

SGX OpenGL ES 2.0 Reference Driver

Software Functional Specification

Copyright © 2011, Imagination Technologies Ltd. All Rights Reserved.

This document is confidential. Neither the whole nor any part of the information contained in, nor the product described in, this document may be adapted or reproduced in any material form except with the written permission of Imagination Technologies Ltd. This document can only be distributed to Imagination Technologies employees, and employees of companies that have signed a Non-Disclosure Agreement with Imagination Technologies Ltd.

Filename : SGX OpenGL ES 2.0 Reference Driver.Software Functional Specification.doc
Version : 1.0.93 External Issue
Issue Date : 12 Aug 2011
Author : PowerVR

Contents

1.	Introduction	4
1.1.	Scope	4
1.2.	Related Documents	4
2.	Overview	5
2.1.	Functionality Summary	5
2.2.	Software Module Description	6
2.2.1.	OpenGL ES 2.0 Reference Driver	6
2.2.2.	GLSL ES Compiler	7
2.2.3.	Back-end Linker.....	8
2.2.4.	EGL	8
2.2.5.	PowerVR Services Glue	8
2.3.	Development constraints.....	8
3.	Conformance	9
3.1.	Compliance.....	9
4.	Extensions	10
4.1.	PowerVR Texture Compression.....	13
4.2.	Other extensions	14
5.	Implementation Dependent Values	15
6.	Debug, Metrics and Profiling	17
6.1.	Debug	17
6.2.	Metrics	17
6.3.	Profiling.....	17
7.	Configurations	18
7.1.	Build time configurations	18
7.2.	Runtime configurations.....	18
8.	Memory allocations	21
8.1.	Device memory allocations for internal driver use	21
8.1.1.	Allocations in 1.2 DDK.....	21
8.1.2.	Allocations in 1.3+ DDK (including performance improvements).....	22
8.2.	Device memory allocations for application supplied data	23
8.3.	Host memory allocations for internal driver use	23
8.4.	Host memory allocations for application supplied data	23
Appendix A.	OGL ES Reference Driver Code Modules	25
Appendix B.	Sample Metric Data	26
Appendix C.	Sample Profile Data	28

List of Figures

Figure 1	OpenGL ES 2.0 Driver Architecture Overview	6
----------	--	---

Glossary of Terms

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

Table 1

1. Introduction

1.1. Scope

This document describes the high level design and functional specification of the OpenGL ES 2.0 reference SGX device driver.

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. Overview

OpenGL ES 2.0 is an API which, primarily through sub-setting of OpenGL, attempts to provide an API that can be efficiently implemented within the constraints of an embedded platform, but still retain the core functionality and feature set of OpenGL, giving application developers an immediate familiarity with the API. It also strives to eliminate legacy fixed function vertex and fragment processing.

The SGX OpenGL ES 2.0 reference driver will be designed and constructed based on knowledge developed in the following processes:

- Deep experience of the OpenGL and OpenGL ES standards, including interaction with the ARB, and promoter level membership of the Khronos Group, which controls and extends the OpenGL ES standard.
- Development experience of full featured OpenGL ES 1.1 implementations.
- Development experience of full featured OpenGL 2.0 implementations.
- Porting of these implementations and associated services to multiple disparate operating and windowing systems.
- Development of OpenGL and OpenGL ES implementations for multiple generations of PowerVR hardware.
- Active interaction with the Khronos group's OpenGL ES standards development work.

This experience allows the development of a reference driver with sufficient modularity to ensure rapid portability to different environments, but sufficient internal coherence to ensure that performance is not sacrificed to modularity.

2.1. Functionality Summary

The following table lists the functionality summary of the OpenGL ES 2.0 reference driver with respect to the OpenGL ES 2.0 specification. Functionality beyond the OpenGL ES 2.0 specification is described in the section on [Extensions](#). Functionality that falls short of the OpenGL ES 2.0 specification is described in the section on [Conformance](#).

Functionality	Description
General	
Profiles	Common 2.0
EGL	
Render Surfaces	Frame Buffer Objects + Pluggable Window System surfaces
Render Formats	Supports 565, 1555, 4444 and 8888 RGB(A) formats
Multiple Context	Support for concurrent 3D applications
Vertex Processing	
Programmable Vertex Shaders	Yes
Viewport Clipping	Yes
Polygon offset	Yes
Rasterisation	
Multi-Sampling	Support for x4 anti-aliasing
Hidden Surface Removal	Z buffering with all compare modes and polygon offset
Polygon culling	Front and Back face polygon culling
Primitives	Native support for indexed triangle and line strips, points, fans, and triangle lists
Programmable Fragment Shaders	Yes

Functionality	Description
Texture Wrap Modes	All wrap modes
Texture filter modes	All modes
Texture Formats	All formats
Blending	All render target blend modes
Color Mask	Full color mask support
Extensions	See Section 4

2.2. Software Module Description

The following diagram 'Figure 1 OpenGL ES 2.0 Driver Architecture Overview' details an overview of the architecture of the OpenGL ES 2.0 reference driver.

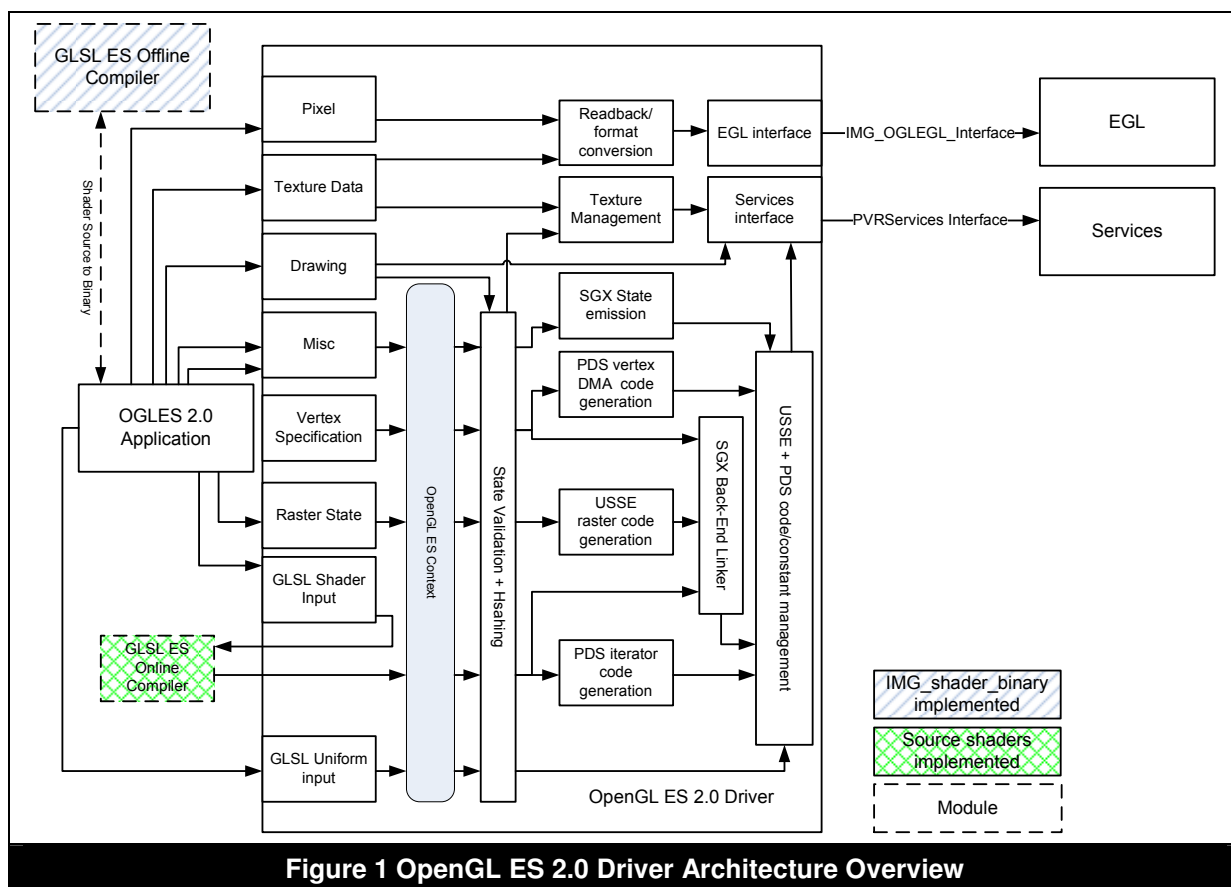


Figure 1 OpenGL ES 2.0 Driver Architecture Overview

2.2.1. OpenGL ES 2.0 Reference Driver

The reference driver contains the following functionality blocks (see SGX OpenGL ES 2.0 Reference Driver Software Functional Specification for details)

- OpenGL ES state machine
- SGX raster code set-up and raster state management
- Linking of SGX code from binary input/compiler output for vertex/fragment shaders
- SGX constant management
- SGX vertex DMA code generation and management
- Texture management
- Interfaces to TA and Render

The reference implementation contains the OpenGL ES API 2.0, in the *common profile* (only valid profile). This component is where the primary development effort of the driver resides.

OpenGL ES 2.0 allows an application to query strings to identify the driver that the application is running on. There are 5 standard queryable strings; these are the version string, the vendor string, the renderer string and the extension string. These are obtained by calling `glGetString` with `GL_VERSION`, `GL_SHADING_LANGUAGE_VERSION`, `GL_VENDOR`, `GL_RENDERER` and `GL_EXTENSIONS` respectively.

Version String

The version string returned by OpenGL ES drivers indicates the version of the API, while the shading language version string indicates which version of the GLSL ES language is supported:

- "OpenGL ES 2.0"
- "OpenGL ES GLSL 1.0"

Vendor String

The vendor string is not restricted and may be replaced by the IP customer. The reference driver string will be: "Imagination Technologies"

Renderer String

The renderer string will likely be the primary method by which developers can identify the underlying hardware, and as such is controlled so that IP customers should only modify the string in a restricted manner. It is unlikely that end developers will have any other manner to identify the underlying PowerVR devices (for example the `DeviceID` register).

The renderer string contains the text "PowerVR", followed by text to identify the core, its major version and any further variant. The customer may prepend text to the front of the renderer string. The following are examples:

- "ACME OGLES on PowerVR SGX530"
- "Magic3DWidget (TM) PowerVR SGX520"

The reference OpenGL ES driver contains the string:

- "Reference PowerVR SGX530" (for a 2 pipe SGX)

Extension String

The extension string is a space separated alphabetically sorted concatenation of the extension names supported by the driver. See the section on extensions for details of extensions supported.

Shader input

The driver can optionally support source shaders, binary shaders or both. An IMG binary format has been defined, and an extension allows access to this functionality. The driver also supports the ability to extract the shader binary after compilation. The binary format has a format version which must be compatible with the DDK which loads the binary. In pre 1.5 DDK releases, this version was equivalent to the exact DDK revision number. In 1.5+ DDK releases, this version is independent of the DDK revision, and should vary at a much slower rate. This allows binary shaders to be created which will be loadable by multiple versions of the DDK. The binary format also has a HW version which must be compatible with the DDK which loads the binary. In pre 1.5 DDK releases, this version was equivalent to the exact HW version. In 1.5+ DDK releases, this version allows compatible cores to be supported with a single binary. The list of compatible core versions can be found in the DDK in `binshader.c` in the `opengles2` directory.

2.2.2. GLSL ES Compiler

If source shaders are supported by the OpenGL ES 2.0 driver, then this component will need to be integrated into the driver. It will accept GLSL ES language source and generate optimised SGX USSE code appropriately. It will leave space for the back-end linker to insert texture and vertex format conversion code as necessary. If binary shader formats are supported, a binary input format will

defined for the driver. The GLSL ES language will be the input to an external compiler which generates this binary format.

2.2.3. Back-end Linker

This component will patch texture format conversion code into pre-generated USSE vertex and fragment code. It will also patch the destination of fragment shaders, to allow optimal insertion of driver generated back-end raster code.

2.2.4. EGL

EGL is the component that allows the OpenGL ES application to control the configuration attributes with which OpenGL ES rendering will happen, in terms of colour depth, Z-depth, support for different rendering surfaces (window/pixmap/pbuffer), and controls where the render results will appear. See the SGX EGL 1.0 Reference Driver, Software Architecture Specification.doc for details.

2.2.5. PowerVR Services Glue

The PowerVR Services glue layer exists to abstract in a platform independent method access to the PowerVR services. This layer in general should be very thin and may simply abstract the calling method by which the services are obtained, and whether on a given platform the services exist in kernel or user mode, or if they share a common address space with the reference driver. This glue layer is defined in the PowerVR Services Porting Guide for the appropriate OS.

2.3. Development constraints

To ensure the above stated portability the following constraints are applied to the development of the PowerVR OpenGL ES 2.0 reference implementation.

- As far as reasonably practical code should be written in a platform neutral style. The Non-windows typedefs and variable naming notation from the PowerVR C Style Guide should be adopted where practical.
- Particular care must be paid to the library entry points to ensure that they are as re-targetable as possible.
- The graphics context (GC) must only ever be retrieved from TLS in the entry point function. There after it must always be passed deeper into the library on the stack. The retrieval from TLS should be macro-ised to simplify porting.
- The code should be extremely well commented specifically in places where care is required to mate OpenGL ES functionality to SGX functionality.
- Code should be written for compactness, but care should be taken that this does not unduly obfuscate the code.
- Floating-point code should where possible be isolated into specific files.

3. Conformance

The Khronos Group publishes a test suite to allow implementers of OpenGL ES to badge their products as conformant. The current suite is at version 3.

The reference drivers will aim to pass within these criteria. There are some bugs filed against version 3 of the conformance tests. A known issue which can cause failures at certain resolutions is https://cvs.khronos.org/bugzilla/show_bug.cgi?id=7474

3.1. Compliance

The SGX OGLES 2.0 reference implementation is not compliant with the specification in the following areas:

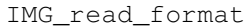
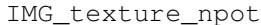
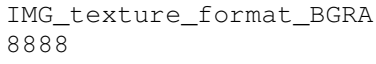
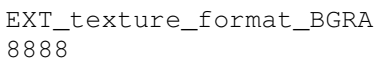
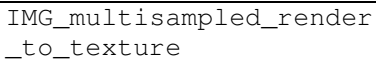
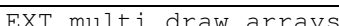
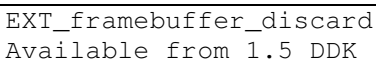
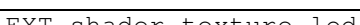
Feature	Description
Non-power of two cube maps	
Polygon offset	

4. Extensions

The following table illustrates the intended extension support.

Extension type	Extension name	Functionality
OES Ratified	OES_rgb8_rgba8 Available from 1.0 DDK	Enables rendering to RGB8 and RGBA8 renderbuffers
	OES_depth24 Available from 1.0 DDK	Enables rendering to 24 bit depth buffers attached to a framebuffer object
	OES_vertex_half_float Available from 1.0 DDK	Allows 16 bit floating point data to be supplied for vertex components
	OES_texture_float OES_texture_half_float Available from 1.0 DDK	Allows 16 and 32 bit floating point textures to be supplied to the GL
	OES_element_index_uint Available from 1.0 DDK	Allows 32 bit indices to be supplied for DrawElements calls
	OES_mapbuffer Available from 1.0 DDK	Allows vertex buffer object buffers to be mapped for the application to write directly into.
	OES_fragment_precision_high Available from 1.0 DDK	Allows the high precision qualifier to be used in a fragment shader
	OES_compressed_ETC1_RGB8_texture Available from 1.0 DDK	Allows ETC1 precompressed data to be supplied to the GL
	OES_EGL_image Available from 1.2 DDK	Allows texture and renderbuffer targets to be created from EGL images
	OES_required_internal_format Available from 1.3 DDK	Guarantees the internal format of textures
	OES_get_program_binary Available from 1.4 DDK	Allows retrieval of program binaries after shader compilation
	OES_packed_depth_stencil Available from 1.4 DDK	Allows use of a combined depth & stencil buffer for framebuffer objects
	OES_depth_texture Available from 1.4 DDK	Allows depth textures to be supplied to the GL.
	OES_standard_derivatives Available from 1.4 DDK	Allow standard GLSL derivatives to be used in a fragment shader
	OES_vertex_array_object Available from 1.5 DDK	Allows applications to rapidly switch between different sets of vertex array states, by encapsulating each set of vertex array states into one vertex array object.
	OES_EGL_sync Available from 1.6 DDK	Allows EGL to insert fences and query completion in the GL command stream.
Other	IMG_shader_binary Available from 1.0 DDK	Allows a binary shader format to be used with the GL.
	IMG_texture_compression_pvrtc Available from 1.0 DDK	Allows PVRTC precompressed data to be supplied to the GL.

	IMG_texture_stream2 Available from 1.3 DDK	Allows texture data to be streamed to the GL from a bufferclass device.
	IMG_program_binary Available from 1.5 DDK	Allows a binary program format to be used in conjunction with OES_get_program_binary

Other	 Available from 1.4 DDK	Increases the number of format/type combinations for glReadPixels
	 Available from 1.3 DDK	Allows mipmap filters to be used with non-power-of-two textures.
	 Available from 1.3 DDK  Available from 1.4 DDK	Allows texture data to be supplied in BGRA format.
	 Available from 1.6 DDK	Allows multiampled renders to textures bound to framebuffer objects with implicit multisample resolution when the texture is referenced.
	 Available from 1.3 DDK	Allows multiple draw calls to be issued in one GL call.
	 Available from 1.5 DDK	Allows framebuffer attachments to be discarded after use
	 Available from 1.6 DDK	Allows texture lookup functions with explicit LOD to be used in fragment shaders.

Following are a list of Khronos defined OpenGL ES 2.0 optional extensions which will NOT be supported:

Extension type	Extension name	Functionality	SGX520/530/531/540	SGX535
OES Ratified	OES_stencil14	Allows a render buffer to be created in a 4 bit stencil buffer format.	Not Supportable	Not Supportable
	OES_stencil11	Allows a render buffer to be created in a 1 bit stencil buffer format.	Not Supportable	Not Supportable
	OES_depth32	Enables rendering to 32 bit depth buffers attached to a framebuffer object	Not Supportable	Not Supportable
	OES_texture_float_linear	Allows filtering between texels and filtering between mipmap levels to be applied to 16 and 32 bit floating point textures.	Not Supportable	Not Supportable
	OES_texture_3D	Allows 3D textures to be supplied to the GL and 3D lookups to be performed in a shader.	Not Supportable	Supportable (Note 1)
	OES_texture_npot	Allows non-power of two textures to be mipmapped and wrapped with non-clamp modes.	Not Supportable	Not Supportable
	OES_standard_noise	Allows standard (not invariant) GLSL noise to be used in a shader	Supportable (Note 2)	Supportable (Note 2)
	OES_shadow	Allows standard GLSL shadow lookups to be used in a shader	Supportable (Note 2)	Supportable (Note 2)
	OES_data_type_10_10_10_2	Allows 10_10_10_2 texture format and vertex attrib formats to be supplied to the GL.	Supportable (Note 2)	Supportable (Note 2)
OES (currently unratified)	OES_visibility_query	Allows predicated rendering and visibility feedback	Supportable (Note 2)	Supportable (Note 2)

Note 1: To maintain consistency between the reference DDK for SGX variants, the common subset of extensions will be supported.

Note 2: Although these extensions are supportable by the HW, this release of the DDK implementation will not expose them. They may be exposed in a future DDK release.

4.1. PowerVR Texture Compression

PowerVR Texture Compression (PVR-TC) will be exposed as enumerated formats through the OpenGL ES compressed texture interface. The presence of the formats will be detectable through the vendor extension "IMG_texture_compression_pvrTC". Because PVR-TC is non-symmetric between

encoding and decoding the driver will not provide compression of supplied data through the *glTexImage2D* call with the PVR-TC internal format token. All compression must be done offline via the tools provided by PowerVR Developer Relations. In addition, the PVR-TC format cannot efficiently support loading of non-complete levels, so *glCompressedTexSubImage2D* will only support the case where the sub-image equals the level size.

4.2. Other extensions

See the OpenGL ES 2.0 spec for details of the optional extensions.

5. Implementation Dependent Values

State	Value	Comment
SUBPIXEL_BITS	4	
MAX_TEXTURE_SIZE	2048 / 4096	4096 for SGX XT cores and SGX DX10 cores (which means SGX543/544/545/554)"
MAX_CUBE_MAP_TEXTURE_SIZE	2048 / 4096	4096 for SGX XT cores and SGX DX10 cores (which means SGX543/544/545/554)"
MAX_VIEWPORT_DIMS	2048 / 4096	4096 for SGX XT cores and SGX DX10 cores (which means SGX543/544/545/554)"
ALIASED_POINT_SIZE_RANGE	0	
	511	
ALIASED_LINE_WIDTH_RANGE	1	
	16	
MAX_ELEMENTS_INDICES	65536	
MAX_ELEMENTS_VERTICES	65536	
SAMPLE_BUFFERS	0	
	1	
SAMPLES	0	
	4	2x2 anti aliasing
COMPRESSED_TEXTURE_FORMATS	GL_COMPRESSED_RGB_PVRTC_2BPPV1_IMG	
	GL_COMPRESSED_RGBA_PVRTC_2BPPV1_IMG	
	GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG	
	GL_COMPRESSED_RGBA_PVRTC_4BPPV1_IMG	
	GL_ETC1_RGB8_OES	
NUM_COMPRESSED_TEXTURE_FORMATS	5	
RENDERER	PowerVR SGX 535	
SHADING_LANGUAGE_VERSION	OpenGL ES GLSL 1.00	
VENDOR	Imagination Technologies	

State	Value	Comment
VERSION	OpenGL ES 2.0	
MAX_VERTEX_ATTRIBS	8	
MAX_VERTEX_UNIFORM_VECTORS	128	
MAX_VARYING_VECTORS	8	
MAX_COMBINED_TEXTURE_IMAGE_UNITS	8	
MAX_VERTEX_TEXTURE_IMAGE_UNITS	8	
MAX_TEXTURE_IMAGE_UNITS	8	
MAX_FRAGMENT_UNIFORM_VECTORS	64	
RED_BITS, GREEN_BITS,BLUE_BITS,ALPHA_BITS	8,8,8,8	
	4,4,4,4	
	5,5,5,1	
	5,6,5,0	
DEPTH_BITS	0, 16, 24	
STENCIL_BITS	0, 8	
IMPLEMENTATION_COLOR_READ_TYPE_OES	GL_UNSIGNED_BYTE	for ARGB8888 surface
	GL_UNSIGNED_SHORT_4_4_4_4	for ARGB4444 surface
	GL_UNSIGNED_SHORT_5_6_5	for RGB565 surface
IMPLEMENTATION_COLOR_READ_FORMAT_OES	GL_BGRA	for ARGB8888 surface
	GL_BGRA	for ARGB4444 surface
	GL_RGB	for RGB565 surface

6. Debug, Metrics and Profiling

6.1. Debug

The driver will implement synchronous debug messages out in debug builds where the underlying operating system allows, for instance in a desktop Windows environment the *OutputDebugString* function would be used. If a synchronous method is not available a non-synchronous method will be used instead (i.e. *printf*). The messages will be in one of five categories summarised as follows:

Message Level	Usage
Verbose	High frequency message, used for specific debugging
Message	Informational message that may be of use to developer
Warning	Recoverable error, or warning of incorrect behaviour
Error	Unrecoverable error, not fatal
Fatal	Unrecoverable error, leading to a fault

By default the driver will not emit messages of less severity than warning, but this default level is controllable.

6.2. Metrics

Metrics describes the collection of data that is of direct interest for assessing performance, both hardware and software, as a function of the data collectable from the OpenGL ES driver. Metrics data is output in a textual form at driver termination. The data by default is collected for all frames, but on systems that support registries or initialisation files the data can be collected over a single range of frames. Metric data will be available in debug and timing builds. The metric collection relies on the presence of a high-resolution system timer. Sample data from an existing PowerVR OpenGL ES driver is presented in Appendix B to illustrate the information that may be presented.

6.3. Profiling

Profiling describes the collection of data indicating how the OpenGL ES API is being used with emphasis on data that is useful to both driver developers and application developers to efficiently use the API on SGX hardware. Profile data is output in a textual form at driver termination. The data by default is collected for all frames, but on systems that support registries or initialisation files the data can be collected over a single range of frames. Profiling is available in debug and metrics builds. Some of the profiling data relies on the presence of a high-resolution system timer. Sample data from an existing PowerVR OpenGL ES driver is presented in Appendix C to illustrate the information that may be presented.

7. Configurations

7.1. Build time configurations

There are three build targets for any given MBX HW configuration and environment configuration these are:

Target	Description
Debug	Non-optimised build, debug statements compiled in, profiling and metric collection present
Timing	Optimised build, profiling and metric collection present, no debug statements
Release	Optimised build, no profiling or metric collection present, no debug statements

In addition to the overall build targets listed above the following compile time configurations exist, these should be set as C pre-processor defines from the target environment makefile:

Configuration	Description
TBD	

7.2. Runtime configurations

The OpenGL ES driver will read run time configuration data from the registry or initialisation file on systems that support these during driver initialisation. Systems that don't support such capabilities may not be runtime configurable.

There are several different classes of use for runtime configuration parameters. These include performance tuning, data collection, debugging, and application compatibility. Of these, only performance tuning and application compatibility will be available in the *Release* build target.

Runtime configuration options and their function and usage are listed below:

Option	Default Value	Function	Usage
DumpProfileData	0	Emit profile data at profile end frame or application normal termination	DataColl
ProfileStartFrame	0	Frame to start collecting profile and metric data	DataColl
ProfileEndFrame	0xffffffff	Frame to stop collecting profile and metric data	DataColl
DefaultVertexBufferSize	200*1024	Defines the default size of the internal vertex and index buffers. This affects the number of vertices for non VBO draw calls which can fit in the buffer before a TA kick is required. ¹	PerfTune
MaxVertexBufferSize	800*1024	Defines the maximum size of the internal vertex buffer. This can grow from the default size if a slow "batching" path is detected.	PerfTune
DefaultIndexBufferSize	200*1024	Defines the default size of the internal vertex and index buffers. This affects the number of indices for non VBO draw calls which can fit in the buffer before a TA kick is required. ¹	PerfTune

Option	Default Value	Function	Usage
DefaultPDSBufferSize	50*1024	Defines the default size of the internal vertex state/constant buffer. . This affects the number of draw calls which can fit in the buffer before a TA kick is required. ¹	PerfTune
DefaultPregenMTECopyBufferSize	50*1024	Defines the default size of the internal MTE copy state program buffer. This affects the number of state changes which can fit in the buffer before a TA kick is required. ¹	PerfTune
DefaultVDMBufferSize	20*1024	Defines the default size of the internal vertex control buffer. This affects the number of draw calls which can fit in the buffer before a TA kick is required. ¹	PerfTune
PDSFragBufferSize *	50*1024	Defines the default size of the internal fragment state/constant buffer. This affects the number of draw calls with different state which can fit in the buffer before a 3D kick is required. This buffer is only used for special objects.	PerfTune
ParamBufferSize **	1.6 DDK onwards: 1*1024*1024 From 1.0 DDK - > 1.4 DDK: 4*1024*1024	Defines the default size of the internal shared parameter buffer.	PerfTune
MaxParamBufferSize**	4*1024*1024	Defines the maximum size of the internal parameter buffer. 0 allows growth until memory is exhausted.	PerfTune 1.6 DDK onwards
DisableHWTextureUpload	0	Allows disabling of the use of the HW for texture uploads	Debug
DisableHWMipGen	0	Allows disabling of the use of the HW for mipmap generation	Debug
FlushBehaviour	0	Allows glFlush/glFinish behaviour to be changed to kick TA (FlushBehaviour=1) or kick 3D (FlushBehaviour=2). Note this only affects EGL Window Surfaces which are not used by multiple client APIs.	PerfTune
KickTAMode ***	4	Allows TA to be kicked more frequently to enable low latency scheduling.	PerfTune
KickTAThreshold ***	3	Threshold to use for TA kicking	PerfTune
ForceExternalZBuffer	0	Force upfront creation of an external Z-buffer and use it for SPM events or fragment buffer overflows (pre 1.5 DDK only)	1.0 DDK -> 1.4 DDK
ForceNoExternalZBuffer	0	Do not allow the on-demand creation of an external Z-buffer (pre 1.5 DDK only)	1.0 DDK -> 1.4 DDK

Option	Default Value	Function	Usage
ExternalZBufferMode****	1	Controls the creation and use of an external Z-buffer (post 1.4 Beta DDK only, overrides settings for ForceExternalZBuffer and ForceNoExternalZBuffer in 1.4 RC DDK)	1.4 RC DDK onwards

* Defined per surface in the EGL module.

** Defined in the EGL module.

*** See KickTAMode Table Below

**** See ExternalZBufferMode Table Below

¹ There is no direct mapping between the number of draw calls and the number of TA kicks. It will differ for all applications.

KickTAMode	Meaning	KickTAThreshold Interpretation
0	Don't submit extra TA kicks	None
1	Kick TA every N primitives (draw calls)	Number of primitives before next kick
2	Kick TA every N indices	Number of indices before next kick
3	Kick TA heuristically based on primitive count of previous frame to try to generate N TA kicks per frame	Number of kicks to try to generate per frame
4	Kick TA heuristically based on index count of previous frame to try to generate N TA kicks per frame	Number of kicks to try to generate per frame

ExternalZBufferMode	Allocation	Use	Meaning
0	On demand	Always	External Z-buffer is allocated after an SPM event or before a fragment buffer overflow, and then always used
1	On demand	As needed	External Z-buffer is allocated after an SPM event or before a fragment buffer overflow, and then only used for future SPM events or fragment buffer overflows
2	Up front	Always	External Z-buffer is allocated when EGL surface is created, and then always used
3	Up front	As needed	External Z-buffer is allocated when EGL surface is created, and then only used for SPM events or fragment buffer overflows

ExternalZBufferMode	Allocation	Use	Meaning
4	Never	Never	External Z-buffer is never created or used

8. Memory allocations

This section describes some of the memory allocations made by the SGX driver. These allocations are in two main forms: device memory and host memory.

8.1. Device memory allocations for internal driver use

The following table details some of the significant device memory allocations made by the driver for its own use:

8.1.1. Allocations in 1.2 DDK

Allocation	Storage for	Default size	Controlled by
Parameter buffer	SGX internal parameter data	4 MB	<pre>#define EGL_DEFAULT_PARAMETER_BUFFER_SIZE in include\srvcontext.h Apphint: "ParamBufferSize" (EGL module)</pre>
Vertex buffer	Dynamic vertex data	200 KB	<pre>ui32Default = 200*1024; in opengles2\misc.c Apphint: "DefaultVIBufferSize"</pre>
Index buffer	Dynamic index data	200 KB	<pre>ui32Default = 200*1024; in opengles2\misc.c Apphint: "DefaultVIBufferSize"</pre>
VDM buffer	VDM input control stream	10 KB	<pre>ui32Default = 10*1024; in opengles2\misc.c Apphint: "DefaultVDMBufferSize"</pre>
Vertex PDS buffer	Vertex state and constants	50 KB	<pre>ui32Default = 50*1024; in opengles2\misc.c Apphint: "DefaultPDSVertBufferSize"</pre>
Pixel PDS buffer	Dynamic PDS pixel programs	50 KB	<pre>#define EGL_DEFAULT_PDS_FRAG_BUFFER_SIZE in include\srvcontext.h Apphint: "PDSFragBufferSize" (EGL module)</pre>

Allocation	Storage for	Default size	Controlled by
Vertex USSE code heap	Static USSE vertex programs	32 KB, grows to maximum of 2 MB	<code>#define</code> SGX_VERTEXSHADER_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h
Pixel USSE code heap	Static USSE pixel programs	32 KB, grows to maximum of 5 MB	<code>#define</code> SGX_PIXELSHADER_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h
Vertex PDS code heap	Static PDS vertex programs	32 KB, grows to maximum of 32 MB	<code>#define</code> SGX_PDSVERTEX_CODEDATA_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h
Pixel PDS code heap	Static PDS pixel programs	32 KB, grows to maximum of 32 MB	<code>#define</code> SGX_PDSPIXEL_CODEDATA_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h

8.1.2. Allocations in 1.3+ DDK (including performance improvements)

Allocation	Storage for	Default size	Controlled by
Parameter buffer	SGX internal parameter data	4 MB	<code>#define</code> EGL_DEFAULT_PARAMETER_BUFFER_SIZE in include\srvmcontext.h Apphint: "ParamBufferSize" (EGL module)
Vertex buffer	Dynamic vertex data	200 KB	ui32Default = 200*1024; in opengles2\misc.c Apphint: "DefaultVertexBufferSize"
Index buffer	Dynamic index data	200 KB	ui32Default = 200*1024; in opengles2\misc.c Apphint: "DefaultIndexBufferSize"
VDM buffer	VDM input control stream	20 KB	ui32Default = 20*1024; in opengles1\misc.c Apphint: "DefaultVDMBufferSize"
Vertex PDS buffer	Vertex state and constants	50 KB	ui32Default = 50*1024; in opengles2\misc.c Apphint: "DefaultPDSVertBufferSize"
Pregen MTE Copy Buffer	Static MTE copy state programs	50KB	ui32Default = 50*1024; in opengles2\misc.c Apphint: "DefaultPregenMTECopyBufferSize"

Allocation	Storage for	Default size	Controlled by
Pixel PDS buffer	Dynamic PDS pixel programs	50 KB	#define EGL_DEFAULT_PDS_FRAG_BUFFER_SIZE in include\srvcontext.h Apphint: "PDSFragBufferSize" (EGL module)
Vertex USSE code heap	Static USSE vertex programs	32 KB, grows to maximum of 2 MB	#define SGX_VERTEXSHADER_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h
Pixel USSE code heap	Static USSE pixel programs	32 KB, grows to maximum of 5 MB	#define SGX_PIXELSHADER_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h
Vertex PDS code heap	Static PDS vertex programs	32 KB, grows to maximum of 32 MB	#define SGX_PDSVERTEX_CODEDATA_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h
Pixel PDS code heap	Static PDS pixel programs	32 KB, grows to maximum of 32 MB	#define SGX_PDSPIXEL_CODEDATA_HEAP_SIZE in services4\srvm\devices\sgx\sgxconfig.h

8.2. Device memory allocations for application supplied data

The OpenGL ES driver will allocate device memory to store application supplied data such as textures and vertex buffer objects.

The size of these allocations depends on the size of the application supplied data and on associated OpenGL ES state (e.g. internal texture format).

8.3. Host memory allocations for internal driver use

The following table details some of the significant host memory allocations made by the driver for its own use:

Allocation	Storage for	Size	Controlled by
OpenGL ES context	OpenGL ES state	Approximately 14 KB	C-structure

8.4. Host memory allocations for application supplied data

The OpenGL ES driver will allocate host memory to store application supplied texture data. After the texture data is copied to device memory, the host copy is freed. The size of these allocations depends on the size of the application supplied data and on associated OpenGL ES state (e.g. internal texture format).

Appendix A. OGL ES Reference Driver Code Modules

The following table lists all the source files compiled to construct the OpenGL ES 2.0 reference driver. Each entry is accompanied by a short description:

Source File	Description
TBD	

Appendix B. Sample Metric Data

PVR:	Total Frames		1
PVR:	Profiled Frames	(0-1)	1
PVR:	Average Elapsed Time per Profiled Frame (ms)		0.1262
PVR:	Average Driver Time per Profiled Frame (ms)		50.3829
PVR:	Estimated FPS		7922.6231
PVR:	Statistics per profiled frame [Calls/Time(ms)]		
PVR:	Total Prepare to draw	3/	0.0789
PVR:	Total SGXKickTA	1/	0.0618
PVR:	Total Renders	1.0000/	-
PVR:	Total Wait for 3D	1/	0.0005
PVR:	Total Wait for TA	3/	0.0007
PVR:	Total ValidateState()	2/	0.3528
PVR:	- Setup_Streams()	2/	0.0014
PVR:	- Setup_AttribPointers()	2/	0.0014
PVR:	- Setup_TextureState()	2/	0.2396
PVR:	- Setup_VertexShader()	2/	0.0506
PVR:	- Setup_RenderState()	2/	0.0023
PVR:	- Setup_FragmentShader()	2/	0.0542
PVR:	Total EmitState()	2/	0.1534
PVR:	- WritePDSPixelShaderProgram()	2/	0.0104
PVR:	- WritePDSVertexShaderProgram()	2/	0.0082
PVR:	- PatchPDSVertexShaderProgram()	0/	0.0000
PVR:	- WritePDSUSEShaderSAProgram(Frag)	2/	0.0124
PVR:	- WritePDSUSEShaderSAProgram(Vert)	2/	0.0263
PVR:	- WriteMTEState()	2/	0.0041
PVR:	- WriteMTEStateCopyPrograms()	2/	0.0807
PVR:	- WriteVDMControlStream()	6/	0.0064
PVR:	USE Codeheap Alloc - Fragment	5/	0.0130
PVR:	USE Codeheap Alloc - Vertex	7/	0.0072
PVR:	PDS Codeheap Alloc - Fragment	3/	0.0025
PVR:	Total NamesArray operations	98/	0.1256
PVR:	Total Frame Resource Manager ops.	16/	0.0178
PVR:	Total Frame Resource Manager Waits	3/	0.0018
PVR:	Total Code Heap operations	33/	0.3096
PVR:	Total Primitive Draw	2/	0.5424
PVR:	Total Vertex Data Copy	2/	0.0050
PVR:	Total Index Data Generate/Copy	2/	0.0002
PVR:	Array Triangles	2/	0.5405
PVR:	PDS Variant hit/miss totals		
PVR:	PDSPixelShaderProgram - variant hit	0	
PVR:	PDSPixelShaderProgram - variant miss	2	
PVR:	Statistics per call [Maximum time (ms) in a single call]		
PVR:	Max Prepare to draw	0.078284	
PVR:	Max SGXKickTA	0.061839	
PVR:	Max Wait for 3D	0.000499	
PVR:	Max Wait for TA	0.000619	
PVR:	Max ValidateState()	0.280907	
PVR:	Max EmitState()	0.121094	
PVR:	Max NamesArray operation	0.091802	
PVR:	Max Frame Resource Manager op.	0.005438	
PVR:	Max Frame Resource Manager Wait	0.000858	

```

PVR: Max Code Heap operation                                0.135283
PVR:
PVR: Buffer Statistics                                     [  kB/kB per Profiled
Frame  ]
PVR: VDM Control                                           0.0303/      0.0303
PVR: MTE State                                             0.0371/      0.0371
PVR: Vertex data                                           0.0566/      0.0566
PVR: Index data                                            0.0059/      0.0059
PVR: PDS vertex data                                       0.2617/      0.2617
PVR: PDS fragment data                                    0.1211/      0.1211
PVR: USE Codeheap - Fragment                              0.1250/      0.1250
PVR: USE Codeheap - Vertex                               0.4922/      0.4922
PVR: PDS Codeheap - Fragment                             0.1250/      0.1250
PVR:
PVR: Average TexImage Load MTexels/s                      42.5662
PVR:
      Texture Statistics                                     [  Calls/perCall-Time|Texels  /
perFrame-Time/Texels  ]
PVR: TexImage                                             1/      0.0241/      1024/
0.0241/ 1024.00
PVR: TexSubImage                                         0/      0.0000/      0/
0.0000/ 0.00
PVR:
      Texture Statistics                                     [  Calls/Time per Call/Time per
Frame (ms)]
PVR: CopyImage                                           0/      0.0000/      0.0000
PVR: AllocateTexture                                    1/      0.1456/      0.1456
PVR: TranslateLoadTexture                                1/      0.0856/      0.0856
PVR: Load Ghost Texture                                 0/      0.0000/      0.0000
PVR: Texture Readback                                    0/      0.0000/      0.0000
PVR:
      Shader Statistics                                     [  Calls/Avg time per Call/Max time
per call/Time per Profiled Frame (ms)]
PVR: CompileShader                                       3/      16.5214/      32.6514/
49.5641
PVR: LinkProgram                                         2/      0.0127/      0.0178/
0.0254
PVR: ShaderSource                                        3/      0.0048/      0.0066/
0.0143
PVR:
PVR: Texture allocation HWM = 8195 bytes
PVR:
PVR: Memory Stats
PVR: -----
PVR:
PVR: High Water Mark = 2123670 bytes
PVR:
  
```

Appendix C. Sample Profile Data

```

Profiling (per context)
=====

Draw calls profiling

DrawArrays          Vertices      Calls      Vertices/Frame      Calls/Frame
GL_TRIANGLES        6          2           6                   2

Entry point profiling (Times in uSx10)

Function                                Calls  Time/Call  Calls/Frame  Time/Frame
glAttachShader                        4    0.001009      4    0.004034
glBindAttribLocation                  4    0.006524      4    0.026097
glClear                              1    0.123660      1    0.123660
glClearColor                         1    0.000629      1    0.000629
glCompileShader                       3   16.626373     3   49.879119
glCreateProgram                       2    0.001089      2    0.002178
glCreateShader                        3    0.001324      3    0.003972
glDisableVertexAttribArray           1    0.000390      1    0.000390
glDrawArrays                         2    0.258635      2    0.517271
glEnableVertexAttribArray             3    0.000225      3    0.000676
glGetAttribLocation                   2    0.001188      2    0.002376
glGetError                           1    0.000052      1    0.000052
glGetProgramiv                       4    0.000568      4    0.002272
glGetShaderiv                        3    0.004838      3    0.014515
glGetUniformLocation                 8    0.000700      8    0.005599
glLinkProgram                        2    0.011744      2    0.023487
glShaderSource                       3    0.004051      3    0.012154
glTexImage2D                         1    0.023300      1    0.023300
glTexParameterf                      4    0.000260      4    0.001040
glUniform1i                          2    0.001742      2    0.003483
glUniform1f                          2    0.000522      2    0.001045
glUniformMatrix4fv                   4    0.000905      4    0.003618
glUseProgram                         4    0.000711      4    0.002844
glValidateProgram                    2    0.001172      2    0.002345
glVertexAttribPointer                 3    0.000752      3    0.002256

State combinations used in rendering

State      0
DITHER     x

State      0
Usage Count 2

```