WHITE PAPER

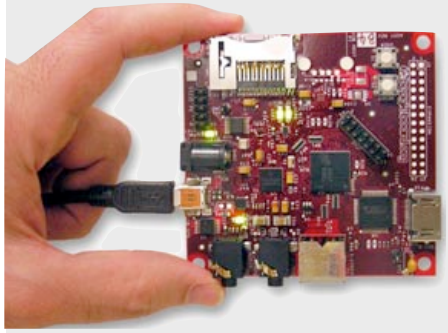
ESC Boston 2008
Class: ESC-341

By **Diego Dompe**
Embedded Software Team Lead
RidgeRun

Introduction to creating 3D UI with BeagleBoard

Executive Summary

BeagleBoard has many features supporting powerful embedded graphics thanks to the OMAP™ 3 architecture. But what are the options to integrate these capabilities with the open source software stacks for Linux? This document provides an overview of the OMAP 3 graphics hardware capabilities and introduces existing open source projects leveraging these features into exciting embedded Linux solutions.



Introduction

BeagleBoard features powerful graphics capabilities thanks to the use of the Texas Instruments OMAP™ 3530 application processor containing an an integrated SGX hardware graphics accelerator. The SGX hardware is available to programmers through the standard OpenGL® ES 1.1 or 2.0 for 3D graphics and OpenVG™ for 2D scalable vector graphics.

Linux is currently the main operating system of choice for BeagleBoard, and OpenGL ES and OpenVG are recent application programming interfaces (APIs) to the Linux software graphics arena. Therefore it requires some background on Linux software stacks to properly evaluate how to create graphic user interfaces that may leverage the hardware and software features for the specific product requirements.

This document provides an overview of the software and hardware features available on the OMAP 3 platform relevant to consider when creating 3D user interfaces, a summary of Linux's graphics software stacks, and an introduction to creating 3D enabled user interfaces with full-featured multimedia capabilities using open source projects. The document focus is on BeagleBoard development but the contents are applicable to any OMAP 3 based platform containing an SGX accelerator.

This document assumes a basic knowledge of computer graphics and Linux programming skills. Be sure to use wikipedia if you find any term that you are unfamiliar with.

OMAP 3 Multimedia Architecture and Linux

The OMAP 3 applications processor powering BeagleBoard provides several features that enables the creation of a new generation of embedded devices delivering state of the art graphics and processing capabilities while conserving battery power. These features include:

- SGX graphics hardware accelerator: enables use of the industry standard OpenGL ES and OpenVG graphics.
- C64x DSP: enables programming of signal processing algorithms. These algorithms uses a standardized programing interface thanks to Texas Instruments CodecEngine.
- Cortex™ A-8 processor featuring Neon™ SIMD instruction set, provides accelerated instructions and libraries to perform media processing functionality. ARM® provides libraries using the standard OpenMax™ DL API to accelerate common media formats.

However these features require proper support by the operating system and application software stacks to allow effective development. Linux provides standardized software APIs that should be used in order to gain quick integration with existing open source projects; for example using video for linux 2 (V4L2) interface for video capture devices provides easy integration with standard video processing applications.

The use of widely accepted open source APIs with strong community support pays off when developing embedded Linux solutions, so it is important to make an analysis of the options available when interfacing with hardware in order to maximize the software potential in the open source arena.

Although this document focus on the graphic software stack, it is important to consider how easy will it be to integrate the graphic solution with the other hardware interfaces: how can you create an user interface with 3D acceleration that can display video streams from the DSP codecs? How can you use the 3D acceleration and reuse existing open source software solutions like toolkit widgets, multimedia components and HTML render engines?

We will address the solutions for these questions by examining the software interfaces available and providing an overview of options to solve them, while balancing their advantages and disadvantages.

SGX Graphics Accelerator

The graphics accelerator of the BeagleBoard is an SGX coprocessor that provides an standard OpenGL ES 1.1 and 2.0 interfaces and OpenVG interfaces. You can find a useful introduction to key concepts related to the SGX hardware acceleration in this reference.

Texas Instruments plans to provide open source Linux kernel drivers for the SGX along with binary-only libraries toward the end of 2008 that can be used with BeagleBoard.

As a standard OpenGL ES software stack, the main interface to working with the driver is the EGL™ (http://www.khronos.org/egl/), which provides the interface to the underlaying native platform windowing system. On the OMAP 3, the native platform is the display subsystem frame buffer, thus the EGL™ implementation will render directly to the hardware as a window surface. This fact is important for properly interfacing the graphic driver to the standard Linux graphics software stacks.

Linux graphics software stacks

Before evaluating how to approach creating 3D user interfaces with the available hardware, it is necessary to survey the existing graphics software stacks to gain important background information.

There are several open source graphic software stacks for embedded Linux. The graphic stack is composed by several components and some of these components can be shared between the stacks:

- The graphic environment: provides the basic libraries and programs that provide the windowing functionality (if windowing is available), input device management, and event management.
- Support libraries: provide some miscellaneous functionality like advanced rendering capabilities (vector graphics), font management and rendering, multimedia frameworks or internationalization (aka i18n) support. Many of these libraries are shared between several graphics software stacks like freetype/fontconfig (de facto font libraries) or cairo (vector graphics library).
- Widget toolkits: provide widget sets to create graphical user interfaces.

Since some of the components are highly reusable; they can be deployed with different functionalities in different graphic stacks (for example cairo can be used with either X, DirectFB or clutter, just configured with a different backend). Also depending on the functionality provided by the graphic stack not all the components (or equivalents) are found on all solutions.

The main open source graphic environments available for embedded Linux are:

- X windows system: the venerable X protocol server implementations (usually the X.org package) is used on several embedded Linux systems. X allows reuse of existing open source toolkits and applications. Although the network oriented overhead of the server is often a concern in embedded systems, the increasing processing power available is dissipating the worry. X servers are particular useful in environments where is required to run several graphical applications simultaneously (like mobile internet devices, MID for short). The are several widget toolkits and support libraries to create interfaces based on X: Gtk+, QT, FLTK, WxWidgets, Motif, etc.
- Qtopia: the embedded version of the QT widget toolkit is capable of running directly over the Linux frame buffer interface, enabling porting or application development in a familiar C++ toolkit. Qtopia is an integrated graphic software stack providing the graphic environment, support libraries and widget toolkit.
- DirectFB: is a project gaining share on the embedded Linux space that provide some high level abstractions over the bare Linux framebuffer interface. Is particularly popular as is possible to compile the Gtk+ toolkit library to use it as a backend without requiring an X server, eliminating the network overhead of the X protocol while remaining capable of running standard Linux desktop applications.

It is important to consider some of the standard support libraries that play a role into integrating hardware features with the applications:

- cairo: is a 2D vector graphics library designed to use hardware acceleration when available. Cairo is a popular render library used by several open source projects like Gtk+ and WebKit (the HTML render engine behind Apple's Safari and iPhone™). There are some experimental work on cairo to

support OpenVG rendering that could be used to improve the performance of standard Linux applications when running on the OMAP 3 platform.

- Gstreamer: is a popular multimedia framework for Linux with a modular design that allows use of plugins that can bring the hardware acceleration features (like DSP codecs, or Neon accelerated codecs) to applications in a transparent manner. Gstreamer is used by several open source projects and applications.

3D acceleration on Linux and OpenGL ES

Linux 3D graphics typically uses the X window system thanks to the GLX libraries that provide a binding to connect OpenGL with the X window system. GLX allows X client applications to either send 3D rendering commands to a remote server or to render directly to the hardware if the client and server are running on the same machine and the driver supports the Direct Rendering Interface (DRI). However the GLX protocol is not compatible with OpenGL ES yet, and there are not known efforts to add support for it at the protocol level.

The DRI interface is currently being redesigned by Tungsten Graphics with the Gallium3D project opening the opportunity for a future implementation of a state tracker that can map the GLX/DRI commands into OpenGL ES commands, but at the time of this writing is only a planned research area and there is no public available code for it on the git repositories of the project.

Since the existing composite window managers for X (like compiz-fusion) relay on the AIGLX architecture that requires GLX, is currently not possible to bring this type of X acceleration to a platform running OpenGL ES. Another possible approach would have been to implement a mapping of the glitz library for the OpenGL ES and use the early Xegl server, but the Xgl code has been recently removed from X.org project for being unmaintained.

3D Interfaces without X server: meet clutter project

The clutter project (http://clutter-project.org) is an open source library for creating fast, visually rich and animated graphical user interfaces. Clutter provides an abstraction library (called the COGL) with a simple 3D API that is translated to OpenGL or OpenGL ES (and also supports direct calls to the specific underlaying API).

One important aspect of clutter is that it reuses several widely used open source support libraries. These libraries provide familiar and standard programming APIs that simplify reusing existing open source components:

- cairo rendering: actors (more on actors later) can be rendered using the cairo graphic library (allowing potential for future OpenVG acceleration).
- pango support: pango is the same i18n text rendering library using by the Gtk+ toolkit, allowing complete and powerful text rendering.
- gstreamer integration: clutter provide gstreamer actors that integrate with custom plugins allowing multimedia resources to be rendered directly on the 3D space.
- webkit backend for clutter: an experimental project that allows creation of webkit actors for web content rendering in 3D spaces.

Clutter already provides a backend support for EGL and is usable today with the OMAP 3 platform.

There are some other interesting features at clutter that are worth to mention:

- There is an optional Gtk+ clutter widget that can be used in the context of a standard Gtk+ environment (like Gtk+ over X, or Gtk+ over DirectFB) to embedded clutter scenes.
- Clutter provides integration with a 2D physics library called box2d that allows the creation of feedback effects based on physics behaviors (actors rotating over axes or bouncing at reaction of touchscreen movements).
- Unlike 3D acceleration available on the X server, clutter provides an animation API for the elements on screen (think CoreAnimation on OS X). This design simplifies the creation of stunning visual interfaces.

Of course clutter has his own limitations (due the youthful nature of the clutter project):

- Only one application can run at the same time using the full screen on the EGL backend (EGL doesn't support multiple clutter stages).
- Clutter EGL backend only provides touchscreen input support through tslib. However the author of this paper is working on adding generic Linux event interface support to the back end that will support keyboards, keypads, mouses and any input device implementing this kernel interface.
- Clutter doesn't provide a widget library beyond basic actors (simple labels and image renderings). However an example widget library adding more complex widgets exists on the development repositories called tidy.
- For touchscreen based devices there is a lack of an onscreen keyboard.

Putting together the graphics stack for your OMAP 3 solution

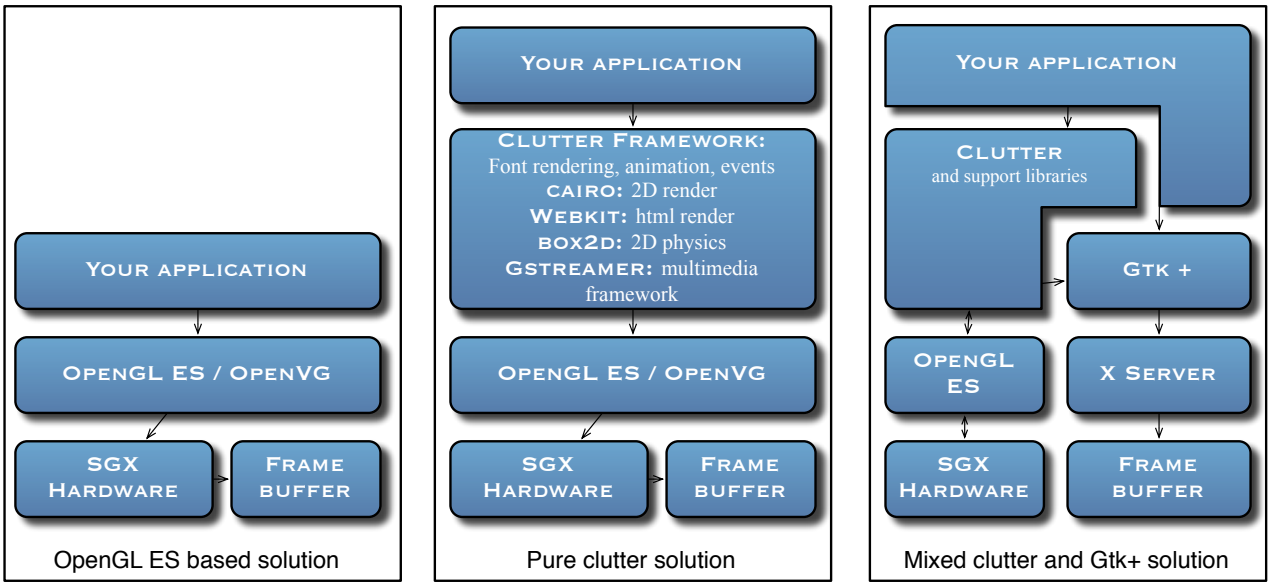
The rest of this document will focus on presenting an introduction on how to create some simple examples using clutter with the EGL driver for the OMAP 3 platform. Recall the possible approaches based on the technical background provided above:

- Bare OpenGL ES solution: one may wonder why not render directly using the OpenGL ES API. This approach make sense if your application displays data in 3D or game development without need for interactive interface rendering text or capturing events: OpenGL ES doesn't provide any support for event handling (or even event picking), or text handling.
- Pure clutter solution: provides a powerful 3D interface with good integration of open source components but has limitations in the following scenarios:
  - If your application relays heavily on standard widgets (which are not available). Not very good idea if you have to fill forms on the screen since you will have trouble displaying forms unless you write some classes and implement an on screen keyboard (or have a physical one on your device).
  - If you need a multi-application sharing the screen scenario like a MID device: clutter only allows one application to be running at the same time. However is worth to analyze the requirement of

multi-application support in the post iPhone era (although some people doesn't notice, the iPhones only run one application at the time in full screen, using some clever tricks to work with this model).

- Mixed Gtk+ with clutter solution: if the restrictions of the pure clutter solution are too much for your application, then you use Gtk+ (either over DFB or over X server) and create clutter animations inside widgets embedded in your Gtk+ application.

The following diagram graphically shows the software stack layout options to make the choices easier to understand:



Since clutter has an existing solution for creating user interfaces with OpenGL ES using standard open source software solutions we will focus on explore how to create applications with clutter using the pure clutter approach. The clutter over Gtk+ approach is left as an exercise to the reader.

Below you can see two screen shots of clutter based applications:

- The image at the left is prototype of the application launcher from moblin.org project. It uses a pure clutter approach.
- Ubuntu network remix application launcher is done with the clutter and Gtk+ approach and is used to launch the applications on the UME Ubuntu distribution.

Getting started with clutter

Clutter is a C programming API based on standard Gtk+ technologies like the GObject system. This allows clutter to easily provide several language bindings like perl or Vala (a popular C# like programming language in the Gnome community). Writing applications in languages other than C could be easier, however for simplicity on the dependencies we will explore the basic concepts with C code.

Clutter support different backends for rendering and target windowing systems. For the OMAP SGX driver clutter should use the EGL backend. The EGL backend runs directly over the framebuffer, therefore it could be only one application using clutter at the same time.

We will survey clutter programming concepts to gain understanding on his capabilities; this will provide the reader the knowledge to assess the potential of the platform. We will not focus on example code since this is available on the SD card accompanying your BeagleBoard, plus the tutorial and documentation pointed at the references section.

Building clutter for the BeagleBoard

To build clutter for BeagleBoard the best approach is to install all the dependencies with OpenEmbedded environment, then download and compile from BeagleBoard the latest clutter release. Note that this requires to have the PowerVR SGX SDK for OMAP 3 available so clutter can find the right headers. We used the following configure line to build the 0.8 release:

```
./configure --host=arm-none-linux-gnueabi --with-flavour=eglnative --with-
gles=2.0 --with-imagebackend=internal CPPFLAGS= -IGFX_Linux_SDK/OGLES2/
SDKPackage/Builds/OGLES2/Include/ LDFLAGS= -LGFX_Linux_SDK/OGLES2/
SDKPackage/Builds/OGLES2/LinuxOMAP3/Lib -lEGL -lIMGegl -lGLESv2 -lsrv_um -
Wl,--rpath-link -Wl,GFX_Linux_SDK/OGLES2/SDKPackage/Builds/OGLES2/
LinuxOMAP3/Lib -lpthread -lz ac_cv_func_malloc_0_nonnull=yes
```

For development process is usually useful to install clutter on your host machine with standard OpenGL backend. This allows to build and design the application functionality on the host machine and later move the code to the target board.

Use autotools as the build system for your applications make the cross compilation simpler, so it is recommended.

Clutter 101

Clutter API design is aimed to simplify the process of create objects that can be animated on a canvas. This canvas is called the *stage* and the objects on it are *actors*. The actors can be moved in 3D space unlike traditional 2D canvas. This model is more simple approach than standard 3D APIs since you don’t require to manage certain 3D elements like the visual context or other ambient factors.

Clutter is a retained mode graphics API, meaning the stage retain a complete model of the objects to be rendered and this can be manipulated.

Actors can be rectangles, images, text or any other content rendered by extra actors installed; for example gstreamer can render video into the gstreamer actor, and webkit actor can render HTML pages.

Clutter abstracts the underlaying OpenGL version providing his own interface for 3D functionality that maps to the target GL implementation. It provides scale, rotation, clipping and translation transformation, as well support for framebuffer objects and shader language API. Some actors implements a container interface that allow to perform graphic operations of all the actors included on the container.

The event interface maps the underlaying backend event interface and provides an efficient picking mechanism to notify actors and containers about input events. The event chain starts with the stage and goes down to the specific actor receiving the event (if parent actors haven't respond to the event).

The most distinctive functionality of clutter is his support for animation, and it provides several tools to accomplish this:

- Timelines are animation functions that can be specified from the number of frames and the frames per second. On every animation moment it will invoke a callback function that perform the required transformations. Timelines can be set to loop.
- Scores are grouping objects for timelines that allow to run several timelines at once or in a sequence.
- Effects and behaviors are some more complete animation functions intent to provide some typical animation transformations over properties of the actors.

An clutter extension that is very popular is the box2d integration. Box2d is an open source 2d physics engine for games that has been adapted to clutter to provide physics 2D constrains and behavior to the actors. This allows to easily create effects when using the interface that are not possible with any other open source graphic library.

All this features provide a good base to start developing applications, but some more complex widgets may be required by some applications. The clutter project repositories are full of small applications and proof of concepts that are not yet integrated in clutter but will help you if you are looking for more complex widgets.

References

The following references will be useful to start your software development:

- *OMAP35x Applications Processor 2D/3D Graphics Accelerator (SGX)*, Technical Reference Manual, http://focus.ti.com.cn/cn/lit/ug/spruff6b/spruff6b.pdf
- *Introduction to Graphics Software Development for OMAP™ 2/3*, White Paper, http://focus.ti.com.cn/cn/lit/wp/spry110/spry110.pdf
- Cairo graphics web site, http://www.cairographics.org
- Clutter project web site, http://www.clutter-project.org (visit the documentation reference and browser the repository to find examples and proof of concepts)

- Programming with clutter, online tutorial, http://www.openismus.com/documents/clutter_tutorial/0.8/docs/tutorial/html/

About the author

Diego Dompe is the embedded software team leader at RidgeRun Engineering, where he research and integrates open source technologies for embedded Linux from the low-level drivers to the application software stacks. He is finishing his master degree on Computer Science at the Costa Rica Institute of Technology.