when and how to effectively implement the dimming models and at the same time reduce the usability impact. We know from previous Section 2.4 that reducing usability on the phones will not make this approach useful, hence, to implement this model successfully, this approach is extremely useful when the user is strained by low battery condition where the user is more willing to trade usability for extra usage time.

Hence by implementing this approach in low battery condition, we are able to save power on these mobile device, prolong their available usage time at a usability level that the user is comfortable and willing to accept to.

4.3   Requirements

To guide us in the design, the proposed design will have the following requirements that the system will adhere to.

1. **User Friendly**. The proposed design must not affect the end user experience in unacceptable ways. However, with proposed design, the user will only be encountering the dimmed screens only at low battery conditions where the impact of usability is acceptable to the user to prolong extended battery life.

However, the envision system will not dimmed the screens when the mobile device is not battery constraint as not to affect the end user experience. This validate this acceptability using both the researcher determination as well as the user. We will demonstrate this in the following Chapter 7.

2. **Significant Power Savings**. The proposed design should have significant power savings across the system not just on OLED display component. This indicates that power savings in the OLED will translate to a savings across the system, hence bring power consumption per application down.

3. **Generalisable**. In this proposed design, the proposed system will be application-agnostic instead of an application-specific approach. The system will be implemented such that we affect all installed binaries (default and 3rd-party) instead of tuning each application.

4. **Low Computational Requirements**. It must be computationally efficient, i.e., It cannot save OLED display power at the expense of incurring a significant CPU or other resource energy cost. We proposed to inline the dimming using existing available alpha-blending APIs within draw process of Android applications.

4.4   Determining the Range Of Savings

In order to determine the amount of savings is possible using dimming, we conducted a simple test using the Monsoon Power Monitor to determine the amount savings achievable using the dimming approach. Shown in the Figure 4.2, as we gradually dimmed from 0% to 100% using a constant dimming approach where we dimmed the screen uniformly. We are able to achieve savings up to 51% from the baseline. However, it should be noted that due to the emissive nature of the OLED display, under different apps, power savings differs depending on their displayed content.
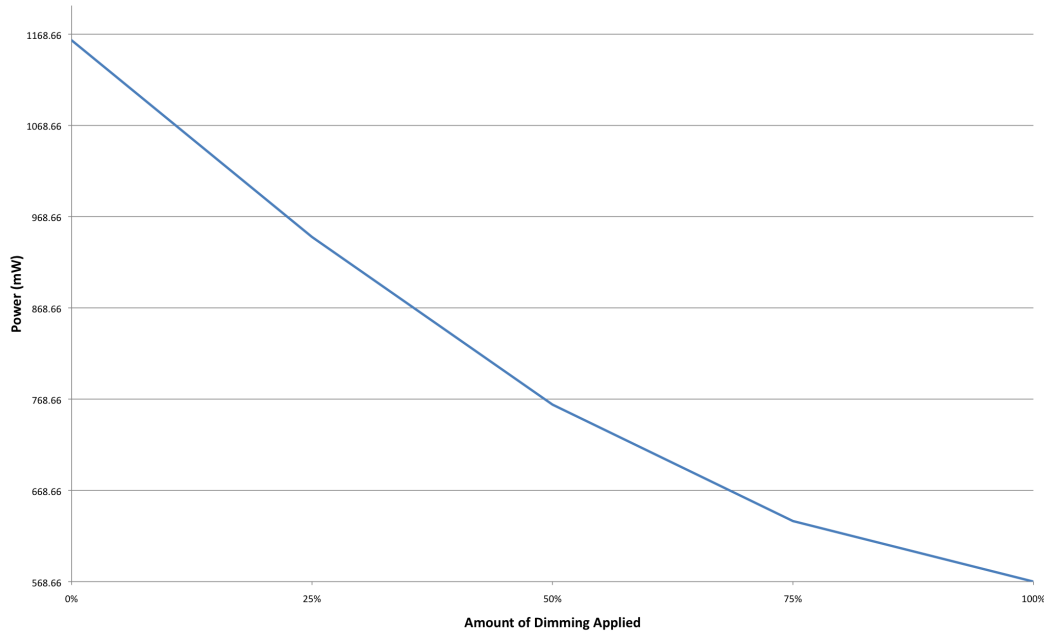
Figure 4.2: Constant dimming applied to Facebook. Power consumption drops as the the screen is progressive dimmed from 0% to 100% (where screen is black at 100%)

Figure 4.3 shows the display of the mobile device at different levels of dimming during the test. Although, significant power savings can achieve when we increase the level of dimming, this comes at the cost of usability where the user now have a harder time using their screen (strain to the eyes), display looses fidelity above 75% dimming.

4.5 Determining the Region Of Interest

To understand the dimming methods that are likely to prove successful, we first carefully surveyed the top 20 applications in each of the 26 Google Play application categories (520 apps in total). Our analysis shown in Figure 4.4 revealed the following:

1. A majority of applications (64%) place their new content either at the top or bottom portions of the screen. For a reasonably large set of applications (29%) (e.g. book readers), the new content was on the entire screen. There
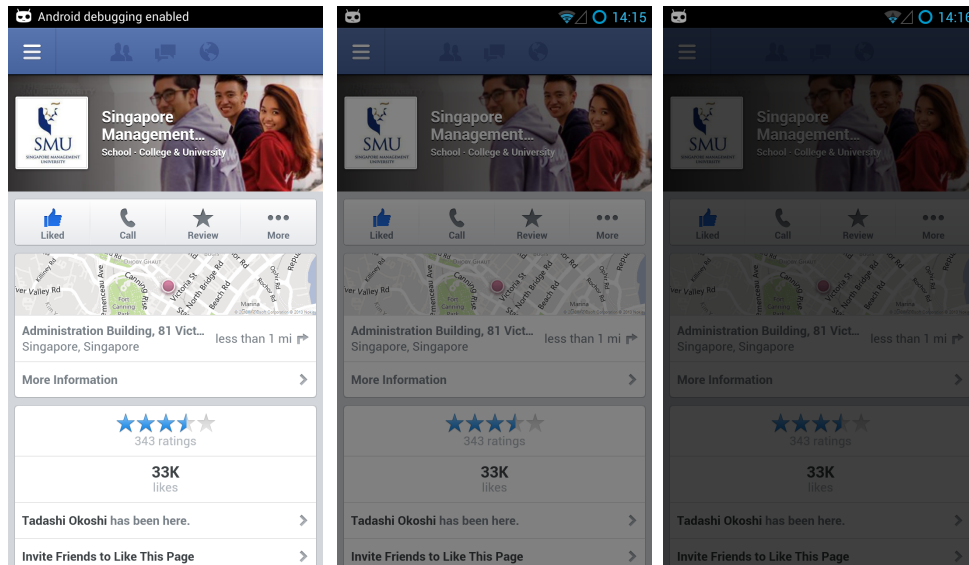
a) Original	b) 50% Constant Dimming	c) 75% Constant Dimming

Figure 4.3: Effects of constant dimming at 50% and 75% for Facebook

were also a few applications (7%) (e.g. wallpapers) that were meant to be run in the background and could be safely dimmed or even turned off when power conservation becomes crucial.

2. In addition, most applications (69%) used scrolling to access new content, while a few applications (30%) (book readers again) used page flips (where you swipe across the screen and the whole page refreshes with new content). An even smaller set (1%) (*Ebay*) used a combination of both scrolling and page flips.

3. Finally, a significant majority of applications were principally focused on digital consumption and are thus *read-only* (77%), with very little user input (beyond navigation controls). However, a few applications (23%) (e.g. *WhatsApp* and *Twitter*) required the user to provide a lot of input.

Content Location

Interaction Mode

Content Consumption
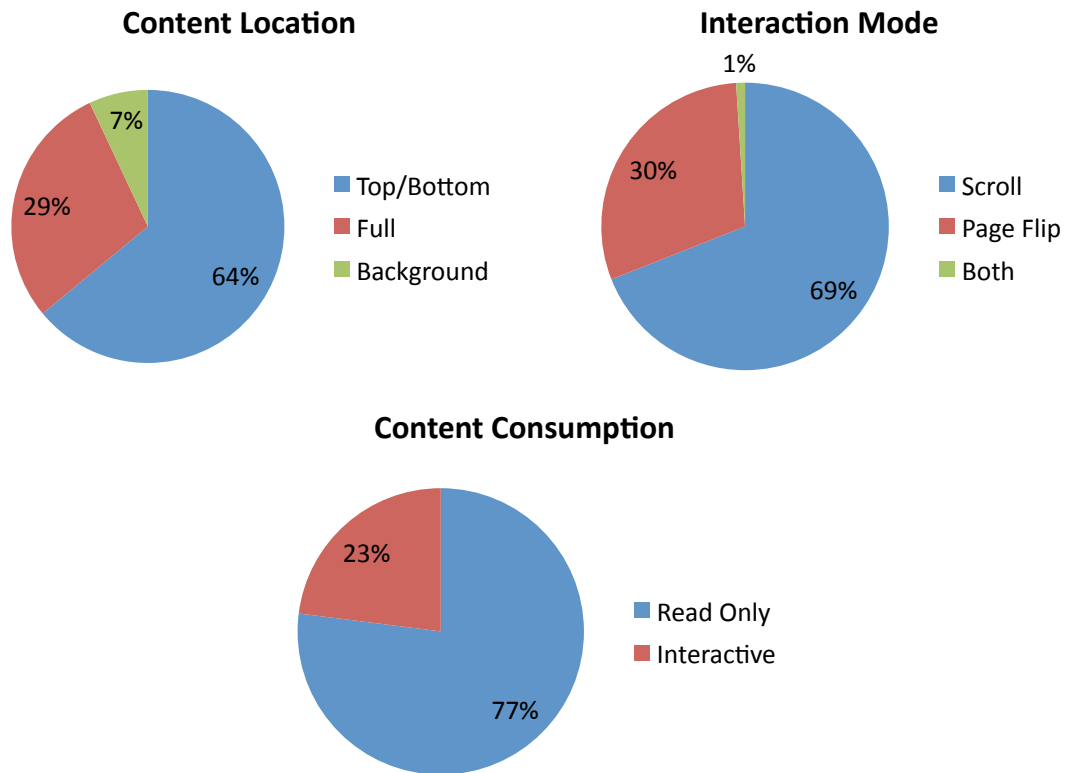
Figure 4.4: Analysis of mobile applications. Majority of the applications have new content at the top/bottom (64%), using scrolling (69%) and are read only (77%)

4.5.1 Simple Region Of Interest Model

Based on the our analysis, we developed a simple Region-Of-Interest (ROI) model that divides the screen into regions that we can apply the dimming accordingly. It is seen that the new content is generally located in the top and bottom of screen and a large portion of the apps uses scrolling that moves the older content into these top or bottom regions with large portion of the applications are content-consumption based rather than content-creation.

From this knowledge, we divide the screen into four main areas see in Figure 4.5. A dimming region at the top, clear region, dimming region at the bottom and a black plateau region. All four regions are adjustable and/or can be removed based on the app's specified ROI regions. The regions are description as follows:

1. Top Gradient Area. The top gradient region dims from 90% dim to 0% from the top to the bottom edge of this region. This region covers areas of screen
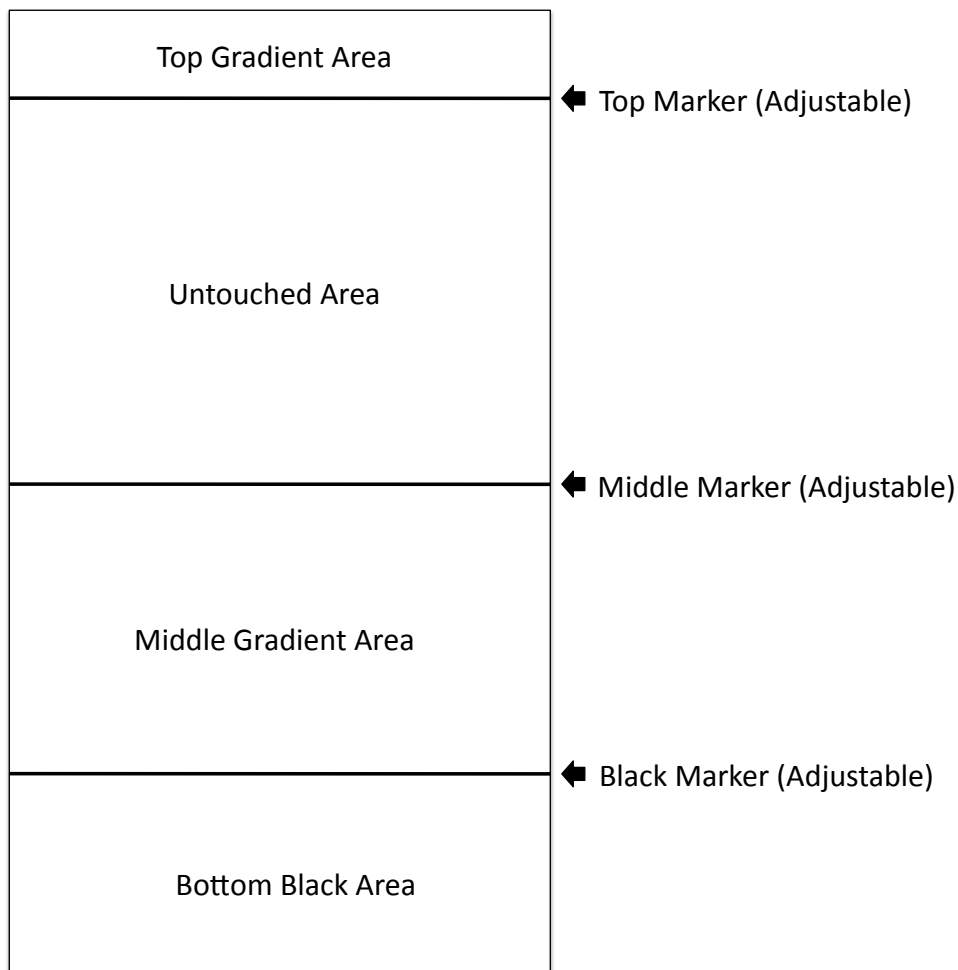
Figure 4.5: The Dimming Regions. *Top Gradient Area* and *Middle Gradient Area* dims their area from brightness to 90% dim where *Bottom Black Area* turns of the pixel in this area and *Untouched Area* remains clear

that usually does not have any information such as logo bars, dividing bars, etc or status indicators. Having this region gives us the flexibility to increase the dimming to the top part of the screen. The gradual dimming (gradient) provides a means to enable dimming while at the same time still allow the user to observe the content instead of a blanket uniform dimming

2. Untouched Area/Clear Region. The clear region does not have any dimming modification and remains clear to the user.

3. Middle Gradient Area. Similar to the dimming region at the top, it dims from 0% to 90% dimmed following the Clear region. It gradually dimmed

(gradient) the screen such that the user is still able to observe the content until the region edge.

4. Bottom Black Area. The black plateau region turns off all pixel in this region and no content is shown. This region provides the maximum power savings.

To illustrate how this simple model translates into dimming filters that is being deployed, we show two examples. First example, we wish to create a dim filter that only dims 50% of the bottom screen and leaves the 50% untouched. To do so, we adjust the Middle Marker to the mid point and remove the Top Gradient Area and Bottom Black Area and we will get the desired effect shown in Figure 4.6.

Original | 50-50 Profile Top and Bottom Region is removed | Outcome

Figure 4.6: Applying dimming filter with 50% Untouched + 50% Middle Gradient

In the second example, if we are to be more aggressive in our power saving, we can include all four areas into the dim filter. This will include the Top Gradient Area, Untouched Area, Middle Gradient Area and the Black Area. This will result in the shown outcome in Figure 4.7 where the bottom area is now black (pixels turn off) with also the Top Gradient Area applied to the top of the screen.

| Original | All four areas included | Outcome |
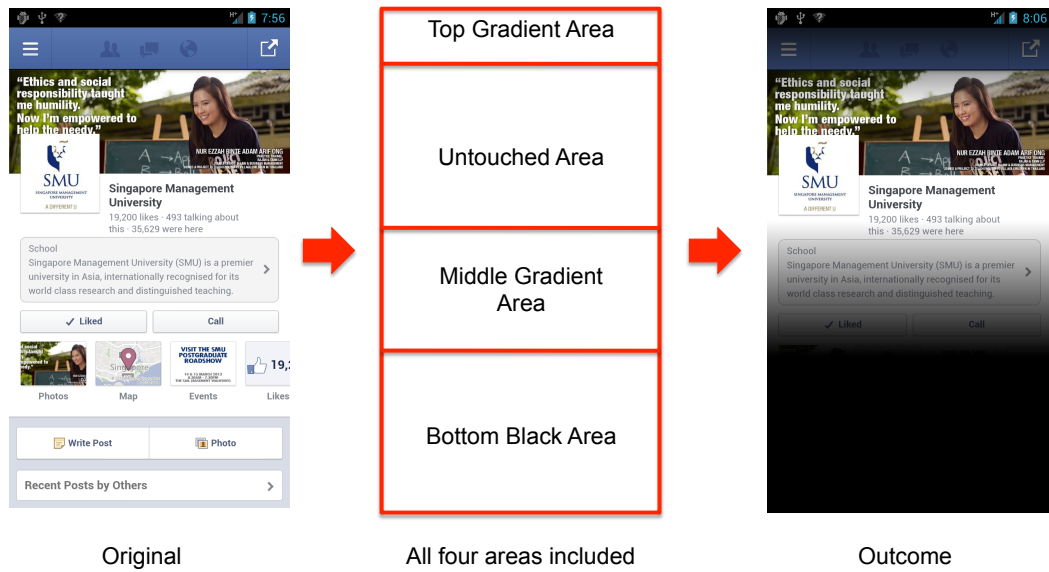
Figure 4.7: Applying dimming filter with custom effect with all four regions included

4.6 Proposed Architecture

The design first addresses generalisability and low computational requirements, to effectively achieve this requirements, the implementation has to be incorporated within the OS stack of the mobile device. This will allow an application-agnostic solution as well as keeping computation low and simple without the need to change each and individual application.

The design is broken down into the following: Profile Generation Module and Deployment Modification as shown in the following Figure 4.8. These two modules are kept separate such that the profiles can be easily generated on any platform. The Profile Generation Module is lightweight operation that can be easily be deployed as an simple application in both desktop and mobile hardware. This allows the Deployment Module flexibility to be deployed as either a kernel level solution or application-level solution.
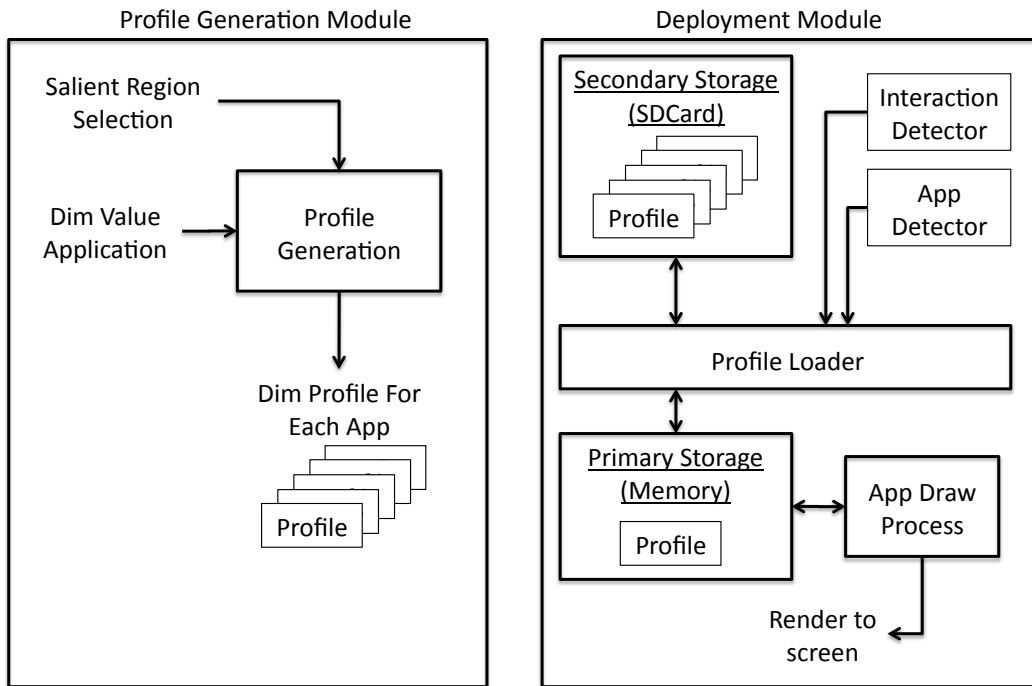
Figure 4.8: Profile Generation and Deployment Modules

4.6.1 The Profile

In order to understand the two modules, we need to review how the four areas under our ROI model is organised into a single entity known as a profile. Each profile contains the definitions of each of the four areas. Once the ROI model is defined for the app to be dimmed, we translate each of the areas and organised them into a profile for this particular app.

Using this profile with the simple ROI model approach, we can now easily build any profile for any application. Hence, this allows us to build a power saving solution while maintaining a usable screen for the user based on the saliency characteristic of each application.

4.6.2 Profile Generation Module

The profile generation module seen in Figure 4.8 is responsible for creating a dimming profile for each application. To setup a profile, the user would use this Profile Generation module to determine which area of the screen would be salient and

28

which area of the screen would be dimmed. As discussed in the previous section, the four regions would be determine and set by the user for each application.

The user does not need to understand the low level semantics with only need to adjust the markers (Figure 4.5). Once the markers are set and the appropriate dim value is determine, the Profile Generation module would generate all the necessary dimming boxes that is needed by these 4 dimming regions. This design is highly flexible as the dimming boxes are flexible to implement different kinds of dimming patterns hence, the system design is flexible enough to accommodate future changes.

Using the Profile Generation Module, the system is capable of adapting the simple ROI model into any application, this allow the user or the developer to quickly customised a profile for any application. As the design is build on a need-only basis, only applications that as an associated profile will be dimmed, applications without will not be modified. The mechanics of how this achieved is explained in the next section.

4.6.3 Deployment Module

In our design, the deployment module is developed and implemented as a kernel-level solution (within the OS stack). The profiles that are generated by the Profile Generation Module is first transferred and stored within the secondary storage on the mobile device (usually the SDCard medium). This allows us to reduce the active memory usage by loading only the profiles that are needed when dimming is required. Individual profile is then loaded into active memory if the app detector has detect if the app has been loaded. In addition, an interaction detector is used to detect user interaction to switch on and off dimming to reduce the impact on usability.

Once the conditions have been satisfied, the profile is loaded into memory. As the solution implements within the OS stack, we implement the dimming routines

within the App draw process, our dimming profiles (within the memory) is applied to the screen. Hence, if the system does not detect a profile is available, no profile will be loaded and no dimming will be applied, allowing the user to use the mobile device normally.

In addition, we further improve the usability by implementing a "flip" feature by pressing a hardware button, the dimming "flips", i.e., the bottom portion is clear and the top portion is dimmed. (The topmost status bar dimming never flips). This approach handles the majority of applications that have new content on either the top or bottom portions of the screen. In addition, the flip button makes it possible to view content anywhere on the screen — albeit with more effort and usability impact seen in Figure 4.9.

Dim Filter Applied Normally   Dim Filter "Flipped"

Figure 4.9: Using the "Flip" Feature for Dimming. Once the physical volume buttons are depressed, volume increase flips the dimming filter and volume decrease flips back to original

To ensure that scrolling is still easy, removes all dimming the moment the screen

is touched (so that all content and scrollbars become instantly visible). Moreover, to allow easy entering of user input, does not dim any of the virtual keyboards shown in Figure 4.10.

| Dim Filter Applied | Dim Filter Removed During Screen Interaction (Touch) | Dim Filter Removed When Keyboard Is Active |
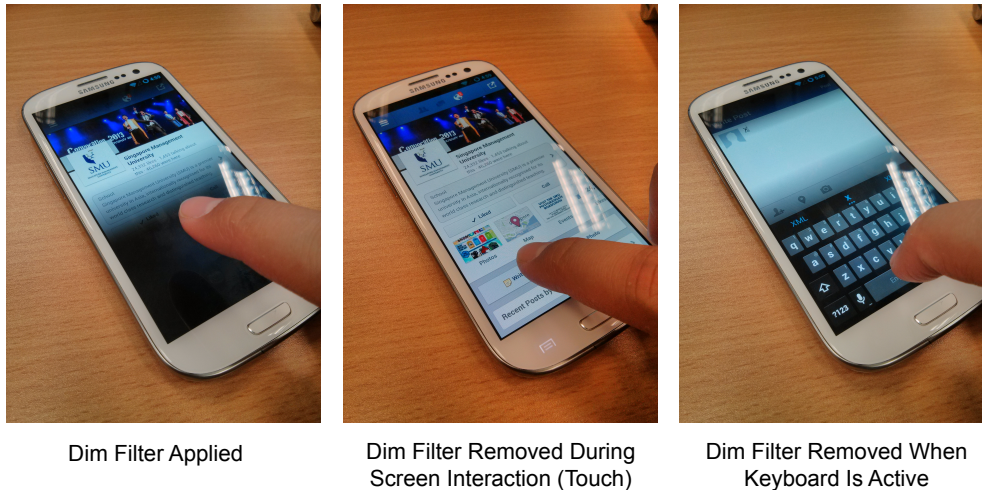
Figure 4.10: Conditions where dimming is not applied

Using this design, we able to build an application agnostic solution where we now affect all mobile applications using the implementation in the OS stack. In addition, by using the profile approach and checking the availability of the profiles during application start, we reduce unnecessary computation such that only applications with profiles are modified.

Chapter 5

Implementation

In this chapter, the implementation of the design for both Profile Generation Module and Deployment Module are described. In addition, the chapter provides additional description of how the various design requirements are achieved in the implementation phase.

5.1 Profile Generation Module

5.1.1 Default vs. App-Specific Profiles

The dimming profile determines the pattern of darkening that is applied to the display. At the high level, each profile is organised into four different portions of the screen (as discussed in Section 4.5.1):

1. Top Gradient Area.

2. Untouched Area.

3. Middle Gradient Area.

4. Bottom Black Area.

However, since the Untouched Area or Clear Region does not have any dimming, only 3 areas need to be considered for dimming or turn off: emphTop &

Middle Gradient Area and *Bottom Black Area*. This provides both flexibility and efficiency in handling and loading of the profile during runtime. For the two dimming regions and black plateau region, two values can be specified:

1. The size of the region (Defined via rectangle defined by its four coordinates).

2. The intensity level at the end of the dimming region, i.e., the alpha blending should blend from 100% intensity to x% intensity (where x is the value specified) at the edge of the dimming region. By default, this value is set to 90%. The black plateau region is set to 100%

However, due to limitation of the Android's Alpha-Blending API, the desired gradient effect (Dimming in gradual intensity) using the API is not achievable (no such effect available). Hence, in order to achieve the gradient effect, the Top & Middle Gradient Areas are decomposed into multiple smaller areas with increasing dimming intensity to create the desired effect.

Each dimming region is bounded by a rectangular box that defines the region that would be dimmed. Hence, by setting up each region carefully, the required gradient effect dimming pattern can be achieved, shown in Figure 5.1.

The computing of these dimming boxes is transparent to the user, once the region is defined, each of the dimming regions and black plateau region will be broken into a series of dimming boxes. This will be finally stored as a single profile for the application.

To handle the flipping feature (that flips the dimming profile to handle full screen content), an additional profile known as a mirror profile is computed. This mirror profile is simply an inverted dimming profile. During an activation of the volume button, either the increase or decrease volume button, the system will activate one of these two profiles which will be rendered onto the screen.

Touch events such as screen scrolling and onscreen keyboard disables the dimming. For any touch operation, dimming is disabled when any finger is on the screen and re-enable once the finger is removed. Similarly, the dimming is disabled
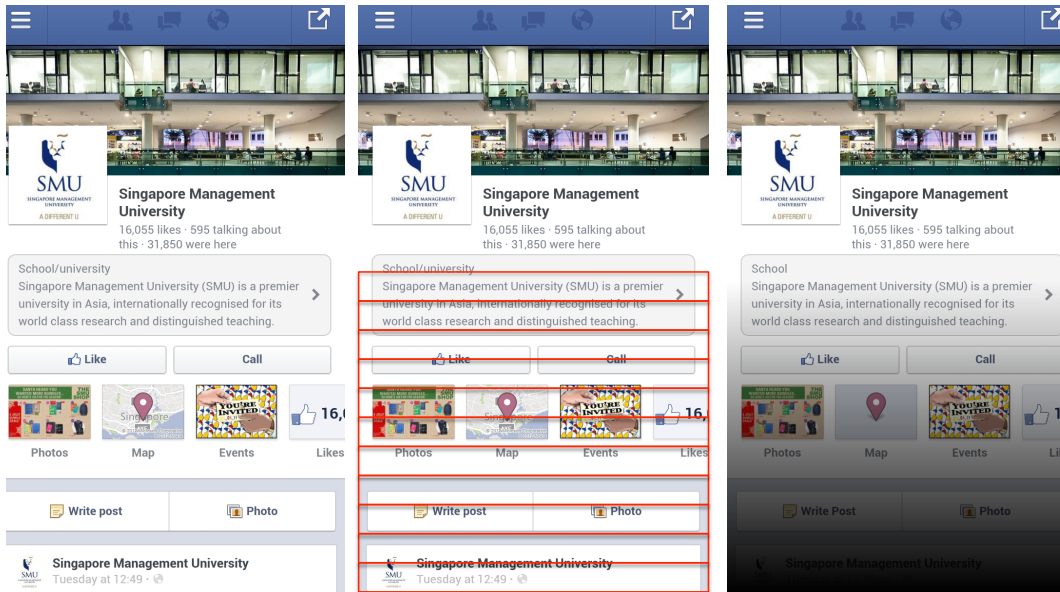
| Before | OLED Profile Containing The Dimming Regions | After |

Figure 5.1: Profiles containing the dimming regions. Each region has a different dimmed value thus creating the gradient effect

when the onscreen keyboard is enabled. This is a usability consideration, to reduce usability impact when the user is interacting with the phone.

Both flipping feature, touch events capturing and onscreen keyboard is reviewed in later Section 5.2.5 where event capturing is implemented within the OS stack.

5.1.2 Default Profile

The default profile is *application independent*, this meant that this is a standard profile with only dimming pattern that can be used across all applications. The dimming pattern is as defined as follows: The top dimming region size was set to 50% with no dimming. The size of the bottom-dimming region was set to 50% (i.e., half of the screen) and the plateau region was set to the bottom of the screen (i.e., there was no area of the visible screen that was turned off). Finally, the gradient of the bottom-dimming region was set to 10%. This profile leaves the top 50% of the screen unmodified while dimming the bottom 50% gradually from 100% intensity

to 10% intensity at the very bottom of the screen. The effect of this conservative profile is shown in Figure 5.2.

a) WhatsApp Messenger   (b) Adobe Reader   (c) Firefox browser

(d) BBC News   (e) Facebook   (f) Gmail
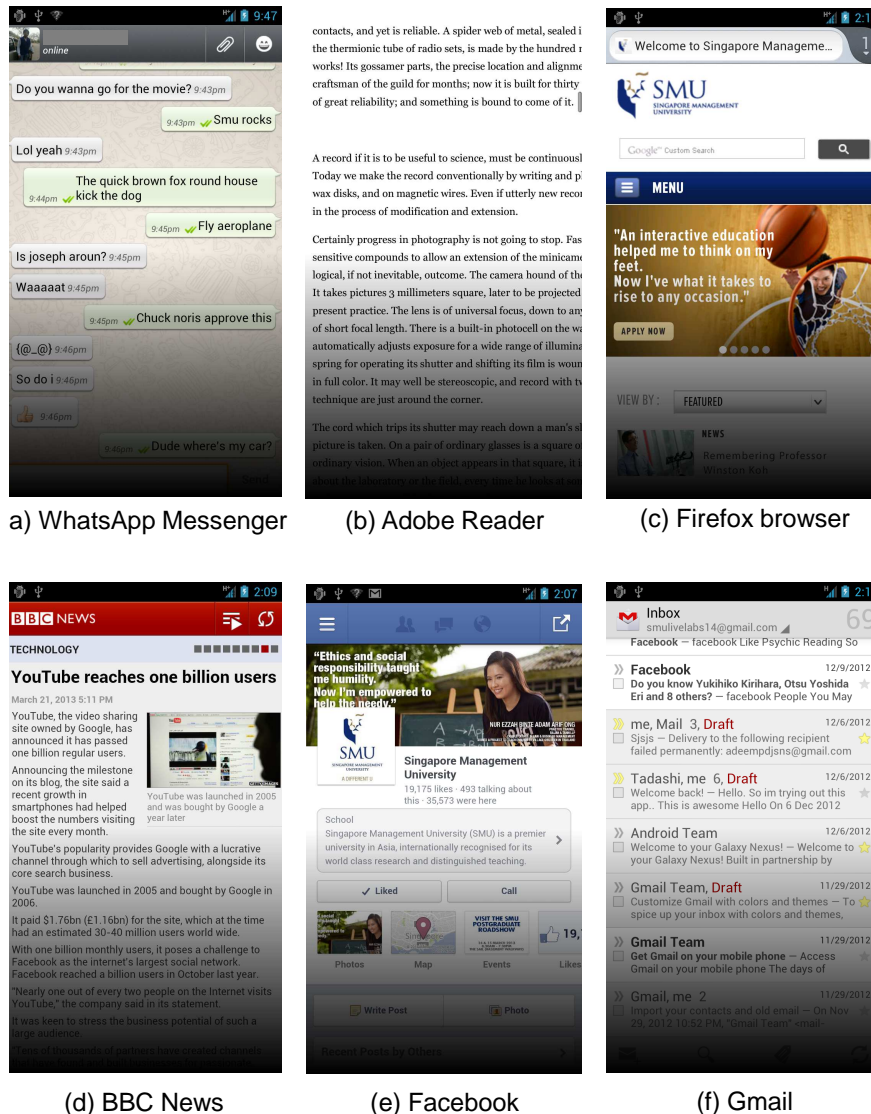
Figure 5.2: Default Profile (One Size Fits All). Applied to 6 different apps.

5.1.3   Application-Specific "Customised" Profiles

The default profile provides an one size fits all approach that quickly provides a quick dimming solution to any applications. However, in situations where default profile is not appropriate or sufficiently optimised, the system allows either the Application developer or the user to specify per-application dimming profiles. Such

per-application dimming profiles are creating by altering the default values of the *top dimming region*, the *middle clear region*, and the *bottom dimming region*, in terms of both the region size and the final intensity level. These application-specific or custom profiles aims to provide the maximum dimming effect to achieve power saving while at the same time reduce the usability impact to the user show in Figure 5.3

a) WhatsApp Messenger   (b) Adobe Reader   (c) Firefox browser

(d) BBC News   (e) Facebook   (f) Gmail
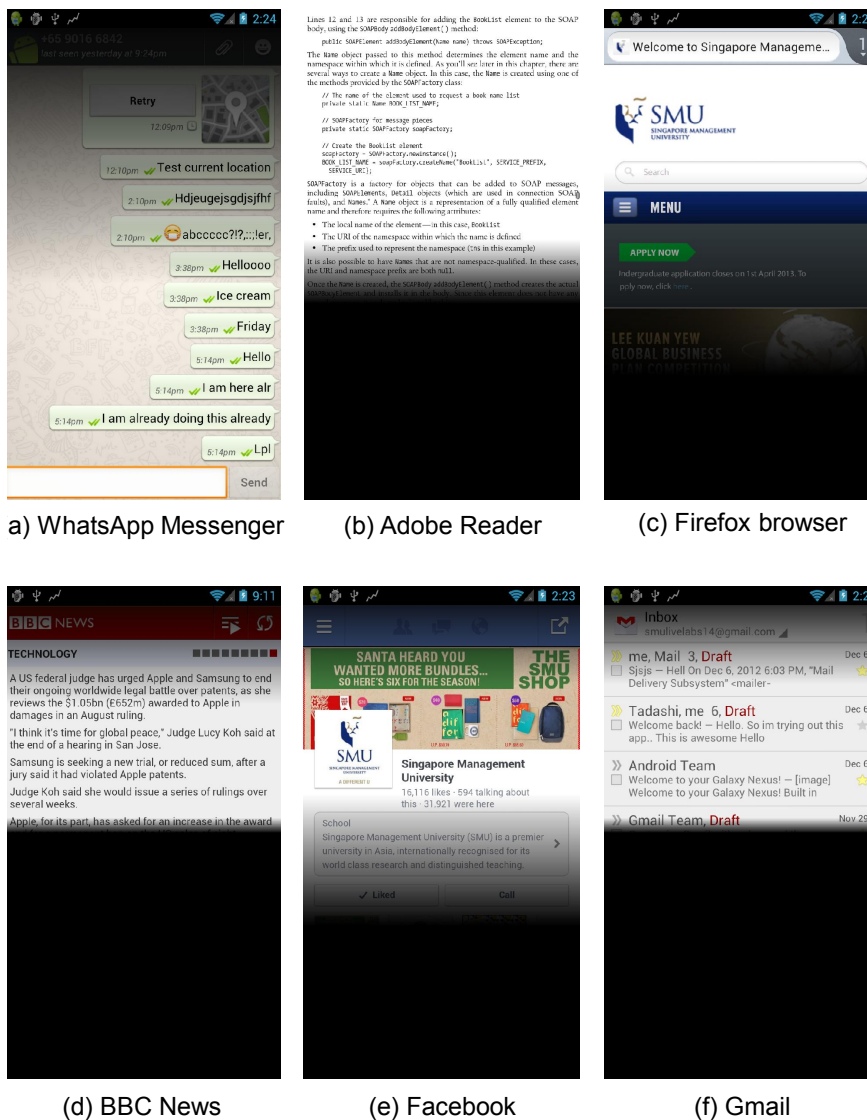
Figure 5.3: Customised Profiles deploy for different applications

5.1.4   Creating and Deploying the Profiles

Both Default and Custom dimming profile defines the four regions (Top Gradient, Untouched, Middle Gradient and Black) in terms of both the size of the region and the intensity value as explained in the previous sections. Once the dimming regions are defined, individual dimming boxes are computed, stored and transferred to the phones. There are two approach on how this is done:

Desktop Computer Tool

For the desktop computer, a simple Java-based tool was developed to allow users to create either the Default or Customised profiles for any application they choose. The interface of this tool is shown in Figure 5.4. This tool works on a desktop/laptop with Java runtime installed. The resulting profiles created are then transferred to the phone. The tool allows users to easily specify the three portions stated above for any application.

The users need only to adjust the three main sliders to mark the four regions, additional optional adjustments to the dimming effect are provided should the user wish to customised the gradient dimming. For every dimming setting, the tool shows the expected effect on the application and the expected power savings achievable (by interpolating a-priori power measurements of various settings for that applications). When the user is comfortable with their profile, they can click a button to generate and deploy the profile to their phone. Figure 5.3 illustrates the resulting application profiles that we (as researchers) created, for 3 out of the total set of 6 applications that we investigated.

Mobility Tool

For mobile platform, a similar tool was developed for the mobile device. The interface is shown in Figure 5.5. The user is presented with three markers (horizontal lines) in which they can adjust by moving the markers to their desired location to

| Dimming Region At The Top |
| Clear Region |
| Dimming Region At The Bottom |
| Plateau Region (Black Area) |

Step 1:
Define the Locus
and Dim Areas

Step 2:
Create Dimming
Profiles

Step 3:
Deploy on
Mobile Device
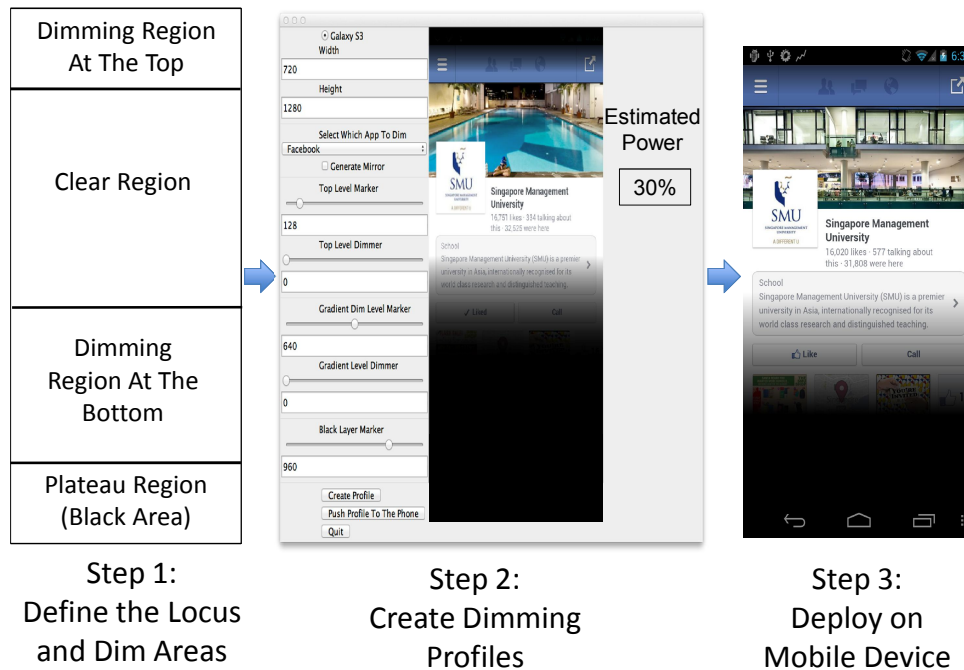
Figure 5.4: Applying application-specific profiles using the *Desktop Tool*

mark the Top Gradient, Untouched, Middle Gradient and Black area (users can also move the markers out of the screen to denote that that region is not utilised). Once the user is satisfied with the markers position, he can preview and deploy the profile to the appropriate application directly on the mobile device.

5.1.5 Profile File Format

In this section, we review the profile file format that hold the dimming boxes. Each profile file is stored as a plain text file (extension .txt) where the profile's filename is denoted by the application's name that the profile is meant for (e.g. A dimming profile for Facebook will be stored as com.facebook.katana.txt).

As described in previous section, once the user has defined the areas with either the desktop or mobile tool, these regions are further broken down into smaller dimming boxes to allow regions that have a gradient dimming effect, the code responsible for breaking down into the dimming boxes using the sliders is shown in Appendix A. Figure 5.6, shows the content of a Default (one size fits all) profile for Facebook.

| a) Dimming Selector | b) Adjust the markers | c) Preview and Deploy |
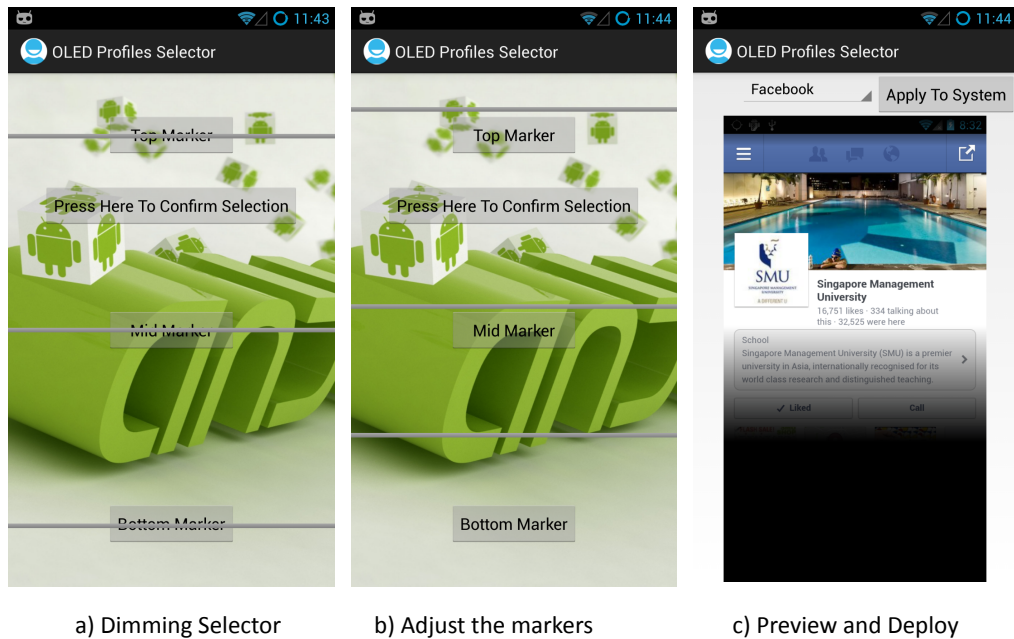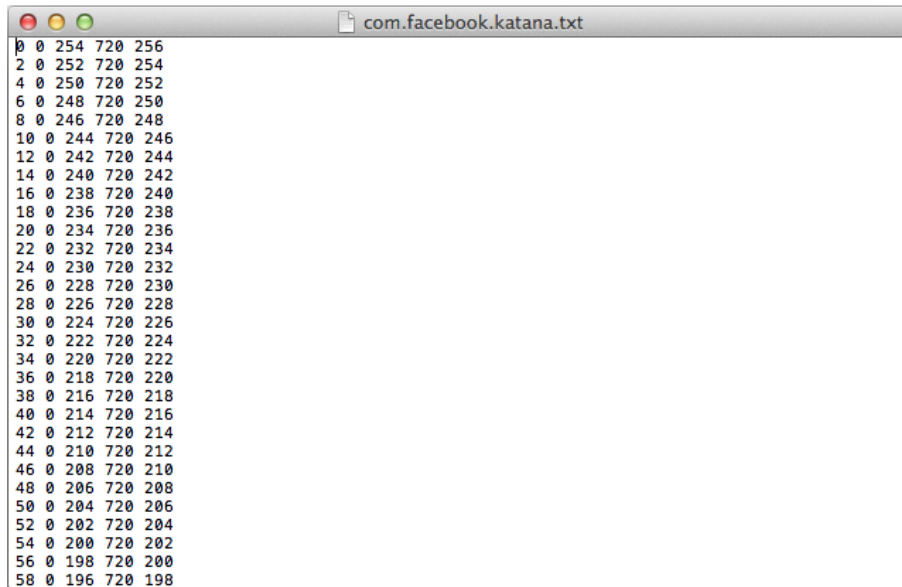
Figure 5.5: Applying application-specific profiles using the *Mobility Tool*

Each line within the file denotes a dimming box and is arranged in a CSV-like (comma-separated values) order. The dim value followed by the coordinates of the dimming box within the resolution of the display as follows: *Dim-Value Left Top Right Bottom*. The Dim-Value is restricted by the Alpha-blending API within the range of 0 to 255 (8 bits). Similarly, for the dimming box coordinates, the box is restricted by the display resolution (e.g. On Samsung Galaxy S3, the max resolution is 720x1280)

For the black plateau region, dim value of 255 with the dimming box will allow to generate the black region. Similarly, there is no need to define for clear region in the profile file. As seen in Figure 5.6, a series of dimming boxes with an increasing dim value allow us to generate the required gradient effect for the Top and Middle Gradient Area. Once the profile file is generated and deploy to the phone, it is then parsed by the Deployment Module line by line to apply the dimming effect on to the screen described in Section 5.2.4.

```
0 0 254 720 256
2 0 252 720 254
4 0 250 720 252
6 0 248 720 250
8 0 246 720 248
10 0 244 720 246
12 0 242 720 244
14 0 240 720 242
16 0 238 720 240
18 0 236 720 238
20 0 234 720 236
22 0 232 720 234
24 0 230 720 232
26 0 228 720 230
28 0 226 720 228
30 0 224 720 226
32 0 222 720 224
34 0 220 720 222
36 0 218 720 220
38 0 216 720 218
40 0 214 720 216
42 0 212 720 214
44 0 210 720 212
46 0 208 720 210
48 0 206 720 208
50 0 204 720 206
52 0 202 720 204
54 0 200 720 202
56 0 198 720 200
58 0 196 720 198
```

Figure 5.6: An example of a profile file for Facebook using Default Profile. Each line is denote by *Dim-Value Left Top Right Bottom* where the dim value is followed by the coordinates of the dimming box. Each line specify one box.

5.2 Deployment Module

In this section, the implementation detail of the Deployment Module is reviewed. In which, the Android primer is first reviewed to established how Android is organised internally before the kernel changes are reviewed.

5.2.1 Android Primer

The Android operating system from Google is an open source Linux based mobile operating system [21]. However, the Linux kernel is not directly exposed to the user or developer. Instead, Android provides abstraction to third party developers develop applications to execute on Android Runtime (virtual machine). Hence, instead of compiling applications into native environment, Android utilised the Dalvik virtual machine that works with byte code that is compiled from Java. The operating system (OS) stack is divided into several layers shown in Figure 5.7.

The application framework, core libraries are written in Java that provides the necessary APIs to build applications. These applications are then compiled into byte
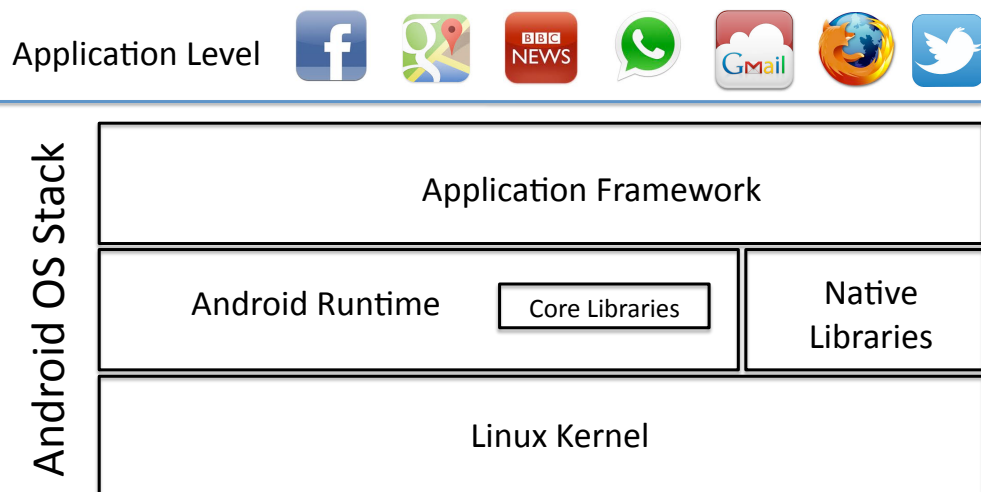
Figure 5.7: The Android Stack. Modifying the *Core Libraries* allow the dimming to be deployed as application agnostic solution

code and run within the Android Runtime that take advantage of the native libraries and Linux kernel to execute the applications. Hence, to achieve an application-agnostic design, our solution needs to be deployed within the Core Libraries, such that any applications that use these APIs will now include our solution.

5.2.2 Implementing Within The OS Stack

In order to apply the dimming to the screen as well as gathering inputs from the user, the implementation is done within the Core Libraries. Specifically, the required set of APIs are modified within the Core Libraries to allow a phone wide effect shown in Figure 5.7. This modified API allows access to both user key presses (to detect swipes, etc.) and the display framebuffer. This was a careful architectural choice and offers advantages over the two likely alternatives:

1. Implementing in the lower levels of the stack (such as the kernel video drivers): This would have provided more direct access to the image frame-buffer, to allow a more direct manipulation. However, this also meant additional processing is needed to interpret low-level pressure data with the touch sensors. Hence, although access to kernel video drivers provides access to

framebuffer it comes at the expense of semantically meaningful key presses (which would now be reported as raw pressure values), making it difficult to adjust the dimming function in response to user interactions.

2. Implementing in user space: This would require application recompiling of every application on the user's phone as this is the only method to inject OLED modifications into third party applications. However, this would limit generalisability for example, Android system applications cannot be easily recompile unless root access is guaranteed on user phones to extract the necessary applications. In addition, problems with recompiling with correct signatures as well as obfuscation also limit generalisability. Another problem is recompilation would incurred significant performance penalty as the OLED routines now have to be added within the third party applications.

Under the Android system, all widgets that is used in any Android applications are a direct or indirect subclass of the superclass *View*. The View class is a superclass that is part of the Core Libraries, this meant both interacting widgets like buttons and non interact widgets such as *Layout* classes (e.g. LinearLayout, RelativeLayout, etc) inherits from this superclass. In addition, the View superclass processes touch events such as onClick that is feed by Android system services.

In Android rendering process, the widget draw process is based on a parent-child relationship. Hence, all View objects (including direct or indirect subclass) are directly chain together under this draw tree. Thus any widgets declared within the draw tree will always have a root View object. This *View* API design simplifies our modification, thus we only require to modify a singular class (*View*) to create all the necessary effect.

However, modification in this superclass can result in performance problem due to the subclasses that inherits from it. As all of the user interface widgets inherit (directly or indirectly) from this superclass, any improper modification might result multiple executions of the same code causing performance penalty. Hence,

our modification is done such that we only apply the dimming on selected widget components and we only apply when the profile is available for the application.

5.2.3 Profile Check and Preload

The first step in the Deployment Module is to check for available profile and load the profile into memory. As discussed, we carefully modify the View class constructor such that we do not execute unnecessary code in the subclass. To do this, we utilise the draw tree to avoid any redundant computation that impede performance. Any widgets that is part of the draw tree will first check the root View before any processing, once any processing is completed, the results are stored only in root View of the draw tree.

In the implementation, during the first instantiation of the View object within the constructor, the process will call our *readInDimArea()* method, see Figure 5.8, that will first check if the profile file is available. If not available, no further processing will be done, the process execute as per normal.

```
/**
 * Simple constructor to use when creating a view from code.
 *
 * @param context The Context the view is running in, through which it can
 *        access the current theme, resources, etc.
 */
public View(Context context) {
    mContext = context;
    mResources = context != null ? context.getResources() : null;
    mViewFlags = SOUND_EFFECTS_ENABLED | HAPTIC_FEEDBACK_ENABLED;
    // Set some flags defaults
    mPrivateFlags2 =
            (LAYOUT_DIRECTION_DEFAULT << PFLAG2_LAYOUT_DIRECTION_MASK_SHIFT) |
            (TEXT_DIRECTION_DEFAULT << PFLAG2_TEXT_DIRECTION_MASK_SHIFT) |
            (PFLAG2_TEXT_DIRECTION_RESOLVED_DEFAULT) |
            (TEXT_ALIGNMENT_DEFAULT << PFLAG2_TEXT_ALIGNMENT_MASK_SHIFT) |
            (PFLAG2_TEXT_ALIGNMENT_RESOLVED_DEFAULT) |
            (IMPORTANT_FOR_ACCESSIBILITY_DEFAULT << PFLAG2_IMPORTANT_FOR_ACCESSIBILITY_SHIFT);
    mTouchSlop = ViewConfiguration.get(context).getScaledTouchSlop();
    setOverScrollMode(OVER_SCROLL_IF_CONTENT_SCROLLS);
    mUserPaddingStart = UNDEFINED_PADDING;
    mUserPaddingEnd = UNDEFINED_PADDING;

    readInDimArea(context.getPackageName());
    readInMirrorDimArea(context.getPackageName());
}
```

Figure 5.8: Modifying the View constructor to check and load in the profile data. This is to avoid performance penalty due to re-execution of the code in the sub-classes as well as avoiding expensive file operation. The method readInMirrorDimArea(), is the same operation for loading mirror profiles for dimming flipping.

If the profile file is available, it will then check if the root View already contains

the data. If available, no further processing is done, otherwise it will read from file and the stored the data into active memory. Please refer to the Appendix B.1 for full implementation. This avoid unnecessary code re-execution in the draw tree and avoid expensive file read operation to load the profile data into active memory.

5.2.4   Extending the Android Draw Process
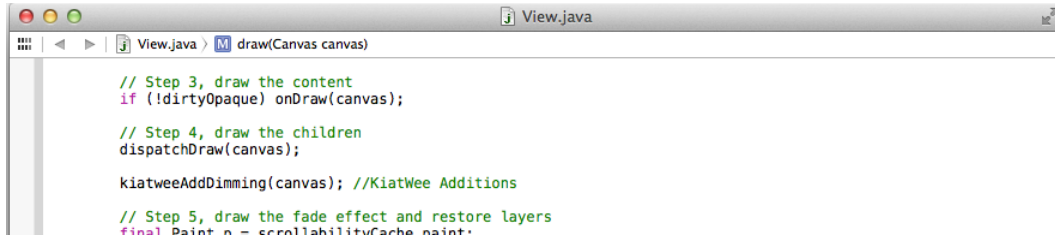
Our solution applies the dimming on any applications by extending the Android draw process that renders application content onto the screen. Standard Android applications that use widgets to draw content on the screen are arranged internally in a parent-child relationship with each widget being a child of the master "View object" [20] structure as a tree.

At runtime, Android draws all the widgets in this tree onto the screen, in a recursive way, starting from the parent. Once all the children for the entire rendering tree have been added to the framebuffer, Android goes ahead and renders the entire framebuffer to the display.

As all user interface widgets are direct or indirect subclasses from the View superclass, we can easily modify the *draw(Canvas canvas)* method and implement our dimming routines. During a draw operation in the draw tree, the root View will execute its own draw method. Once its draw operation is completed, it dispatches an event to the next widget in the parent-child chain in the draw tree. It repeats for the next widget in a recursive manner until all widget is processed.

Our modification takes advantage of this draw tree by only executing our dimming at the end of all draw process. In addition, we only execute on the main widget that utilise the whole display resolution, this only allow us to apply the dimming to the final content once. Our method, *kiatweeAddDimming(canvas)* is added after the *dispatchDraw(canvas)* so that we only execute after the next widget's draw process has been executed seen in Figure 5.9.

As the draw process is a recursive operation, each widget will execute its own *ki-*

```
● ● ●                          [j] View.java                                    ⤢
|||  ◄  ▶  | [j] View.java › [M] draw(Canvas canvas)

        // Step 3, draw the content
        if (!dirtyOpaque) onDraw(canvas);

        // Step 4, draw the children
        dispatchDraw(canvas);

        kiatweeAddDimming(canvas); //KiatWee Additions

        // Step 5, draw the fade effect and restore layers
        final Paint p = scrollabilityCache.paint;
```

Figure 5.9: Modifying the Draw method to implement the dimming profile. Dimming method, *kiatweeAddDimming(canvas)*, is applied after the dispatchDraw such that the child widget is first rendered before dimming is applied.

atweeAddDimming(canvas) method after its child's draw operation has completed. Hence, our method will only execute on the main widget that is higher up in the draw tree that utilise the whole display resolution. This main widget may not necessary be the root View as different developers creates their UI differently. This effectively only apply the dimming once and prevents re-execution of the code in draw tree.

In implementation, *kiatweeAddDimming(canvas)* method will first check for the scroll flag, if it is turn on, no dimming will be applied. If not, it continues to check if the user has initiated to flip the dimming. This allow us to apply the correct dimming filter (the original or mirror profile) later on. Second, we obtain the widget resolution to determine if this widget utilise the whole screen. If if does, we proceed to dim the screen by applying the dimming boxes sequentially (more details in the next Section 5.2.4. The method utilise different configuration parameters for different machine, shown in Appendix B.2, this set of parameters is meant for Samsung Galaxy S3.

Utilising Alpha Blending

The system applies a *dimming profile* to the final root display object using the well-known alpha blending technique, which uses a special colour channel to gradually adjust the opacity and translucency of the individual pixels of the screen image. In our solution, black colour is used to blend with the displayed content (black content turns off pixels on OLED).

45

There are existing efficient Alpha-Blending APIs within the Android Core Libraries, the API requires the region to be blended as well as the opacity/translucency value to be set and the API performs the blending. However, as discussed earlier, these APIs is limited in its functionality as the gradient effect is not available. Hence, to achieve the desired effect, dimming regions are furtherer divided into dimming boxes to achieve various dimming effects.

In implementation, we utilised Alpha-Blending APIs under the *Paint* class in the *android.graphics* package. Using this Paint class, we first set Paint colour to black. Next, since our profile data is already prepared in the following format *Dim-Value Left Top Right Bottom* (discussed in Section 5.1.5), we can easily set the Alpha channel with the Dim-Value with the canvas colour to transparent and the proceed to draw the rectangle (dimming box) onto the canvas with the provided coordinates, see in Figure 5.10. The dimming is applied after all dimming boxes in memory has been parsed and draw onto the screen creating all the necessary effect. For full implementation see Appendix B.2.2.


```java
if((viewHolder.getWidth()==viewHolderProtraintWidth && viewHolder.getHeight()==viewHolderProtraintHeight) ||
    (viewHolder.getWidth()==viewHolderLandscapeWidth && viewHolder.getHeight()==viewHolderLandscapeHeight))
{
    //Step 1: If rect holder is null means no dim for this widget, return.
    if(kwD_rectHolder==null)
        return;

    //Step 2: Check if View is a valid view covering the whole screen.
    Paint paint = new Paint();
    paint.setColor(Color.BLACK);

    //Step 3: Apply Dim area
    canvas.drawColor(Color.TRANSPARENT);

    int len=kwD_rectHolder.size();

    for(int i=0;i<len;i++)
    {
        paint.setAlpha(kwD_alphaHolder.get(i)); // trasparenza

        canvas.drawRect(kwD_rectHolder.get(i), paint);
    }

    //kwsavecanvas=canvas.save();
    //drawnBeforeFlag=true;
}
```

Figure 5.10: Part of the kiatweeAddDimming() that shows the Alpha-blending process. Each dimming box is sequentially applied to the canvas with the specified alpha-blending values.

5.2.5 Capturing Events

In addition to extend the draw process, we instrument various key system events in order capture and to aid in the dimming procedure. These includes:

1. Touch Event Dispatch. This is instrumented to give us the event when a touch event has been fired, see Figure 5.11, for implementation refer to Appendix B.3.1. This aid us to switch on or off the dimming filter, this is to reduce the usability impact when the user is interacting with the content such as scrolling. We keep scroll flag (boolean value) to indicate if the user is touching the screen. This is later used draw process to apply the profile or not (in previous Section 5.2.4).

2. Key Dispatch. Key dispatch event inform us when a physical button on the device is being utilised, see Figure 5.12, for implementation refer to Appendix B.3.2.This instrumentation allow hooks to be placed with the even volume buttons, this fires the event for the mirror profile to be applied so as to "flip" the dimming filter and vice versa. Similar to the touch event, we use flip flag (boolean value) that is also use in the the draw process to determine if the mirror profile should be use (in previous Section 5.2.4).

5.3 Putting it all Together

To illustrate how the system work, an use case with the process flow is shown in Figure 5.13 and Figure 5.14. This show how the system works with all the pieces connected.

If a user that is using the proposed mobile system wishes to use Facebook, BBC News, Gmail and WhatsApp even when the user's mobile device is battery constraint. The user would first utilised either desktop or mobile tool to a series of four profiles for this four applications and deploy these four profiles into the mobile

Figure 5.11: Part of the *dispatchTouchEvent()* method that shows instrumentation that captures and touch interaction and store it under scroll flag (kiatweeOnScrollFlag) within the root View.

device. Using the tool and adjusting the markers, the user can define a dimming profile that it acceptable, save and deploy to the mobile device's SDCard.

The user then proceeds to use this mobile device normally and reaches a point where the mobile device enters a battery constraint condition (As an example, less than 10% battery life). If the user still wishes to use one of the four applications, for example, Facebook then during the Start Up process, the system will now check for a profile (either a "Customised" or "Default" profile).

If available, the system applies the profile using the draw process and applies the dimming filter using the alpha-blending and renders the output to the display. If not, it leaves the application unmodified.

Figure 5.12: Part of the *dispatchKeyEvent()* method that shows instrumentation that captures and volume button interaction and store it under flip flag (kaitweeOnView-PortFlipFlag) within the root View.

As the dimming is applied, display power is reduced allowing more extended phone life, allowing the user to continue to use the applications compared to when the user continue to use the screen without dimming.

In critical battery situations like this, we believe that users will be willing to lose some of their screen real estate if it meant that a) they could still use their applications effectively, and b) it extends their battery life by about 20% longer. In future versions, the system could query a web service for an appropriate profile (the profile could also be downloaded when installing the application).
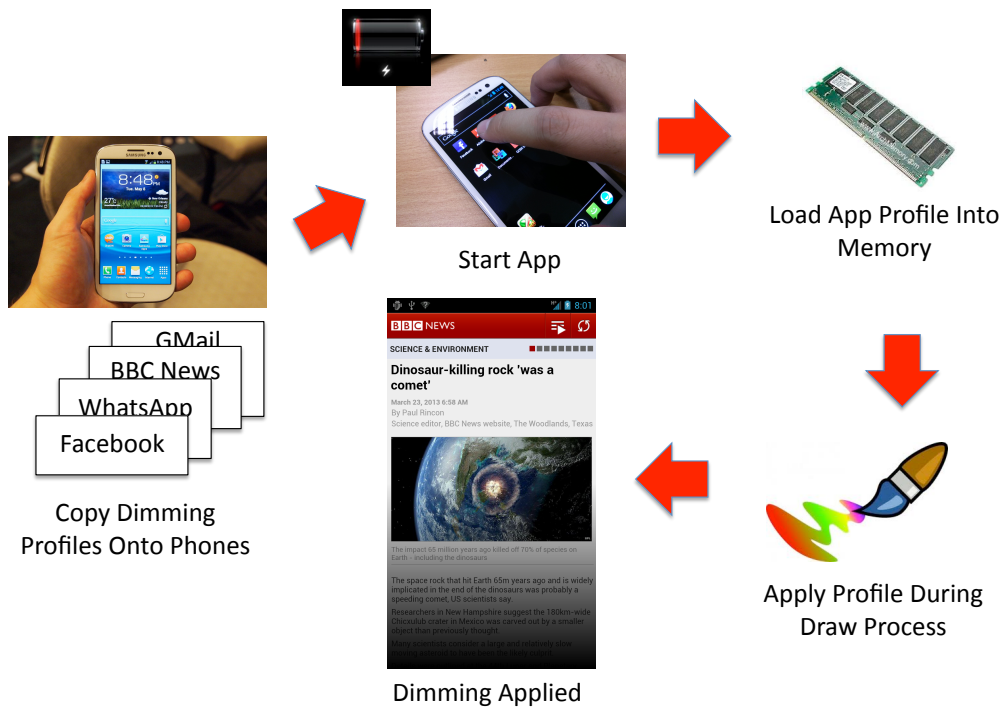
Figure 5.13: How the dimming is applied during battery constraint scenario. Dimming is applied only to applications with profiles during low battery condition
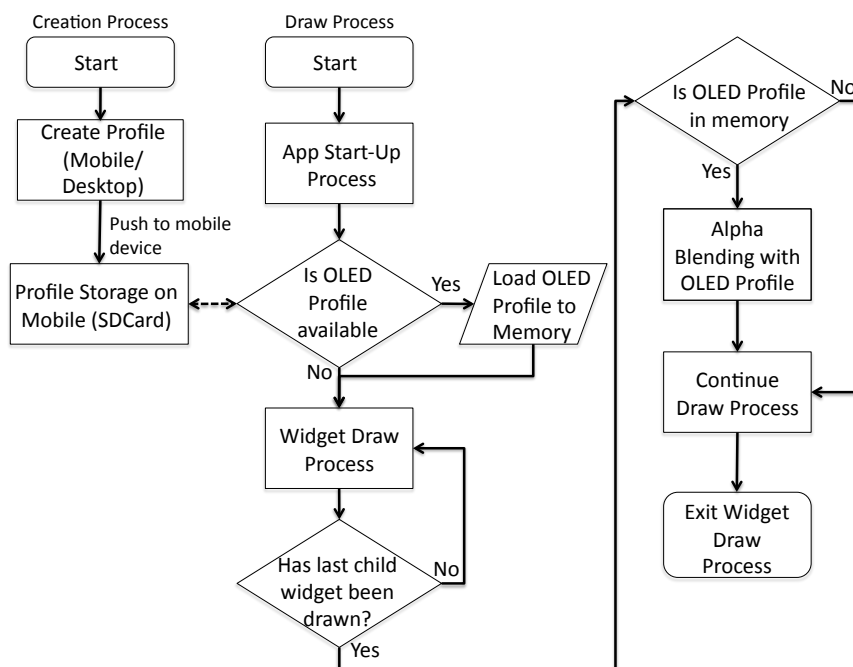
Figure 5.14: Process flow chart of creating, deploying and runtime operation of dimming

Chapter 6

Performance Evaluation

In this chapter, an quantitative evaluation was conducted on the system where detailed power performance of the system was analysed. In which, power savings was determined for no technique, default and custom profiles.

6.1 Evaluation Methodology

In this experiment, we utilise the Samsung Galaxy S III (GS3) Smartphone, which is a popular OLED based mobile device as our test device and for the user study. The GS3 uses a 4.8 inch HD Super-AMOLED (1280x720 resolution) display. The modifications implemented on the View class is subsequently build on Android version 4.2.2 using CyanogenMod i9300 variant. CyanogenMod is an open source operating system that is based on the Android platform, it is chosen for its readily available online resource and comprehensive support from the CyanogenMod community [11].

6.1.1 Applications Selection

In our evaluation for both power savings and usability study, we evaluated using 15 popular applications from the Android app store (Google Play). The Android app store covers 26 categories (games has a separated category) from books, busi-

ness, comics, communications to launcher widgets. In our work, we selected the
top applications from the top categories that the users would normally use on their
mobile device. We cover large popular categories such as books, business, com-
munication, finance, media & video, news, productivity, shopping and social. The
applications from these categories are shown in Table 6.2, that adequately cover the
entire application space.

6.1.2 Power Measurements

We used the Monsoon external hardware power monitor [32] to collect accurate
power measurements. The Monsoon power monitor show in Figure 6.1 replaces
the battery as the primary power source of the mobile device. Hence, we are able
to determine how much power the mobile device draws from the Monsoon power
monitor during normal operation as well as during dimming operations.
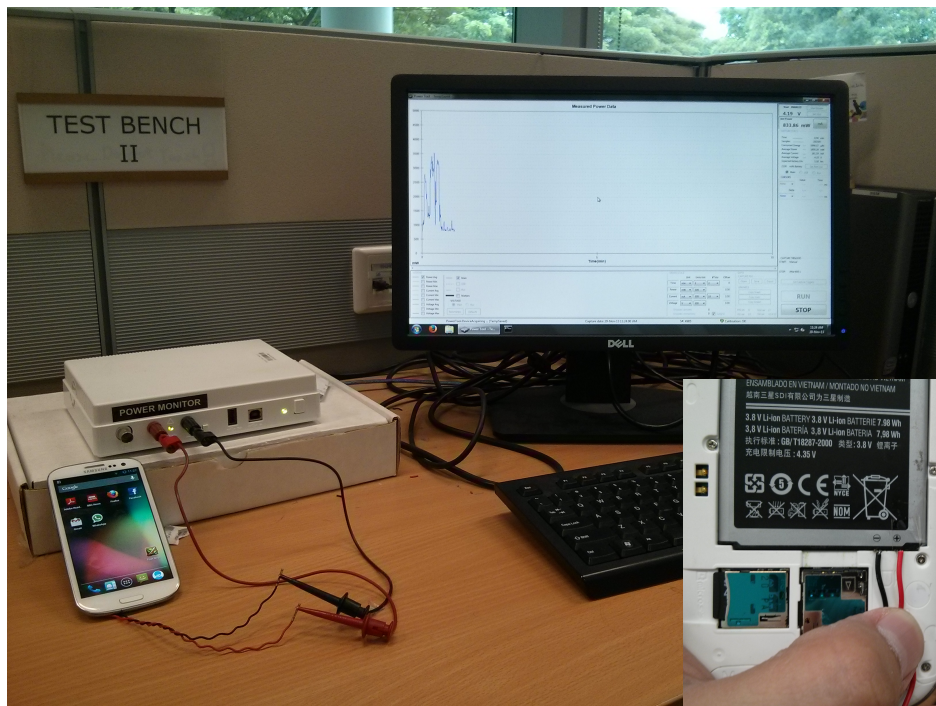
Figure 6.1: Monsoon Power Monitor measuring power consumption on the Sam-
sung Galaxy S3. The Monsoon Power Monitor replaces the battery as a power
source allowing us to measure consumed power

As the power drained measurement is for the whole phone, to determine display measurement, we have to first measure a baseline reading that is without any technique for each application. We then subsequently compare subsequent measurement relative to this baseline. Due to this nature, we have to keep the measuring time consistent as well as ensuring that no other applications or processes running. In the following result sections, we provide more details on their respective methodology.

Name	Category	One Minute Continuous Usage Scenario	Completing A Specific Task Scenario
Aldiko Book Reader	Books	Page 14 of a eBook ("Oliver Twist")	Flip to 1st page and read the 1st page
Documents To Go 3.0	Business	Viewing 4 sample files provided by the app (2 Excel, 1 Word, 1 Powerpoint)	Open a specific word document and read to specified point
Gmail	Communication	Inbox listing	Browse content of a selected email
Firefox Browser	Communication	Main pages of Google, Yahoo SG, Wikipedia, Amazon US, and Endgadget	Use Google, search: cnn and browse to www.cnn.com
WhatsApp Messenger	Communication	Chat screen with keyboard active	Browse to specific thread and post "test" reply
OCBC Bank	Finance	Current balance screen	Browse and read transaction history
YouTube	Media and Video	Portrait mode playback of "Gangnam Style" video	Portrait mode playback of "Gangnam Style" video
BBC News	News	Article view page of 5 long lived articles	Read top three news articles completely
Adobe Reader	Productivity	Page 10 of a pdf file ("Java Cryptography")	Read default document to specified point
Dropbox	Productivity	File listing page	Browse contents of specific folder
ES File Explorer	Productivity	File listing page	Browse folder contents of directory on sdcard
Calendar	Productivity	Weekly schedule page	Browse event calendar for a specific day
eBay	Shopping	Individual Item description page	Browse printer section to a particular printer sale
Facebook	Social	Timeline of the 5 most popular Facebook pages	Read top 15 postings of a specific user
Twitter	Social	Timeline screen	Browse top 15 tweets of a specific user

Table 6.1: For each application, we conducted our continuous power measurements using the scenario(s) in the "One Minute Continuous Usage Scenario" column and our task completion measurements using the scenario(s) in the "Specific Task Scenario" column

Table 6.2: Applications Used For This Evaluation

6.2 Power Savings

The first evaluation was to identify how much OLED display power the system could save. We conducted two main sets of experiments.

6.2.1 Measurement Details

In the first set of experiments, we used the generic "Default" profile (that gradually dimmed only the bottom half of the screen) for all the applications listed in Table 6.2.

For each application, we ran the application without the system for one minute and with the system for one minute. In both cases, we ensured that no other applications or processes were running (the phone was rooted and we manually killed all background processes etc.). For each test, we repeated for at least 3 times to ensure that the readings we obtain are consistent and the average is taken. For readings that are not consistent (e.g. Power spikes due to Android system processes) are dropped and test is repeated.

To ensure repeatability across experiments, we did the following: for each application, we left the application on the "main" page. In particular, this was the page which was the most frequently accessed page when the application was used normally. For example, for the *Aldiko Book Reader* and *Adobe Reader*, this was the page that showed the contents of the book or pdf being read (and not the main menu page).

In the second set of experiments, we used a per-application "Customised" profile, which we created to save as much power as possible while still preserving (from our perceptual standpoint) application usability. These profiles could thus be thought off as a form of usable "upper bound" to demonstrate the potential of the system.

In both experiments, measured power is compared against the baseline (one minute without the system). This gave us the power savings across the phone re-

sulting from the display changes.

6.2.2   Results: The System Saves Significant Display Power

Table 6.4 shows the average power consumption when running each application without the dimming ("Unmodified"), with the "Default" profile, and with the "Customised" profile. The table also shows the % improvement in power consumption between the "Default" profile and the unmodified version ("(2) over (1)" in the table), between the "Customised" profile and the unmodified version ("(3) over (1)" in the table), and between the "Customised" and "Default" profiles ("(3) over (2)" in the table). We show both the % improvement and the absolute difference in % savings (in brackets) for this last case.

In terms of power savings for the "Default" profiles or "Custom" profiles are not consistent, this is mainly due to how the application is developed. For example, applications that are mainly dominated by white colour theme such as Adobe reader, book readers, etc usually have better power savings when dimming is applied compared to darker theme applications.

In addition, the results show that the "Default" profile's power savings ranged from 1.5% (*WhatsApp*) to 32.5% (*Twitter*). The poor performance of *WhatsApp* arose because the "Default" profile only dims the bottom half of the screen. However, the *WhatsApp* test scenario has the virtual keyboard active on most of the bottom half of the screen, rendering this system (which does not dim the keyboard) largely ineffective. However, even with *WhatsApp*'s poor performance, on average, the "Default" profile saves about 23% of the OLED display power. This translates to savings of anywhere between 15% to 20% or more of the total power consumption of the phone when the screen is on (depending on which other components are also active).

Application	One Minute Continuous Usage Scenario						Specific Task Scenario			
	Power Consumption (mW)			% Improvement			% Improvement			
	Base (1)	"Default" (2)	"Customised" (3)	(2) over (1)	(3) over (1)	(3) over (2) (% & Diff.)	"Default" Time	"Default" Energy	"Customised" Time	"Customised" Energy
Aldiko Book Reader	1952.30	1337.65	1236.27	31.48	36.68	16.52 (5.20)	-0.37	23.21	0.49	29.06
Documents To Go 3.0	1620.04	1357.11	1267.67	16.23	21.75	34.01 (5.52)	-0.17	11.79	0.18	7.23
Gmail	1707.48	1243.77	1006.59	27.16	41.05	51.14 (13.89)	0.90	19.34	-10.09	16.62
Firefox Browser	1703.89	1255.24	1047.32	26.33	38.53	46.33 (12.20)	1.09	10.22	-12.75	3.39
WhatsApp Messenger	1237.10	1218.18	952.98	1.53	22.97	1401.31 (21.44)	-1.10	21.82	-1.45	15.99
OCBC Banking	1696.96	1249.17	1036.19	26.39	38.94	47.56 (12.55)	-0.59	20.98	-0.75	29.31
YouTube	1452.80	1113.60	787.30	23.35	45.81	96.18 (22.46)	0.01	27.88	0.02	36.81
BBC News	1550.99	1118.97	881.51	27.85	43.11	54.79 (15.26)	4.77	27.36	5.09	42.37
Adobe Reader	1923.19	1437.42	1261.15	25.26	34.42	36.26 (9.16)	2.58	5.31	4.95	7.82
Dropbox	1921.80	1358.99	1284.40	29.29	33.17	13.25 (3.88)	-1.57	15.79	0.46	14.23
ES File Explorer	889.71	790.46	768.43	11.16	13.63	22.13 (2.47)	-0.07	2.78	0.44	12.10
Calendar	1520.55	1149.23	1092.58	24.42	28.15	15.27 (3.73)	-0.16	22.02	0.68	24.71
eBay	1766.78	1259.86	1238.60	28.69	29.90	4.22 (1.21)	0.26	8.02	-0.14	4.62
Facebook	1557.35	1288.49	1041.78	17.26	33.11	91.83 (15.85)	0.28	14.47	-1.01	8.93
Twitter	1823.38	1230.84	1020.18	32.50	44.05	35.54 (11.55)	0.73	31.57	-0.33	38.71
Average	—	—	—	23.26	33.68	—	0.44	17.50	-0.95	19.46

Table 6.3: The table shows the average values across all test scenarios for each application (Table 6.2). The bracket values for the "(% & Diff.)" column are the % savings difference between the "Customised" and "Default" profiles for the continuous measurement scenario. For the task scenario, all values are the % improvement over the base case (without dimming running) — negative values indicating an increase in the time or power consumption.

Table 6.4: Measurement Results

57

The "Customised" profiles obtained higher power savings. In particular, *What-sApp* increased from 1.5% to almost 23% as the *WhatsApp* "Customised" profile, shown in Figure 5.3, darkens areas of less interest to the user that are not covered by the virtual keyboard. On average, the "Customised" profile saved about 34% of the OLED display power (with a maximum savings of about 45% for *YouTube*). This translates to saving 20% to 30% of the phone's total power consumption.

6.3 Task Completion Time

In the second evaluation, we identify how much impact to the user task time when the dimming is applied. We conducted two main sets of experiments as well.

6.3.1 Measurement Details

In first set of experiment, we utilise the "Default" profile and perform a task on this application across one minute. Each task is unique and typical to the application. For example, using Facebook, we read the top 15 postings in the timeline. We measured the both the time and energy (mJ) needed to perform this task. In which, we repeat the task again for "Default" and "Custom" profile, for the Facebook example, we read the posting in the clear region and subsequently move the unread stuff into this clear region. We repeat each task for at least 3 times to make sure the results are consistent (without any Android system processes interferences), results that are inconsistent are discarded and test is repeated.

Table 6.2 lists the "One Minute Continuous Usage Scenario" we used for each application. Note: for four applications, *Documents to Go 3.0*, *Firefox Browser*, *BBC News*, and *Facebook*, the content displayed could change dramatically and this affected the power savings. Hence, for these four applications, we used multiple compelling test scenarios as described in Table 6.2. The measurement results shown for these four applications are the average values across all the different test scenarios for that application. We did not use different test scenarios for the other dynamic-content applications applications such as *eBay*, *Adobe Reader*, and *Twit-*

ter as our testing showed no significant power variation even across different pages containing fairly standard amounts of text.

6.3.2   Results: But What About The Time To Finish Tasks?

For the results shown above, the applications were used continuously for 1 minute each. It is, however, important to ascertain if such dimming impacts *task effectiveness*, i.e., will the partial dimming of the screen result in a much higher task-completion time (as the user has to scroll unnecessarily to find things), thereby increasing the overall *energy* consumption?

To evaluate this, we tested all 15 applications with a series of realistic tasks (described in the "Completing a Specific Task Scenario" column in Table 6.2) without dimming and with both the "Default" and "Customised" profiles. For each application, we measured the energy consumption (in mJ) as well as the time to completion of each task.

The 'Specific Task Scenario' column of Table 6.4 shows the results of this experiment as a % improvement over the same task being run for the same application without the system active. (negative numbers indicate the time or energy consumed increased by that % value).

Our experiment revealed that the task completion time increased by 0.44% and -0.95%, on average, when using the "Default" and "Customised" profiles respectively. Even in the cases when it increased or decreased task completion times (e.g., an increase of 12.75% for *Firefox* and 10% for *Gmail*, a decrease of 4.77% for *BBC News*), the absolute time increase was at most 1 to 2 seconds only and within the error margin.

In our experiment, we notice that due to the fact that we do not dim the screen when the user is interacting with the screen, this indirectly aid the task. For example, reading when scrolling up the content helps to reduce time spent in reading. In addition, there are less scrolling action as we do not dim the screen when the user is interacting, hence, content is usually move into the clear region once without any

correction.

This suggests that the system has minimal impact on task completion times. However, even with this minimal impact, the "Default" and "Customised" profiles still saved, on average, 17.50% and 19.52% of the overall display energy consumption respectively. Coupled with the higher energy savings achievable when using applications continuously, we posit that using system regularly will achieve significant energy savings with minimal impact on task completion times.

6.4 Comparison with other techniques

As stated earlier, there are other techniques such as colour remapping and other ways of dimming the screen that can, in principle achieve the same results as this system. In this section, we discuss how these other approaches compare with this system. All of these techniques are orthogonal to and can be used concurrently — with each technique benefitting from the strengths of the other. However, we do not evaluate concurrent usage in this thesis.

6.4.1 Colour Mapping

We were unable to directly compare with colour remapping systems such as Chameleon [14] as these systems are extremely complicated and hard to replicate. For example, Chameleon was implemented and evaluated on just a single custom browser (Fennec browser on a Google Nexus One phone) that first study the power-colour relationship per device model before implementing colour mapping. Instead, we point out that their reported power savings of $\approx 40\%$ when browsing is a little higher (but albeit with a much more significant engineering effort).

6.4.2 Resolution Reduction and Uniform Dimming

Uniform dimming and resolution reduction are traditional methods in OLED dimming. Resolution reduction has two main forms, either a reduction in pixel resolution or spatial resolution. In this comparison, we looked at pixel resolution re-

duction. For uniform dimming, the whole display is dimmed uniformly across the screen. In fact, uniform dimming implementation can be found in almost all OLED smartphones (screen is dimmed when no interactivity is detected).

6.4.3 Measurement Details

However, it is possible to compare existing dimming techniques such as uniform dimming as well as resolution reduction. To compare against other dimming techniques, we choose to perform a static screen test. This test utilised capturing screen shots and replaying back the screen shots on the screen. We choose this approach as power management feature such as uniform dimming feature does not provide adequate controls during power measurement to allow un-bias power comparison. In addition, significant changes has to be implemented to allow resolution reduction to be deploy to the device.

We performed a task on 6 applications (Facebook, WhatsApp, BBC News, Gmail, Adobe Reader, Firefox), these applications are chosen as they are the top and popular applications on the app store and they cover across different categories.

We first captured screen shots on each key application screen for that task. We then applied the appropriate dimming routines to these screen shots. These routines consists of uniform dimming, resolution reduction, as well as our default and custom profiles seen in Figure 6.2.

We then replayed the screen shots on a real phone (to simulate the task being performed) and measured the power consumed by the display. Each screenshot is measured across one minute, repeated at least 3 times to ensure results are consistent. The power savings is then calculated by comparing against the baseline (original screenshot). This gave us an excellent idea of the power savings achievable by each dimming method across 6 applications. However, without a real implementation and user study, we make no claims as to the usability of these approaches.

The results are shown in Table 6.5 for different dimming percentages. We observe that the "Default" profile has comparable power savings to either a 20% re-

(a) Original   (b) Default Profile   (c) Custom Profile

(d) Uniform Dimming (50%)   (e) Resolution Reduction (50%)
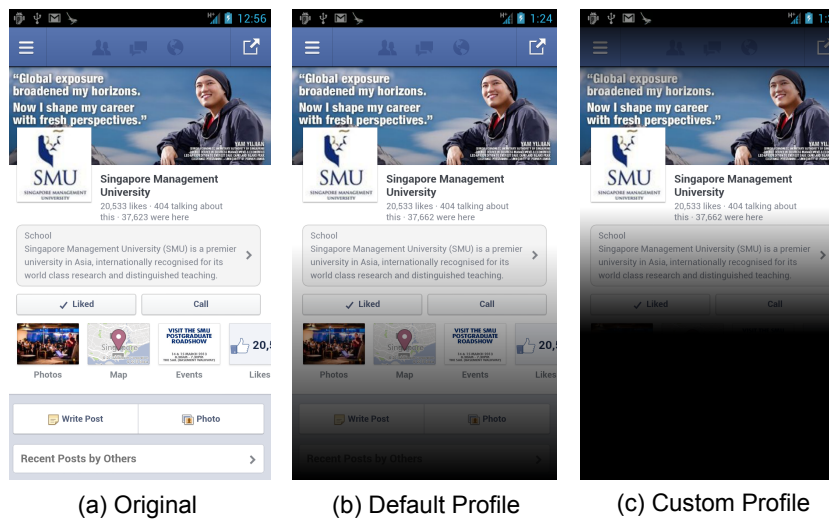
Figure 6.2: Uniform Dimming & Resolution Reduction (Unused areas are set to black) compared to the originals and our Default and Custom profiles

duction in either resolution or screen brightness while the "Customised" profile is comparable to a 30% resolution or screen brightness reduction. We suspect that both the resolution and screen brightness reduction methods will impact usability far more than this system as they do not differentiate between important and less important areas of the screen. Hence, key content could be too dim for the user to see (using uniform dimming) or too small for the user to interact with (using resolution reduction). However, this validation is left as future work.

Application	Baseline (mW)	System		Uniform Dimming (% Reduction)					Resolution Reduction (% Reduction)				
		Default	Customised	10	20	30	40	50	10	20	30	40	50
Facebook	1627.94	27.85	42.05	15.67	27.72	37.42	44.51	51.06	13.66	23.53	32.25	40.59	46.79
WhatsApp Messenger	1682.60	25.31	18.46	14.17	26.36	36.71	45.16	51.53	14.71	24.62	33.43	41.54	48.54
BBC News	908.45	9.03	23.84	6.87	13.43	19.02	23.08	27.27	9.15	14.69	19.30	24.33	28.54
Gmail	1780.11	24.82	43.47	14.74	26.45	37.41	46.22	53.47	13.19	23.84	33.23	41.99	49.57
Adobe Reader	773.56	9.73	12.43	5.60	7.75	12.28	15.60	19.20	8.75	12.44	15.62	18.92	21.27
Firefox Browser	1534.48	23.02	29.70	12.31	23.97	33.83	41.81	49.40	11.81	21.79	30.23	38.57	45.43
Average Improvement	—	19.96	28.33	11.56	20.94	29.45	36.06	41.99	11.88	20.15	27.34	34.32	40.02

Table 6.5: Comparing the Power Savings with Other Dimming Methods. All values shown are the % improvement in Power consumption relative to the Baseline

Chapter 7

User Study

Having established that the system is indeed effective in saving display power, we now present the results of our qualitative evaluation through a user study designed to evaluate if the system is acceptable to end-users. The user study was designed to answer two questions:

1. **Are the "Default" and "Customised" profiles usable?**: We compared both profiles against the base application.

2. **Are Supplied Profiles Good Enough?**: The most useful systems tend to be the ones that requires no user intervention. As such, the second experiment investigated if the "Default" and "Customised" profiles created by the research team (and which could, in theory, be created easily by application developers) are good enough for end-users.

7.1 User's demographic profile

In total, we had 30 undergraduate participants (18 males and 12 females) from SMU's Information Systems school that is proficient with Smartphones. Figure 7.1 shows the break down of the demographics.

From the pre-test demographics survey, 96% of the participants recharged their phones at least once a day, 66% of the participants agreed or strongly agreed that they reduced their application usage to save power, and 63% of the participants agreed or strongly agreed that their phones did not have sufficient battery capacity

Types of Phones

2%
6%
8%
37%
47%

- iPhone
- Android
- Blackberry
- Windows Phone 7 or 8
- Other

Frequency of Recharging Phones

3%
27%
70%

- Multiple times every day
- Once a day
- A few times per week

Intentional Reducing App Usage

4 — Strongly disagree
3 — disagree
3 — neutral
11 — agree
9 — Strongly agree

Battery Life Is Insufficient

1 — Strongly disagree
5 — disagree
5 — neutral
8 — agree
11 — Strongly agree

Figure 7.1: Demographics of users

for their daily usage patterns. This shows a consistent user behaviour that users are willingly to perform some degree of tradeoff for longer battery life.

7.2 User Study Methodology

For the user study, we reduced the set of applications tested to just 6 for two experiments. This allowed us to obtain enough participant data points per application without requiring an extremely large user base. We chose 6 popular applications that collectively covered a very broad application range: *WhatsApp Messenger*, *Facebook*, *BBC News*, *Gmail*, *Adobe Reader*, and *Firefox Browser*.

To avoid fatigue, each participant tested just 4 applications. In total, each of the 6 applications was tested by 15 participants. The user study procedure was as follows: Each user was asked to answer a small pre-test demographics questionnaire and then perform two experiments. Each participant was paid SGD $10 ($\approx$USD $8) for taking part in the study.

7.3 Experiment 1: Is the system Usable?

This first experiment was designed to evaluate the usability of the system, with both the "Default" and a research team-supplied "Customised" profile. We asked each participant to first use the unmodified version of the application for 30-50 seconds to re-familiarise themselves with these popular applications, this step is necessary for users who have not use any of these applications before.

Each participant then used both modified versions of the application (running the system with either the "Default" or "Customised" profile) for a short period of time (about 2 to 3 minutes per modified application). Users are free to use the application in any way they please, we do not restrict the user on how they should use the application.

To minimise bias, we counter-balanced the order of using the modified versions among the participants, i.e., half the participants used the applications in the order {Unmodified, "Default", "Customised"}, whereas the other half used the order {Unmodified, "Customised", "Default"}.

Right after completing the use of each modified version, they had to answer the following two questions using a 5-point Likert scale:

- 5 — Strongly Agree

- 4 — Somewhat Agree

- 3 — Neutral

- 2 — Somewhat Disagree

- 1 — Strongly Disagree

Question 1 was *"This version of the application is as usable as the original version"* while Question 2 was *"The most important portions of the application are still visible"*.

We understand that the positive phrasing of the questions, coupled with a direct comparison instead of an objective score, could lead to experimental bias. However, because each participant was already very familiar with every test application (all 6 are very popular mobile applications) and because of our intentional bias against the system (see below), we believe that the results collected are still quite meaningful.

No specific training was provided to the participant for any part of this experiment. Note: to avoid further bias, we did not tell participants either that the system was for saving display power or the amount of power savings achieved. As such, even the participants who guessed that the system was power-related did not know the actual gains for either the "Default" or the "Customised" profiles. This was an intentional bias to avoid making the users more receptive to the system due to its significant power savings.

7.4   Result: The System is Perceived As Usable

Figure 7.2 shows the perceived usability of the system for both profiles and the 6 test applications. In the Figure, higher values are better (indicating that the users agree with the question "This version of the application is as usable as the original version"). We observe, that in every case, the "Default" profile achieved acceptable usability scores (every value is above the neutral 3 middle mark).

This is a very encouraging result as it suggests that the system is still usable even when saving significant display power (average 23% savings for this profile). However, the "Customised" profile that the research team provided was rated as below neutral for 4 of the 6 applications. This was because the "Customised" profiles turned off parts of the screen and the participants found this jarring — especially since they did not know the purpose and effectiveness of the systems.

Similar, for question, "The most important portions of the application are still visible", showed the same pattern shown in Figure 7.3. This shows the "Default" profile performs better than "Custom" with most of the important visual components
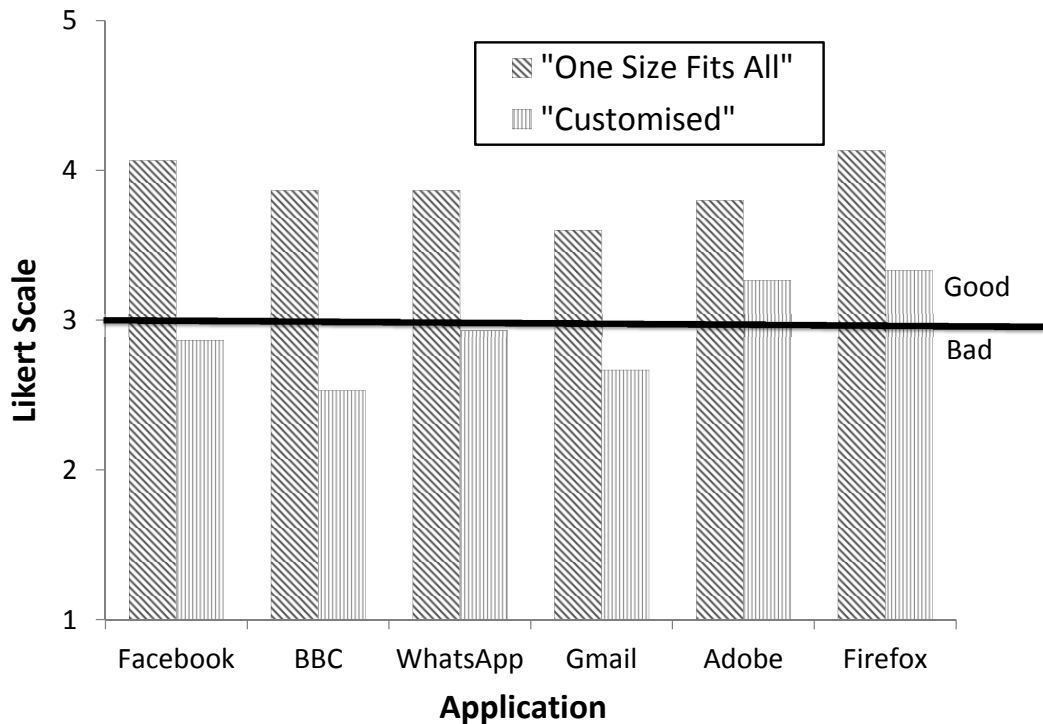
Figure 7.2: Perceived usability of the system. The self-reported scores for the question "This version of the application is as usable as the original version" using a 5 point Likert scale (5–Strongly Agree to 1–Strongly Disagree

still visible and usable to the user. While "Custom" profiles was designed by the research team with aim of balancing power savings with usability, the rating was lower not only due to the jarring interface (users did not know the purpose), the "Custom" profile may dim visual components that the user felt important.

However, as we shall see next, when the participants understood the purpose of the system and were able to pick their own "best" profiles, more than half of them picked profiles that were at least as aggressive as the default profile, while a handful picked profiles that were even more aggressive (in terms of darkening the screen) than our "Customised" profiles.

7.5 Experiment 2: Are Supplied Profiles Good Enough?

In the second experiment, our goal was to understand if the participants, after being told what the system did, could design their own profiles to achieve what they

Are the most important portions of the application still visible on the screen?
(higher better)
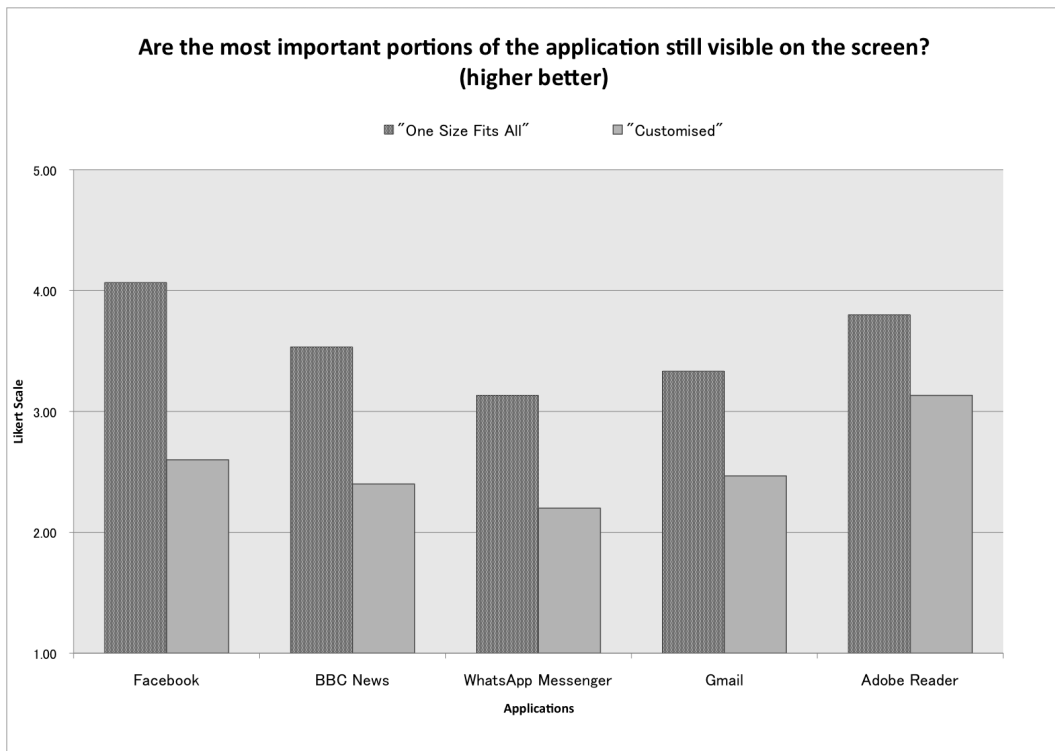
Figure 7.3: Perceived usability of the system. The self-reported scores for the question "The most important portions of the application are still visible" using a 5 point Likert scale (5–Strongly Agree to 1–Strongly Disagree

believe to be the best balance of power savings and usability. In particular, this experiment would allow us to understand if the participants would voluntarily pick settings for the system that either matched or exceeded either our "Default" or "Customised" profiles — instead of deciding that the most usable form of the system would be to turn off all the dimming settings.

For experiment 2, each participant was asked to customise the dimming profile for 3 applications. We only chose 3 applications as each participant was given 5 to 10 minutes per application to create and test their "best" dimming profile. We counterbalanced the order of the applications presented to each participants.

The experiment started by explaining the system power saving technique to the participants. Each participant was then given a PowerPoint slide deck, that included embedded Visual Basic Application (VBA) scripting, see Figure 7.4, allowing them to customise, via movable sliders, the three dimming areas for each application used in this experiment. The dimming areas were:

1. The size of the top dimming area

2. The size of the bottom dimming area

3. The plateau position in the bottom dimming area.
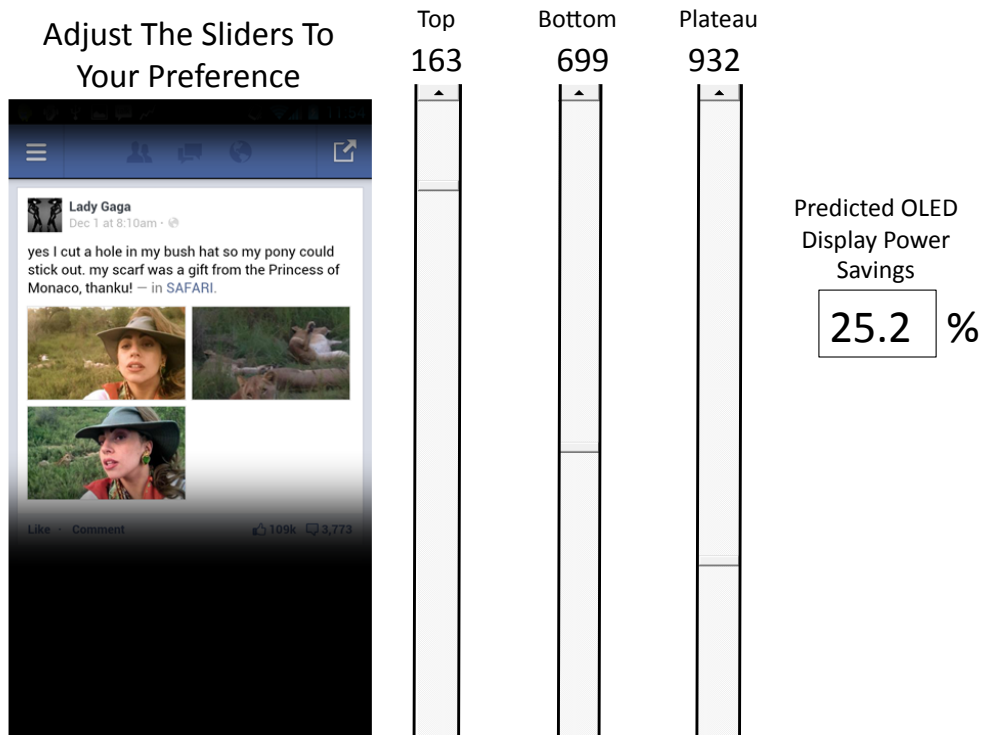
Figure 7.4: VBA Tool for user to control amount of dimming to be deployed

All user manipulations to the sliders immediately changed the image on the slide to show the resulting screen output, as well as provided an updated estimate for the resulting OLED display power savings. (These predictions were computed by interpolating real measurement studies done with various slider settings for each application.) When a participant wanted to test her PowerPoint settings, she could transfer and deploy it onto a Galaxy S III phone and test the application to see if the system settings were optimal for her. If not, the participant could iterate through the whole process until she obtained her "best" profile.

Application	Minimum (%)	Median (%)	Maximum (%)
Facebook	5.68	18.74	47.03
WhatsApp	0.38	16.67	29.39
BBC News	9.02	35.19	43.17
Gmail	1.87	25.54	45.06
Adobe Reader	0.00	32.05	54.67
Firefox	2.67	27.82	31.40
Average	3.27	26.00	41.79

Table 7.1: Power Savings Achievable by Participant Generated Profiles. The table shows the % improvement of the minimum (least aggressive), median, and maximum (most aggressive) participant generated profiles relative to the baseline unmodified power consumption values (using the values shown in Table 6.4) for the same application. We used the same testing procedure used to generate the "Continuous Usage Scenario" entries in Table 6.4. Note: for *Adobe Reader*, one participant (the min. value) decided that the best profile would be to turn off the systems.

7.6   Result: Users Can Pick Effective Profiles

In total, 15 profiles (each from a different user) were created for each of the 6 test applications. Table 7.1 shows the results of this experiment and we observe that at least half the users (looking at the "Median" values) picked profiles that achieved comparable power savings to our "Default" profile. In the best case ("Maximum" column), some users were even dimming more aggressively than our "Customised" profile!

Overall, the median % power improvement with the participant supplied profiles was similar to that achievable with the "Default" profile — indeed many of the user profiles looked very similar to the "Default" profile. This suggests that developers can supply profiles with their applications that are acceptable to many users. At the other extreme, the users who wanted extreme power savings ended up with profiles that looked and behaved (in terms of power savings) very similar to the research-team created "Customised" profiles, see Figure 7.5 — again supporting the claim that developer supplied profiles might be sufficient.

However, there were also some users who picked very conservative profiles.

We will need further studies to understand if this attitude changes under low power situations. Overall, this experiment suggests that developer-supplied profiles, for both the "Default" and extreme power savings "Customised" cases, might be good enough for the majority of users. This greatly reduces the system entry barrier as users need not supply their own specific profiles.

Facebook   BBC News   GMail
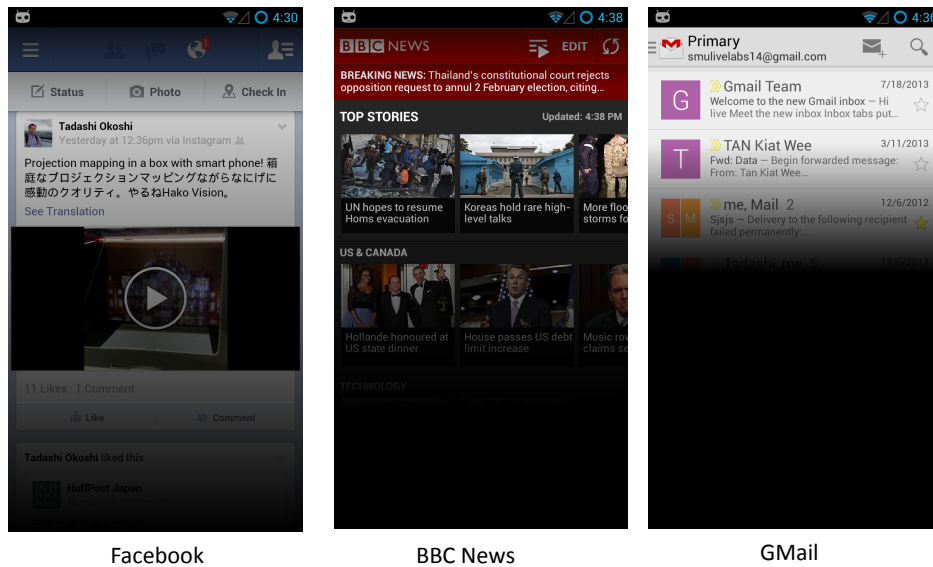
Figure 7.5: Aggressive profiles defined by users. Shown 3 examples of profiles defined by users which save more power than our Custom profiles.

7.7 Result: Feedback of Users

At the end of the user study, we solicited feedback from the user to enquire about the solution. Generally, users are willing to trade away some part of the screen estate (dimming) to save power, seen in Figure 7.6 with positive feedback, *"Very novel idea and I am sure people will find it extremely useful."* However, users provide feedback on the solution which can be generally categorised as follow:

Context and User Aware There are significant group of feedback that indicates that the solution must take into account of the context of the application. This meant that certain parts of the screen should not be dimmed due to important

visual information may be blocked. In addition, solution needs to take into account of how users read and use the application.

Due to the problem that each user and application is different this may be a hard problem to achieve even with significant machine learning with a human expert user (to make the final decision).

Examples of such feedback, *"it depends on the screen size and the type of application and the users. for reading apps, i might not be a fast reader with my eyes running in z-form and thus i can use this power saving method as i usually only need to focus on certain portion of the screen whereas other people may be more used to scheming the entire article"*

User Customisable Controls Another feedback wish the solution to have more user customisable controls. This includes a more flexible way of choosing regions other than the solution's 4 regions, user ability to adjust the amount of dimming on the fly as well as customisable to different applications and even widgets. This aspect of more controls can be easily be implemented should the solution be deployed. Example of such feedback, *"User should get to decide how much of the screen is dimmed down. Varies depending on which application. Should not apply to home screen. Can be used like a screensaver for apps that provide a feed."*

Negative Feedback There are also feedback on solution that this may not work in certain applications such as video applications (Youtube) which is correct as salient region in videos are hard to determine. There are also one feedback that they would rather use hardware solution to extend battery, *Refer not in this approach to improve the battery use life, rather than enhance the hardware or other technology to boost the battery usage life..*
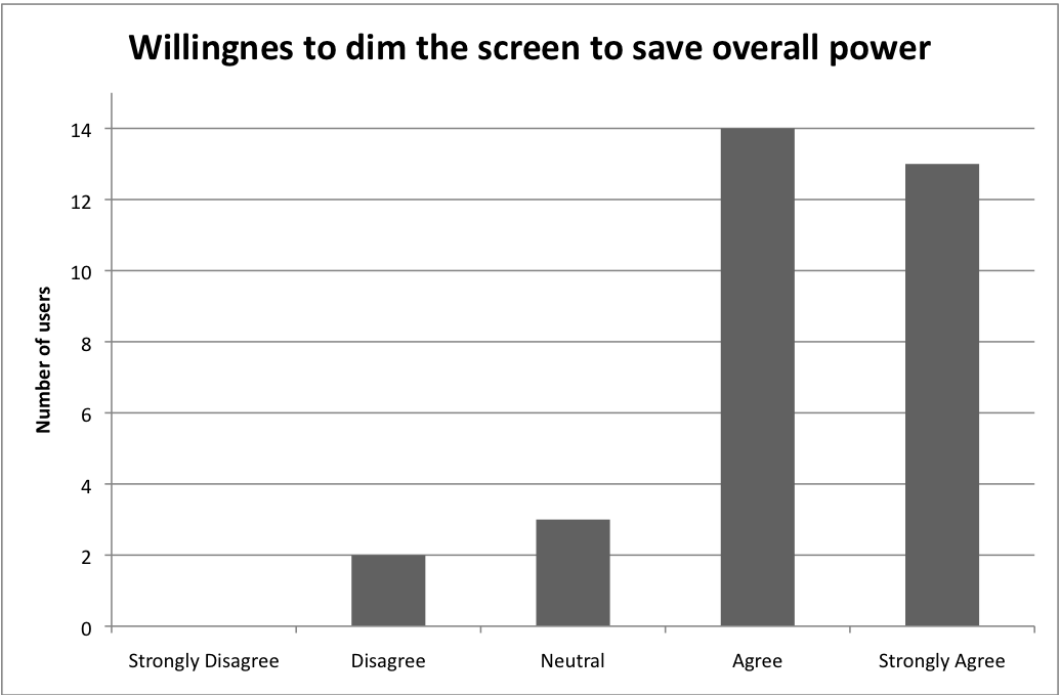
Figure 7.6: Willingness to dim the screen to save power.

Chapter 8

Supporting Games

In this work, we have seen the solution implemented on applications that utilised standard Android widgets that allow manipulation through the draw process. However, the other key application category that was not supported in this study is the game category. Although, the same strategy can be applied to games, it is much harder to support this class of applications as they have very different ROI.

In terms of the region of interest (ROI), games can be broadly classified into those that continuously divert user attention to random screen areas (most casual games (*Angry Birds, Cut The Rope*, etc.) do this) or those that focus attention to the middle of the screen (most first person shooting and driving games do this) see Figure 8.1.

Hence, the simple ROI model that we derived for top-down or bottom-up dimming approaches used for applications won't work for games. Unfortunately, these different ROIs makes it much harder to produce a game-agnostic solution.

8.1 Salient Type: Centred-Based Games

Games in this category, usually have their salient regions located in the middle of the screen (First player Shooters, Driving Simulation, etc) [15]. Hence, to effectively dim to save power, we can dimmed the areas outside the central salient region to save power.

To better understand how to effectively apply this dimming filter without im-
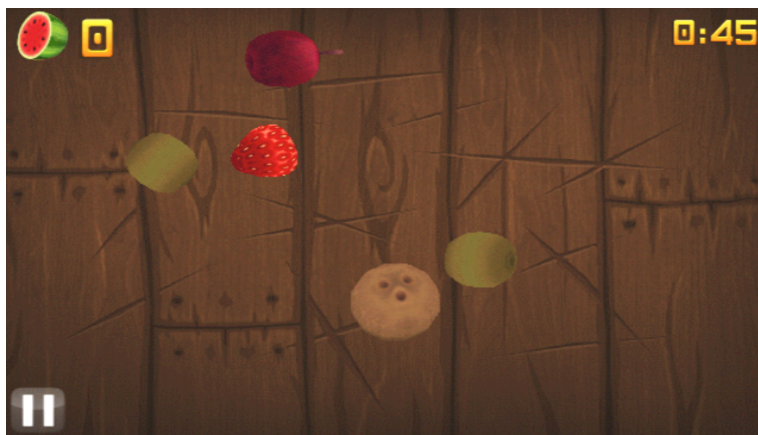
Random
(Fruit Ninja)

Centered
(Quake 3)

Figure 8.1: Different salient region in games. Random salient region are found in games such as Fruit Ninja where the focus is on the fruits. Centred salient region are found in games such as Quake 3 where the focus is in middle

pacting gaming experience, we have to look at the locus of attention to determine this central ROI. In this review, we used this dimming approach on First Player Shooter (FPS) game (Quake 3) on the OLED mobile device (HTC Nexus One).

8.1.1 Locus of Attention

In earlier chapter, large numbers of applications were reviewed to determine a simple ROI model, the other approach is to determine the ROI is via tracking the eye gaze of the user when they use an application. El-Nasr et al. [15] has performed eye-tracking study of users playing a FPS game (Halo II) and found that the main area of focus, within these games, was the centre of the screen. Their study showed that the resolution of the main area of interest (the locus of attention) was approx-

imately 300x220 pixels when the game resolution was 640x480 pixels. This locus was centred on the crosshair of the game located at the centre of the screen (coordinate {320,240}).

For this study, we utilised this knowledge on a Nexus One Smartphone that uses the OLED display with a resolution of 800x480 pixels. We thus linearly scaled the results listed above and identified a locus of interested of around 300x180 pixels, centred at coordinate {400,240} as shown in Figure 8.2.

Figure 8.2: Locus of attention

The implication is that the remainder of the screen commands less interest to the user when playing the game. However, this does not imply that these non-locus

77

areas are not useful during game play. Indeed, peripheral visual attention allows users to see the movement of artefacts as they come in and out of the locus of attention. Thus removing the non-locus areas completely, as shown in Figure 8.2, is not practical.

8.1.2 Power Saving Technique

To achieve a good balance between power saving and preserving the user's peripheral vision, we dim the areas outside the locus of attention – allowing us to achieve power savings. However, the challenge then becomes to affect this dimming without distracting the user (for example, naive dimming can can cause flickering as the user moves around).

To address this concern, we use a gradient dimming approach where areas close to the locus of attention are dimmed less than areas further away. We ensure that artefacts at the edge of the screen are still visible by ensuring that there is a minimum level of brightness that is applied to every pixel as shown in Figure 8.3. This minimum brightness level differs based on the OLED manufacturer and needs to be calibrated once for every new display (from a different manufacturer).

8.1.3 Adapting to Game Play Situations

The dimming mechanism can only be applied when the game is actually being played and even then, only in situations where the user's viewpoint is changing (either because the user is moving or looking around). When the user is stationary, we do not dim the display as prior work [15] has noted that the user's attention is spread across the entire screen when they are stationary.

8.1.4 Adaptive Framework

Our final implemented framework is shown in Figure 8.4 and implements the following: First, the system remains in normal power mode when the user is stationary within the game. Second, dimming is performed incrementally as the user initiate

Figure 8.3: Gradual dimming from *Locus of Attention* to the edges

movements and vice versa (brightening the screen) as the user stops moving. This
approach reduces user distraction as the user will not see the screen abruptly dim or
brighten.

8.1.5   Implementation

We tested our solution using Kwaak 3 [28], an Android version of the Quake 3
Arena [24] game. This game belongs to the First-Player Shooters (FPS) genre which
demands both the most phone resources as well as the most user attention during
game play. We believed these genre characteristics make Kwaak 3 an ideal test
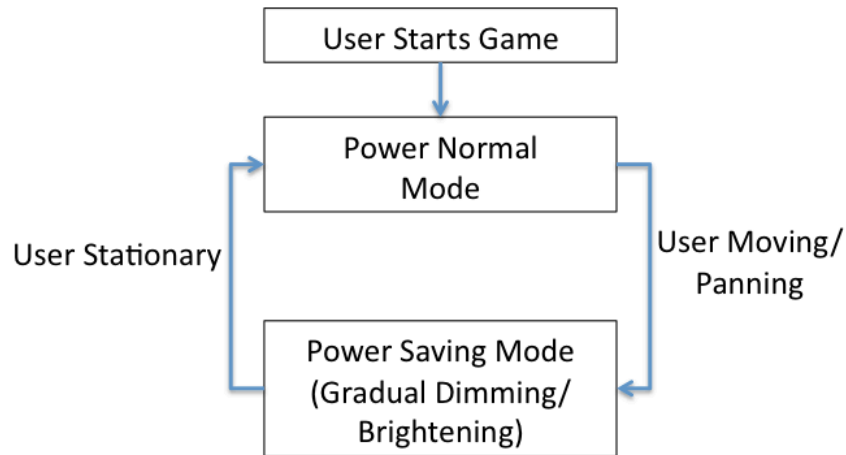
Figure 8.4: The Adaptive Framework. Dimming only is applied during game motion and is implemented slowly across time to reduce distraction to the user

app and that any results achieved with Kwaak 3 can be generalised to other app categories.

Kwaak 3 Overview

Quake 3 Arena was developed by Id Software in 1999 and was a commercially successful FPS game. Id Software released the Quake 3 game engine under the GNU General Public License in 2005. It was then ported to Android and made publicly available by the Kwaak 3 [28] project. Kwaak 3 has two main game components: an app layer that sits on top of Android's Application Framework API and a native component developed using Android's Native Development Kit (NDK).

Kwwak 3 Modification

Our dimming technique utilises the OpenGL API that is already present in Android's API stack. We achieve our dimming by performing alpha blending (to change the transparency of specific pixels) on the areas outside the locus of attention. To control the dimming, we added code to monitor the game's movement controls (the dimming is activated when the user moves and removed when the user stops). In particular, we monitored both the forward and backward controls mapped to the on-screen trackball as well as the pan controls that were mapped to screen

gestures.

Alpha Blending Implementation

In our game implementation, alpha-blending is utilised again to achieved the gradual dimming effect from the locus of attention to the edge. However, in this game implementation, we are using the OpenGL alpha-blending APIs instead of the Core Libraries. This is because the game's rendering engine was developed with the OpenGL APIs. To obtain our gradual dimming effect:

1. Similar to our Core Libraries implementation, we have to compute a series of dimming boxes due to the limitation in the alpha-blending APIs. The dimming boxes originates from the locus of attention to the edge of the screen as shown in Figure 8.5. The maximum number of boxes is set to 60. This number was derived experimentally so that each larger box still had the same aspect ratio as the screen (3:5). For the Nexus One (800x400 pixel display with a locus of attention of 300x180 pixels), this resulted in 60 boxes. In addition, we experimented with less than 60 boxes but found that the resulting images showed noticeable user artefacts (boundaries of the dimmed boxes becoming visible etc.).

2. Second, alpha blending is then applied to these boxed areas using the following heuristics:

   (a) The outermost box is set to the brightness level that is dim but still bright enough for the user to see.

   (b) The remaining 59 boxes (between the outermost box and the locus of attention) are then brightened linearly (with each box having a different brightness value) until the box next to the locus of attention is just a little dimmer than the locus.

   This heuristics ensures that we incrementally dim the screen from the locus of

81

attention (brightest area) to the edge of the display (dimmest yet still visible area).
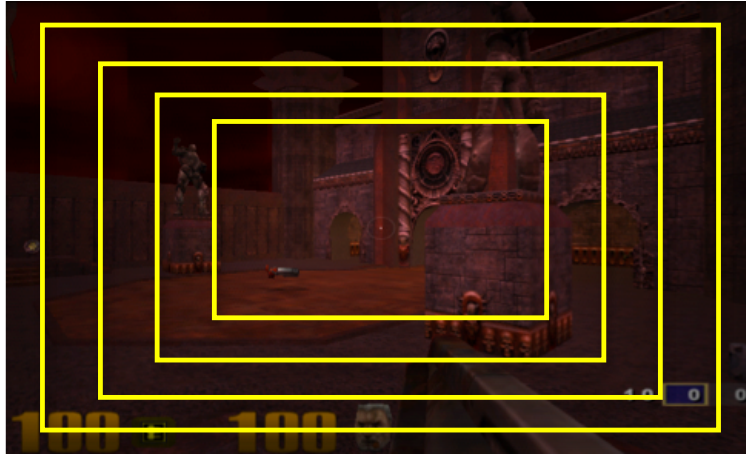
Figure 8.5: An example of the series of Dimming Boxes used to create the gradient effect

8.1.6 Evaluation

We focused on two evaluation goals:

1. To test the effectiveness of the power saving technique. In particular, how much power can we save, when playing a game, with our technique.

2. How playable is the game when our technique is active?

We tested both these goals via a user study involving 30 participants.

Participant Selection and Devices

The user study participants were separately elicited from the undergraduate and postgraduate population at Singapore Management University for the game study. The basic requirement for the user study was that each participant needed to be comfortable with Smartphone. The evaluation was performed on a single Google Nexus One Android Smartphone to ensure that the results were not skewed by inconsistent hardware or settings between phones (such as different OLED displays, OS version or settings).

In total, we obtained 30 participants for our user study (18 males, 12 females) who were within the age range of 16 to 27. All our participants had prior experience with playing mobile games, with the majority preferring to play causal/puzzle mobile games. The majority indicated that they preferred to play mobile games during "quieter" moments when they were on public transport, queuing, or resting. The majority of the participants were also spent more than 15 minutes per day playing mobile games.

Experiment Settings

During the user study, each participant had to play two games of Kwaak 3, each lasting 15 minutes, that used the exact same map and configurations. One game used our technique and one did not. We used PowerTutor [42] to measure the component (display, CPU, network, etc.) power consumption of each game. The user was not told which version of the game they were trying and the order of the games (our version and normal) was counter-balanced between users to reduce bias.

To identify game playability issues, we tracked the number of kills (elimination of the enemy in the game) achieved by the user within the 15 minute playing window for both game versions. This provided an objective measure of whether players could achieved the same level of success with our version of the game (as compared to the default). We also surveyed each user to learn their qualitative opinion of the playability of each game version.

User Study Procedure

Each user study experiment was conducted in the following order.

1. The participant was briefed on the project and motivation for this study. They then signed the consent form.

2. Each participant was shown a demo of the unmodified Kwaak 3 game on the Google Nexus One Smartphone.

3. The participant was then allowed to try the game for a while to get used to the game mechanics, controls and map before the actual test. This allowed every participant to become somewhat familiar with the game even if they had no prior experience with it. After about 5 to 10 minutes of acclimatisation, the actual test would begin.

4. After each game session, the user had to complete a questionnaire that asked them to rate the playability of that game.

5. After completing both game sessions, the user had to fill up a final overall feedback form.

Each user study took about one hour in total and each participant was paid USD $8 for their participation. Finally, we did not influence the participants while they were playing the game – in particular, they were allowed to use whatever game play style (aggressive, cautious, camping, etc.) that they wished.

8.1.7 Results

In this section, we present and discuss the results from our user study.

Power savings

Figure 8.6 shows the power savings achievable by our method (The bars for "After" are with our technique turned on). We show both the total power savings as well as the savings for just the OLED power consumption. On average, across all participants, our methods managed to save about 11% of the OLED display power that resulted in a 4% total power savings.

Playability

To understand the playability impact of our solution, we reviewed the in-game performance (in terms of the number of "kills") of each player both with and without our system. The mean kill score across both game variants, across all participants,
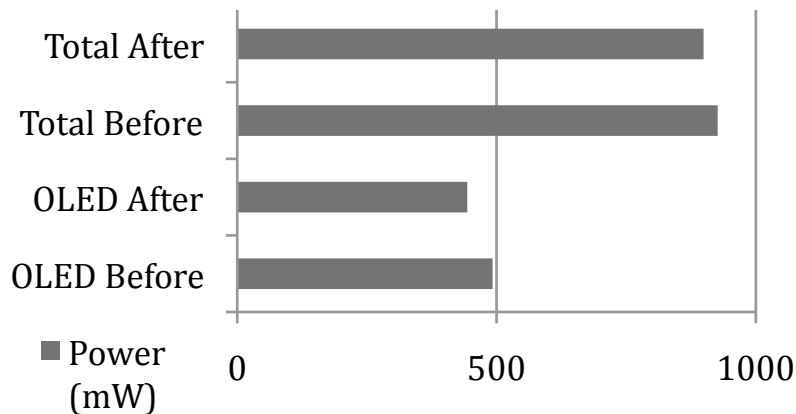
Figure 8.6: Power savings

was consistently low at 0.8 (original) and 1.2 (with our system) with a median of 0.5 and 1 respectively. This difference was statistically insignificant (p-value of 0.51 using a pairwise t-test). This indicated that our system had no impact on the player's kill totals. However, the low number of kills per player was a cause for concern as we expected many more kills to be obtained in a 15 minute game window.

To understand the reason for the low number of kills, we reviewed the survey feedback and noted that almost all the users had trouble with the controls for the game. In particular, most users (80% of the participant pool) found the game controls hard to master. They thus were unable to navigate or shoot effectively in the game and this contributed to their low kill scores. Hence, overall, we could not reach any definitive conclusion on the impact of our solution on the playability of the game.

8.1.8   Discussion

In this section, we discuss the lessons learned during this study.

The True Cost of Power Savings

We could have significantly increased the OLED display power savings from 11% by dimming the rest of the screen extremely aggressively – albeit at the cost of playability. We believe that this tradeoff could be integrated within the game settings and provided as an option to the user – save minimal power with no playability

85
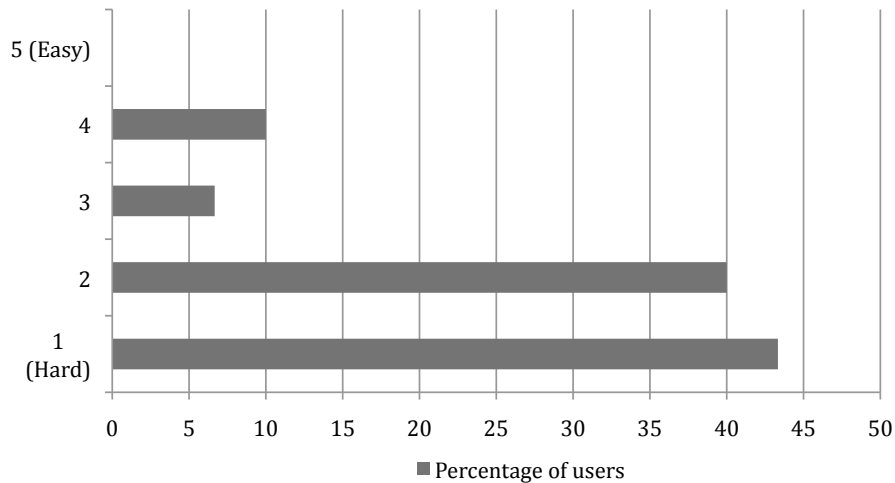
Figure 8.7: Ease of Controls

impact or save more power with playability impact.

Our overall power savings is limited to 4% as we need additional CPU cycles to perform our alpha blending-based screen dimming. However, we believe that this CPU cost can be reduce by utilising Android's native layer SDK (which provides power efficient image manipulation methods). Finally, another factor for our lower power savings was that quite a few participants (36%) used a passive game play style where they did not move around much (possibly because they were not comfortable with the game controls). Our scheme would save more power for players who were constantly moving about the game world.

Playability Has Many Aspects

We found that our participant reported usability issues arose from Kwaak 3's controller implementation and not from our power dimming implementation. This was because Quake 3 was never designed for mobile devices (that did not have a keyboard and mouse) and thus the movement controls had to be artificially mapped onto a few physical buttons, an on-screen joystick, and touch gestures.

As a result, our participants had major difficulties switching between the various control schemes when playing the game (moving around, shooting enemies, etc.). This resulted in the overall low number of kills per participants. Indeed, as

86

noted above, 36% of the participants adopted a more passive game play style. We investigated the control issues in more detail and found that most participants found it much easier to track and engage enemies while they were stationary (Figure 8.8) compared to when they were moving (Figure 8.9). This was because the movement and tracking controls were laid out differently (movement used an on-screen joystick while tracking required screen swipes).

Overall, our experience is that there could be significant control issues inherent to playing complicated games on mobile devices. Hence, for our future tests, we will provide participants with external keyboards and mice to eliminate control issues.
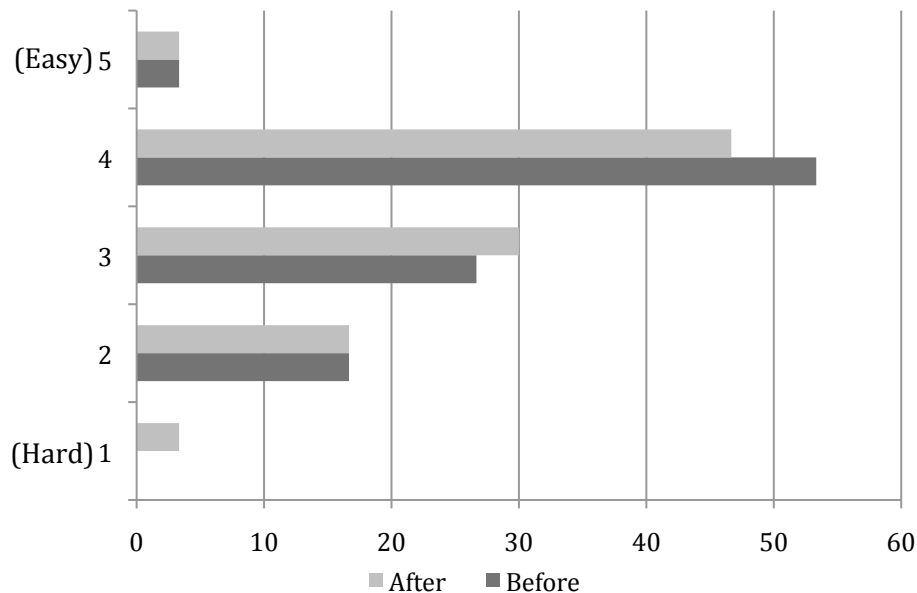
Figure 8.8: User ability to track when stationary

8.1.9 Generalisability of Technique

The technique discussed so far was tested with one game using data from an eye tracking model developed for FPS games. In order to generalise it, we would need to first conduct additional eye tracking user studies to determine where mobile users look at when they are using applications from other genres on their mobile devices. We can then use this information to determine the appropriate locus of attention
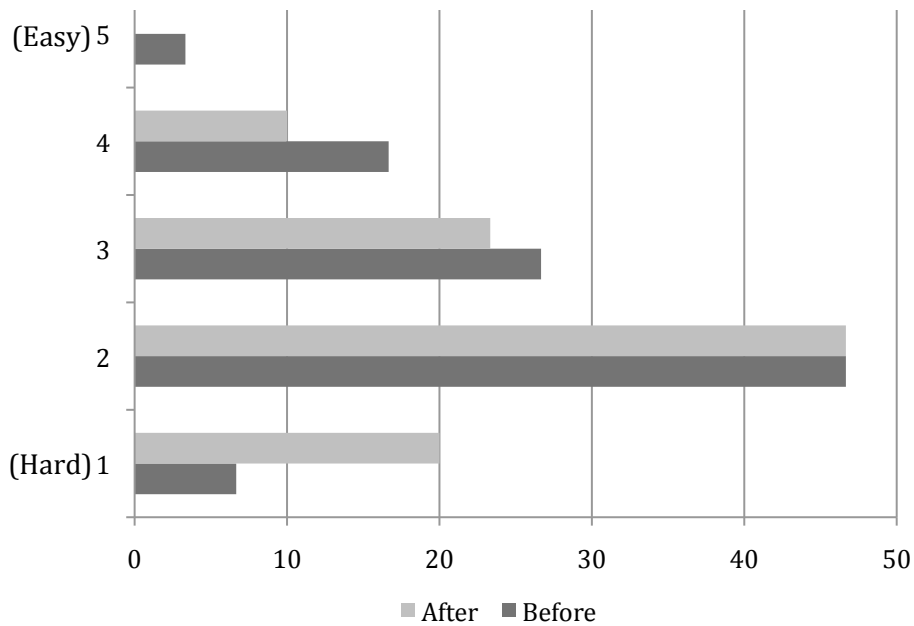
Figure 8.9: User ability to track when moving

for other application genres. Next, we can make use of existing mobile OS features and notifications to improve our dimming mechanism. For example, we can perform gradual dimming when the user is scrolling (the faster the scroll, the dimmer it gets). We can also achieve very CPU-efficient dimming by implementing our mechanisms into the framebuffer pipeline of the mobile OS. By utilising these behavioural and low-level OS features, we believe we can extend our dimming technique to many other applications. We plan to investigate these techniques in future work.

8.1.10 Conclusion

This work has demonstrated that we can save the power consumption of OLED displays, during game play, by gradually dimming the edges of screen when the user is moving based on a centred oriented salient region. We implemented our solution on a Nexus One Android phone with the Kwaak 3 FPS game. We conducted a user study, involving 30 participants, to test our solution. The study showed that our method reduced the overall system power consumption by 4% without incurring and significant game playability impact. However, more study needs to be done to build an comprehensive solution as peripheral vision outside the locus of attention can

differ between games thus the amount and dimming pattern would differ between games as well.

8.2 Salient Type: Random-Based Games

In this category, the salient regions are harder to define based on the game type and their interaction model. For example, in side scrolling games like *Super Mario Bros, Metal Slug* have interactivity content across the screen and due to the scrolling nature, it is harder to dimmed the screens due to the high dynamic nature of the game.

However, it is still possible to implement dimming in this type of games by shifting the focus to non-dynamic content. For example, in some casual games, there are foreground and background elements. Hence, the best way to save power, is to dim the non dynamic content such as background images while leaving the foreground objects clear (e.g., leave the bids, pigs, and structures clear in *Angry Birds* but dim everything else).

However, doing this requires understanding not only the game nature and context but the exact composition of the pixels in the game's framebuffer as well (as the foreground objects could be anywhere on screen). This makes the task harder than expected as games use the OpenGL-ES and not the standard rendering framework. While we have extended to support OpenGL-ES games shown in *Fruit Ninja* Figure 8.10 where we dimmed the background and leave the foreground objects untouched, this continues to be work-in-progress.

8.3 On Going Work

The games category is the most dynamic category within any popular application stores and as well as the most challenging in terms of power management. In this chapter, we review region of interest differs widely between games and although it is possible to applied dimming as a solution but it remains a ongoing work in

Before
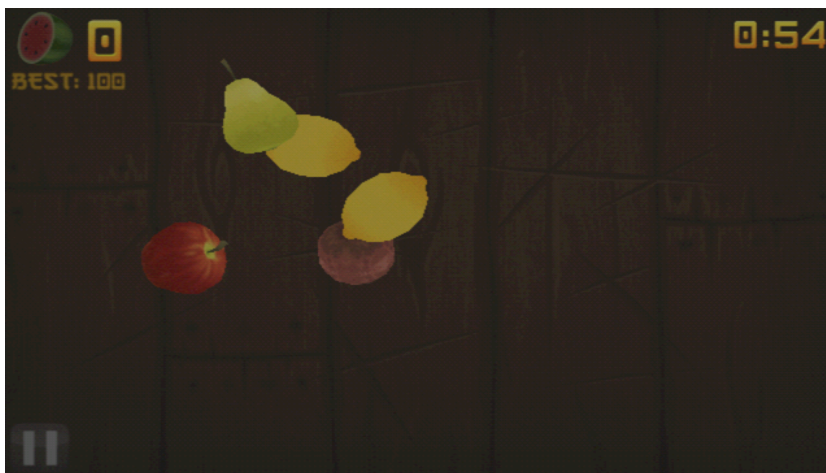
After

Figure 8.10: Fruit Ninja dimming applied to the background when visual attention is placed on the fruits

progress to develop a game agnostic solution.

Chapter 9

Dissertation Conclusion and Future Work

9.1 Conclusion

We presented a novel system that saves OLED display power by dimming the screen in effective and efficient ways by exploiting the *saliency* properties of individual applications as well as user behaviour. We do this by dimming regions of low interest while keeping region of interest untouched, we reduce the impact to usability by activating the dimming during low battery condition, in addition, we do not dim the screen when the user is interacting with the screen and using the onscreen keyboard to further reduce usability impact.

We evaluated the performance of the system with 15 popular applications. Our evaluation showed that system saves, on average, between 23 to 34% of the OLED display power when using applications with minimal impact on task completion times. As stated earlier, OLED displays consume as much as 67% of the overall system power [6]. Hence, the power savings translates to at least a few extra hours of phone use. In addition, we showed, via a user study with 30 participants, that the usability of system was still quite high and that developer-supplied profiles might be good enough. Overall, the results are very positive and demonstrate that the system is effective at saving OLED display power without affecting usability significantly.

Furthermore, our review of dimming in games through our review of the Kwaak 3 dimming implementation. We showed that it is possible for developers to bundle

useful dimming profiles together with their applications — thus reducing the burden on the user. However, region of interest is harder to determine in games due to different game category as such, a game agnostic solution remains in the pipeline for future work.

9.2 Operational Issues

While our results suggest that this system is quite effective and usable, this system still has some practical limitations. In addition that game solution is still in the works, this study was done in a well-lit office environment. To generalise our solution, we need to conduct studies in other indoor and outdoor environments to better determine if dimming effects needs to be strengthen or weaken depending on lighting condition. In addition, the profiles currently only work in portrait mode and will need to be modified to support landscape orientations, and then tested for effectiveness.

The user study was a key component in evaluating the efficacy of this system. However, it also has its limitations. While our user population was large, it comprised solely of undergraduate technically oriented students; it is possible that a general or larger population might show different results.

Finally, the user study lasted only about 30 minutes; it is possible that the results would be quite different if this system was deployed and used daily. Clearly, additional studies are needed to ascertain the true power savings that can be achieved under normal daily use.

9.3 Future Direction

In this work, we look at the tradeoffs between usability and power savings through the use of dimming on OLED displays. There are few directions that we expand this work in the following areas:

9.3.1   Content and Context Aware

Seen in our game review, we saw that the game's content and context can affects the user interactivity, such as adopting a sit and wait approach rather than a rush tactic in Quake 3. Hence, by understanding the application content and context, this information can be extremely valuable to aid in power savings, for example earlier works utilised in game player positions to help save power [1].

Hence, the same approach can be used in application and games to help in display power savings. For example, we can utilise in game lighting to aid in power savings such as shadow casting allow us to turn off pixels. Similarly, any applications that emphasised on any important content can use the foreground and background approach to dim any unrelated content to emphasised the foreground content.

9.3.2   Usability-Aware Power Management

Usability-Aware takes the approach that power savings can be achieved when we take into consideration the usability characteristic between the user and their mobile hardware. The intuition stems from the fact that users are willingly to trade off usability when face with resource constraint. Hence, we can also observe usability and make intelligent choices when to save power.

One example is a simple extension for the OLED work is to use occlusion dimming. Occlusion dimming observes the fact that when the user is interacting with the mobile device, the user's fingers will always occlude the area underneath. This presents a good opportunity to dim or turn off the screen. Seen in Figure 9.1, the user fingers are on the screen, hence, there is no need to display any areas underneath the screen.

We proposed to approach usability-aware power management on two fronts: Minimising wastage when resource abundant and Maximising usage life when face with resource constraint.

Occlusion Dimming when finger is on screen

Turn off or dim areas under the finger.

Figure 9.1: Occlusion Dimming

Maximising usage life when resource constraint

Our OLED work can be broadly categorised under this area. However, we can also expand the same concept to other areas such as into other display technology, Wi-Fi/cellular radio and processing. As an example, on LCD display, we can modified content and backlight to adapt the content to reflective colours to use natural lighting to read content during battery constraint condition. Similarly, if users are using wearable computers, interaction and content can be diverted to these wearables during battery constraint conditions.

Minimising wastage when resource abundant

On the other end, usability can be utilised to minimise wastage on resource during resource abundant period (e.g. Fully charged battery). In most modern devices,

some level of power savings features already reduces battery wastage such as automatically screen dim, sleep mode, etc. However, these usually activates after inactivity. Our proposed approach looks at enabling saving during active usage. A simple example uses the previous wearable computer scenario, where messages and notifications can be directed to these wearables reducing the need for users to use their mobile devices, this indirectly, reduce the frequency for user to look at their mobile device.

Other techniques can make use of human perception properties such as optical illusion effects and/or change blindness to aid in power savings. For example, using change blindness such as gradual dimming white spaces and/or increasing black font size over a period of time would enable power savings on OLED devices without impacting the user due to the slow gradual change.

Appendix A

Profile Generation

Profile generation is part of Section 5.1.4. This chapter shows how the user defined sliders in the Desktop or Mobile tool generates the required profile text files that is used in the deployment module.

A.1 Dim Profile Calculation

The dim profile calculation is encoded within the *DimCalculation* class. To utilise it, developers need only to do the following:

```
//Calculate the actual dim area, store
    this data, only apply to the system
    when user click the button
DimCalculation dimHandle=new
    DimCalculation();
String[] data=dimHandle.buildDimRect(2,
    deviceWidthResolution,
    deviceHeightResolution, rectLocus,
    rectBlack, 0, 0);
```

The result is presented in the String array, data. Using any CSV tool, we can deploy this data into a normal text file. This text file needs to be denoted by the application name that the dimming is applied to. For example, a dimming profile for Facebook will be stored as com.facebook.katana.txt.

Note: Method *buildDimMirrorRect()* is not shown here. This method is used for creating the mirror profiles, however, it uses the same approach as *buildDimRect()*. The only difference is the dimming boxes generate is just in reverse order.

A.1.1 Method Details: buildDimRect()

Method *buildDimRect(int boxWidth, int displayWidth, int displayHeight, ClsRect LocusArea, ClsRect PureDarkArea, int alphaSeedTop, int alphaSeedGradient)* is responsible for building the String array data.

The String array data consists of each String with the format *Dim-Value Left Top Right Bottom* (See Section 5.1.5 for more details)

The slider markers in the Desktop or Mobile tool can be easily translate into two main area, LocusArea and PureDarkArea (clear region and black plateau region). These two main areas is defined in code using our ClsRect class. Our ClsRect class is an implementation of the Android Rect class (under package android.graphics), this class simply holds the rectangular coordinates of the four edges of these two regions. Once we known the dimension of these two regions, we can easily compute the other 2 regions (Top gradient region and Middle gradient region).

Parameters displayWidth and displayHeight refers to the maximum resolution of the display of your device. Both alphaSeed parameters indicate how fast the gradients should be reached to the edge (range 0 to 100). The bigger the seed, the faster it reaches 90% dim. alphaSeedTop refers to the Top gradient region.

Lastly, as mentioned in Section 5.1, the gradient regions are made up of smaller dimming boxes. Hence, the boxWidth parameter indicates the height of these dimming boxes.

This method builds all the necessary dimming boxes in another subroutine, *buildRectDimAreas()* and formats into the String array.

```java
public String[] buildDimRect(int boxWidth, int
    displayWidth, int displayHeight, ClsRect
    LocusArea, ClsRect PureDarkArea, int alphaSeedTop,
    int alphaSeedGradient)
{
    //Step 1: Call Basic Building Rect List First.
    rectHolder=new ArrayList<ClsRect>();
    alphaHolder=new ArrayList<Integer>();
    buildRectDimAreas(boxWidth, displayWidth,
        displayHeight, LocusArea, PureDarkArea,
        alphaSeedTop, alphaSeedGradient);
```

```
//Step 2 Translate Build Rect List to String
    data
int size=rectHolder.size();

String data[]=new String[size];
for(int i=0;i<size;i++)
{
        String tempHolder="" + alphaHolder.get(i)
            + " " + rectHolder.get(i).getRect();
        data[i]=tempHolder;
}

return data;
}
```

Note: rectHolder and alphaHolder are global variables.

A.1.2 Method: buildRectDimAreas()

This is the important routine that builds the necessary dimming boxes. Data is stored directly into the global variables rectHolder and alphaHolder, this reduces memory pressure when deploy on to the mobile platform.

The private method *buildRectDimAreas(int boxWidth, int displayWidth, int displayHeight, ClsRect LocusArea, ClsRect PureDarkArea, int alphaSeedTop, int alphaSeedGradient)* uses the same parameters from method *buildDimRect()*, see previous Section A.1.1 for parameter details.

This method is broken into 3 main areas:

1. Computing the Top Gradient Region

2. Computing the Middle Gradient Region

3. Computing the Black Region

The clear region does not require any dimming, hence no processing is needed for this region.

All 3 areas consists of nearly the same flow structure. First, given the parameters LocusArea and PureDarkArea, we can compute the Top & Middle Gradient Region

and Black region. Second, given the boxWidth, we can divide the Top and Middle Gradient region into individual dimming boxes.

Lastly, given the nature of the region, the dimming is applied accordingly. For Top Gradient Region, the dimming is applied in reverse from 90% dim to 0% across the dimming boxes. For Middle Gradient Region is from 0% to 90%. Since the dark region is black, we only need to set the last dimming box to the PureDarkArea dimension and set the dimming to 100%.

```java
private void buildRectDimAreas(int boxWidth, int
    displayWidth, int displayHeight, ClsRect LocusArea,
    ClsRect PureDarkArea, int alphaSeedTop, int
    alphaSeedGradient)
{
    final int locusLeft=LocusArea.getCorner_Left();
    final int locusTop=LocusArea.getCorner_Top();
    final int locusRight=LocusArea.getCorner_Right();
    final int locusBottom=LocusArea.getCorner_Bottom();

    final int blackLeft=PureDarkArea.getCorner_Left();
    final int blackTop=PureDarkArea.getCorner_Top();
    final int blackRight=PureDarkArea.getCorner_Right();
    final int blackBottom=PureDarkArea.getCorner_Bottom();

    int boxResolution=0;
    float alphaResolution=0;

    int leftShifter;
    int rightShifter;
    int topShifter;
    int currentAlpha;
    int bottomShifter;

    float floatCurrentAlpha=0.0f;


    //Top Gradient Box
    //Based on boxWidth, calculate number of boxes
        required.
    boxResolution=(int)
        Math.ceil((double)locusTop/(double)boxWidth);
    System.out.println("Top Gradient Box Resolution: " +
        boxResolution);

    if(boxResolution!=0)
    {
            //int readjusted_boxResolution=(int)
```

```java
        Math.ceil((double)boxResolution*((double)alphaSeedTop/100));
    //alphaResolution=(int)
        Math.ceil((double)256/(double)readjusted_boxResolution);
        //Give the number of boxes, calculate the
        alpha from light to dark to the edge.

    alphaResolution=(float)256/(float)boxResolution;
        //Given the number of boxes, calculate the
        alpha from light to dark to the edge.
    System.out.println("Top Gradient Alpha
        Resolution: " + alphaResolution);

    //Setup the box boundaries that does not change.
    leftShifter=locusLeft;
    rightShifter=locusRight;
    topShifter=locusTop;

    //Reset Alpha to zero
    currentAlpha=0;
    floatCurrentAlpha=0.0f;

    //Plus Alpha is the throttling of the dimming
        (achieve dark edge faster) for this area based
        on user's selection
    int plusAlpha=(int)
        Math.ceil(256d*(double)alphaSeedTop/100);

    //Draw the boxes
    for(int i=0;i<boxResolution;i++)
    {
        ClsRect rect=new ClsRect(leftShifter,
            topShifter-boxWidth, rightShifter,
            topShifter);
        rectHolder.add(rect);
        //alphaHolder.add(currentAlpha+plusAlpha);
        alphaHolder.add(currentAlpha);

        topShifter-=boxWidth;

        //currentAlpha+=(alphaResolution+plusAlpha);
        floatCurrentAlpha+=alphaResolution+plusAlpha;
        currentAlpha=(int)
            Math.floor(floatCurrentAlpha);
    }
}


//Middle Gradient Box
//Reset box and alpha resolution back to zero.
```

```java
boxResolution=0;
alphaResolution=0;

//Based on boxWidth, calculate number of boxes
   required for this section.
boxResolution=(int)
   Math.ceil(Math.abs((double)blackTop-(double)locusBottom)/(double)k
System.out.println("Middle Gradient Box Resolution: "
   + boxResolution);

//Given the box resolution, find out the resolution
   for the alpha for each incremenet.
alphaResolution=(float)256/(float)boxResolution;

System.out.println("Alpha Resolution: " +
   alphaResolution);


//Setup box value that does't change.
bottomShifter=locusBottom;
rightShifter=locusRight;
leftShifter=locusLeft;

//Reset Alpha values
currentAlpha=0;
floatCurrentAlpha=0.0f;

//Plus Alpha is the throttling of the dimming
   (achieve dark edge faster) for this area based on
   user's selection
int plusAlpha=(int)
   Math.floor(256*(double)alphaSeedGradient/100);


//System.out.println("Box Diff: " +
   (boxResolution-readjusted_boxResolution));

//Draw the boxes
for(int i=0;i<boxResolution;i++)
{

      ClsRect rect=new ClsRect(leftShifter,
         bottomShifter, rightShifter,
         bottomShifter+boxWidth);
      rectHolder.add(rect);
      //alphaHolder.add(currentAlpha+plusAlpha);
      alphaHolder.add(currentAlpha);
      //System.out.println("Alpha: " + currentAlpha);

      bottomShifter+=boxWidth;
```

```java
            floatCurrentAlpha+=alphaResolution+plusAlpha;
            currentAlpha=(int)
                Math.floor(floatCurrentAlpha);
        }



        //Bottom Black Box
        if(boxWidth>1)
        {
            ClsRect rect=new ClsRect(blackLeft, blackTop,
                blackRight, blackBottom);
            rectHolder.add(rect);
            alphaHolder.add(255);
        }
        else
        {
            boxResolution=0;
            alphaResolution=0;

            //Get the number of Rect(boxes) to draw
            boxResolution=(int)
                Math.ceil(Math.abs((double)blackTop-(double)blackBottom)/(doub
            System.out.println("Black Box Resolution: " +
                boxResolution);


            bottomShifter=blackTop;
            rightShifter=locusRight;
            leftShifter=locusLeft;

            for(int i=0;i<boxResolution;i++)
            {

                ClsRect rect=new ClsRect(leftShifter,
                    bottomShifter, rightShifter,
                    bottomShifter+boxWidth);
                rectHolder.add(rect);
                //alphaHolder.add(currentAlpha+plusAlpha);
                alphaHolder.add(255);
                //System.out.println("Alpha: " + currentAlpha);

                bottomShifter+=boxWidth;

            }
        }


    }
```

Appendix B

View Class Modifications

Modifications to the View class under the Android Core Libraries. Only dimming modifications made to the View class are listed, for implementation details please refer to Section 5.

B.1 Profile Check and Preload

B.1.1 Constructor Modification

Modified constructor to execute our solution to check and preload the profile during any Android widget loading.

```
/**
 * Simple constructor to use when creating a view from
   code.
 *
 * @param context The Context the view is running in,
   through which it can
 *      access the current theme, resources, etc.
 */
public View(Context context) {
   mContext = context;
   mResources = context != null ? context.getResources()
      : null;
   mViewFlags = SOUND_EFFECTS_ENABLED |
      HAPTIC_FEEDBACK_ENABLED;
   // Set some flags defaults
   mPrivateFlags2 =
         (LAYOUT_DIRECTION_DEFAULT <<
            PFLAG2_LAYOUT_DIRECTION_MASK_SHIFT) |
         (TEXT_DIRECTION_DEFAULT <<
            PFLAG2_TEXT_DIRECTION_MASK_SHIFT) |
         (PFLAG2_TEXT_DIRECTION_RESOLVED_DEFAULT) |
```

```
            (TEXT_ALIGNMENT_DEFAULT <<
               PFLAG2_TEXT_ALIGNMENT_MASK_SHIFT) |
            (PFLAG2_TEXT_ALIGNMENT_RESOLVED_DEFAULT) |
            (IMPORTANT_FOR_ACCESSIBILITY_DEFAULT <<
               PFLAG2_IMPORTANT_FOR_ACCESSIBILITY_SHIFT);
    mTouchSlop =
       ViewConfiguration.get(context).getScaledTouchSlop();
    setOverScrollMode(OVER_SCROLL_IF_CONTENT_SCROLLS);
    mUserPaddingStart = UNDEFINED_PADDING;
    mUserPaddingEnd = UNDEFINED_PADDING;


    readInDimArea(context.getPackageName());
    readInMirrorDimArea(context.getPackageName());
  }
```

B.1.2 Method: readInDimArea()

Read and load the dimming profiles into memory with checks to prevent re-execution of code in subclasses as well as prevent multiple expensive IO operations. readInMirrorDimArea() provides the same functionality.

```
/**
    * Read in the dim profiles from the SDCard Path
      /OLEDProfiles/ into the memory.
    * Please ensure that the /OLEDProfiles/ profile is
      created.
    *
    * @param packageName The package name (App) that we are
      going to load in the profile
    */
  private void readInDimArea(String packageName)
  {
     //Log.i("KiatWee", packageName);

     //Customisation For Weird App Behaviours
     if(packageName.equals("com.halfbrick.fruitninjafree"))
        //Dialog box
        return;
     else if(packageName.equals("android")) //System
        return;
     else if(packageName.contains("com.android")) //System
        return;
     else if(packageName.contains("com.cyanogenmod"))
        //System
        return;
```

```java
 File sdcardKW =
    Environment.getExternalStorageDirectory(); //Get
    the SDCard path (where the OLED Profiles are
    stored)

//Get the profile (text) file
File file = new File(sdcardKW, "/OLEDProfiles/" +
    packageName + ".txt");

//Log.i("Data: ", file.getAbsolutePath());

if(file.exists()==false) //Exit if not exist
    return;

try
{
    //Get the root view
    View rootView=getRootView();
    //View rootView= (View) getRootNode();

    //Check if root view has the data
    if(rootView.alphaHolder.size()==0 &&
        rootView.rectHolder.size()==0)
    {
        //Log.i("KiatWee", "Load in root. " +
            this.toString());

        //Load from file
        BufferedReader br = new BufferedReader(new
            FileReader(file));
        String line;

        while ((line = br.readLine()) != null)
        {
            //Log.i("Data: ", line);

            //Data is CSV-like format, data seperated by
                space.
            //Process the data in the following format
            //alpha-value rect-left rect-top rect-bottom
                rect-right
            //Process the data into alpha value and Rect
            String[] splitHolder=line.split(" ");
            int
                alphaValue=Integer.parseInt(splitHolder[0]);
            Rect rect=new
                Rect(Integer.parseInt(splitHolder[1]),
                Integer.parseInt(splitHolder[2]),
                Integer.parseInt(splitHolder[3]),
                Integer.parseInt(splitHolder[4]));
```

```
                alphaHolder.add(alphaValue);
                rectHolder.add(rect);
            }

            br.close();

            //Store the data into root so that we only need
                to store in one location. No need to store
                in every View. This is because it will be
                very memory expensive to store this repeated
                information.
            rootView.alphaHolder=alphaHolder;
            rootView.rectHolder=rectHolder;
        }
    }
    catch (IOException e)
    {
        Log.e("KiatWeeOLED", e.getMessage());
    }
}
```

B.2 Extending the Android Draw Process

B.2.1 Draw Method Modification

Modification to the *draw(Canvas canvas)* method to add in our dimming subroutines, *kiatweeAddDimming(Canvas canvas)*.

```
/**
 * Manually render this view (and all of its children)
     to the given Canvas.
 * The view must have already done a full layout before
     this function is
 * called. When implementing a view, implement
 * {@link #onDraw(android.graphics.Canvas)} instead of
     overriding this method.
 * If you do need to override this method, call the
     superclass version.
 *
 * @param canvas The Canvas to which the View is
     rendered.
 */
public void draw(Canvas canvas) {
    final int privateFlags = mPrivateFlags;
    final boolean dirtyOpaque = (privateFlags &
        PFLAG_DIRTY_MASK) == PFLAG_DIRTY_OPAQUE &&
```

```java
        (mAttachInfo == null ||
            !mAttachInfo.mIgnoreDirtyState);
mPrivateFlags = (privateFlags & ~PFLAG_DIRTY_MASK) |
    PFLAG_DRAWN;

/*
 * Draw traversal performs several drawing steps
     which must be executed
 * in the appropriate order:
 *
 *    1. Draw the background
 *    2. If necessary, save the canvas' layers to
    prepare for fading
 *    3. Draw view's content
 *    4. Draw children
 *    5. If necessary, draw the fading edges and
    restore layers
 *    6. Draw decorations (scrollbars for instance)
 */

// Step 1, draw the background, if needed
int saveCount;

if (!dirtyOpaque) {
    final Drawable background = mBackground;
    if (background != null) {
        final int scrollX = mScrollX;
        final int scrollY = mScrollY;

        if (mBackgroundSizeChanged) {
            background.setBounds(0, 0, mRight - mLeft,
                mBottom - mTop);
            mBackgroundSizeChanged = false;
        }

        if ((scrollX | scrollY) == 0) {
            background.draw(canvas);
        } else {
            canvas.translate(scrollX, scrollY);
            background.draw(canvas);
            canvas.translate(-scrollX, -scrollY);
        }
    }
}

// skip step 2 & 5 if possible (common case)
final int viewFlags = mViewFlags;
boolean horizontalEdges = (viewFlags &
    FADING_EDGE_HORIZONTAL) != 0;
boolean verticalEdges = (viewFlags &
```

```java
        FADING_EDGE_VERTICAL) != 0;
if (!verticalEdges && !horizontalEdges) {
    // Step 3, draw the content
    if (!dirtyOpaque) onDraw(canvas);

    // Step 4, draw the children
    dispatchDraw(canvas);

    kiatweeAddDimming(canvas); //KiatWee Additions

    // Step 6, draw decorations (scrollbars)
    onDrawScrollBars(canvas);

    // we're done...
    return;
}

/*
 * Here we do the full fledged routine...
 * (this is an uncommon case where speed matters less,
 * this is why we repeat some of the tests that have
   been
 * done above)
 */

boolean drawTop = false;
boolean drawBottom = false;
boolean drawLeft = false;
boolean drawRight = false;

float topFadeStrength = 0.0f;
float bottomFadeStrength = 0.0f;
float leftFadeStrength = 0.0f;
float rightFadeStrength = 0.0f;

// Step 2, save the canvas' layers
int paddingLeft = mPaddingLeft;

final boolean offsetRequired =
    isPaddingOffsetRequired();
if (offsetRequired) {
    paddingLeft += getLeftPaddingOffset();
}

int left = mScrollX + paddingLeft;
int right = left + mRight - mLeft - mPaddingRight -
    paddingLeft;
int top = mScrollY + getFadeTop(offsetRequired);
int bottom = top + getFadeHeight(offsetRequired);
```

```java
if (offsetRequired) {
   right += getRightPaddingOffset();
   bottom += getBottomPaddingOffset();
}

final ScrollabilityCache scrollabilityCache =
   mScrollCache;
final float fadeHeight =
   scrollabilityCache.fadingEdgeLength;
int length = (int) fadeHeight;

// clip the fade length if top and bottom fades
   overlap
// overlapping fades produce odd-looking artifacts
if (verticalEdges && (top + length > bottom -
   length)) {
   length = (bottom - top) / 2;
}

// also clip horizontal fades if necessary
if (horizontalEdges && (left + length > right -
   length)) {
   length = (right - left) / 2;
}

if (verticalEdges) {
   topFadeStrength = Math.max(0.0f, Math.min(1.0f,
      getTopFadingEdgeStrength()));
   drawTop = topFadeStrength * fadeHeight > 1.0f;
   bottomFadeStrength = Math.max(0.0f, Math.min(1.0f,
      getBottomFadingEdgeStrength()));
   drawBottom = bottomFadeStrength * fadeHeight >
      1.0f;
}

if (horizontalEdges) {
   leftFadeStrength = Math.max(0.0f, Math.min(1.0f,
      getLeftFadingEdgeStrength()));
   drawLeft = leftFadeStrength * fadeHeight > 1.0f;
   rightFadeStrength = Math.max(0.0f, Math.min(1.0f,
      getRightFadingEdgeStrength()));
   drawRight = rightFadeStrength * fadeHeight > 1.0f;
}

saveCount = canvas.getSaveCount();

int solidColor = getSolidColor();
if (solidColor == 0) {
   final int flags = Canvas.HAS_ALPHA_LAYER_SAVE_FLAG;
```

```java
    if (drawTop) {
        canvas.saveLayer(left, top, right, top +
            length, null, flags);
    }

    if (drawBottom) {
        canvas.saveLayer(left, bottom - length, right,
            bottom, null, flags);
    }

    if (drawLeft) {
        canvas.saveLayer(left, top, left + length,
            bottom, null, flags);
    }

    if (drawRight) {
        canvas.saveLayer(right - length, top, right,
            bottom, null, flags);
    }
} else {
    scrollabilityCache.setFadeColor(solidColor);
}

// Step 3, draw the content
if (!dirtyOpaque) onDraw(canvas);

// Step 4, draw the children
dispatchDraw(canvas);

kiatweeAddDimming(canvas); //KiatWee Additions

// Step 5, draw the fade effect and restore layers
final Paint p = scrollabilityCache.paint;
final Matrix matrix = scrollabilityCache.matrix;
final Shader fade = scrollabilityCache.shader;

if (drawTop) {
    matrix.setScale(1, fadeHeight * topFadeStrength);
    matrix.postTranslate(left, top);
    fade.setLocalMatrix(matrix);
    canvas.drawRect(left, top, right, top + length, p);
}

if (drawBottom) {
    matrix.setScale(1, fadeHeight *
        bottomFadeStrength);
    matrix.postRotate(180);
    matrix.postTranslate(left, bottom);
    fade.setLocalMatrix(matrix);
    canvas.drawRect(left, bottom - length, right,
```

```
           bottom, p);
    }

    if (drawLeft) {
        matrix.setScale(1, fadeHeight * leftFadeStrength);
        matrix.postRotate(-90);
        matrix.postTranslate(left, top);
        fade.setLocalMatrix(matrix);
        canvas.drawRect(left, top, left + length, bottom,
            p);
    }

    if (drawRight) {
        matrix.setScale(1, fadeHeight * rightFadeStrength);
        matrix.postRotate(90);
        matrix.postTranslate(right, top);
        fade.setLocalMatrix(matrix);
        canvas.drawRect(right - length, top, right,
            bottom, p);
    }

    canvas.restoreToCount(saveCount);

    // Step 6, draw decorations (scrollbars)
    onDrawScrollBars(canvas);
}
```

B.2.2   Method: kiatweeAddDimming()

Apply the dimming profile onto the screen. Uses the scroll flag to determine any touch operation and flip flag to determine if user uses the volume button. Method has been configure for Samsung Galaxy S3 deployment, however, other options are available for other devices.

```
/**
 * Actual Dimming Function to dim the screen using the
   canvas. Called from the onDraw method.
 * To understand this method, understand how Android
   draw the View. Quick guide: All views are structured
   via the layout.xml. Like any XML processing DOM/SAX
   parsing, the tree has to be traverse from the root,
   each view is successively drawn one ontop of each
   other until the whole XML tree is parsed.
 * Hence to dim effectively, we have to apply the dim at
   the correct location and applied it only once
   instead of applying everytime a draw occurs which
```

```java
       inccured a computational penalty. Hence, this
       function only dims at the correct location where the
       View occipies the device's width and height.
  *
  * @param canvas The Canvas to which the View is
       rendered. In this method, we piped in the canvas
       from onDraw to do the actual dimming.
  */
private void kiatweeAddDimming(Canvas canvas)
{
   //Initialize local Rect and Alpha holders for local
      processing without the affecting the main store.
   List<Rect> kwD_rectHolder; //Alpha Rect holders,
      holds the actual rect for alpha blending. Refer to
      alpha blending. A rect is basically a rectangle
      that covers the area you want to alpha-blend.
   List<Integer> kwD_alphaHolder; //Holds each alpha
      value for each rect.

   View viewHolder = this; //Get a view handle. For
      accessing the global variables. Do note, in
      Android using Get/Set functions is computationally
      expensive, please refer to Google's Android
      Performance guide to understand why we are doing
      this instead of building Get/Set functions.

   View rootView=getRootView(); //Get the root view. For
      processing rootView variables, to determine
      width/height and apply dim at the correct location

   //Turn on/off occulsion dimming under finger
   if(rootView.kiatweeOnScrollFlag)
   {
      canvas.drawColor(Color.TRANSPARENT);

      Paint paint = new Paint();
      paint.setColor(Color.BLACK);

      float radius=kwpressuredata * 80;

      canvas.drawCircle(kwlocX, kwlocY, radius, paint);

      return; //Do not apply dimming so that screen is
         not dimmed during on Touch events.
   }

   if(hasRectMirror &&
      rootView.kiatweeOnViewPortFlipFlag==false) //Check
      if there is a mirror component so that we can
      apply the dim accordingly.
```

```
{
    //System to use the mirror to dim the upper part
        of the screen.

    //Read from Root view holder.
    kwD_rectHolder=rootView.rectHolderMirror;
    kwD_alphaHolder=rootView.alphaHolderMirror;

    //kwD_rectHolder=rectHolderMirror;
    //kwD_alphaHolder=alphaHolderMirror;
}
else
{
    //If no mirror, just dim the lower areas

    //Read from Root view holder.
    kwD_rectHolder=rootView.rectHolder;
    kwD_alphaHolder=rootView.alphaHolder;

    //kwD_rectHolder=rectHolder;
    //kwD_alphaHolder=alphaHolder;
}

//Log.i("Kiatwee", "W: " + viewHolder.getWidth() + ",
    H: " + viewHolder.getHeight());

//1184, 1038, 1134

//Customization for Browser due to weird behaviour so
    additional customisation needed.
int viewHolderProtraintWidth=0;
int viewHolderProtraintHeight=0;
int viewHolderLandscapeWidth=0;
int viewHolderLandscapeHeight=0;
String packName=mContext.getPackageName();

//Galaxy Nexus
/*
viewHolderProtraintWidth=720;
viewHolderProtraintHeight=1184;
viewHolderLandscapeWidth=1184;
viewHolderLandscapeHeight=720;

if(packName.equals("com.google.android.youtube"))
{
    //720x633, 187, 112, 405
    viewHolderProtraintWidth=720;
    viewHolderProtraintHeight=633;
    viewHolderLandscapeWidth=633;
    viewHolderLandscapeHeight=720;
```

```java
        }
        */

        //Galaxy S3
        viewHolderProtraintWidth=720;
        viewHolderProtraintHeight=1280;
        viewHolderLandscapeWidth=1280;
        viewHolderLandscapeHeight=720;

        if(packName.equals("com.google.android.youtube"))
        {
           //720x633, 187, 112, 405
           viewHolderProtraintWidth=720;
           viewHolderProtraintHeight=729;
           viewHolderLandscapeWidth=729;
           viewHolderLandscapeHeight=720;
        }

        //if(packName.equals("com.facebook.katana"))
        //{
        //    //720x633, 187, 112, 405
        //    viewHolderProtraintWidth=720;
        //    viewHolderProtraintHeight=1281;
        //    viewHolderLandscapeWidth=1281;
        //    viewHolderLandscapeHeight=720;
        //}

        //Galaxy Tab 7.7
        /*
        viewHolderProtraintWidth=736;
        viewHolderProtraintHeight=1280;
        viewHolderLandscapeWidth=1280;
        viewHolderLandscapeHeight=736;
        */

        if((viewHolder.getWidth()==viewHolderProtraintWidth
           &&
           viewHolder.getHeight()==viewHolderProtraintHeight)
           ||
           (viewHolder.getWidth()==viewHolderLandscapeWidth
           &&
           viewHolder.getHeight()==viewHolderLandscapeHeight))
        {
           //Step 1: If rect holder is null means no dim for
              this widget, return.
           if(kwD_rectHolder==null)
              return;


           //Step 2: Check if View is a valid view covering
```

```
        the whole screen.
    Paint paint = new Paint();
    paint.setColor(Color.BLACK);



    //Step 3: Apply Dim area
    canvas.drawColor(Color.TRANSPARENT);

    int len=kwD_rectHolder.size();

    for(int i=0;i<len;i++)
    {
        paint.setAlpha(kwD_alphaHolder.get(i)); //
            trasparenza

        canvas.drawRect(kwD_rectHolder.get(i), paint);
    }

    //kwsavecanvas=canvas.save();
    //drawnBeforeFlag=true;
    }
}
```

B.3   Capturing Events

B.3.1   Touch Event Dispatch

Modified to the Touch dispatcher that is fired when the touch event arrives for this widget. Any touch operation is captured and stored within the root View scroll flag.

```
/**
   * Pass the touch screen motion event down to the target
     view, or this
   * view if it is the target.
   *
   * @param event The motion event to be dispatched.
   * @return True if the event was handled by the view,
     false otherwise.
   */
  public boolean dispatchTouchEvent(MotionEvent event) {

    //KiatWee Start Additions
    //Log.i("kwOLED", "dispatch Touch"); //KiatWee

    //Get the tge touch event to:
    //1) Turn on or off dimming on the screen during on
       Touch events
```

```java
//2) Turn on or off occulsion dimming under the finger
switch (event.getAction())
{
    case MotionEvent.ACTION_UP:
        kwpressuredata=0.0f;
        View rootView1=getRootView();
            rootView1.kiatweeOnScrollFlag=false;
                //KiatWee
            rootView1.invalidate();
            //Log.i("kwOLED", "Action Up. " +
                mContext.getPackageName());


        break;
    case MotionEvent.ACTION_MOVE:
        kwlocX=0.0f;
            kwlocY=0.0f;
            kwpressuredata=0.0f;
    case MotionEvent.ACTION_DOWN:

        kwlocX=event.getX();
            kwlocY=event.getY();

        if(OcculDimFlag)
        {
            kwpressuredata=event.getPressure(); //Turn
                on occulsion
        }
        else
        {
            kwpressuredata=0.0f; //Turn off occulsion
        }

            View rootView2=getRootView();
            rootView2.kiatweeOnScrollFlag=true;
                //KiatWee
    //rootView2.kiatweeOnScrollFlag=false;
        //KiatWee: Premeant Dim Turn On
            rootView2.invalidate();
    //Log.i("kwOLED", "Action Down. " +
        mContext.getPackageName());

        break;
}
//KiatWee End Additions

if (mInputEventConsistencyVerifier != null) {
    mInputEventConsistencyVerifier.onTouchEvent(event,
        0);
}
```

```
        if (onFilterTouchEventForSecurity(event)) {
            //noinspection SimplifiableIfStatement
            ListenerInfo li = mListenerInfo;
            if (li != null && li.mOnTouchListener != null &&
                (mViewFlags & ENABLED_MASK) == ENABLED
                    && li.mOnTouchListener.onTouch(this, event))
                    {
                return true;
            }

            if (onTouchEvent(event)) {
                return true;
            }
        }

        if (mInputEventConsistencyVerifier != null) {
            mInputEventConsistencyVerifier.onUnhandledEvent(event,
                0);
        }
        return false;
    }
```

B.3.2 Key Dispatch

Modified to the key dispatcher that is fired when any physical key/button event arrives for this widget. We filtered for volume button detection, volume up and down is captured and stored within the root View flip flag.

```
/**
   * Dispatch a key event to the next view on the focus
     path. This path runs
   * from the top of the view tree down to the currently
     focused view. If this
   * view has focus, it will dispatch to itself. Otherwise
     it will dispatch
   * the next node down the focus path. This method also
     fires any key
   * listeners.
   *
   * @param event The key event to be dispatched.
   * @return True if the event was handled, false
     otherwise.
   */
public boolean dispatchKeyEvent(KeyEvent event) {

    //KiatWee Start
```

```java
//Log.i("KiatWee", "dispatchKeyEvent event: " +
   event.getKeyCode());

//Get the keyevent KiatWee Additions
//Purpose: To do the dim mirror so that the dim area
   can flipped up or down
switch (event.getKeyCode())
{
   case KeyEvent.KEYCODE_VOLUME_DOWN:
      if(hasRectMirror)
      {    //kiatweeOnViewPortFlipFlag
         View rootView2=getRootView();
         rootView2.kiatweeOnViewPortFlipFlag=true;
                  rootView2.invalidate();
      }


      break;
   case KeyEvent.KEYCODE_VOLUME_UP:
      if(hasRectMirror)
      {    //kiatweeOnViewPortFlipFlag
         View rootView2=getRootView();
         rootView2.kiatweeOnViewPortFlipFlag=false;
                  rootView2.invalidate();
      }
      break;
}
//KiatWeeEnd

if (mInputEventConsistencyVerifier != null) {
   mInputEventConsistencyVerifier.onKeyEvent(event,
      0);
}

// Give any attached key listener a first crack at
   the event.
//noinspection SimplifiableIfStatement
ListenerInfo li = mListenerInfo;
if (li != null && li.mOnKeyListener != null &&
   (mViewFlags & ENABLED_MASK) == ENABLED
      && li.mOnKeyListener.onKey(this,
         event.getKeyCode(), event)) {
   return true;
}

if (event.dispatch(this, mAttachInfo != null
      ? mAttachInfo.mKeyDispatchState : null, this)) {
   return true;
}
```

```java
        if (mInputEventConsistencyVerifier != null) {
            mInputEventConsistencyVerifier.onUnhandledEvent(event,
                0);
        }
        return false;
    }
```

Appendix C

User Study

In this chapter, we reviewed the user study material used in solution. The user study material is created with the help of Tadashi Okoshi of Keio University.

C.1 Demographics Questions

The list of demographics data points and questions as follows:

1. The student's email address.

2. The student's gender

   - male

   - female

3. The student's school

   - Accountancy

   - Lee Kong Chian School of Business

   - Economics

   - Information Systems

   - Social Sciences

   - Law

4. The student's Year of study

- 1st year

- 2nd year

- 3rd year

- 4th year

5. How do you think of your proficiency level in smart phones?

   - I have never used smartphones

   - novice

   - average

   - expert

6. Which smart phones have you used?

   - iPhones

   - Android

   - Blackberry

   - Windows Phone 7 or 8

   - Others

7. Please write down which smart phones you are currently using.

8. I intentionally reduce my application usage on my smart phone to save battery power.

   - Strongly Disagree

   - Disagree

   - Neutral

   - Agree

   - Strongly Agree

121

9. How often do you recharge your phone during the day?

- Multiple times every day

- Once a day

- A few times per week

10. I feel that my phone has insufficient battery power for my usage patterns.

- Strongly Disagree

- Disagree

- Neutral

- Agree

- Strongly Agree

11. Application 1: How often do you use the "Facebook" (not "Messenger") app on your smart phone?

- More than 10 times per day

- Between 3 to 10 times per day

- Just a few times per week

- Just a few times per month

- Just 1 or 2 times per day

12. Application 2: How often do you use the "WhatsApp Messenger" app on your smart phone?

- More than 10 times per day

- Between 3 to 10 times per day

- Just a few times per week

- Just a few times per month

- Just 1 or 2 times per day

13. Application 3: How often do you use the "BBC News" app on your smart phone?

    - More than 10 times per day

    - Between 3 to 10 times per day

    - Just a few times per week

    - Just a few times per month

    - Just 1 or 2 times per day

14. Application 4: How often do you use the "Fruit Ninja" app on your smart phone?

    - More than 10 times per day

    - Between 3 to 10 times per day

    - Just a few times per week

    - Just a few times per month

    - Just 1 or 2 times per day

15. Application 5: How often do you use the "Gmail" app on your smart phone?

    - More than 10 times per day

    - Between 3 to 10 times per day

    - Just a few times per week

    - Just a few times per month

    - Just 1 or 2 times per day

16. Application 6: How often do you use the "Adobe Reader" app on your smart phone?

- More than 10 times per day

- Between 3 to 10 times per day

- Just a few times per week

- Just a few times per month

- Just 1 or 2 times per day

17. Application 7: How often do you use the "Firefox" app on your smart phone?

- More than 10 times per day

- Between 3 to 10 times per day

- Just a few times per week

- Just a few times per month

- Just 1 or 2 times per day

18. Application 8: How often do you use the "Kwaak" app on your smart phone?

- More than 10 times per day

- Between 3 to 10 times per day

- Just a few times per week

- Just a few times per month

- Just 1 or 2 times per day

C.2 Experiment 1 Questions

1. Q1: Is the application still usable (the unmodified version was shown at the start of this trial)?

- Strongly Disagree

- Disagree

- Neutral

- Agree

- Strongly Agree

2. Q2: Are the most important portions of the application still visible on the screen?

    - Strongly Disagree

    - Disagree

    - Neutral

    - Agree

    - Strongly Agree

3. Q3: Is the application still usable (the unmodified version was shown at the start of this trial)?

    - Strongly Disagree

    - Disagree

    - Neutral

    - Agree

    - Strongly Agree

4. Q4: Are the most important portions of the application still visible on the screen?

    - Strongly Disagree

    - Disagree

    - Neutral

    - Agree

    - Strongly Agree

5. What application did you just try?

- Facebook

- WhatsApp Messenger

- BBC News

- Fruit Ninja

- Gmail

- Aldiko

- Firefox

- Qwaak

6. Which mode do you prefer (2nd or 3rd)?

C.3 Experiment 2 Questions

1. Q1: I am willing to tradeoff some part of the screen in order to save overall power.

   - Strongly Disagree

   - Disagree

   - Neutral

   - Agree

   - Strongly Agree

2. Q2: I find the usability versus power savings tradeoffs with this approach acceptable.

   - Strongly Disagree

   - Disagree

   - Neutral

   - Agree

- Strongly Agree

3. Q3: Any comments/thoughts/feedback on the technique?

C.4   Screenshots of the user study

LiveLabs User Study: Demographics Survey

Please answer your questions and push "Submit" button to finish.

* Required

Your email address *

[   ]

Your gender *

○ Male

○ Female

Your school *

[Accountancy ▼]

Year of study *

[1 ▼]

How do you think of your proficiency level in smart phones? *

○ I have never used smart phones

○ novice

○ average

○ expert

Which smart phones have you used? *

☐ iPhones

☐ Android

☐ Windows Phone 7 or 8

☐ Blackberry

☐ Others

LiveLabs User Study: Experiment 1

Select your application that you just tried

What application did you just try? *

Facebook ▼

Usability of Application during the 2nd trial

Q1: Is the application still usable (the unmodified version was shown at the start of this trial)? *
1:Strongly disagree - 2:Disagree - 3:Neutral - 4:Agree - 5:Strongly agree

        1 2 3 4 5

Strongly disagree ○ ○ ○ ○ ○ Strongly agree

Q2: Are the most important portions of the application still visible on the screen? *
1:Strongly disagree - 2:Disagree - 3:Neutral - 4:Agree - 5:Strongly agree

        1 2 3 4 5

Strongly disagree ○ ○ ○ ○ ○ Strongly agree

Usability of Application during the 3rd trial

Q3: Is the application still usable (the unmodified version was shown at the start of this trial)?
1:Strongly disagree - 2:Disagree - 3:Neutral - 4:Agree - 5:Strongly agree

        1 2 3 4 5

Strongly disagree ○ ○ ○ ○ ○ Strongly agree

Q4: Are the most important portions of the application still visible on the screen?
1:Strongly disagree - 2:Disagree - 3:Neutral - 4:Agree - 5:Strongly agree

        1 2 3 4 5

Strongly disagree ○ ○ ○ ○ ○ Strongly agree

LiveLabs User Study: Experiment 2

Q1: I am willing to tradeoff some part of the screen in order to save overall power. *

1:Strongly disagree - 2:Disagree - 3:Neutral - 4:Agree - 5:Strongly agree

	1	2	3	4	5	
Strongly disagree	O	O	O	O	O	Strongly agree

Q2: I find the usability versus power savings tradeoffs with this approach acceptable. *

1:Strongly disagree - 2:Disagree - 3:Neutral - 4:Agree - 5:Strongly agree

	1	2	3	4	5	
Strongly disagree	O	O	O	O	O	Strongly agree

Q3: Any comments/thoughts/feedback on the technique?

Submit

Powered by Google Docs

Report Abuse - Terms of Service - Additional Terms

Bibliography

[1] B. Anand, K. Thirugnanam, L. T. Long, D.-D. Pham, A. L. Ananda, R. K. Balan, and M. C. Chan. ARIVU: Power-aware middleware for multiplayer mobile games. In *Workshop on Network and Systems Support for Games (NetGames)*, Taipei, Taiwan, Nov. 2010.

[2] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan. Adaptive display power management for mobile games. In *Proc. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, Bethesda, MA, June 2011.

[3] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: interfaces for better power management. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 23–35, New York, NY, USA, 2004. ACM.

[4] Athi. *Colour Image Segmentation*, Sep. 2009. `http:// www.mathworks.com/matlabcentral/fileexchange/ 25257-color-image-segmentation`.

[5] Azevedo, A., Cornea, R., Issenin, I., Gupta, R., Dutt, N., Nicolau, A., Veidenbaum, and A. Architectural and compiler strategies for dynamic power management in the copper project. In *Proc. Conf. on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA)*, Maui, HI, Jan. 2001.

[6] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proc. USENIX Technical Conference*, Berkeley, CA, June 2010.

[7] S. Chandra. Wireless network interface energy consumption: implications for popular streaming formats. *Multimedia Systems*, 9:185–201, Aug. 2003.

[8] N. Chang, I. Choi, and H. Shim. Dls: dynamic backlight luminance scaling of liquid crystal display. *IEEE Trans. VLSI Systems*, 12(8):837–846, Aug. 2004.

[9] X. Chen, J. Zheng, Y. Chen, M. Zhao, and C. J. Xue. Quality-retaining oled dynamic voltage scaling for video streaming applications on mobile devices. In *Proc. Design Automation Conference (DAC)*, San Francisco, CA, June 2012.

[10] W.-C. Cheng, Y. Hou, and M. Pedram. Power minimization in a backlit TFT-LCD display by concurrent brightness and contrast scaling. In *Proc. Conf. Design, Automation and Test in Europe (DATE)*, Paris, France, Feb. 2004.

[11] CyanogenMod. *CyanogenMod Android Fork*, Jul. 2012. `http://www. cyanogenmod.org/`.

[12] M. Dong, Y.-S. K. Choi, and L. Zhong. Power modeling of graphical user interfaces on oled displays. In *Design Automation Conference (DAC)*, San Francisco, California, July 2009.

[13] M. Dong, Y.-S. K. Choi, and L. Zhong. Power-saving color transformation of mobile graphical user interfaces on oled-based displays. In *Proc. Symp. Low Power Electronics and Design (ISLPED)*, San Fancisco, CA, Aug. 2009.

[14] M. Dong and L. Zhong. Chameleon: A color-adaptive web browser for mobile OLED displays. In *Proc. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, Bethesda, MA, June 2011.

[15] M. S. El-Nasr and S. Yan. Visual attention in 3d video games. In *Proc. Conf. on Advances in Computer Entertainment Technology (ACE)*, Hollywood, CA, June 2006.

[16] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 48–63, New York, NY, USA, 1999. ACM.

[17] S. Frintrop, E. Rome, and H. I. Christensen. Computational visual attention systems and their cognitive foundations: A survey. *ACM Trans. Appl. Percept.*, 7(1):6:1–6:39, Jan. 2010.

[18] Gartner. *Worldwide Sales of Mobile Phones Declined 3% in 3rd Quarter of 2012; Smartphone Sales Increased 47%*, Nov. 2012. `http://www.gartner.com/it/page.jsp?id=2237315`.

[19] F. Gatti, A. Acquaviva, L. Benini, and B. Ricco'. Low power control techniques for tft lcd displays. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '02, pages 218–224, New York, NY, USA, 2002. ACM.

[20] Google. *How Android Draws Views*, Nov. 2012. `http://developer.android.com/guide/topics/ui/how-android-draws.html`.

[21] Google Inc. *Android Operating System*, Jan. 2014. `http://developer.android.com/index.html`.

[22] H. Han, Y. Liu, G. Shen, Y. Zhang, and Q. Li. Dozyap: power-efficient wi-fi tethering. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 421–434, New York, NY, USA, 2012. ACM.

[23] T. Harter, S. Vroegindeweij, E. Geelhoed, M. Manahan, and P. Ranganathan. Energy-aware user interfaces: an evaluation of user acceptance. In *Proc. Conf. on Human Factors in Computing Systems (CHI)*, Vienna, Austria, Apr. 2004.

[24] Id Software. *Quake 3 Arena Source Code*, Jul. 2010. `http://ioquake3.org/`, Version 3.21.

[25] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(11):1254 –1259, nov 1998.

[26] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans. Pattern Analysis & Machine Intel.*, 20(11):1254 –1259, Nov. 1998.

[27] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami. Energy trade-offs in the IBM wristwatch computer. In *Proc. Symp. on Wearable Computers (ISWC)*, Washington, DC, Oct. 2001.

[28] kwaak3. *Port of Quake3 to Android.* http://code.google.com/p/kwaak3/.

[29] Lucsk-ho. *An Example of Fixations and Saccades Over Text.*, Feb. 2007. http://en.wikipedia.org/wiki/File:Reading_Fixations_ Saccades.jpg.

[30] Matthew Rollings. *Photo taken with a 200x digital microscope by Matthew Rollings [1]. It shows the RGB arrangement of the pixels on Google's Nexus One AMOLED screen with PenTile matrix pixel arrangement.*, Jun 2010. http://en. wikipedia.org/wiki/File:Nexus_one_screen_microscope.jpg.

[31] J. Melnick. *Smartphone Market Will Ring Up Largest Share of the OLED Display Market Through 2017.* Lux Research Inc., June 2012. http://goo.gl/brWOuZ.

[32] Monsoon Solutions Inc. *Monsoon Power Moniter.* http://www.msoon.com/ LabEquipment/PowerMonitor/.

[33] B. Noble, M. Satyanarayanan, and M. Price. A programming interface for application-aware adaptation in mobile computing. In *Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, MLICS '95, pages 57–66, Berkeley, CA, USA, 1995. USENIX Association.

[34] S. Pasricha, M. Luthra, S. Mohapatra, N. Dutt, and N. Venkatasubramanian. Dynamic backlight adaptation for low-power handheld devices. *Design & Test of Computers*, 21(5):398–405, Sept.-Oct. 2004.

[35] A. Rahmati, A. Qian, and L. Zhong. Understanding human-battery interaction on mobile phones. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, MobileHCI '07, pages 265–272, New York, NY, USA, 2007. ACM.

[36] P. Ranganathan, E. Geelhoed, M. Manahan, and K. Nicholas. Energy-aware user interfaces and energy-adaptive displays. *IEEE Computer*, 39(3):31 – 38, Mar. 2006.

[37] D. Shin, Y. Kim, N. Chang, and M. Pedram. Dynamic voltage scaling of oled displays. In *Proc. Design Automation Conference (DAC)*, San Diego, CA, June 2011.

[38] J. Shinar. *Organic Light-Emitting Devices: A Survey.* Springer, 2003.

[39] M. Stemm, P. Gauthier, D. Harada, and Y. H. Katz. Reducing power consumption of network interfaces in hand-held devices. In *Workshop on Mobile Multimedia Communications*, Princeton, New Jersey, Sept. 1996.

[40] K. W. Tan and R. K. Balan. Adaptive display power management for OLED displays. *ACM Comput. Commun. Rev.*, 42(4):485–490, Sept. 2012.

[41] K. N. Truong, J. A. Kientz, T. Sohn, A. Rosenzweig, A. Fonville, and T. Smith. The design and evaluation of a task-centered battery interface. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, Ubicomp '10, pages 341–350, New York, NY, USA, 2010. ACM.

[42] University of Michigan. *A Power Monitor for Android-Based Mobile Platforms.*
`http://powertutor.org/`.

[43] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 179–192, New York, NY, USA, 2009. ACM.

[44] X. Xie, H. Liu, S. Goumaz, and W.-Y. Ma. Learning user interest for image browsing on small-form-factor devices. In *Proc. Conf. on Human Factors in Computing Systems (CHI)*, Portland, OR, Apr. 2005.

[45] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 315–330, New York, NY, USA, 2010. ACM.