A Study and Implementation of VGA Multi-Resolution on Android Platform

Zhihua Xie[1], Ning Li[2], Lie Luo[3]

[1,2,3]Department of Computer Science and Technology
Wuhan University of Technology, Wuhan, China
[1]15827423370@163.com, [2]ln@whut.edu.cn, [3]986030252@qq.com

***Abstract:* At present, Android system can only support fixed resolution. This paper discusses a universal and concise solution to solve this problem, and the realization of Android VGA multi-resolution is described in detail. In order to solve the problem efficiently, we present a method with presetting resolution and reboot-confirmation strategy. This method involves multiple layers of the Android system architecture. We conducted a study on the Android system framework and graphics system architecture. Combined with Framebuffer driver technology of the Linux Kernel, the study applies the presetting resolution and reboot-confirmation method by using the existed framework. According to experiment result, the design achieved the expected goal, and it ensured the stable operation of the system. This solution is universal for all Android systems. In a way, it has reference value and practical value for making Android system support multi-resolution.**

***Keywords:* Android graphics system;Linux drivers; Framebuffer;HAL layer;Framework layer**

I. INTRODUCTION

Google's Android operating system (OS) led as the No. 1smartphone platform with 51.5 percent platform market share, according to comScore [1], and also spawned countless Android-derived devices. However, with the wide variety of derivatives, the graphics display system of Android is inadequate in some demand. Multi-resolution is needed on Android-derived devices, such as TV box, mini PC, thin client. They need to be connected with a plurality of screens with different resolutions.The main work of this paper is to design and implement system VGA multi-resolution function of Android, and to give the experiment result based on studying the Android system framework and graphics display system. In addition, it is related with the actual conditions of Android projects.

The design architecture of Android graphics system is the same as the common embedded graphics system, that is using the hierarchical structure underlying hardware, graphics engine and graphics applications [2]. As shown in Figure 1, the graphics display system of Android adopt the

Client/Server architecture. Surface Flinger [3] works as the server-side of the Android in the Framework layer, and the Client part is the applications. In the graphics displaysystem of Android, the layer of application includes several different types of application. The first one is used to show the applications with common UI interface. The UI interface is usually RGB format data which can be transmitted to the kernel without complicated process. The second is related with the video applications, such as the camera, video playback applications. They need to deal with massive YUV data. When we open these applications, YUV data chunks through the Overlay can be sent to the kernel to display. The third is that the applications needs the process of 2D, 3D. Have been processed by 2D, 3D, we will find there is no difference between the third type and Common UI application in the next process.

In Android graphics system, Surface plays a crucial role, it is a core part of the whole graphics system[4]. SurfaceFlinger's main function is to composer for the layer which come from the client side, and perform composer's process [5]. During the development process, Surface merging can be in the form of software, or you can also use Overlay mode (hardware mode). Whether software mode or Overlay mode, they all need to use the graphics display functions integrated in SurfaceFlinger. In addition, SurfaceFlinger can invoke ordinary Gralloc hardware abstraction layer to merge Surface, and can also call to OpenGL.

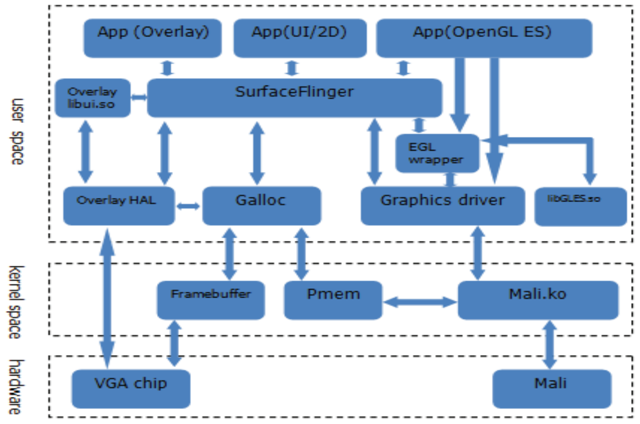


Figure. 1. Graphics System Architecture Diagram.

In the hardware abstraction layer of graphics display system, Grallocmodule encapsulates the Framebuffer device file operations. After the system starts to work, Gralloc will open the Framebuffer device of the kernel, configure the parameters of the device and initialize it. The location of the Overlay module and Gralloc are in the same level. The former can call to the kernel of the Framebuffer driver through system. At the same time, it provides the data channel and control channel to the upper layer. The hardware abstraction layer associated with OpenGL is also in graphics display system of Android, which is not open source. In the process of practical projects, the development will meet certain difficulties if we can't get Gralloc corresponding source code. In the design of system VGA multi-resolution function, we need to keep away from the difficulty that we are unable to modify Gralloc source.

In Android system, the display driver using the Linux standard display device driver Framebuffer[6]. In the Linux kernel, the Framebuffer device is an abstraction of the underlying hardware LCD. And the kernel offers the upper application Framebuffer interface operations, which makes the development of Linux user space can ignore the differences in the underlying hardware. Framebuffer driver involves three pieces of frame buffer application. And every frame buffer will register buffer address space, the refresh rate of LCD, the resolution of the output and so on. In the process of kernel development, we can modify the Framebuffer driver to specify these parameters to control the output of LCD.

II. Discussed problems

The graphics display system of Android System is not perfect, because the hardware abstraction layer and layer Framework of Android do not directly support system resolution of dynamic selection interface. The process of resolution initialization is as follows: 1.The corresponding display driver in the kernel is loaded after the Android device is powered on. Framebuffer driver registers the system resolution, and determines the size of the buffer system graphics, memory areas and other parameters. 2.Upper Linux kernel boot is complete, the Android system calls to display the hardware abstraction layer interfaces. The process will save Framebuffer information of the kernel, and remain the same in the system operation stage.

Thus, Android fixed resolution and the graphic buffer cannot be changed. From the analysis of the above process, we can find that there are four types of potential solutions to solve the problem of multi-resolution:

A. Solution 1

We can unregister the modules related to graphics display function of Android, and then register it again to switch resolution dynamically. On the condition that the hardware abstraction layer of Android has fixed the parameters about resolution, if we want to switch resolution when the system is running, we can unregister the graphics buffer of the system, the Framebuffer of the kernel and so on. Then we can set up a new system resolution and initialize the Android display system again. It will switch resolution of Android when we keep the system on. However, Android display system involves many modules, like the Linux kernel, the Android system layer. It is very complicated to implement this solution, and it is not conducive to rapid development.

B. Solution 2

The system default resolution can be stored in the hardware module. In the aspect of hardware design, we can choose to add EEPROM storage module and provide the corresponding communication interfaces of the hardware (such as I2C, SPI interface), so that the kernel driver can read and write system default resolution. And at the same time, we should create the corresponding device file of the kernel driver for the upper application to operate. Before Android devices run for the first time, we will put the default resolution information into the EEPROM memory. Then the devices will read the resolution information in the Linux driver when the devices run. Finally the information will be fixed in the hardware abstraction layer for system resolution. The Android application layer will also provide interfaces for users to switch resolutions. The resolution we choose will be written into the EEPROM memory so that the system will start with the default resolution next time.

C. Solution 3

We can use the internal Flash of system to store the default resolution. After the kernel boots, the display driver of kernel establish system resolution by reading the default resolution. But it is hard to guarantee to load the Framebuffer driver before Flash driver. That is to say when the Framebuffer driver need to read the resolution, file system must has been normally loaded. In terms of actual projects, some vendors will load the Flash driver in the form of a module after the Linux kernel boots for security and other factors. Although, display driver is loaded in the kernel, then the Linux kernel display driver cannot read information from Flash.

D. Solution 4

Some improvement upon the Solution 3 will be shown in Figure 2. The display driver is modular in Solution 4. The module of display driver will be loaded in the Linux kernel space after a period of time when the module of display driver is loaded. This ensures the Flash driver can be successfully loaded, and display driver can get the correct default resolution of Flash.

In Solution 1, resolution can be modified dynamically without restarting the system. However, it is very difficult to implement this solution in the actual process, and it is not conducive to rapid development, because the display architecture of Android is not suitable now. It needs to restart the system to read the default resolution in the Solution 2. At the same time, it also needs to add additional hardware. Solution 3 and 4 are software solution and more universal, without using extra hardware. Solution 4 is modified on the basis of Solution 3, the former uses internal Flash memory of system as a storage area for resolution information, loads the modular driver of Framebuffer in kernel space. Considering the method should be universal and feasible for development, Solution 4 is the ideal solution.

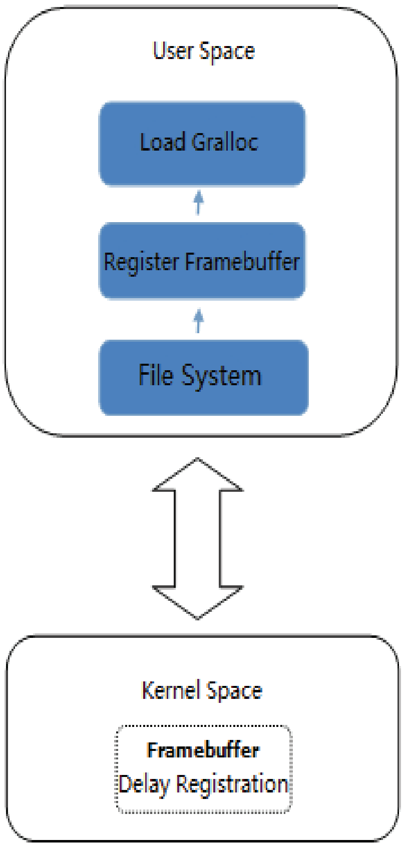


Figure. 2. Principle Diagram of Solution 4

Compared with the above four kinds of solutions, fake-resolution [7] is also a good solution of multi-resolution which has been proposed. When we use the fake-resolution, Android will start with a fixed resolution and output different images to different screens through the algorithm of zoom. It has the advantage that all the resolutions for images can be generated by algorithm without restarting system. But the images will be out of shape and the extra algorithm will reduce the efficiency of graphic system. In Solution 4, we need to restart Android to initialize graphic system again. However the output of Solution 4 will not be out of shape which we think is more important to users.

III. Design of Multi-Resolution

The content above has discussed the solution of multi-resolution of Android in detail. The following will elaborate realization of each layer of the system in detail.

A. Design of Kernel Driver

The bottom of the graphics display system architecture in kernel is the LCD controller, which is responsible for accepting the kernel of image data and outputting it to the LCD screen according to certain display timing. The kernel space of display architecture realizes fb driver and fbmem driver. The former is mainly used for registration, cancellation of the Framebuffer through the fb_info information.

Here fb_info includes Framebuffer information. Its member fb_var_screeninfo records the display control information which user can modify, such as the pixels of each row and column. Its member fb_fix_screeninfo records controller parameters which users can not modify, such as the physical address and length of the buffer [8]. fb_info is visible to fbmem driver, and the latter package the operations of fb_info. fbmem drivers has provided a set of common file interfaces to operate LCD, such as read, write, ioctl etc. Framebuffer determines the resolution during the process of registration and the LCD output timing related will be recorded into the fb_info. Therefore, we need to read the preset resolution and initialize the corresponding timing parameters to determine the system resolution before the Framebuffer driver registers.

The driver used to register Framebuffer is amended as follows: Create configuration file vga.conf of the preset resolution to the Android /data partition of file system, and read the configuration file to obtain the corresponding timing parameters of VESA[9] standard.

The format of configuration file: "lmode=x, rmode=y, flag=0/1". As shown in Table 1, when the using flag is 0, it indicates that no new resolution is set; When the using flag is 1, it stands for a new resolution is set. After the device is restarted, a new resolution mode will be set for the system resolution mode.

TABLE I: File Format of Resolution Configuration

Old Mode (lMode)	New Mode (rMode)	Using Flag (flag)	Next Action
1280x720	1280x720	0	Use lMode
1280x720	1920x1080	1	Use rMode

When compiling the kernel source code, Framebuffer need to be compiled into a driver module(Lcdc.ko), and it should be loaded in the user space of Linux. After Linux starts, Linux user space will execute "init" process as the first program for Android initialization. The "init" process will parse the init.rc to execute the corresponding operations for initialization. Therefore, we can use the command for loading driver module to load Lcdc.ko in init.rc:

```
    insmod /system/lib/modules/NandFlash.ko
    insmod /system/lib/modules/Lcdc.ko
    insmod /system/lib/modules/Mali.ko
```

NandFlash.ko is a Flash drive module. System will mount a corresponding system partition after NandFlash.ko is loaded. While the Mali.ko is the graphics driver. The loading time of Lcdc.ko module must be earlier than that of Flash driver module and later than the Mali.ko graphics driver. Only loading the Flash driver is completed, can the configuration file of resolution be read under the /data partition. At the same time, if the Framebuffer driver is not registered, the graphics driver will be incorrectly loaded. Therefore the loading time of Lcdc.ko must be between them. At this point, before the Android system Gralloc module is loaded, we can read the configuration file of resolution and register the Framebuffer to the kernel.

B. Design of Hardware Layer

Android HAL(hardware abstraction layer) is running in user space of Linux, which is the abstraction and encapsulation to the device file operations of kernel, and it provides interfaces to upper layer. The HAL methods are implements in C/C++ and stored in the native library file in the format .so for Linux system[10]. The HAL module has its own ID number, and each HAL module can be compiled into a dynamic link library. The JNI layer uses the generic function hw_get_module(ID, &module) in HAL to query module and loads module into memory, and obtain the function symbols. Thus it can realize Java programs of the framework layer call to C or C++ methods of the hardware abstraction layer. After the JNI layer loads the HAL module, the HAL functions can be easily used in JNI. Meanwhile the functions can be provided to the framework layer.

The main work of hardware abstraction layer is to encapsulate the operations of /data/vga.conf file. We should add VGA hardware abstraction layer module based on the framework standard of the hardware abstraction layer and encapsulate three methods: setMode, keepCurMode, recoveryOldMode. The setMode function is used to set a new resolution. The keepCurMode function is used to save the current resolution. The recoveryOldMode is used to restore the previous resolution. The struct below includes the three abstract methods defined the HAL:

```
struct vga_device_t{
 ...
 int (*setMode) (struct vga_device_t *dev, char *val);
 void (*keepCurMode) (struct vga_device_t *dev);
void (*recoveryOldMode) (struct vga_device_t *dev);
};
```

C. Design of Framework Layer

Due to the presence of the JNI layer, a connection is established between native interface and the Java world. The JNI layer further encapsulates the functions of VGA hardware abstraction layer module, SetMode, keepCurMode, recoveryOldMode, which turn into three JNI methods.The framework layer of Android calls to the interfaces of hardware abstraction layer through JNI and it provides a uniform interface for the development of Android application layer. Then we can use these interfaces in application layer.

However in Android system, hardware service (vga.conf can be regarded as the abstract file of hardware, and this design is more universal) is generally running to provide services for a variety of applications in a separate process. Therefore, the communication between applications and this hardware service need to be realized through a proxy. We need to define the communication interface IvgaService.aidl and realize these interfaces in VgaService.

D. Design of Framework Layer

The application layer uses the user-defined widget of Listview to show different resolution options, sets selection events which is related to ListItem, and calls to AlertDialog to confirm whether the user restart the machine to set new resolution or not. After system shows AlertDialog, if the user has not made relevant selection within 15 seconds, system will cancel the tips; If the user chooses to restart in time, system will call to the Vgaservice service of framework layer to set up a new resolution.

Strategy of reboot-confirmation: After system sets up a new resolution and restarts, it is necessary to restore the original resolution automatically in a certain time, for display device may not necessarily support the new resolution. Otherwise we must find another suitable display device to set the resolution of our system. This design enables users who make the wrong choice to restore the original resolution.

In order to realize the strategy of reboot-confirmation, we choose to register a Boot broadcast receiver in the system. After the system starts, the Boot broadcast receiver will receive a Boot broadcast, and the configuration file will be read to determine whether calling the AlertDialog options dialog or not. After a AlertDialog appears, if the users manually select the confirm to save current resolution, AlertDialog will not appear to users at the next time when system starts. Otherwise, the system will reboot after 15 seconds and restore the original resolution.

Thus, the realization of basic user interaction has been realized through multi-resolution selection and strategy of reboot-confirmation. It can guarantee the system steadily run after changing the resolution.

IV. EXPERIMENT

We conducted an experiment to find out whether Solution 4 can steadily realize system multi-resolution functions or not. Figure 3 is the rendering when system resolution is 1280x720. Figure 4 is resolution selection interface in the application. The system will restart automatically after the user select a resolution and conform it.

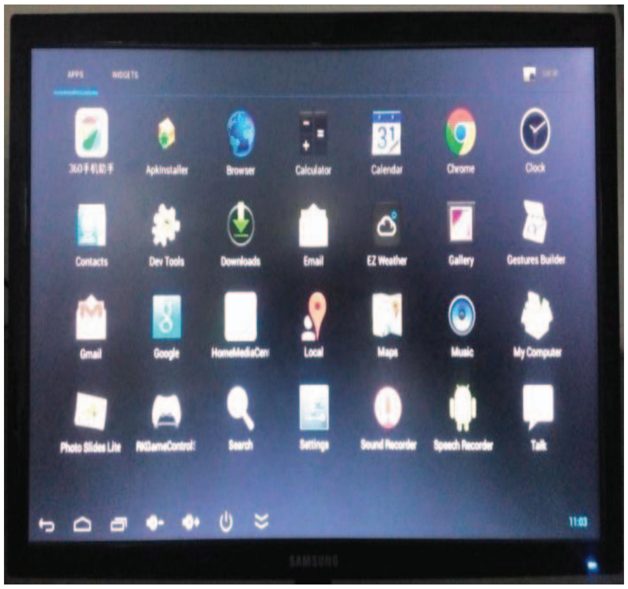
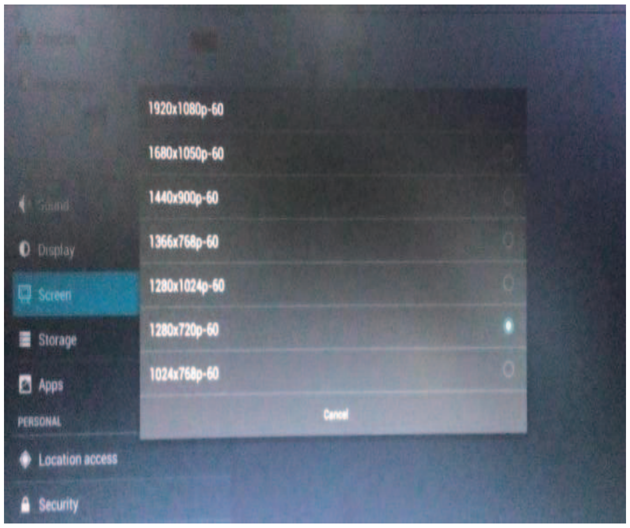
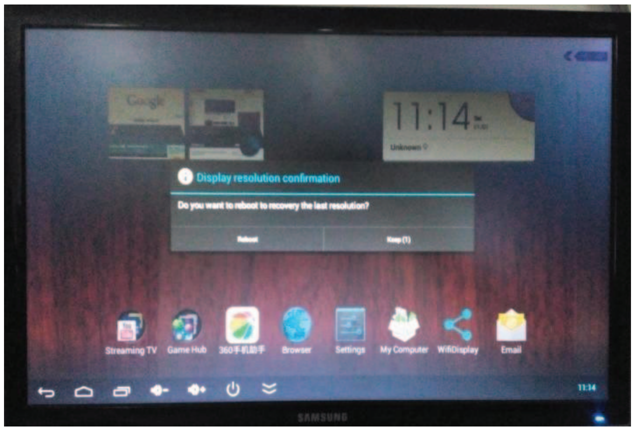


Figure. 3. 1280x720p Resolution



1920x1080p-60
1680x1050p-60
1440x900p-60
1366x768p-60
1280x1024p-60
[✓ 1280x720p-60]
1024x768p-60

Figure. 4. Selection Interface of Resolutions



Display resolution confirmation

Do you want to reboot to recovery the last resolution?

Reboot | Keep (1)

Figure. 5. Conformation of Resolution

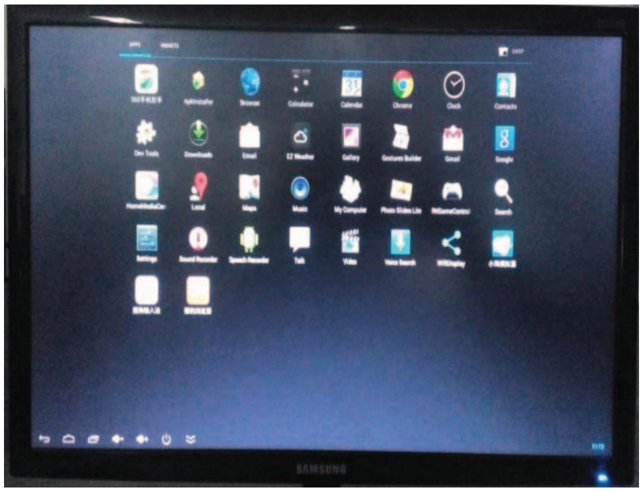


Figure. 6. 1920x1080p Resolution

Figure 5 is the prompt dialog box. It appears after the system restarts and receives the Boot broadcast. The dialog box is used to confirm whether to use a new set of resolution or not. Figure 6 is the rendering when system resolution is 1920x1080. It indicates the system completed the modification of resolution after we choose and confirm the 1920x1080 resolution. Experiment proves that the solution is stable and feasible.

V. CONCLUSIONS

The paper has proposed a universal and concise solution about multi-resolution, and discussed the implement of the solution from the perspective of system. Multi-resolution selection is an indispensable function for personal computers. With the rapid development of mobile equipment, Android plays a important role. More and more customers regard it as a small PC system. It is necessary for Android to support multi-resolution, which is an improvement for the Android platform. Before Google releases more perfect Android code, this paper presents a more general solution for multi-resolution and it has the certain reference significance.

REFERENCES

[1] Eddy, Nathan, Android First in Smartphone OS Market Share, Apple Top OEM, eWeek, 2014, (1): 1-1.

[2] Wang Xianchun, Cai Jianhua, Hu Weiwen and Guo Jierong, Research on Embedded Graphical System Based on ARM and Linux, Micro Computer Information. Vol. 23(2007).

[3] SurfaceFlinger, https://code.google.com/p/ics-custom-services/source/browse/Surfaceflinger/SurfaceFlinger.cpp.

[4] Tieyuan Yin, Mingjun Su, Research and Implementation of the Android graphics system on the ARM platform, Applied Mechanics and Materials Vols. 263-266, 2013, pp 1442-1446.

[5] ZHU Xiao-dong, YU Song-nian, The Horizontal and Vertical Screen to Switch the Display Based on the Android, *Applied Mechanics and MaterialsVols*. 313-314, 2013, pp 1393-1397.

[6] Yu Hongdan, YANG Bi, Research and Application for Framebuffer in Embedded GUI, Journal of Chengdu University of Information Technology. Vol. 25, 2010.

[7] LIU Kun, NIU Wen-liang, CHEN Jing-xia, The Improved Algorithms Based on Android Display System in HDMI Stick, Science Technology and Engineering. Vol.13, No.27, September 2013.

[8] Li Zhi-hui, Xian Jinlong, The monochrome LCD Driver Design and Implementation Based on Framebuffer, TELKOMNIKA, Vol.11, No.9, September 2013, pp. 4918~4924.

[9] VESA, https://en.wikipedia.org/wiki/VESA.

[10] Maoqiang Song, Wenkuo Xiong, Xiangling Fu, Research on Architecture of Multimedia and Its Design Based on Android, Internet Technology and Applications, 2010 International Conference on.