



Achieving High Performance in Embedded Graphics Without Specialised Hardware.

FST White Paper

Robi Karp / 5th June, 2008

www.fst.net

Copyright © 2008 Fluffy Spider Technologies – All Rights Reserved

No part of this document may be reproduced in any form except with the written permission of FST.

This document is Confidential.

EXECUTIVE SUMMARY

Consumer expectation of new products is focussed towards not only the functionality of the product, but also towards the method of delivery of the functionality. The consumer expects a rich, dynamic experience which is responsive with natural animations and useful visual effects. However as consumer expectations increase, there is tremendous pressure on OEMs to continuously lower the total production costs. This paper describes some of the optimizations achieved by *FST FancyPants* so that it is able to deliver a consistently high performance user experience without the associated high end hardware requirement.

FancyPants is a resource-efficient platform for achieving outstanding graphical results in run-time environments that have limited resources. Designed from the ground up for the embedded market, *FancyPants* is a highly optimised suite of products that removes unnecessary overhead to deliver a faster and smaller platform.

DISCLAIMER

The views, analyses and conclusions presented in this document are solely those of the author and do not necessarily represent the views of any other party. While all possible care was taken to ensure the accuracy of the information in this document no warranty is provided with respect to that information. FST shall not be liable for any errors or omissions contained in this document. FST reserves the right to modify this document at any time in any way it wishes.

© COPYRIGHT FST

This document is the property of Fluffy Spider Technologies. This document may not be copied, in part or whole, without the written permission of Fluffy Spider Technologies.

ABOUT FST

FST has been developing embedded software since 1995. FST's deep knowledge of software efficiency coupled with embedded systems and Linux proficiency led to the creation of FancyPants, the ruthlessly efficient platform for embedded Linux graphics and multimedia, first launched in 2003.

Working closely with global corporations including VeriFone and Toshiba, FST has bred an embedded graphics community through its partners and is working with customers world wide.

FancyPants is the next generation high performance multimedia and graphics platform for embedded applications. Comprising a powerful SDK

to simplify the development of rich user interfaces, and an optimised runtime environment, for maximum performance, FancyPants is used by innovative creators of consumer, commercial and industrial devices.

WHY DO WE NEED EFFICIENCY?

The user interface is the ultimate driver for applications and services. In the hands of the consumer, an intuitive and dynamic interface will guide and manage, leading the consumer by visual cues, available options and enough feedback.

Additionally, the interface will make the consumer *want* to use a device and service. Giving the consumer familiar themes and pleasing visuals appeals to the senses and reaches out to more than just a choice based on functionality.

To be able to deliver this kind of user experience means that the *framework*, or *platform*, on which the user interface is built supports the requirements of this user experience. In other words: To have a rich and dynamic interface at the highest, consumer facing, level, you need to have the capabilities to support this interface at the building block, lowest, level.

Software for embedded devices is different to software for the desktop or for web applications. Resources, such as CPU speed, available memory and even the availability of chips like floating point units, are scarce. Being able to utilize extra CPU cycles, save memory copies, cache data, defer disk writes and offload processing to specialised hardware are just a few optimisations that can be made. Having those optimisations as a part of the platform are critical.

NOT THE DESKTOP

Desktop computers are richer and much more powerful platforms than embedded devices. They have variable configurations with variable screen sizes and applications must support the keyboard and mouse as the primary input modalities.

Rigid Hierarchical / Packing Model

Desktop GUI toolkits usually have a concept of a top level window into which other widgets are placed, for example `GtkWindow`. This derives from the fact that the desktop provides a rich windowed environment which allows for multiple windows.

To manage the placement of widgets within a window the toolkit provides packing widgets. Packing widgets are invisible containers that help with widget layout. Packing widgets are usually constrained within the window, and the widgets packed within are also constrained within the parent packing widget.

Having enforced layout of this sort is at odds with creating a dynamic and unique experience for the user.

FancyPants also has container widgets. However unlike desktop container widgets, *FancyPants* containers are neither mandatory nor ultimately constraining for the application as they can be placed anywhere on the canvas and be moved, scaled and have visual effects applied to them.

Resizing and Layout

A GUI toolkit for the desktop must be flexible in terms of sizing and layout. This flexibility is usually achieved by the layout algorithms employed by

the widgets¹, allowing applications to look good and function correctly at arbitrary scaled window and pixel sizes. In most desktop implementations these algorithms calculate the optimal spacing between widgets.

As noted by Biggs [1] and Janik [2] GTK+ employs the following algorithm for widget resizing, also done at widget creation time. A first pass through the entire widget tree aggregates the size requirements from the widgets. A second pass divides up the available space between the visible widgets according to container specific partitioning of their respective child areas. Some widgets break with the resizing contracts when they occasionally re-queue additional resizes, and apply resizing thresholds which depend on previous size negotiations. The outcome are resizing hysteresis effects, or lack of responsiveness.

On a desktop system, where CPU and memory are not limited, these real time calculations and scaling is possible. However when a desktop toolkit is re-targeted towards embedded device development these algorithms slow down display and application performance by needing to recalculate layout periodically or at each redraw. Furthermore, because layout code is a fundamental part of the desktop toolkit, it cannot be bypassed or removed.

On an embedded device there is a fixed screen size and usually one, or at most two layouts: portrait and landscape. The level of flexibility required on the desktop is unnecessary on the embedded device.

Because *FancyPants* does not have the desktop legacy it does not have any unnecessary code. Desktop cases are not considered and layout code paths are optimized.

1 A “widget” is an on-screen control for example a button, scrollbar or drop-down list.

NO UNNECESSARY OPERATIONS

With limited CPU and memory bus bandwidth, duplication of data and unnecessary operations are not only wasteful, they can adversely affect the performance of the entire system.

The *FancyPants* graphics engine maximizes data throughput and minimizes the time needed to update the display. The intention is to push pixels to the screen faster and redraw only when necessary to offer a more responsive user experience.

Single Copy Pipeline

It is easy to create inefficient code. Some programming languages provide high level abstractions to multiple low-level operations and often these operations involve the copying of data during the construction process.

Redundant copies of data not only increase the working memory required, the very action of copying is expensive.

A 100 pixel by 100 pixel image on a 32 bit display consumes around 40 kilobytes of memory. On a QVGA (640x480) display there can be 24 such images on-screen simultaneously leaving room for spacing, taskbars, etc. This totals to 960 kilobytes of memory. If the images are copied 4 or 5 times during the screen creation and layout process a total of over 4 megabytes may be consumed.

Internally, the *FancyPants* graphics engine does not perform any data copies. The only copy required is to the final destination, the display.

Scene Graph Mode Canvas

Traditional graphics engines work in what is called Immediate Mode. No information about the state of the entire screen is stored at the graphics

engine level forcing calculations at each redraw. At best, traditional engines calculate “damaged” (exposed) regions of the screen to redraw and try to avoid drawing elements that will be completely covered by later draws. At worst, they redraw the entire screen each time.

The *FancyPants* engine uses Scene Graph Mode with a state engine and has knowledge of the entire state of the screen. When it is time to redraw, the graphics engine knows what the final screen will look like, it can calculate the resultant state, and does not draw anything unnecessarily.

Display Tiling

To further speed the drawing process, the *FancyPants* graphics engine splits the display into “tiles” so that only those parts of the display that require updating are redrawn.

This means that even on high resolution displays good frame rates for animation and video can be achieved using a modest CPU if only sections of the display are changing.

CONTENTION and CACHING

Single Thread

Switching between multiple threads creates a Context Switch on most CPU architectures. As discussed in David, 2007 [3], multi-threaded applications on the ARM CPU architecture use context switching which incurs an overhead greater than the minimum required running time.

FancyPants execution occurs within the context of a single operating system thread. This single thread model is portable across operating systems and eliminates contention among threads. IPC via standard mechanisms is supported.

Caching and On Demand Loading

FancyPants uses an image cache. Images are only loaded from a source into memory once and can be referenced more than once. In other words, if the application uses the same image multiple times, no additional image loads are required. This saves on expensive input / output to disk or network stream.

All application data, such as images and fonts, are only loaded on demand thereby saving memory until required. This is a significant consideration during application launch time which can be slowed down if trying to load all the data at start time. Similarly, importers for media types, such as JPEG, also only load on demand.

OTHER FANCYPANTS EFFICIENCIES

Minimal Memory Usage

Every element that appears on-screen is represented by an in-memory structure. This structure maintains information about the on-screen element such as position, size, etc. This information is called the per-object overhead.

The *FancyPants* graphics engine is designed to minimise the per-object overhead. Consuming between just 240 and 400 bytes per-object thousands of on-screen elements can be managed without any noticeable slowdown.

No Unnecessary Encapsulation

Subsystems within *FancyPants* are encapsulated where value is added by that encapsulation. Encapsulating for its own sake adds overhead, code complexity, unnecessary function calls and extra levels of indirection within the code.

Tiny Install Footprint

Only 600 Kb is required for installation of the base system.

CONCLUSION

FST designed *FancyPants* as a high speed, resource-friendly platform, enabling next generation user interfaces for embedded devices. This document is strictly focussed on the efficiencies of the *FancyPants* graphics engine, enumerating the major areas within the graphics engine code where performance is boosted. *FancyPants* applications are extremely responsive, highly appealing and flexible while minimizing memory, CPU cycles and install space.

While only the software implementation of the *FancyPants* graphics engine is discussed in this paper, OEMs can gain further benefits by making use of specialized hardware at the graphics engine level.



REFERENCES

- 1: Billy Biggs, Eclipse on SWT/GTK+ Performance Notes, , <http://vektor.ca/eclipse/gtk-performance-notes.html>
- 2: Tim Janik, Rapicorn - Improving Gtk+, , <http://testbit.eu/wiki/Rapicorn-Improving-Gtk+>
- 3: Francis M. David, Context Switch Overheads on Mobile Device Platforms, 2007, <http://www.cs.huji.ac.il/~feit/exp/expcs07/papers/136.pdf>