SGX OpenGL ES 2.0 Reference Driver

Software Architecture Specification

Filename : SGX OpenGL ES 2.0 Reference Driver.Software Architecture Specification.doc

Version : 1.0.20a External Issue

Issue Date : 14 Jul 2008

Author : PowerVR

Contents

List of Figures

Glossary of Terms

Table 1

Khronos Group	Body that controls the OpenGL ES, OpenVG and OpenMAX specifications
EGL 1.4	The Native Platform Graphics Interface implemented for this project.
OpenGL ES™ 2.0	The programmable 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 2.0 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

The OpenGL ES Shading Language, Khronos Group, Version 13
OpenGL ES 2.0 Specification, Khronos Group, Version 2.0.22 (Full Specification)
EGL 1.4 Specification, Khronos Group, Version 1.0

2. Product Overview

The OpenGL ES 2.0 component delivers a reference OpenGL ES 2.0 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 2.0 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 SGX510, 520, 530 with MMU. The reference OpenGL ES 2.0 DDK development environment is Linux on a Intel P4 PC running a Linux based on a 2.6.22 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 2.0 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.

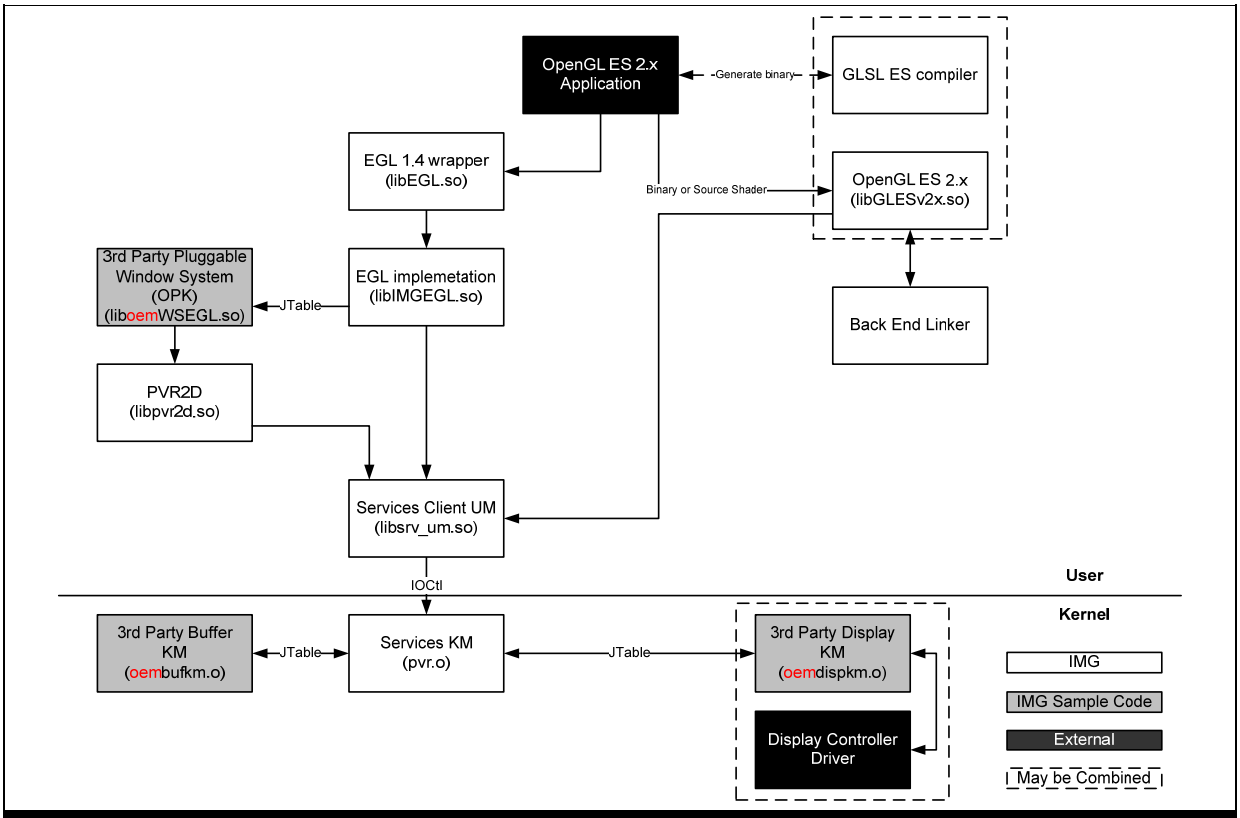

Figure 1 Architecture Overview

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 2.0	
Component Type	Library
Purpose	Provide the OpenGL ES 2.0 API interface to SGX hardware and native platform graphics interface.
Functionality	Provides SGX HW acceleration of OpenGL ES 2.0.
External Interfaces	This component provides a 'C' interface as defined by the OpenGL ES 2.0 and Native Platform Graphics Interface as defined by the Khronos group.

OpenGL ES 2.0	
Internal Interfaces	If GLSL ES source shaders are supported, this component may interface with a GLSL ES compiler component (which would be linked into the library).
Dependencies and Inter-relationships with other major components	This component will interface with a back-end linker component, with the EGL implementation component and with the PowerVR services component.

3.2. Detailed Component Description

The following diagram 'Figure 2 OpenGL ES 2.0 Driver Architecture Overview' details an overview of the architecture of the OpenGL ES 2.0 reference driver.

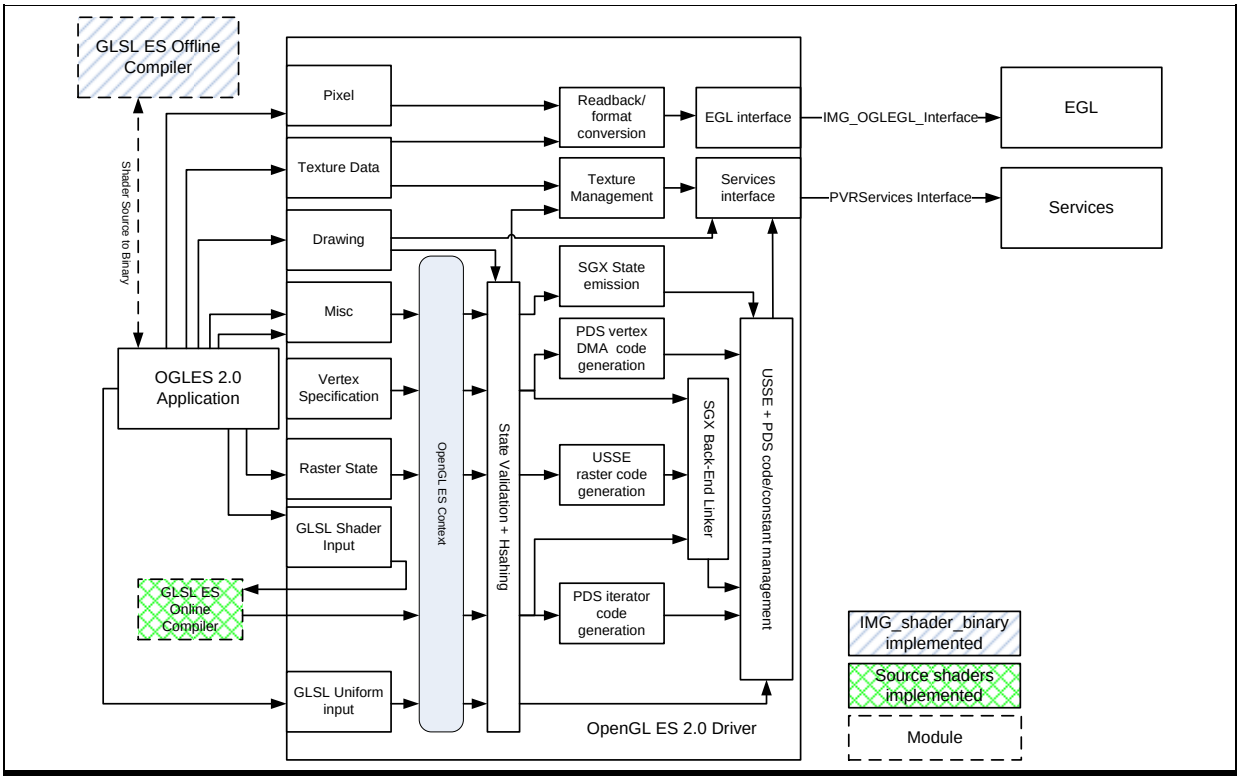

Figure 2 OpenGL ES 2.0 Driver Architecture Overview

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
Functionality	Converts OpenGL ES 2.0 texture data into SGX format, and performs pixel readback from the render surface (including any required format conversion).
Exported Interfaces	None
Imported 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).

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.

PDS Vertex DMA 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.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	None

GLSL ES Compiler (Offline)	
Component Type	Executable
Functionality	The module will accept a file (or pair of files) containing a GLSL ES source shader (or fragment + vertex shader pair). A binary format will be defined which the component will output as a file. This format will contain highly optimised USSE code with spaces for extra code to be added in back-end linking. It will also include information used to generate PDS iterator code, and output selects for MTE state.
External Interfaces	Text input file to binary output file. Binary format to be defined
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	Output will be passed to back-end linker and to PDS iterator code generation module.

GLSL ES Compiler (In driver)	
Component Type	C Code / Support Library (.so)
Functionality	This component will be a module within (or linked to by) the OpenGL ES 2.0 driver. The module will accept a GLSL ES source shader (or fragment + vertex shader pair). The output will be in the same binary format as the offline compiler, and will contain USSE code with spaces for extra code to be added in back-end linking. It will also include information used to generate PDS iterator code, and output selects for MTE state. The generated USSE code trades sub-optimal output against small static/runtime footprint of the compiler module.
External Interfaces	Source shader input from GLES 2.0 to binary output structure.
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	Output will be passed to back-end linker and to PDS iterator code generation module.

PDS Iterator code generation	
Component Type	C Code
Functionality	Generates PDS code to kick the iterators based on the requirements of the pixel shader.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component either interfaces with the GLSL ES compiler, or receives input directly from the shader binary to help generate the PDS iterator code.

USSE raster code generation	
Component Type	C Code
Functionality	Generates the USSE code to perform back end raster operations (e.g. blending, fog).
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component will interface with the back-end linker to link this code to the supplied fragment shader optimally.

Back-end linker	
Component Type	C Code / Support library (.so)
Functionality	Links binary code generated by GLSL ES compiler with raster code generated by OpenGL ES 2.0 driver. Inserts relevant texture format unpacking and vertex format unpacking to fragment and vertex shaders respectively.
External Interfaces	None
Internal Interfaces	None
Dependencies and Inter-relationships with other major components	This component accepts binary code from the compiler and driver components, as well as texture and vertex format specifications from the GL.

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 entry points to optimise the code/state generation.

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.

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.