SGX Android Version Summary

1.8 DDK

Filename : SGX Android Version Summary DDK.Honeycomb and Ice Cream Sandwich.doc

Version : 1.0.7 External Issue

Issue Date : 01 Sep 2011

Author : Imagination Technologies Ltd

Contents

1. Introduction

This document summarises changes to the SGX DDK graphics framework of Android required for “Ice Cream Sandwich” (ICS) version, as such this document covers changes since the Gingerbread version of Android. Although not a target of the 1.8 DDK, a subset of these changes would be applicable to the “Honeycomb” (HC) version of Android and these are detailed in a separate “Honeycomb” section.

2. Ice Cream Sandwich

2.1. Hardware Composer HAL (improved functionality)

2.1.1. ANativeWindow TRANSFORM_HINT

EGL implementations can now see what transform the compositor would apply to a buffer. This is not strictly related to HWC, but it is generally required by it.

This allows the Android DDK to perform a PBE rotation of the output SGX rendering in the client application. The client renders to buffer in the pre-rotated orientation.

This means the compositor no longer has to transform the buffers when compositing. The PBE rotation “cancels out” the device/screen rotation.

This effect is particularly desirable for HWC implementations that cannot handle rotation of GL client buffers themselves. Since the GPU has handled the rotation to output memory, the HWC can directly display the buffer as though it was not transformed.

2.1.2. GRALLOC_USAGE_HW_COMPOSER

A new usage bit (`GRALLOC_USAGE_HW_COMPOSER`) indicates that a buffer may be consumed by the HW composer module. Buffers that are not marked with this bit will never be passed to the HW composer module. Examples of such buffers are those allocated by the “mediaserver” process, which are used as scratch buffers or client application buffers only and are never composited directly.

If the vendor gralloc is allocating buffers in a special way for HW composer inter-op, and this special way is a precious/finite resource, it is useful to be told that some buffers can be exempted from this process. This usage bit controls such exemption.

2.2. New pixel format support

The `HAL_PIXEL_FORMAT_BGRX_8888` (XRGB8888) format is supported by default. This is now the default pixel format for EGLConfig with `r=8,b=8,g=8,a=0`.

The `HAL_PIXEL_FORMAT_BGRA_8888` (ARGB8888) format is now the default pixel format for EGLConfig with `r=8,b=8,g=8,a=8`.

The `HAL_PIXEL_FORMAT_RGBX_8888` (XBGR8888) and `HAL_PIXEL_FORMAT_RGBA_8888` (ABGR8888) formats are obsolete for GPU rendering, and remain only for Skia (SW rendering) compatibility. Gralloc supports allocating these buffers, but they cannot be rendered to with GL.

This change was made for multiple reasons:

- Most frame-buffer hardware uses the ARGB8888 format and is not compatible with ABGR8888. This presented problems in older versions of the platform that lead to duplicate EGLConfig being listed. Applications could only differentiate between the EGLConfig by inspecting the `EGL_NATIVE_VISUAL_ID` attribute.
- The introduction of HWC in Honeycomb and ICS has exacerbated the above issue. Client applications picking an `r=8,b=8,g=8,a=0` or `r=8,b=8,g=8,a=8` EGLConfig would ordinarily receive XBGR8888 or ABGR8888 buffers. The GPU would swizzle the pixels before rendering them to the display.
  But in the HWC path, we cannot utilize the GPU to perform this swizzle, and the display generally cannot perform this task. It is therefore desirable to reduce the number of buffers allocated in these formats.

2.3. EGL_ANDROID_recordable extension

The EGL_ANDROID_recordable EGL extension adds a new EGLConfig attribute, `EGL_RECORDABLE_ANDROID`, which should be set on EGLConfigs that support rendering to matching buffer formats that are suitable for feeding to an OpenMAX video encoder.

There are two ways a vendor can implement the functionality expressed by this extension, explicitly invoked by OpenMAX or implicitly in the GL. The first way is implemented here, as the second way is too expensive and could cause unrequired color converts of large surfaces.

In the first case, the EGL driver can advertise any RGB colorspace formats that the OpenMAX video encoder can handle. The current SGX Android DDK implements this method, and marks all RGB EGLConfig as `EGL_RECORDABLE_ANDROID`.

2.4. Change to cancelBuffer() behaviour

Prior to ICS, cancelBuffer() would be called in eglMakeCurrent() when a surface is made non-current.

Under ICS, Google requested that cancelBuffer() only be called at eglDestroySurface() time, to facilitate pbuffer rendering and other subsequent re-use of a buffer.

The Android DDK changes the locations cancelBuffer() is called when compiled against ICS.

2.5. Removal of ANativeWindow connect()/disconnect() hooks

The Android wrapper EGL implementation will perform the equivalent of these calls automatically for all vendor EGL implementations. Vendor EGL drivers can cease calling them.

The Android DDK does not call the hooks when compiled against ICS.

2.6. Third Party Display Driver (3PDD) changes

2.6.1. Vendor-neutral “next generation” 3PDD

The Android DDK introduces a new side-by-side services API for writing complex 3PDD modules for handling the most advanced hardware composer HAL use cases.

The most visible change is the introduction of a Post2() framebuffer HAL hook. This function is designed to post multiple buffer handles simultaneously, unlike the Google-specified post() function, which posts only a single buffer.

Post2() calls a new services API, `PVRSRVSwapToDCBuffer2()`. This function differs from `PVRSRVSwapToDCBuffer()` in the following ways:

- The concept of a “swap buffer” has been removed.
  A list of `PVRSRV_CLIENT_MEM_INFO` structs must be provided instead.
- Support for clip rectangles has been removed.
- Support for “render IDs” has been removed.
- Private data can be passed through from user-space.

On the kernel side, this new function calls 3PDD ProcessFlip() with a new data structure. The old structure was `DISPLAYCLASS_FLIP_COMMAND`. The new data structure passed by the services queue code is `DISPLAYCLASS_FLIP_COMMAND2`.

The structures can be differentiated by first casting to `DISPLAYCLASS_FLIP_COMMAND` and checking the `hExtBuffer` field. If this handle is `NULL`, cast to `DISPLAYCLASS_FLIP_COMMAND2` and access the new extension fields:

- `pvPrivData`: pointer to kernel copy of user-space private data
- `ui32PrivDataLength`: size of private data in bytes
- `ppvMemInfos`: cast to `PVRSRV_KERNEL_MEM_INFOS` to obtain physical and virtual address information for buffers
- ui32NumMemInfos: size of above array of MemInfos

In general, the private data passed through by `PVRSRVSwapToDCBuffer2()` should relate to the MemInfos passed through. A basic implementation will pass one chunk of private data through for every MemInfo, allowing the display to program itself appropriately for the buffer data/format.

2.6.2. OMAP-specific 3PDD changes

The IMG supplied 3rd party display for OMAP “dc_omapfb3_linux” (omaplfb) implements the above API changes.

This backs directly to TI’s dsscomp, which handles overlay configuration, presentation and synchronization.

The dsscomp programming information is passed through from the hardware composer HAL module through the private data parameter of `PVRSRVSwapToDCBuffer2()`.

This private data contains information about layer dimensions, blend mode, transform and other useful meta-data.

3. Honeycomb

3.1. Hardware Composer HAL (basic functionality)

The HWC enables vendors to bypass GL composition for one or more layers of a scene and composite them with display hardware (or using blit functionality if supported by the GPU).

Imagination currently ships a “reference” composer HAL which does not have any dependencies on SOC display hardware. This module is intended to replace the “composition bypass” modifications made on earlier versions of Android.

At present this “reference” composer HAL requires a small patch against the framework to allow specialised buffers to be allocated for “bypass” buffers. It is not anticipated that Google will merge such a patch.

The HWC API in HC is an early version that lacks some important features for the “reference” composer HAL. In particular, GL clients cannot identify the desired rotation of the screen and therefore cannot render in the correct orientation for direct display of client buffers. It is not anticipated that Google will back port the required changes from ICS to HC.

3.2. New gralloc USAGE bits

3.2.1. GRALLOC_USAGE_EXTERNAL_DISP

Content should be displayed on an external display if available. The SGX DDK does not explicitly handle this usage bit, although we allow buffers to be allocated with it.

3.2.2. GRALLOC_USAGE_PROTECTED

Content is protected (DRM) and must not be mapped to the GPU virtual or user virtual address spaces. Buffers should be read + written only by hardware in kernel mode. The SGX DDK does not explicitly handle this usage bit, although we allow buffers to be allocated with it. The DDK will still map such buffers to the GPU and user mode.

3.2.3. GRALLOC_USAGE_PRIVATE

A new bit field has been provided in the top byte of the usage word for vendor-specific usage bits. The Android DDK currently uses the following bits:

```
#define GRALLOC_USAGE_BYPASS          GRALLOC_USAGE_PRIVATE_0
```

This usage bit replaces the `GRALLOC_USAGE_HW_FB|GRALLOC_USAGE_HW_RENDER` bits used in Gingerbread and Froyo.

```
#define GRALLOC_USAGE_PHYS_CONTIG     GRALLOC_USAGE_PRIVATE_1
```

This usage bit indicates that a buffer is backed by physically contiguous memory. Some HWC implementations can use this to determine if a buffer is compatible with their video decode/encode or display hardware.

3.3. HwRenderer (Hardware accelerated canvas)

Honeycomb introduces an “opt-in” hardware accelerated canvas model which replaces Skia rendering. The Android DDK can support this acceleration in the latest version of MR1 and beyond.

3.3.1. cancelBuffer()

The Android DDK supports the `ANativeWindow::cancelBuffer()` API extension for cancelling de-queued buffers when they are made non-current. Previously, such state changes would result in the composition of these buffers.

The `cancelBuffer()` extension was added in Gingerbread but was rarely utilised. In Honeycomb, rendering artefacts would be obvious across the interface without `cancelBuffer()`, as HwRenderer often makes surfaces current, renders to them, and does not want to composite the result.

For example, lack of support for `cancelBuffer()` results in a flickering artefact in the “Settings” menus.

3.4. GL_OES_EGL_image_external

In Honeycomb and later, all GL implementations must support texturing from specific non-GL texture formats. This implies that gralloc must also support allocation of such buffers.

Support for YV12 is mandatory. Support for one or more SOC/vendor-specific formats is also desirable.

The Android DDK supports the following formats in OpenGL ES 1.1 and OpenGL ES 2.0. The Imagination gralloc also supports allocating buffers in these formats:

- HAL_PIXEL_FORMAT_YV12
- HAL_PIXEL_FORMAT_TI_NV12 (OMAP specific)
- HAL_PIXEL_FORMAT_TI_NV12_PADDED (OMAP specific)

3.4.1. YV12 video buffer format

Support for YV12 is mandated by the multimedia framework.

In early bring-up, where there may be no hardware-assisted video decode, the software video decoder will output YV12 video.

If the video decode IP cannot support a format, the software video decoder will handle it, and output YV12 video.

3.4.2. Vendor-specific buffer format

The Imagination gralloc has a new abstraction layer for handling buffer formats. Vendors can add new buffer formats more easily and without modifying the core of gralloc.

Adding support for formats already supported by OpenGL ES 1.1 and 2.0 (NV12, YUYV, YV12, RGB565, ARGB4444, ARGB8888, ARGB1555, XBGR8888, XRGB8888) but with differing memory layouts (stride, padding) may not require modifications to ANDROID_WSEGL or the OpenGL ES libraries. However, IMG cannot guarantee automatic compatibility and further changes may be necessary.

Adding support for totally new formats involves extensive modifications to OpenGL ES and ANDROID_WSEGL. Such an effort should be undertaken as early as possible in a project.