SGX OpenGL ES 1.1 Reference Driver

Software Architecture Specification

Filename : SGX OpenGL ES 1.1 Reference Driver.Software Architecture Specification.doc

Version : 1.0.9a External Issue External 1.6 Beta.

Issue Date : 16 Apr 2010

Author : PowerVR

Contents

List of Figures

Glossary of Terms

Table 1

Khronos Group	Body that controls the OpenGL ES, OpenVG and OpengMAX specifications
EGL 1.4	The Native Platform Graphics Interface implemented for this project.
OpenGL ES 1.1	The fixed function 3D API implemented for this project OpenGL ES is a trademark of Silicon Graphics, Inc.
SGX	IP that provides 2D/3D/Video acceleration

1. Introduction

1.1. Scope

This document provides a top-level view of the OpenGL ES 1.1 software architecture for SGX with a brief description of the major architectural components. It is assumed that the reader is already familiar with the hardware functionality of SGX and its derivatives.

1.2. Related Documents

OpenGL ES 1.1 Full Specification, Khronos Group, Version 1.1.10
EGL 1.4 Specification, Khronos Group, Version 1.0

2. Product Overview

The OpenGL ES 1.1 component delivers a reference OpenGL ES 1.1 driver for SGX and variants, but as this cannot be developed in isolation of its environment, the driver will also require the presence of other components for services functionality and 2D functionality.

2.1. Goals and Objectives

The major high-level design goals for this project's software architecture are detailed below.

- Low Implementation Risk. The OpenGL ES 1.1 schedule is aggressive; any chosen architecture must be very low risk, which implies it being based on existing knowledge and well understood environments.
- Platform portability. The code must be easily ported to different operating systems potentially including Linux, Symbian, WinCE, others.
- Does not impede performance.

2.2. Product Environment

The product is designed to support the SGX hardware and variants including SGX520, 530, 535 with MMU. The reference OpenGL ES 1.1 DDK development environment is Linux on a Intel P4 PC running a Linux based on a 2.6.14 kernel. This should not be confused with any customer delivery environment to which the reference driver is ported.

2.3. Assumptions, Dependencies and Constraints

2.3.1. Assumptions

- None.

2.3.2. Constraints

- The OpenGL ES 1.1 schedule is aggressive; any chosen architecture must be very low risk. This then requires re-use of successfully integrated components from the target environment as far as possible.

3. Architecture Specification

3.1. Overview

The following diagram 'Figure 1 Architecture Overview' details a top-level overview of the architecture of the software stack, including the OpenGL ES reference driver's position within it.

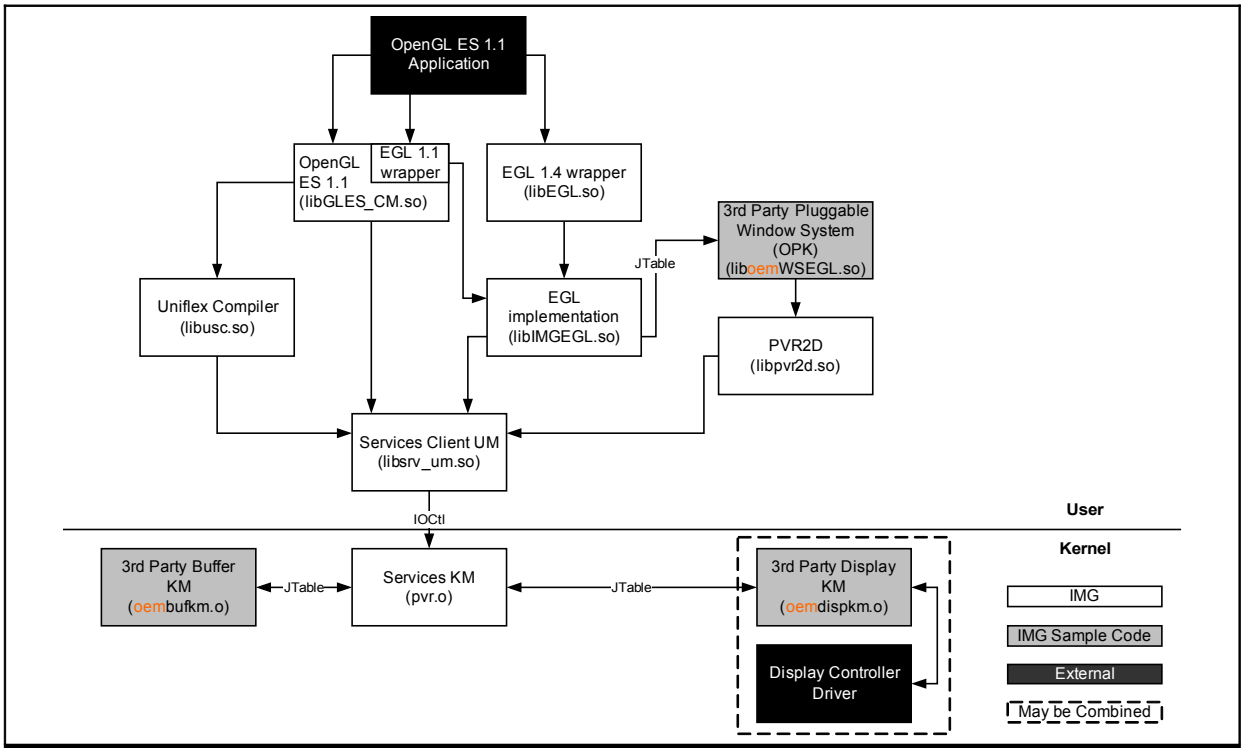

Figure 1 Architecture Overview

Elements in grey are 3rd party components for which example code will be supplied

The architecture chosen meets the design goals defined in section 2.1 (see below for details).

- Proven architecture. The basic architecture chosen has been successfully implemented on a variety of platforms including WinCE, Symbian, Linux and uiTRON. This provides a very low risk design. The architecture re-uses substantial amounts of proven code from the PC SGX OpenGL driver and the MBX OpenGL ES code bases.
- Platform portable. Platform specific code abstracted to services components to simplify porting to another platform.

OpenGL ES 1.1	
Component Type	Library
Purpose	Provide the OpenGL ES 1.1 Common or Common-lite API interfaces to SGX hardware and native platform graphics interface.
Functionality	Provides SGX HW acceleration of OpenGL ES 1.1 Common and Common-lite. This component also provides a thin layer to the EGL implementation for backwards compatibility.
External Interfaces	This component provides a 'C' interface as defined by the OpenGL ES 1.x Common Lite profile, Common profile and Native Platform Graphics Interface as defined by the Khronos group.

OpenGL ES 1.1	
Internal Interfaces	This component layers the EGL 1.0/1.1 interface onto a common EGL implementation module.
Dependencies and Inter-relationships with other major components	This component interfaces with the EGL implementation component and uses the PowerVR services component. The PowerVR services component is used for memory management, power management, synchronisation, device initialisation and various common utility functions.

Table 2

3.2. Detailed Component Description

The following diagram 'Figure 2 OpenGL ES 1.1 Driver Architecture Overview' details an overview of the architecture of the OpenGL ES 1.1 reference driver.

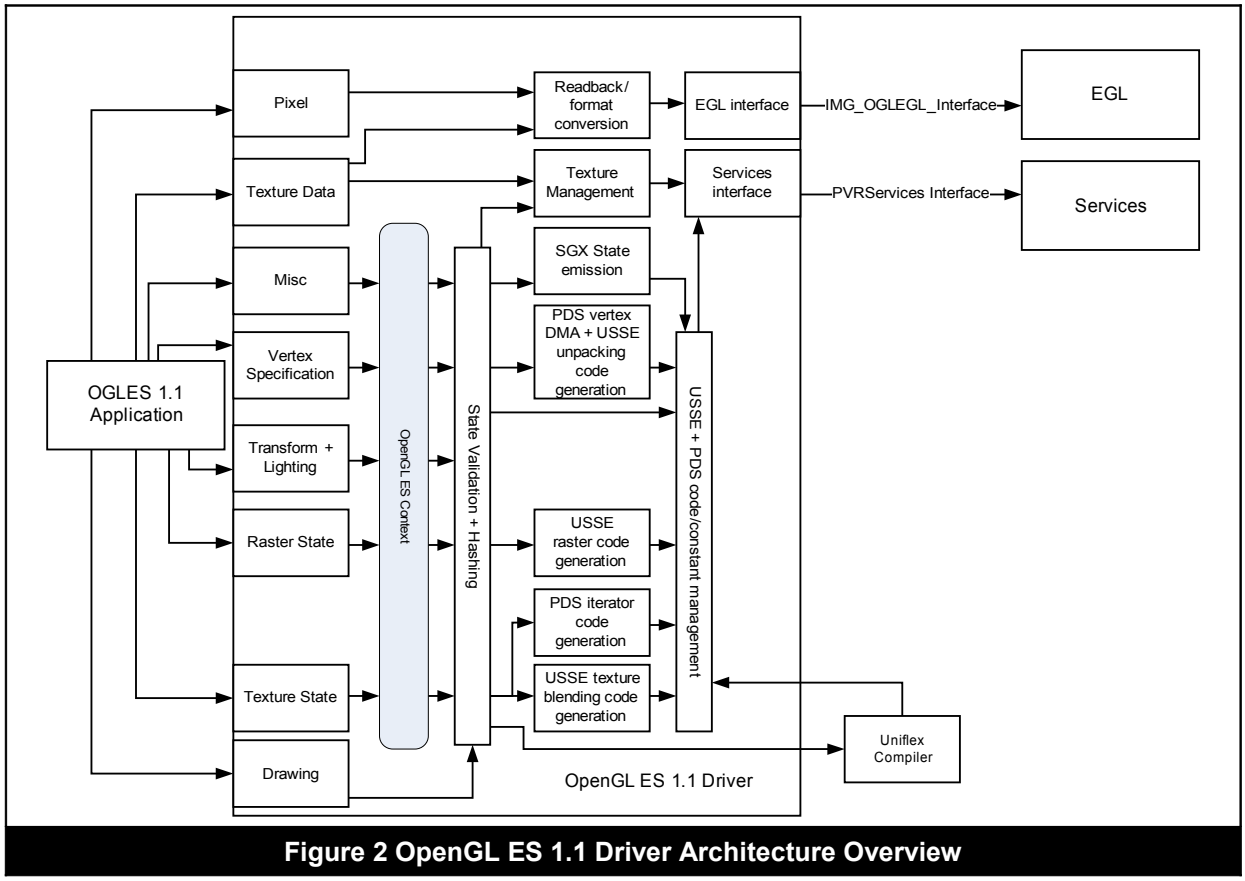

Figure 2 OpenGL ES 1.1 Driver Architecture Overview

The architecture chosen meets the design goals defined in section 2.1 (see below for details).

- Proven architecture. The architecture re-uses substantial amounts of proven code from the PC SGX OpenGL driver and the MBX OpenGL ES code bases.
- Platform portable. Platform specific code abstracted to services components to simplify porting to another platform.

Readback/format conversion	
Component Type	C Code

Readback/format conversion	
Functionality	Converts OpenGL ES 1.1 texture data into SGX format, and performs pixel readback from the render surface (including any required format conversion).
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component interfaces with the EGL implementation component to get information about the current readback surface's attributes (size, format etc).

Table 3

Texture Management	
Component Type	C Code
Functionality	Performs management of texture and subtexture data and performs texture residency + ghosting as necessary.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component interfaces with the services to allocate device memory for textures.

Table 4

PDS Vertex DMA + USSE unpacking code generation	
Component Type	C Code
Functionality	Generates PDS code to copy vertex data from the application supplied buffers (or buffer objects) to the primary attribute buffer. Also generates USSE code to perform any data format conversion as necessary.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	None

Table 5

PDS Iterator code generation	
Component Type	C Code
Functionality	Generates PDS code to kick the iterators based on the requirements of the pixel shader and the outputs of the vertex shader.
External Interfaces	None
Internal Interfaces	None

PDS Iterator code generation	
Dependencies and Inter-relationships with other major components	This component interfaces with texture code generation modules to help generate the PDS iterator code.

Table 6

USSE raster code generation	
Component Type	C Code
Functionality	Generates the USSE code to perform back end raster operations (eg. blending, fog). This will also generate code necessary to perform alpha test.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	None

Table 7

USSE texture code generation	
Component Type	C Code
Functionality	Generates the USSE code to perform fixed function texture blending.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component provides requirements to the PDS iterator code generation module.

Table 8

State Validation and Hashing	
Component Type	C Code
Functionality	Validates state from the context and optimises code and SGX state regeneration using a hash of the current state, and “dirty bits” to signify state changes.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component accepts “dirty bits” from all the state entrypoints to optimise the code/state generation.

Table 9

USSE + PDS code/constant management	
Component Type	C Code
Functionality	Manages and loads USSE constants and code, as well as PDS code.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component interfaces with all the code generation modules and gets code/constant buffers from the services.

Table 10

SGX State Emission	
Component Type	C Code
Functionality	Emits relevant SGX state (ie MTE, ISP, PixelBE) based on various GLES state values.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component interfaces with the code management module to load the state at the relevant time for the appropriate primitives.

Table 11

Uniflex Compiler	
Component Type	C Code / Support Library (.so)
Functionality	This component is a module that is linked to by the OpenGL ES 1.1 driver, and is used to generate the USSE code for performing transform and lighting as per the OpenGL ES 1.1 specification. The module will accept Uniflex instructions and generate USSE output code
External Interfaces	Uniflex instruction input to binary USSE code
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	Output will be passed to the code management module

Table 12

4. Review of this Document

4.1. Signatories

Signed off for document version:	0.7.22b

4.2. Change Control

Any changes to this document will result in a new version with appropriate sign off.

To do this the Signatories Section will need to be duplicated and the dates and version removed. Please leave the old details also present in the document so that the original can signed version can be found.