IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 4 Issue 3, March 2017
ISSN (Online) 2348 – 7968 | Impact Factor (2016) – 5.264
www.ijiset.com

A Comparison between DirectFB and X11 Graphics Library

Sung-Bong Jang[1], Jeong-Mi Kim[2] and Mi-Young Choi[2]

[1]Department of Industry-Academy Cooperation, Kumoh National Institute of Technology,
730701 Gyeong-Buk, South Korea,
[2]Department of Computer Science & Engineering, Korea University,
136713 Seoul, South Korea

Abstract

A performance is a very important criterion when developers implement graphics software on Linux-based mobile system. This paper describes a comparison between X11 and DirectFB graphics library to help them to choose most suitable one fit to their purposes. To compare the libraries, we have conducted an experiment where processing time for GTK widget has been measured on Intel Xscale Processor. GTK is a multi-platform toolkit for creating graphical user interfaces. This is suitable for projects ranging from small on-off projects to complete application suites because it offers a complete set of widgets. By doing this, we would like to get a hint about which graphic library will represent better performance for Linux based embedded systems.

Keywords: *Graphic Performance, Mobile System,*

1. Introduction

In Linux-based embedded system, the graphic performance is an important consideration when developers implement software. Today, some smartphone system adopts WinCE as their operating system. Some developers may be familiar to WinCE's user interface library. However, WinCE is not an open source. It just provides bunch of limited interfaces and applications program interfaces (APIs), and they cannot explore whole inner system structure, and it is impossible to change them to adapt our own systems. In terms of operating system, Linux can be a great alternative. Linux is a perfect open source and they can explore its kernel, and develop their own code by hand. Linux provides two kinds of graphics libraries: X11 and DirectFB. The developers must choose one between them by considering their performance.

X11 is a graphic library system working on Unix operating system, which provides the basic framework for a GUI programming: drawing and moving windows on the screen interacting with a mouse and keyboard. It does not require the user interface, but each client software handle this. We can use the X11-Basic interpreter as a shell. Also for execution of CGI-Scripts. A pseudo compiler is shipped with the interpreter so that you can make stand-alone binaries out of your programs. You can do any data manipulation and you may use external functions and libraries. At least the X11-Basic interpreter is fast and small.

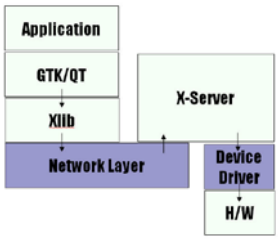


Fig. 1 X11 Graphic System

DirectFB is a thin graphics library to accelerate hardware graphics processing and to support window programming for developers. It is working on top of the graphic framebuffer in the Linux. It keeps and manages its own video graphic memory. In order to control the graphic devices, DirectFB use the kernel interface function provided by the framebuffer device (/dev/fb). This point is a flaw because it always must interwork with a working framebuffer driver. Some hardware chipsets provides developers with a special driver at the kernel layer. If chipset do not provide working driver, they have to use a special framebuffer. Fig. 2 illustrates the basic working architecture of DirectDB.

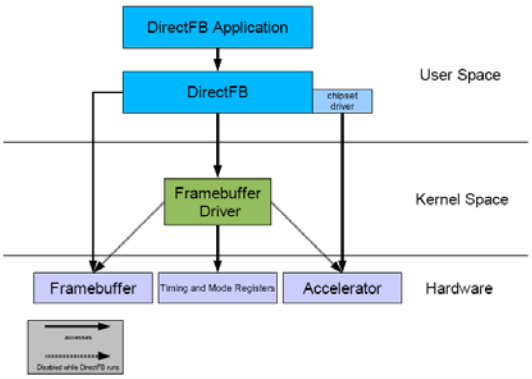


Fig. 2 DirectFB Graphic Structure

This paper describes a performance comparison between X11 and DirectFB over Intel Xscale Processor. We used GTK+ GUI Widgets for test data. GTK+ is a multi-platform toolkit for creating graphical user interfaces. GTK+ is suitable for projects ranging from small on-off projects to complete application suites because it offers a complete set of widgets. By doing this, we would like to determine which graphic library will support better performance for Linux based embedded systems.

2. Related Works

The researches about graphics performance have been much conducted in the past. I.G. Thakkar et al. [1] proposed an enhanced DRAM architecture to improve throughput and energy consumption. The approach is based on the three-dimensional bank organization. In their experiments, they have reached 77.7% delay reduction when processing graphic data. However, in reality, their experiment is based on the simulation. When it is done in real systems, the results may be different. S.-F. Hsiao et al. [2] presented a scheme to reduce the energy consumption of the performance of the Open GL ES 2.0 mobile graphic standard. In the scheme, they stopped using useless execution based on memory partition approach. Therefore, when accessing memory, only a small part of memory is accessed, not whole. To evaluate the approach, they have implemented the real system and they have attained much improvement in processing speed and energy consumption when compared with the conventional scheme. S. F. Oberman [3] analyzes and discuss about an impact of the high latency of the floating-point division in graphic operation. Through this research, they tried to help the designers to make decisions about implementation issues of the graphic hardware. J. Cohen et al. [4] describes the effect and performance of a parallel execution based on graphic processor unit (GPU) when processing graphics data. The GPU used in their discussion is a CUDA processor, which has been developed Nividia. The processor supports multi-threaded and multi-processors. Each processor in CUDA allows the number of 1,024 threads to be forked and executed at the same time. Thread handling and scheduling can be done via hardware, and thus it minimize the processing overhead. D. W. Wong et al. [5] discusses about a simulation scheme to help chip designers to efficiently predict the performance before releasing the graphic chip. Z. Yang et al. [6] discusses about a approach to accelerate the graphic processing based on multi-chip solution. L. Ilya et al. [7] describes a case study about the approach based on GPU to improve the image processing performance.

3. Experimental Setup

For evaluating X11 and DirectDB, GTK+ widgets have been as testing data. GTK+ widgets is a graphic library that can be worked on X11 and DirectFB, which consists of three basic libraries: Glib, Pango, and ATK. Glib refers to the basis library that provides low-level interfaces for operating system functionalities such as threads, memory management, and file system management. Pango provides basic simple graphic functionalities such as text rendering, font management, and layout. The ATKs are interface libraries through which other softwares can access to test widgets easily. The basic procedures to create a widget are as follows.

- Create a widget by calling the function “new()”. This function returns a pointer of a GTK data type.
- Set the appropriate signal handlers to be used
- Assign the initial values to the widget’s attributes
- Combine the created one into a container class by utilizing gtk_container_add() or gtk_box_pack_create function
- Display the created one by using gtk_widget_show() function

The widgets that are used in the experiments are illustrated in Fig. 3.

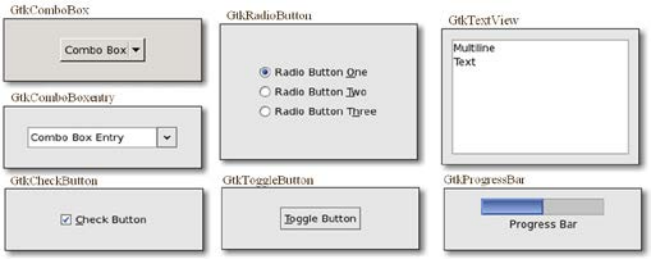


Fig. 3 Widgets used in performance evaluation

To evaluate each graphic library using the basic widgets, we have used the special purpose software, grkperf(). This is a useful software designed to test GTK widgets. By using this tool, we can create common testing environment to execute predefined widgets and define the speed between device and platform. This tool can be helpful to solve the following problems:

- Comparison of the processing time for my hardware or software platform
- Comparison of the processing performance of GTK with different themes

IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 4 Issue 3, March 2017
ISSN (Online) 2348 – 7968 | Impact Factor (2016) – 5.264
www.ijiset.com

- Detecting any remarkable slow widget when using my GTK library.
- Getting information about performance enhancement when GTK is used together with X-server
- Determining whether GTK+ can be used efficiently for different embedded device.

The resulting processing time can be seen at the command prompt at the Linux. In the test, we run the widgets each 100 times and some of widget operations are follows.

- GtkComboBox (on_idle_gtkcombobox_test)

  This test repeated for 10 entries from "Selection 1" to "Selection 10". This test opens and closes GtkComboBox 10 times while selecting next entry.

- GtkComboBoxEntry (on_idle_gtkcomboboxentry_test)

  This test repeated for 10 entries "Selection 1"..."Selection 10" of combo box. This test opens and closes GtkComboBoxEntry 10 times while selecting next entry.

- GtkSpinButton (on_idle_gtkspinbutton_test)

  This function is used to test spin button press. In the test, maximum count number is set to be 1000, and thus when the counter reaches 1000, it is set to be 0.

- GtkProgressBar (on_idle_gtkprogressbar_test)

  The purpose of this function is to evaluate the performance of the progress bar on both platform. In the test, the bar increases by 1%. Whenever bar gets full, its value is set back to 0.

- GtkToggleButton (on_idle_gtktogglebutton_test)

  This is used to test toggle key on and off.

- GtkCheckButton (on_idle_gtkcheckbutton_test)

  This is used to test check button.

- GtkRadioButton

  This is used to simulate the on and off of the radio button.

- GtkTextView

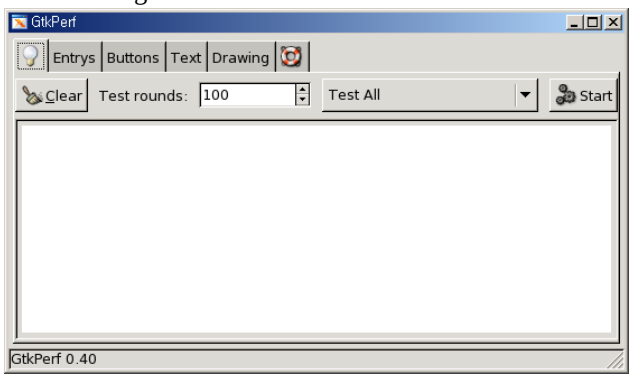
  Using this function, we can test adding and scrolling the simple text input.

- GtkDrawingArea

  Using this function, we are able to test the drawing actions that include line drawing, circle drawing, text input, and pixel buffer operation.

3. Results and Analysis

The testing screen is illustrated in Fig. 4. Fig. 4-(a) represents an initial window for setting various options before starting the experiment. In this window, the user can set the number of test rounds and the widget that they want to test. If the users want to test all widgets, they select “test all” option from lists. Fig. 4-(b) shows the testing results that shows execution time for each widget. In this way, we have collected running time of each widget and total execution time for X11 and DirectFB. We repeated the test for each widget ten times.



(a) User Interfaces for Testing Widgets in GtkPerf()

```
/mnt/nfs/local/bin # ./gtkperf -a
GtkPerf 0.40 - Starting testing: Fri Jan  2 03:37:06 1970

GtkEntry - time:  0.01
GtkComboBox - time:  3.57
GtkComboBoxEntry - time:  2.91
GtkSpinButton - time:  0.43
GtkProgressBar - time:  0.30
GtkToggleButton - time:  0.91
GtkCheckButton - time:  0.93
GtkRadioButton - time:  1.20
GtkTextView - Add text - time:  3.63
GtkTextView - Scroll - time:  2.51
GtkDrawingArea - Lines - time:  2.17
GtkDrawingArea - Circles - time:  3.28
GtkDrawingArea - Text - time:  4.43
GtkDrawingArea - Pixbufs - time:  0.83
 ---
Total time: 27.16
```

(b) Widgets Execution Screen for Processing Time Measurement

Fig. 4 an Execution Screen of the Test

Fig. 5 and Fig. 6 shows the experimental results. Fig. 5 represents the results with double buffering in DirectFB, and Fig. 6 shows those without double buffering in DirectFB. The line with red-color represents average time of second over DFB, and blue-color line represents the processing time of each widget over X11. From the results, we can see that there is a great difference in performance when double buffering is used in DirectFB. However, there

is a little difference when DirectFB does not use a double buffering. In this project, we implemented software double buffering only in the DFB and this is used in conjunction with DMA in intel processor. But in case of X11, we did not use double buffering in video frame processing. When the double buffering is disabled in DirectDB, the performance result is similar to that in X11 as shown in Fig. 6. We see that X11 shows better performance always for both case.

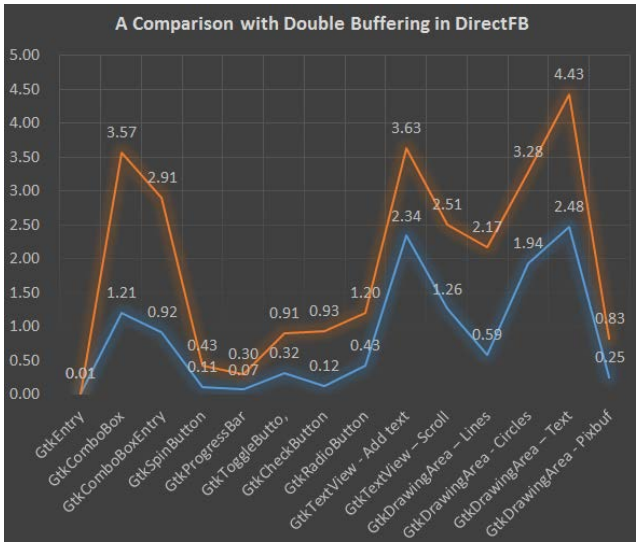


Fig. 5 Experimental Results *with* Double Buffering in DFB

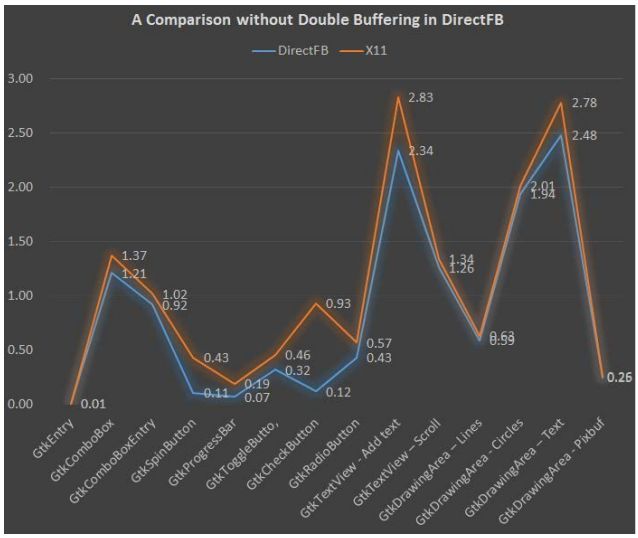


Fig. 6 Experimental Results ***without*** Double Buffering in DFB

In this work, we analyzed the causes of the performance difference from the following two aspects:

- Graphic Software Structure

Basically, X11 library has a network graphic software architecture. This means that if we want to render graphic data to the display, the data should go through network layer. Furthermore, the rendering operation is done based on client-server architecture. This architecture is one of causes which degrade the performance of complex widget processing or three-dimensional graphic that require faster graphic speed. In this project, we predicted that DFB would have faster widgets running time, but the result was opposite. From the results, we concluded that network X11 graphic architecture have made small impact on the performance. If we would have used longer test rounds value or test 3D graphic data, it would make greater impact on the GTK performance.

- Double Buffering

Double buffering is a well-known scheme that is used to get rid of useless artifacts during rendering. The implementation can be done both in hardware and software. In computer graphics, LCD monitors display the visible image on the screen at seventy times per second. This makes it difficult to apply changes to the screen data without the screen rendering the results before the rendering operation finishes. Therefore, useless video noises like flickering and tearing appear in the screen. To remove the noises, software double buffering utilize a system RAM where all rendering data are written into. The process is as follows. First, when a rendering operation is finished, the whole data, or a part of the data, is replicated into the system RAM. Next, whenever they render the video, replication is done in advance before the screen's beam reaches. In the result, the video noises can be avoided if replication is faster than screen beam. The advantage is that the approach causes a higher overhead than in hardware. The hardware implementation for removing noises is called as page flipping. In the scheme, two kind of graphic pages are used. When rendering video, at one instant, only one page of video is being active for display, and at the same time, another background page is being rendered. When the rendering has been finished, the roles of each video data are switched, so that the previously displayed vodeo data is now being updated, and the previously updated video data is now being displayed. By using the scheme, noises will not appear as long as the video data are appropriately switched over during the screen's vertical blank period when there is no data to be rendered. However, the required amount of VRAM size is twice of the single video data scheme.

From the experiment, double buffering is the main cause of performance degradation in DFB which is two times slower than in X11. Therefore, we concluded that

double buffering is more major factor than graphic software architecture which affects the GTK perform performance because the result shows that X11 is two times faster than DFB. Table 1 shows the qualitative analysis of the results of GTK widgets running.

Table 1: The factors affecting the performance

Factors	DirectFB	*X11*
Graphic System Architecture	No	Yes
Double Buffering	Yes	No

From this table, we can see that a double buffering is a major factor and graphic system architecture is a minor factor.

4. Conclusions

The contribution of this paper is to get some hints about which graphic library has a better performance between X11 and DirectFB. The future work is to implement the double buffering in X11, and conduct the comparison with DirectFB. The next work is to compare and evaluate two libraries on mobile processor such as advanced risc machine (ARM) rather than Intel XScale processor.

Acknowledgments

This paper was supported by Research Fund, Kumoh National Institute of Technology.

References

[1] I. G. Thakkar, and S. Pasricha, “A novel 3D graphics DRAM architecture for high-performance and low-energy memory accesses,” in Proceedings of the 2015 33rd IEEE International Conference on Computer Design (ICCD),” 2015, pp. 467 – 470.

[2] S.-F. Hsiao, S.-Y. Li, and K.-H. Tsao, “Low-power and high-performance design of OpenGL ES 2.0 graphics processing unit for mobile applications,” in proceedings of the 2015 IEEE International Conference on Digital Signal Processing (DSP), 2015, pp. 110 -114.

[3] S. F. Oberman, and M. J. Flynn, “Design issues in division and other floating-point operations,” IEEE Transactions on Computers”, 1997, vol.46, no.2, pp. 154 – 161.

[4] J. Cohen, and M. Garland,”Novel Architectures: Solving Computational Problems with GPU Computing,” Computing in Science & Engineering, 2009, vol.11, no.5, pp.58 – 63.

[5] D. W. Wong, and M. Aleksic, “Performance prediction on graphics hardware using software simulation,” in Proceedings of the Canadian Conference on Electrical and Computer Engineering 2001, vol. 2, pp.1235 – 1240.

[6] Z. Yang, M. Rahman, and S. Mourad, “Signal integrity and design consideration of an MCM for video graphic acceleration,” IEEE Transactions on Advanced Packaging, 2001, vol.24, no.3, pp.309 – 316.

[7] L. Ilya, S. S. Ivanovich, K. Y. Viacheslavovich, D. I. Alekseevna, P. V. Vladimirovich, S. L. Valerievna, “ On the possibility of image processing acceleration with the graphic processing unit,” in Proceedings of the 2016 Second International Young Scientists Forum on Applied Physics and Engineering (YSF)”, 2016, pp. 214 – 217.