

Apertis Graphics Architecture Design

Author:	Tomeu Vizoso
Contributors:	Sjoerd Simons
Version:	1.2
Status:	Final
Date:	17 November 2015
Last Reviewer:	Luis Araujo

This proposal was produced exclusively using free and open source software.

Please consider the environment before printing this document.

Table of Contents

Document Change Log.....	3
1 Introduction.....	4
2 Requirements.....	5
3 Overview.....	6
4 Operation of composition.....	7
5 DRI2.....	8
6 Kernel driver.....	9
7 Xorg driver.....	10
8 OpenGL ES v2 implementation and EGL implementation.....	11
9 Standards compliance and upstream code.....	12
10 Glossary.....	13

Index of Illustrations

Illustration A: Block diagram showing how the different components sit on top of each other.....	6
--	---

DOCUMENT CHANGE LOG

Version	Date	Changes
1.2	2015-11-17	<ul style="list-style-type: none">• Updated to new name Apertis• Removed file custom properties (metadata)
1.1	2014-12-11	<ul style="list-style-type: none">• Updated to new template
1.0	2012-05-09	<ul style="list-style-type: none">• Update title and file name to follow Document Naming Scheme

1 INTRODUCTION

The goal of this document is to make explicit some characteristics of the Graphics subsystem that are needed in order to achieve the kind of user experience that Apertis aims for.

2 REQUIREMENTS

To support the proposed Apertis architecture the underlying Graphics stack needs to fulfill the following requirements:

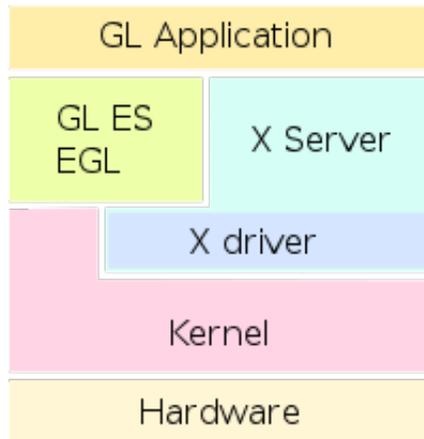
- Provide complete hardware acceleration for Clutter-based applications
- Provide the features needed by Mutter to provide a composited X environment
- Playback of video with hardware acceleration and without any buffer copies between the hardware decoder and the GPU
- Provide the necessary performance for smooth animations and scrolling

3 OVERVIEW

For a platform to support modern rich user interfaces, it needs a very complete and modern graphics stack. This document further specifies what is required from the components and functionality needs to be provided.

The diagram below shows a very simplified view of how the different components interface with each other:

Illustration A: Block diagram showing how the different components sit on top of each other



4 OPERATION OF COMPOSITION

In X, applications draw to top-level windows, which are backed by pixmap. When using a compositing window manager, such as Mutter, this backing pixmap is offscreen. The compositing window manager composites these offscreen pixmaps into a final scene which is rendered into the screen framebuffer. In the case of Mutter this composition is done using GL, (either OpenGL or GL ES).

In order for the composition to be fast enough to sustain smooth animations, the offscreen pixmaps need to be usable by Mutter as GL textures directly, without having to perform any copying of data.

The following section give an overview of the underlying components that are involved in a graphics stack that meets the requirements of Mutter.

5 DRI2

DRI2 (Direct Rendering Infrastructure 2) is an open-source framework to allow X applications to directly access the underlying GL hw without need to pass through the X server. In addition it ensure the smooth cooperation between GL using applications and the Windowing system.

Though not strictly needed, a set of drivers that doesn't make use of DRI2 will be reimplementing a lot of infrastructure, that is available as open-source already. Leading to a longer development time, and require more time to stabilize and integrate properly. Using DRI2 makes sure that all the required functionality is provided.

A detailed view of how the different components in DRI fit together can be found here: http://dri.sourceforge.net/doc/dri_control_flow.html

6 KERNEL DRIVER

A kernel driver is needed for managing the memory used in graphic operations, and to arbitrate access to the GPU. For open-source drivers this is implemented using DRM (Direct Rendering Manager), as with DRI2 it is possible to implement a driver with the necessary functionality which makes no use of this existing infrastructure, but this would lead to a longer development time, requires more time to stabilize and integrate.

7 XORG DRIVER

Compositing window managers rely on several X extensions for their functionality. The two main extensions are XDamage and XComposite. When XComposite is used application render their windows to an offscreen pixmap, which can then be used by the Compositing window manager to composite the final onscreen image. For a compositing window manager to be able to efficiently work, it needs to know which (areas of) offscreen pixmaps need to be redrawn. This functionality is provided by the XDamage extension, which can be used for notifications when certain (areas of) pixmaps get changed. The last extension heavily used by a modern linux graphics environment is the XRender extension to accelerate 2d drawing operations.

To summarize, the Xorg driver needs to implement the extensions XDamage, XComposite and XRender.

For the XRender extension modes that are most often needed are the following:

- 16/32bpp formats at depths 16, 24 & 32
- all of argb/abgr/xrgb/xbgr
- pictop(clear/src/dst/over/overreverse/in/inreverse/out/outreverse/add/saturate)
- repeat modes: none, normal, pad
- filtering: nearest, bilinear
- flags: component alpha, separate alpha map

8 OPENGL ES V2 IMPLEMENTATION AND EGL IMPLEMENTATION

As mentioned previously an implementation using the standard DRI2 and DRM infrastructure is recommended, but regardless of the implementation details it is crucial X11 is properly support as the windowing system.

Further so that Mutter can composite windows efficiently support for the following EGL extensions is required: * EGL_KHR_image_base * EGL_KHR_image_pixmap
Both these extensions should be implemented without requiring any copies.

In order to achieve the smoothness in animations and scrolling that is required, the graphics stack needs to swap buffers on display's Vsync and when using triple buffering the implementation should drop older frames if newer ones are available to prevent lag between system input and the onscreen response.

9 STANDARDS COMPLIANCE AND UPSTREAM CODE

All graphics functionality required by the middleware should be exposed using standard interfaces when applicable (mainly GL ES v2 and EGL) to ensure open-source components can be used without any further modification. If certain standard functionality is only available using proprietary extensions or APIs it will be impossible to merge the required modifications back in the upstream projects, which greatly increases the maintenance required by that project for the middleware platform.

10 GLOSSARY

- DRI: The Direct Rendering Infrastructure is a framework for allowing direct access to graphics hardware under the X Window System in a safe and efficient manner.
- DRI2: Revision of the DRI with improved support for direct rendering to offscreen buffers and compositing.
- DRM: API to be implemented by a kernel module to allow userspace to access graphics hardware.
- XRender: API to be implemented by the X driver to allow compositing operations on drawables.
- XDamage: API provided by the X server to allow applications to track modified regions of drawables.
- XComposite: API provided by the X server for applications to "redirect" their drawing to offscreen buffers, which can be composited later by a compositor.