

# OpenGL ES 2.0 Emulator

Version: 1.2.0

## User Guide



# OpenGL ES 2.0 Emulator

## User Guide

Copyright © 2009-2010 ARM. All rights reserved.

### Release Information

The following changes have been made to this book.

			Change history
Date	Issue	Confidentiality	Change
14 October 2009	A	Non-Confidential	First release for v1.1
9 April 2010	B	Non-Confidential	Updated for v1.2 Beta.
30 July 2010	C	Non-Confidential	Updated for v1.2.0 release. Changed name of installation files.

### Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

### Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

### Product Status

The information in this document is final, that is for a developed product.

### Web Address

<http://www.arm.com>

# Contents

## OpenGL ES 2.0 Emulator User Guide

	<b>Preface</b>	
	About this book .....	viii
	Feedback .....	x
<b>Chapter 1</b>	<b>Introduction</b>	
	1.1 About the OpenGL ES 2.0 Emulator .....	1-2
<b>Chapter 2</b>	<b>Installation and Configuration on Windows</b>	
	2.1 Installing the OpenGL ES 2.0 Emulator on Windows .....	2-2
	2.2 Configuring the OpenGL ES 2.0 Emulator on Windows .....	2-6
	2.3 Building the OpenGL ES 2.0 Cube example on Windows .....	2-9
<b>Chapter 3</b>	<b>Installation and Configuration on Linux</b>	
	3.1 Installing the OpenGL ES 2.0 Emulator on Linux .....	3-2
	3.2 Configuring the OpenGL ES 2.0 Emulator on Linux .....	3-6
	3.3 Building the OpenGL ES 2.0 Cube example on Linux .....	3-9
<b>Chapter 4</b>	<b>Operational modes</b>	
	4.1 Normal and Instrumented operational modes .....	4-2
	4.2 Instrumentation parameters .....	4-3
	4.3 Enabling instrumentation on Windows .....	4-4
	4.4 Enabling instrumentation on Linux .....	4-6
	4.5 Enabling Debugging .....	4-7
	4.6 Building and running the example with instrumentation .....	4-10
<b>Chapter 5</b>	<b>Implementation Information</b>	
	5.1 OpenGL ES 2.0 Implementation information .....	5-2
	5.2 EGL implementation information on Windows .....	5-6

	5.3	EGL implementation information on Linux .....	5-10
<b>Appendix A</b>		<b>Adding registry settings in Windows</b>	
	A.1	Enabling instrumentation .....	A-2
		<b>Glossary</b>	

# List of Tables

## OpenGL ES 2.0 Emulator User Guide

	Change history .....	ii
Table 2-1	Emulator directory file contents .....	2-4
Table 2-2	OpenGL ES 2.0 Emulator library structure .....	2-7
Table 3-1	OpenGL ES 2.0 Emulator directory file contents .....	3-4
Table 3-2	OpenGL ES 2.0 Emulator library structure .....	3-6
Table 4-1	Windows registry keys for enabling offline instrumentation .....	4-4
Table 4-2	Windows registry keys for enabling online instrumentation .....	4-4
Table 4-3	Linux environment variables for offline instrumentation .....	4-6
Table 4-4	Linux environment variables for online instrumentation .....	4-6
Table 4-5	Windows registry key settings for debugging .....	4-8
Table 4-6	Linux environment variables for debugging .....	4-9
Table A-1	Registry file values .....	A-2

# List of Figures

## OpenGL ES 2.0 Emulator User Guide

Figure 2-1	Emulator directory structure .....	2-4
Figure 2-2	Cube example .....	2-9
Figure 3-1	OpenGL ES 2.0 Emulator directory structure .....	3-4
Figure 3-2	Cube image .....	3-9
Figure 4-1	Performance counters .....	4-10
Figure A-1	Registry editor key creation .....	A-2
Figure A-2	Creating a String Value .....	A-3
Figure A-3	Created value .....	A-3
Figure A-4	Edit String dialog box .....	A-3
Figure A-5	Registry file values .....	A-4

# Preface

This preface introduces the *OpenGL ES Emulator 2.0 User Guide*. It contains the following sections:

- *About this book* on page viii
- *Feedback* on page x.

## About this book

This is the *OpenGL ES 2.0 Emulator User Guide*. It provides guidelines for using the OpenGL ES 2.0 Emulator to develop 2D and 3D graphics applications that are targeted to run on an embedded platform. This book is part of a suite belonging to the Mali Developer Tools.

## Intended audience

This guide is written for system integrators and software developers using a PC to develop OpenGL ES 2.0 applications that are targeted to run on an embedded platform.

## Using this book

This book is organized into the following chapters:

### **Chapter 1 Introduction**

Read this for an introduction to the OpenGL ES 2.0 Emulator.

### **Chapter 2 Installation and Configuration on Windows**

Read this for a description on how to install and configure the emulator on Windows.

### **Chapter 3 Installation and Configuration on Linux**

Read this for a description on how to install and configure the emulator on Linux.

### **Chapter 4 Operational modes**

Read this for information about the operational modes available in the OpenGL ES 2.0 Emulator.

### **Chapter 5 Implementation Information**

Read this for information about the implementation of the OpenGL ES 2.0 and EGL APIs in the emulator.

### **Appendix A Adding registry settings in Windows**

Read this for information about how to add registry settings in Windows.

**Glossary** Read this for definitions of terms used in this book.

## Typographical Conventions

The typographical conventions are:

<i>italic</i>	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
<b>bold</b>	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u>monospace</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<i>monospace italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
<b>monospace bold</b>	Denotes language keywords when used outside example code.

- < **and** >            Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:
- MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode\_2>

## Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, <http://infocenter.arm.com>, for access to ARM documentation.

### ARM publications

This guide contains information that is specific to the Mali Developer Tools. See the following documents for other relevant information:

- *Mali GPU Developer Tools Technical Overview* (ARM DUI 501)
- *Mali GPU Performance Analysis Tool User Guide* (ARM DUI 0502)
- *Mali GPU Texture Compression Tool User Guide* (ARM DUI 0503)
- *Mali GPU Shader Developer Studio User Guide* (ARM DUI 0504)
- *Mali GPU User Interface Engine User Guide* (ARM DUI 0505)
- *OpenGL ES 1.1 Emulator User Guide* (ARM DUI 0506)
- *Mali GPU Mali Binary Asset Exporter User Guide* (ARM DUI 0507)
- *Mali GPU Shader Library User Guide* (ARM DUI 0510)
- *Mali GPU Offline Shader Compiler User Guide* (ARM DUI 0513).

### Other publications

This section lists relevant documents published by third parties:

- *OpenGL ES 1.1 Specification* at <http://www.khronos.org>.
- *OpenGL ES 2.0 Specification* at <http://www.khronos.org>.
- *OpenGL ES Shading Language Specification* at <http://www.khronos.org>.
- *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2* (5th Edition, 2005), Addison-Wesley Professional. ISBN 0-321-33573-2.
- *OpenGL Shading Language* (2nd Edition, 2006), Addison-Wesley Professional. ISBN 0-321-33489-2.

## Feedback

ARM welcomes feedback on this product and its documentation.

### Feedback on this product

If you have any comments or suggestions about this product then contact [malidevelopers@arm.com](mailto:malidevelopers@arm.com) and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

### Feedback on content

If you have comments on content then send an e-mail to [errata@arm.com](mailto:errata@arm.com). Give:

- the title
- the number, ARM DUI 0511C
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

# Chapter 1

## Introduction

This chapter provides information about the OpenGL ES 2.0 Emulator, and describes how to start using the tool in your particular workflow.

It contains the following section:

- *About the OpenGL ES 2.0 Emulator* on page 1-2.

## 1.1 About the OpenGL ES 2.0 Emulator

The OpenGL ES 2.0 Emulator is a library that maps OpenGL ES 2.0 API calls to the OpenGL 2.0 API. The graphics card on the PC must support OpenGL 2.0 for the emulator to work.

### 1.1.1 Mali GPU Offline Shader Compiler

The Mali GPU Offline Shader Compiler is provided with the Mali Developer Tools. It translates vertex shaders and fragment shaders written in the OpenGL *ES Shading Language* (ESSL) into binary vertex and fragment shaders.

The Mali GPU Offline Shader Compiler is used by the OpenGL ES 2.0 Emulator and Shader Development Studio to check the syntax of shaders before they are sent for rendering. It compiles each shader in the background and gathers data on any warnings or errors that are generated. If the Mali GPU Offline Shader Compiler is not installed, the OpenGL ES 2.0 Emulator and Mali Development Studio are unable to perform syntax checking on the application shaders.

See *OpenGL ES 2.0 Emulator integration* on page 2-6 and the *Mali GPU Shader Development Studio User Guide*.

# Chapter 2

## Installation and Configuration on Windows

This chapter provides information about installing and configuring the Mali GPU OpenGL ES 2.0 Emulator on Microsoft Windows. It contains the following sections:

- *Installing the OpenGL ES 2.0 Emulator on Windows* on page 2-2
- *Configuring the OpenGL ES 2.0 Emulator on Windows* on page 2-6
- *Building the OpenGL ES 2.0 Cube example on Windows* on page 2-9.

## 2.1 Installing the OpenGL ES 2.0 Emulator on Windows

The installation procedure varies depending on the OS being used. This section describes the installation on Microsoft Windows.

---

**Note**

---

The OpenGL ES 2.0 Emulator has been tested successfully on a 32-bit computer.

---

### 2.1.1 Supported Hardware and Software

The OpenGL ES 2.0 Emulator, Windows version, has been tested with the following hardware and software:

- The OpenGL ES 2.0 Emulator:
  - Windows XP Professional, version 2002, service pack 3
  - NVIDIA GeForce 210 graphics card with driver version 190.45.

---

**Note**

---

The graphics card and driver versions are recommendations. The emulator typically also works with other graphics cards and driver versions provided they support OpenGL 2.0 or above with appropriate extensions. Other Graphics card versions provided with drivers might support OpenGL 2.x with appropriate extensions. The minimum value for x in OpenGL 2.x is 0.

The minimum appropriate extensions are:

- WGL\_ARB\_extensions\_string
- WGL\_ARB\_pixel\_format
- WGL\_ARB\_pbuffer
- WGL\_ARB\_render\_texture
- EXT\_framebuffer\_object.

Wherever possible, update your drivers to the latest version.

---

- The Windows version of the OpenGL ES 2.0 Emulator was built with Microsoft Visual Studio 2005 and links with applications developed using Microsoft Visual Studio 2005.

#### Determining the driver version for the video card

To determine the NVIDIA driver version:

1. Right click on the desktop to open the NVIDIA Control Panel.
2. In the NVIDIA Control Panel, under the Help menu, select **System Information**.
3. Select the **Display** tab.
4. The version number appears as ForceWare version.

To determine the ATI driver version:

1. Right click on the desktop.
2. Select **Catalyst(TM) Control Center**.
3. In the left-hand menu, expand **Information Center**.
4. Under Information Center, select **Graphics Software**.
5. The version number appears as Catalyst(TM) version.

### 2.1.2 Disk requirements

The OpenGL ES 2.0 Emulator requires a minimum of 5MB disk space.

### 2.1.3 Installation procedure

This section describes the installation procedure, it contains the following sections:

- *Installing the OpenGL 2.0 Emulator on Microsoft Windows*
- *OpenGL 2.0 ES Emulator content.*

#### Installing the OpenGL 2.0 Emulator on Microsoft Windows

To install the OpenGL 2.0 Emulator on a Windows system:

1. Go to the Mali Developer Center website at:  
<http://www.malideveloper.com>
2. Select the package to download:  
OpenGL\_ES\_2\_0\_Emulator\_m.n.o.p\_Win32.msi  
where:  
**m** identifies the major version  
**n** identifies the minor version.  
**o.p** identifies the part and build version.
3. Run the OpenGL\_ES\_2\_0\_Emulator\_m.n.o.p\_Win32.msi file by double clicking on it.
4. Select the installation options and click **Finish** to complete the installation.

By default, the OpenGL ES 2.0 Emulator is installed in:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 2.0 Emulator m.n.o\

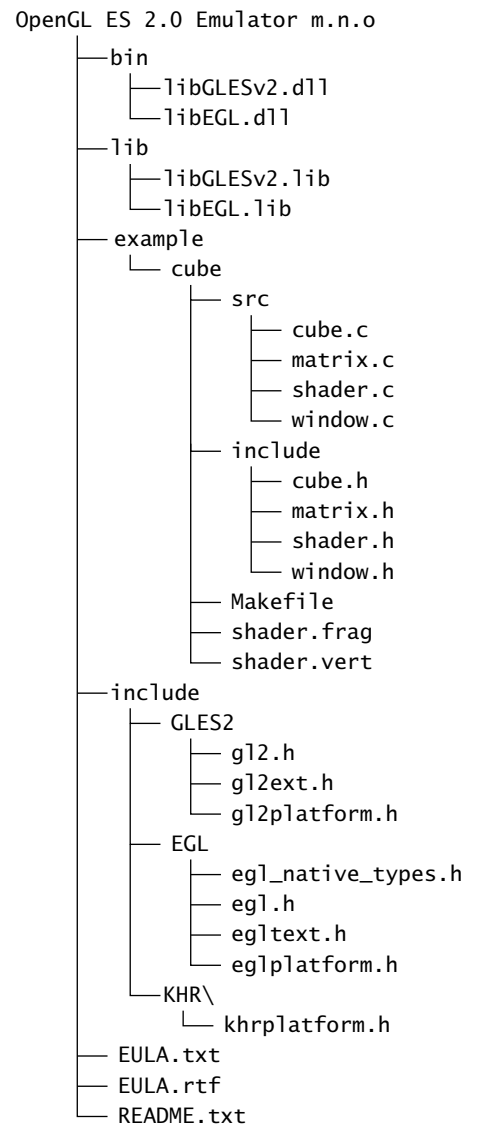
#### OpenGL 2.0 ES Emulator content

The download package contains the OpenGL ES 2.0 Emulator binaries for Windows and a simple example application, based on OpenGL ES 2.0, that runs on the OpenGL ES 2.0 Emulator.

For more information see the *OpenGL ES 2.0 Release Notes*.

Figure 2-1 on page 2-4 shows the directory structure that is created at the path where you installed the emulator. The default installation directory is:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 2.0 Emulator m.n.o\



**Figure 2-1 Emulator directory structure**

Table 2-1 shows the OpenGL ES 2.0 Emulator file contents.

**Table 2-1 Emulator directory file contents**

Filename	Description
bin\libGLESv2.dll	DLL for OpenGL ES 2.0 Emulator
bin\libEGL.dll	DLL for EGL implementation
lib\libGLESv2.lib	Import library for OpenGL ES 2.0 Emulator
lib\libEGL.lib	Import library for EGL implementation
example\cube\src\cube.c	Main cube application source code
example\cube\src\matrix.c	Utility functions for dealing with matrices
example\cube\src\shader.c	Utility functions for dealing with shader files

**Table 2-1 Emulator directory file contents (continued)**

<b>Filename</b>	<b>Description</b>
example\cube\src>window.c	Utility functions for creating windows
example\cube\include\cube.h	Header file for the cube application
example\cube\include\matrix.h	Header file for matrix manipulation code
example\cube\include\shader.h	Header file for shader file manipulation code
example\cube\include>window.h	Header file for window code
example\cube\Makefile	Makefile for cube example
example\cube\shader.frag	Fragment shader for cube example
example\cube\shader.vert	Vertex shader for cube example
include\GL\GL\gl2.h	Khronos header file
include\GL\GL\gl2ext.h	Khronos header file
include\GL\GL\glplatform.h	Khronos header file
include\GL\egl\native_types.h	Khronos header file
include\GL\egl.h	Khronos header file
include\GL\egltext.h	Khronos header file
include\GL\eglplatform.h	Khronos header file
include\KHR\khrplatform.h	Khronos header file
EULA.txt	End User Licence Agreement as a text file
EULA.rtf	End User Licence Agreement as an rtf file
README.txt	README file

## 2.2 Configuring the OpenGL ES 2.0 Emulator on Windows

This section provides information about installing and configuring the Emulator. It contains the following sections:

- *Using the OpenGL ES 2.0 Emulator*
- *OpenGL ES 2.0 Emulator integration.*

### 2.2.1 Using the OpenGL ES 2.0 Emulator

To run any OpenGL ES 2.0 application on the OpenGL ES 2.0 Emulator, you must link it to the OpenGL ES 2.0 Emulator libraries:

- The static libraries, `libEGL.lib` and `libGLESv2.lib`, are linked while building an OpenGL ES 2.0 application.
- The dynamically linked libraries, `libEGL.dll` and `libGLESv2.dll`, must be provided at run-time.

There can be problems caused by running applications on the OpenGL ES 2.0 Emulator if the OpenGL ES 1.1 Emulator is also installed on the PC. To avoid problems, you must do the following before running an application on an emulator:

1. Right-click on **My Computer** → **Properties** → **Advanced** → **Environment Variables**
2. Ensure that the system environment variable `Path` contains the path to `libEGL.dll` and `libGLESv2.dll` before the path to the OpenGL ES 1.1 Emulator libraries.  
The application will use the OpenGL 2.0 DLLs at runtime.
3. Ensure that the system environment variable `GLES2_LIB_DIR` contains the path to `libEGL.lib` and `libGLESv2.lib` before the path to the OpenGL ES 1.1 Emulator libraries.  
The linker will use the OpenGL 2.0 libraries to build the application.

———— **Note** ————

To run an application on the OpenGL ES 1.1 Emulator, reverse the order of the libraries and DLLs. Ensure that the system environment variables point to the OpenGL ES 1.1 Emulator libraries before the OpenGL ES 2.0 Emulator libraries.

### 2.2.2 OpenGL ES 2.0 Emulator integration

This section describes:

- *DLLs and libraries*
- *EGL configuration* on page 2-7
- *EGL context creation* on page 2-8
- *Shader syntax checking by the Mali GPU Offline Shader Compiler* on page 2-8
- *Shader language version* on page 2-8.

#### DLLs and libraries

The OpenGL ES 2.0 Emulator Library consists of two DLLs, corresponding to the separate OpenGL ES 2.0 and EGL 1.3 APIs. For each of these DLLs, there is a corresponding import library for an OpenGL ES 2.0 application to statically link against. Applications must include both of the import libraries in builds to link against the OpenGL ES 2.0 and EGL 1.3 APIs. The DLLs use the `__stdcall` calling convention.

Table 2-2 shows the OpenGL ES 2.0 Emulator library structure

**Table 2-2 OpenGL ES 2.0 Emulator library structure**

Filename	Description
bin\libGLESv2.dll	DLL for OpenGL ES 2.0 Emulator
bin\libEGL.dll	DLL for EGL implementation
lib\libGLESv2.lib	Import library for libGLESv2.dll
lib\libEGL.lib	Import library for libEGL.dll

**Note**

The OpenGL ES 2.0 Emulator can operate in normal or instrumented mode:

- In the normal mode of operation, the OpenGL ES 2.0 Emulator does not dump any performance counters.
- In the instrumented mode of operation, the OpenGL ES 2.0 Emulator dumps performance counters that can be viewed by the Mali GPU Performance Analysis Tool.

See Chapter 4 *Operational modes* for more information.

### EGL configuration

The EGL library supplied with the OpenGL ES 2.0 Emulator supports OpenGL ES 2.0 only. Ensure that, in the OpenGL ES 2.0 application, the attribute list passed as a parameter to `eglChooseConfig` includes the attribute `EGL_RENDERABLE_TYPE` set to the value `EGL_OPENGL_ES2_BIT`.

Example 2-1 shows a coded section.

#### Example 2-1

```

EGLDisplay Display;

EGLint Attributes[] = {
    EGL_RENDERABLE_TYPE, EGL_OPENGL_ES2_BIT,

    EGL_RED_SIZE, 8,
    EGL_GREEN_SIZE, 8,
    EGL_BLUE_SIZE, 8,
    EGL_NONE
};
EGLConfig Configs[1];

EGLint NumConfigs;

...

eglChooseConfig(Display, Attributes, Configs, 1, &NumConfigs);

```

## EGL context creation

The EGL library supplied with the OpenGL ES 2.0 Emulator supports OpenGL ES 2.0 contexts only. It is important to ensure that, in the OpenGL ES 2.0 application, the attribute list passed as a parameter to `eglCreateContext` includes the attribute `EGL_CONTEXT_CLIENT_VERSION` set to the value 2.

Example 2-2 shows a coded section.

### Example 2-2

---

```

EGLDisplay Display;

EGLConfig Configs[1];

EGLint ContextAttributes[] = {
    EGL_CONTEXT_CLIENT_VERSION, 2,
    EGL_NONE
};

...

Context = eglCreateContext(Display, Configs[0], EGL_NO_CONTEXT,
                          ContextAttributes);

```

---

## Shader syntax checking by the Mali GPU Offline Shader Compiler

The shading language for use with the OpenGL ES 2.0 API is *OpenGL ES Shading Language* (ESSL). See the *OpenGL ES Shading Language Specification*. The corresponding shading language for use with OpenGL 2.0 API is *OpenGL Shading Language* (GLSL).

The OpenGL ES 2.0 Emulator validates the shader source:

1. If the Mali GPU Offline Shader Compiler is installed and is present in the PATH environment variable, the OpenGL ES 2.0 Emulator uses this compiler to check the shader syntax for the ESSL code.
2. The OpenGL ES 2.0 Emulator modifies the validated ESSL code to make it compliant GLSL code.
3. The generated GLSL code is passed to the GLSL compiler in the OpenGL graphics driver of your Windows desktop machine.

## Shader language version

The OpenGL ES 2.0 Emulator checks whether the graphics card has version 1.2 of the *OpenGL 2.0 Shader Language* (GLSL) available:

- If version 1.2 is available, this is selected by using `pragma #version 120` in the conversion of the ESSL shader to GLSL code.
- If version 1.2 is not available, the `pragma #version 110` selects GLSL version 1.1. This limits some features.

## 2.3 Building the OpenGL ES 2.0 Cube example on Windows

This section describes how to build the OpenGL ES 2.0 cube example on Windows.

The cube example code for using the OpenGL ES 2.0 Emulator is included in the following directory:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 2.0 Emulator *m.n.o*\example\cube\src

To build and run this example:

1. Ensure the OpenGL ES 2.0 Emulator is installed.
2. Ensure that locations of the DLLs for OpenGL ES 2.0 emulator are added to the system environment variable Path. This is described in *Using the OpenGL ES 2.0 Emulator* on page 2-6.
3. The next steps use Microsoft Visual Studio 2005 to build and run the application:

- a. **Start → All Programs → Microsoft Visual Studio 2005 → Visual Studio Tools → Visual Studio 2005 Command Prompt**

- b. Change directory to the following directory that contains the example:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 2.0 Emulator *m.n.o*\example\cube

- c. To build the example application, type the following command at the command prompt:

```
nmake
```

- d. To run the example application, type the following command at the command prompt:

```
cube.exe
```

- e. An additional window with a spinning, colored cube appears when the example application is running. Figure 2-2 shows an image.

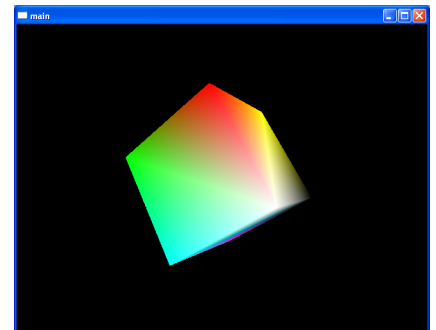


Figure 2-2 Cube example

- f. To end the program, close this window.

# Chapter 3

## Installation and Configuration on Linux

This chapter provides information about installing and configuring the OpenGL ES 2.0 Emulator on Linux OS. It contains the following sections:

- *Installing the OpenGL ES 2.0 Emulator on Linux* on page 3-2
- *Configuring the OpenGL ES 2.0 Emulator on Linux* on page 3-6
- *Building the OpenGL ES 2.0 Cube example on Linux* on page 3-9.

## 3.1 Installing the OpenGL ES 2.0 Emulator on Linux

The installation procedure varies depending on the OS being used. This section describes the installation on Ubuntu 10.04 LTS.

———— **Note** —————

The OpenGL ES 2.0 Emulator has been tested successfully on a 32-bit computer.

---

### 3.1.1 Supported Hardware and Software

The OpenGL ES 2.0 Emulator, Linux version, has been tested with the following hardware and software:

- Ubuntu 10.04 LTS
- NVIDIA GeForce 210 graphics card with driver version 195.36.15

———— **Note** —————

The graphics card and driver versions are recommendations. The emulator typically also works with other graphics cards and driver versions provided they support OpenGL 2.0 or above, with appropriate extensions. The platform must also support GLX 1.4. Wherever possible, update your drivers to the latest version.

---

#### NVIDIA driver version

To determine the NVIDIA driver version on a Linux machine:

1. Open the terminal, and type the following command:  
`nvidia-settings`
2. A dialog box is opened with the card and driver details.

### 3.1.2 Disk requirements

The OpenGL ES 2.0 Emulator requires a minimum of 5MB disk space.

### 3.1.3 Installation procedure

This section describes the installation procedure, it contains the following sections:

- *Installing the OpenGL ES 2.0 Emulator*
- *OpenGL ES 2.0 Emulator content on page 3-3.*

#### Installing the OpenGL ES 2.0 Emulator

To install the OpenGL ES 2.0 Emulator on a Linux system:

1. Go to the Mali Developer Center website at:  
<http://www.malideveloper.com>
2. Select the package to download:  
`OpenGL_ES_2_0_Emulator_m.n.o.p_Linux.tar.gz`

———— **Note** —————

where:

- m*** identifies the major version
- n*** identifies the minor version.

***o.p*** identifies the part and build version.

---

3. To decompress the file, type the following command:

```
tar -zxvf OpenGL_ES_2_0_Emulator_m.n.o.p.Linux.tar.gz
```

———— **Note** —————

You must use GNU tar version 1.16, or a later version, to untar the deliverables, because many versions of tar have problems dealing with very long path names. To find the version of tar being used type `tar --version`

---

After decompressing, the Mali Developer Tools are installed in:

```
ARM/Mali_Developer_Tools
```

By default, the OpenGL ES 2.0 Emulator is installed in:

```
ARM/Mali_Developer_Tools/OpenGL_ES_2_0_Emulator_m.n.o
```

### **OpenGL ES 2.0 Emulator content**

Figure 3-1 on page 3-4 shows the directory structure that is created at the path where you installed the emulator.

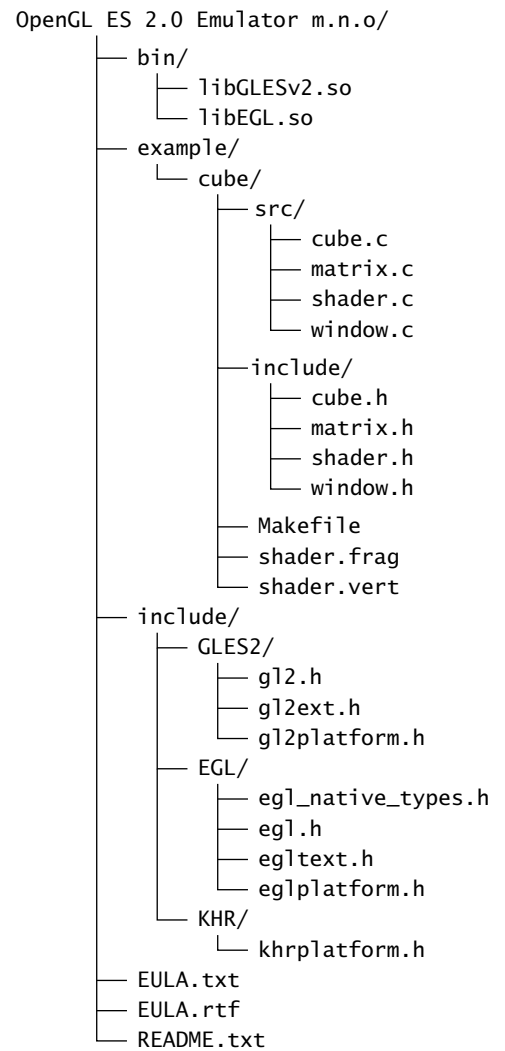
**Figure 3-1 OpenGL ES 2.0 Emulator directory structure**

Table 3-1 shows the OpenGL ES 2.0 Emulator directory file contents:

**Table 3-1 OpenGL ES 2.0 Emulator directory file contents**

Filename	Description
lib\libGLESv2.so	Import library for OpenGL ES 2.0 Emulator
lib\libEGL.so	Import library for EGL implementation
example\cube\src\cube.c	Main cube application source code
example\cube\src\matrix.c	Utility functions for dealing with matrices
example\cube\src\shader.c	Utility functions for dealing with shader files
example\cube\src>window.c	Utility functions for creating windows
example\cube\include\cube.h	Header file for the cube application
example\cube\include\matrix.h	Header file for matrix manipulation code
example\cube\include\shader.h	Header file for shader file manipulation code

**Table 3-1 OpenGL ES 2.0 Emulator directory file contents (continued)**

<b>Filename</b>	<b>Description</b>
example\cube\include>window.h	Header file for window code
example\cube\Makefile	Makefile for cube example
example\cube\shader.frag	Fragment shader for cube example
example\cube\shader.vert	Vertex shader for cube example
include\GLES\gl2.h	Khronos header file
include\GLES\gl2ext.h	Khronos header file
include\GLES\glplatform.h	Khronos header file
include\EGL\egl_native_types.h	Khronos header file
include\EGL\egl.h	Khronos header file
include\EGL\egltext.h	Khronos header file
include\EGL\eglplatform.h	Khronos header file
include\KHR\khrplatform.h	Khronos header file
EULA.txt	End User Licence Agreement as a text file
EULA.rtf	End User Licence Agreement as an rtf file
README.txt	README file

## 3.2 Configuring the OpenGL ES 2.0 Emulator on Linux

This section provides information to configure your system to use the OpenGL ES 2.0 Emulator when the installation is complete. It contains the following sections:

- *Using the OpenGL ES 2.0 Emulator*
- *OpenGL ES 2.0 Emulator integration.*

### 3.2.1 Using the OpenGL ES 2.0 Emulator

Your OpenGL ES 2.0 application must use the OpenGL ES 2.0 libraries:

1. To build an OpenGL ES 2.0 application on the OpenGL ES 2.0 Emulator, you must provide the path to `libGLESv2.so` and `libEGL.so` during link stage. Use the command:  
`-L path_to_emulator/bin`
2. Before you can run OpenGL ES 2.0 applications, the library search path must include the required OpenGL ES 2.0 Emulator libraries. To add the path of the Emulator libraries to the system environment variable `LD_LIBRARY_PATH`:

- The command to do this using the bash Linux shell is:

```
export LD_LIBRARY_PATH=<installation root directory for OpenGL ES 2.0 Emulator>/bin
```

- The command to do this using the tcsh Linux shell is:

```
setenv LD_LIBRARY_PATH <installation root directory for OpenGL ES 2.0 Emulator>/bin
```

For information on building the Cube example, see *Building the OpenGL ES 2.0 Cube example on Linux* on page 3-9.

### 3.2.2 OpenGL ES 2.0 Emulator integration

This section describes:

- *Libraries*
- *EGL configuration*
- *EGL context creation* on page 3-7
- *Shader language version* on page 3-8.

#### Libraries

The OpenGL ES 2.0 Emulator contains two libraries corresponding to the separate OpenGL ES 2.0 and EGL 1.3 APIs.

Table 3-2 shows the OpenGL ES 2.0 emulation library structure

**Table 3-2 OpenGL ES 2.0 Emulator library structure**

Filename	Description
<code>bin\libGLESv2.so</code>	Library for OpenGL ES 2.0 API
<code>bin\libEGL.so</code>	Library for EGL API

#### EGL configuration

The EGL library supplied with the OpenGL ES 2.0 Emulator supports OpenGL ES 2.0 only.

**Note**

Ensure that, in the OpenGL ES 2.0 application, the attribute list passed as a parameter to `eglChooseConfig` includes the attribute `EGL_RENDERABLE_TYPE` set to the value `EGL_OPENGL_ES2_BIT`.

Example 3-1 shows a coded section:

**Example 3-1**


---

```

EGLDisplay Display;

EGLint Attributes[] = {
    EGL_RENDERABLE_TYPE, EGL_OPENGL_ES2_BIT,
    EGL_RED_SIZE, 8,
    EGL_GREEN_SIZE, 8,
    EGL_BLUE_SIZE, 8,
    EGL_NONE
};
EGLConfig Configs[1];
EGLint NumConfigs;

...

eglChooseConfig(Display, Attributes, Configs, 1, &NumConfigs);

```

---

**EGL context creation**

The EGL library supplied with the OpenGL ES 2.0 Emulator only supports OpenGL ES 2.0 contexts.

Ensure that, in the OpenGL ES 2.0 application, the attribute list passed as a parameter to `eglCreateContext` includes the attribute `EGL_CONTEXT_CLIENT_VERSION` set to the value 2.

Example 3-2 shows a coded section:

**Example 3-2**


---

```

EGLDisplay Display;

EGLConfig Configs[1];

EGLint ContextAttributes[] = {
    EGL_CONTEXT_CLIENT_VERSION, 2,
    EGL_NONE
};

...

Context = eglCreateContext(Display, Configs[0], EGL_NO_CONTEXT,
                          ContextAttributes);

```

---

### Shader language version

The OpenGL ES 2.0 Emulator checks whether the graphics card has version 1.2 of the *OpenGL 2.0 Shader Language* (GLSL) available:

- If version 1.2 is available, this is selected by the pragma `#version 120` in the conversion of the ESSL shader to GLSL.
- If version 1.2 is not available, pragma `#version 110` selects GLSL version 1.1, that limits some features.

### 3.3 Building the OpenGL ES 2.0 Cube example on Linux

This section describes how to build the OpenGL ES 2.0 cube example on Linux.

The cube example code for using the OpenGL ES 2.0 Emulator is included in the directory:

```
<installation root directory for OpenGL ES 2.0 Emulator>/example/cube
```

To build and run this example:

1. Ensure that OpenGL ES 2.0 Emulator is installed.
2. Ensure that the system environment variable `LD_LIBRARY_PATH` is set to the path of the OpenGL ES 2.0 libraries. See *Using the OpenGL ES 2.0 Emulator* on page 3-6.
3. Navigate to the directory where the example is present:  

```
cd <installation root directory for OpenGL ES 2.0 Emulator>/example/cube
```
4. Build the cube application:  

```
make
```
5. To run the example application, type the following command:  

```
./cube
```

An additional window with a spinning, colored cube appears when the example application is running. Figure 3-2 shows an image of the example running.

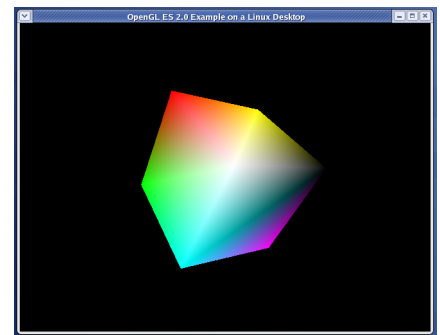


Figure 3-2 Cube image

To end the program, click anywhere on the window.

# Chapter 4

## Operational modes

This chapter provides information about the OpenGL ES 2.0 Emulator operational modes. It contains the following sections:

- *Normal and Instrumented operational modes* on page 4-2
- *Instrumentation parameters* on page 4-3
- *Enabling instrumentation on Windows* on page 4-4
- *Enabling instrumentation on Linux* on page 4-6
- *Enabling Debugging* on page 4-7
- *Building and running the example with instrumentation* on page 4-10.

## 4.1 Normal and Instrumented operational modes

This sections describes the operational modes of the OpenGL ES 2.0 Emulator, it contains the following sections:

- *Normal operation*
- *Instrumented operation.*

### 4.1.1 Normal operation

In normal operation, the OpenGL ES 2.0 Emulator does not dump any performance counters that can be viewed by the Performance Analysis Tool.

### 4.1.2 Instrumented operation

This section describes instrumented operation. It contains the following sections:

- *OpenGL ES 2.0 and EGL counters*
- *Mali GPU Hardware counters*
- *Instrumentation modes.*

#### OpenGL ES 2.0 and EGL counters

In instrumented operation, the OpenGL ES 2.0 Emulator can dump performance counters. These counters provide statistics about OpenGL ES 2.0 and EGL. You can view the saved data with the Mali GPU Performance Analysis Tool.

The counters that are dumped are:

- OpenGL ES 2.0 counters
- EGL counters.

#### Mali GPU Hardware counters

Because Mali GPU Hardware counters are specific to Mali hardware, they are not dumped by the OpenGL ES 2.0 Emulator. The OpenGL ES 2.0 Emulator uses desktop graphics cards that cannot provide information related to Mali hardware.

If a dump file is viewed using Performance Analysis Tool, the hardware counters are grayed out and disabled to highlight this. However, you can view all the OpenGL ES 2.0 and EGL specific counters.

ARM provides instrumented drivers for Mali Hardware that can dump the values of the hardware counters for viewing in the Performance Analysis Tool. If the hardware counters are available, the display of the hardware counters is automatically enabled in Performance Analysis Tool

#### Instrumentation modes

The OpenGL ES 2.0 Emulator offers the following modes of instrumentation:

##### Online mode

in online mode, the Performance Analysis Tool connects to the OpenGL ES 2.0 Emulator over a network using a TCP socket.

##### Offline mode

in offline mode, the Emulator writes counter data to a disk file which can be imported into the Performance Analysis Tool.

## 4.2 Instrumentation parameters

This section describes the parameters used to activate instrumentation. It contains the following sections:

- *Offline Instrumentation parameters*
- *Online Instrumentation parameters.*

Instrumentation is controlled by a number of parameters. Because the parameters are not created during installation, you must add them manually to enable Instrumentation output.

How you set these parameters depends on the OS you are using:

- On Windows, use the standard Windows regedit tool to create the parameters as registry keys. The keys must have the DWORD type and a value of 1, to enable the corresponding debugging operations.
- On Linux, add the parameters as environment variables, each with a value of 1, to enable the debugging operations.

### 4.2.1 Offline Instrumentation parameters

The parameters you must set to enable offline Instrumentation are:

MALI\_DUMP\_FILE

This is the name of the file that holds the performance data dumped by the OpenGL ES 2.0 Emulator. You can use the Performance Analysis Tool to view this file.

MALI\_DUMP\_FRAMEBUFFER

This flag enables dumping of the output framebuffer images.

MALI\_DUMP\_NAME

This the name you give to your dump data.

MALI\_DUMP\_PRINT

This flag enables printing the dump data onto the standard output stream.

MALI\_RECORD\_ALL

This flag enables dumping of performance data.

### 4.2.2 Online Instrumentation parameters

You must set the following parameters to enable online Instrumentation:

MALI\_DUMP\_NET

Enables or disables the exposing of performance counter data over a TCP connection.

MALI\_DUMP\_PORT

The TCP port number that the OpenGL ES 2.0 Emulator listens on for connections from the Performance Analysis Tool.

## 4.3 Enabling instrumentation on Windows

This sections describes:

- *Enabling offline instrumentation on Windows*
- *Enabling online instrumentation on Windows.*

### 4.3.1 Enabling offline instrumentation on Windows

Instrumentation is activated by the Windows registry settings. You must create these manually to enable Instrumented operation.

To activate offline Instrumentation on Windows:

1. Run `regedit` from a command prompt
2. Create the following key inside the key `HKEY_CURRENT_USER`:  
`Software\ARM\MaliSDK\OGLES2EMUL\Instrumentation`
3. Add the values listed in Table 4-1 under the key:

**Table 4-1 Windows registry keys for enabling offline instrumentation**

Name	Type	Data
MALI_DUMP_FILE	REG_SZ	MaliCount.ds2
MALI_DUMP_FRAMEBUFFER	REG_SZ	FALSE
MALI_DUMP_NAME	REG_SZ	OGLES20EMUL
MALI_DUMP_PRINT	REG_SZ	FALSE
MALI_RECORD_ALL	REG_SZ	TRUE

### 4.3.2 Enabling online instrumentation on Windows

To activate online Instrumentation on Windows:

1. Run `regedit` from a command prompt
2. Create the following key inside the key `HKEY_CURRENT_USER`:  
`Software\ARM\MaliSDK\OGLES2EMUL\Instrumentation`
3. Add the values listed in Table 4-2 under the key:

**Table 4-2 Windows registry keys for enabling online instrumentation**

Name	Type	Data
MALI_DUMP_NET	REG_SZ	TRUE
MALI_DUMP_PORT	REG_SZ	<port number>

Where <port number> is the number of a free TCP port, for example 3471.

4. Launch the application. There is no output from the application because it is waiting on the Performance Analysis Tool.
5. Use the Mali-GPU Performance Analysis Tool to connect to the application. Use the port number specified in `MALI_DUMP_PORT`.

See the *Mali Performance Analysis Tool User Guide* for more information.

## 4.4 Enabling instrumentation on Linux

This sections describes:

- *Enabling offline instrumentation on Linux*
- *Enabling online instrumentation on Linux.*

On Linux, instrumentation is controlled by environment variables, you must manually add these to enable Instrumented operation.

### 4.4.1 Enabling offline instrumentation on Linux

To enable offline instrumentation, set the environment variables listed in Table 4-3:

**Table 4-3 Linux environment variables for offline instrumentation**

Name	Value
MALI_DUMP_FILE	MaliCount.ds2
MALI_DUMP_FRAMEBUFFER	0
MALI_DUMP_NAME	OGLES20EMUL
MALI_DUMP_PRINT	0
MALI_RECORD_ALL	1

### 4.4.2 Enabling online instrumentation on Linux

To enable online instrumentation, set the environment variables listed in Table 4-4:

**Table 4-4 Linux environment variables for online instrumentation**

Name	Value
MALI_DUMP_NET	1
MALI_DUMP_PORT	<port number>

1. Launch the application. There is no output from the application because it is waiting on the Performance Analysis Tool.
2. Use the Mali-GPU Performance Analysis Tool to connect to the application. Use the port number specified in MALI\_DUMP\_PORT.

See the *Mali Performance Analysis Tool User Guide* for more information.

## 4.5 Enabling Debugging

This section describes debug settings. It contains the following sections:

- *Debugging settings*
- *Enabling Debugging on Windows* on page 4-8
- *Enabling Debugging on Linux* on page 4-8.

### 4.5.1 Debugging settings

Debugging is controlled by a number of parameters, These are not created during installation, so you must manually add these to enable debugging output.

How you set these parameters depends on the OS you are using. To enable debugging in the OpenGL ES 2.0 Emulator on Windows, add them as registry keys. On Linux, add them as environment variables.

The feedback provided by the OpenGL ES 2.0 Emulator depends on the OS it is running on.

The following parameters control debugging:

#### EGL\_INTERNAL\_REPORT

This reports errors returned by the underlying calls to the underlying windowing system. These errors are always returned through the normal EGL error reporting mechanism. The report to standard error might provide more information on the exact cause of the failure.

#### EGL\_OUT\_OF\_MEMORY\_REPORT

Memory allocation failures within the EGL Library are reported. These are always reported as EGL\_BAD\_ALLOC. The report to standard error indicates the size and purpose of the failed allocation.

#### EGL\_WARNING\_REPORT

The EGL Library outputs warnings about limitations of the library. This includes warnings about limitations and failure to set EGL\_RENDERABLE\_TYPE or EGL\_CONTEXT\_CLIENT\_VERSION.

#### GLES2\_INTERNAL\_REPORT

This reports errors returned by the underlying calls to the underlying windowing system. These errors are always returned through the normal EGL error reporting mechanism. The report to standard error might provide more information on the exact cause of the failure.

The Windows version of the EGL library uses the WGL windowing system, and the Linux version of the library uses GLX.

#### GLES2\_OUT\_OF\_MEMORY\_REPORT

Memory allocation failures within the OpenGL ES 2.0 Emulator are reported. These are always reported as EGL\_BAD\_ALLOC. The report to standard error indicates the size and purpose of the failed allocation.

#### GLES2\_WARNING\_REPORT

Any failure to run the Mali GPU Offline Shader Compiler is reported to standard error. See *Shader syntax checking by the Mali GPU Offline Shader Compiler* on page 2-8.

---

**Note**

---

The following key formats are deprecated but still supported:

- EGLInternalReport
  - EGLOutOfMemoryReport
  - EGLWarningReport
  - GLES2InternalReport
  - GLES2OutOfMemoryReport
  - GLES2WarningReport.
- 

## 4.5.2 Enabling Debugging on Windows

To enable debugging in the OpenGL ES 2.0 Emulator on Windows, you must add parameters as keys to the Windows registry. If a parameter is enabled, the OpenGL ES 2.0 Emulator writes debugging information to the standard error stream. Table 4-5 shows the keys.

You can use the standard Windows regedit tool. You must create the registry keys with the DWORD type, and a value of 1, to enable the corresponding debugging.

**Table 4-5 Windows registry key settings for debugging**

Registry settings
HKEY_CURRENT_USER\Software\ARM\MaliSDK\OGLES2EMUL\Debug\EGL_INTERNAL_REPORT
HKEY_CURRENT_USER\Software\ARM\MaliSDK\OGLES2EMUL\Debug\EGL_OUT_OF_MEMORY_REPORT
HKEY_CURRENT_USER\Software\ARM\MaliSDK\OGLES2EMUL\Debug\EGL_WARNING_REPORT
HKEY_CURRENT_USER\Software\ARM\MaliSDK\OGLES2EMUL\Debug\GLES2_INTERNAL_REPORT
HKEY_CURRENT_USER\Software\ARM\MaliSDK\OGLES2EMUL\Debug\GLES2_OUT_OF_MEMORY_REPORT
HKEY_CURRENT_USER\Software\ARM\MaliSDK\OGLES2EMUL\Debug\GLES2_WARNING_REPORT

---

**Note**

---

The following key formats are deprecated but still supported:

- EGLInternalReport
  - EGLOutOfMemoryReport
  - EGLWarningReport
  - GLES2InternalReport
  - GLES2OutOfMemoryReport
  - GLES2WarningReport.
- 

## 4.5.3 Enabling Debugging on Linux

To enable debugging in the OpenGL ES 2.0 Emulator on Linux, you must add parameters as environment variables. If a parameter is enabled, the OpenGL ES 2.0 Emulator writes debugging information to the standard error stream. Table 4-6 on page 4-9 shows the available parameters.

Set the environment variables to 1 to enable reporting.

**Table 4-6 Linux environment variables for debugging**

<b>Name</b>	<b>Value</b>
EGL_INTERNAL_REPORT	1
EGL_OUT_OF_MEMORY_REPORT	1
EGL_WARNING_REPORT	1
GLES2_INTERNAL_REPORT	1
GLES2_OUT_OF_MEMORY_REPORT	1
GLES2_WARNING_REPORT	1

## 4.6 Building and running the example with instrumentation

You can see the performance counters by building and running the Cube example with instrumentation activated.

1. To build the Cube example, follow the steps outlined in *Building the OpenGL ES 2.0 Cube example on Windows* on page 2-9 or *Building the OpenGL ES 2.0 Cube example on Linux* on page 3-9.
2. Set up the emulator with online or offline instrumentation. See *Enabling instrumentation on Windows* on page 4-4 or *Enabling instrumentation on Linux* on page 4-6.
3. Run the Cube example.
4. If you are using online instrumentation, you can see the performance counters immediately in the Performance Analysis Tool.

If you are using offline instrumentation, a MaliCount.ds2 file is created containing the counter information that you can load into the Performance Analysis Tool.

---

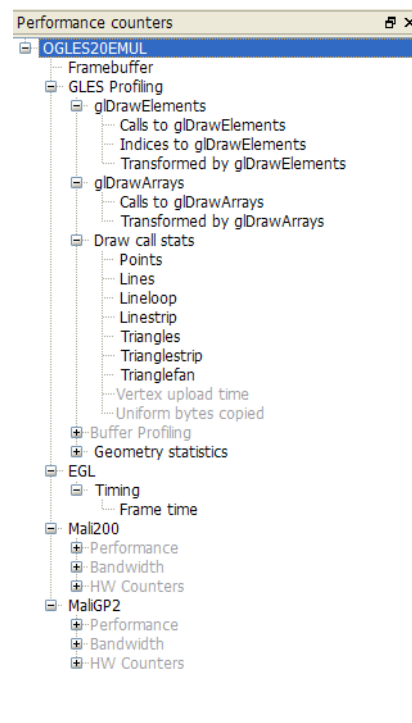
### Note

---

See the *Mali GPU Performance Analysis Tool User Guide* for information on how to use Performance Analysis Tool to analyze performance data.

---

Figure 4-1 shows a screen capture of the performance counters.



**Figure 4-1 Performance counters**

---

### Note

---

The Mali200 and MaliGP2 counters are disabled in the OpenGL ES 2.0 Emulator dump files. These counters are Mali hardware specific and can only be enabled and viewed when the application is run on Mali GPU hardware.

---

# Chapter 5

## Implementation Information

This chapter provides implementation information about OpenGL ES 2.0 and EGL APIs in the emulator. It contains the following sections:

- *OpenGL ES 2.0 Implementation information* on page 5-2
- *EGL implementation information on Windows* on page 5-6
- *EGL implementation information on Linux* on page 5-10.

## 5.1 OpenGL ES 2.0 Implementation information

The OpenGL ES 2.0 Emulator converts OpenGL ES 2.0 API calls to OpenGL 2.0 calls. These OpenGL 2.0 calls are handled by the Windows graphics drivers.

Because of the difference in specifications, OpenGL ES 2.0 parameters are not always compatible with OpenGL 2.0. The API call conversion checks OpenGL ES 2.0 parameters, and rejects invalid OpenGL ES 2.0 parameter values.

The OpenGL ES 2.0 Emulator depends on the functionality of the OpenGL 2.0 implementation provided by the graphics card drivers. In some cases, this dependency can lead to limitations in the OpenGL ES 2.0 implementation. This occurs when the behavior of the graphics card drivers differs from the OpenGL ES 2.0 specification.

This section describes:

- *General limitations*
- *NVIDIA GeForce 210 graphics card with driver version 190.45 for Windows* on page 5-3.

### 5.1.1 General limitations

This section describes:

- *Implementation-specific behavior*
- *glShaderBinary always fails*
- *Fixed-point data gives reduced performance* on page 5-3
- *Shader precision qualifiers are ignored* on page 5-3
- *glGetShaderPrecisionFormat values* on page 5-3
- *glGenerateMipmap performance* on page 5-3
- *Compressed texture formats* on page 5-3
- *OpenGL GLSL Shader compiler errors reported by the OpenGL ES 2.0 Emulator cannot be easily mapped onto the original source code* on page 5-3
- *Multiple threads and multiple contexts* on page 5-3.

#### Implementation-specific behavior

Where the OpenGL ES 2.0 specification permits implementation-specific behavior, the behavior is usually determined by the underlying driver. The behavior of the graphics card drivers can differ from the behavior of Mali drivers and hardware. This includes implementation-dependent limits, for example:

- texture sizes
- extensions
- mipmap level calculation
- precision of shaders
- framebuffers.

#### glShaderBinary always fails

Because of the incompatibility between binary formats for different graphics engines, the OpenGL ES 2.0 Emulator provides support for ESSL shader source code only and does not provide support for compiled Mali-200 or Mali-400 MP shader binaries. The call `glShaderBinary` has no functionality and always returns the error `GL_INVALID_ENUM` because no binary formats are supported.

### Fixed-point data gives reduced performance

OpenGL 2.0 does not provide support for fixed-point data, but this is required by the OpenGL ES 2.0 specification. The OpenGL ES 2.0 Emulator converts fixed-point data and passes it to OpenGL 2.0. For the OpenGL ES 2.0 Emulator, fixed-point data gives lower performance than floating-point data. This effect is stronger if you use a client-side vertex array rather than a vertex buffer object. The OpenGL ES 2.0 Emulator must convert a client-side vertex array on each draw call, because the client application might modify the data between draw calls.

### Shader precision qualifiers are ignored

The `lowp`, `mediump` and `highp` qualifiers in the *OpenGL ES 2.0 Shading Language* (ESSL) have no equivalents in the *OpenGL 2.0 Shading Language* (GLSL), and are removed. Because precision of shader variables is implementation dependent in OpenGL 2.0, shader variables might not have the minimum range or precision required by the ESSL specification.

### glGetShaderPrecisionFormat values

`glGetShaderPrecisionFormat` returns the same values as the Mali-200 driver, but the actual range and precision depends on the underlying OpenGL 2.0 driver. There is no equivalent query mechanism for the OpenGL 2.0 driver.

### glGenerateMipmap performance

`glGenerateMipmap` is slow, because it must work around a defect in the NVIDIA and ATI drivers.

### Compressed texture formats

`GL_OES_compressed_ETC1_RGB8_texture` is supported, but it is treated internally as RGB8. It is therefore possible to mix this format with uncompressed RGB8 texture data in ways that can cause either an error or an incomplete texture when used with Mali drivers on Mali GPU hardware.

### OpenGL GLSL Shader compiler errors reported by the OpenGL ES 2.0 Emulator cannot be easily mapped onto the original source code

Shader compiler error line numbers reported from the underlying OpenGL graphics driver might not match because of the translation of ESSL to GLSL. This defect is documented in the *OpenGL ES Emulator 2.0 Errata*.

### Multiple threads and multiple contexts

Multiple threads and multiple contexts are not supported and might lead to unpredictable behavior.

## 5.1.2 NVIDIA GeForce 210 graphics card with driver version 190.45 for Windows

These are defects in the driver and that might change between driver releases:

- *Driver settings* on page 5-4
- *Framebuffer object with depth and stencil buffer not supported* on page 5-4
- *Framebuffer object with stencil buffer and no depth buffer not supported* on page 5-4
- *Image attachment* on page 5-4
- *Clipping to the viewport* on page 5-4
- *Link failures with attribute aliasing* on page 5-4
- *Link can succeed with undefined varying variables* on page 5-4

- *Driver adjustment of texture filtering, anti-aliasing and anisotropic filtering* on page 5-5.

### Driver settings

For the most conformant results, some settings must be changed in the NVIDIA control panel. To do this, right click on Desktop, select **NVIDIA control panel**.

1. Under Adjust image settings with preview, select **Use the advanced 3D settings**, then select **Take me there**.
2. Set **Anisotropic filtering** to **Application-controlled**.
3. Set **Antialiasing - Gamma correction** to **Off**.
4. Set **Antialiasing - Mode** to **Application-controlled**.
5. Set **Antialiasing - Transparency** to **Off**.

### Framebuffer object with depth and stencil buffer not supported

ARM does not support using both a depth and a stencil buffer in a framebuffer object.

#### ———— Caution ————

The emulator does not detect the use of a framebuffer object with depth and stencil buffer. It does not give an error message. If you use a framebuffer object with depth and stencil buffer, this might result in unpredictable behavior.

### Framebuffer object with stencil buffer and no depth buffer not supported

If you use a stencil buffer, but no depth buffer, in a framebuffer object, the OpenGL ES 2.0 Emulator reports `GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT` even though the attachment is complete.

### Image attachment

If you attach a depth-renderable image to `GL_COLOR_ATTACHMENT0` or a color-renderable image to `GL_DEPTH_ATTACHMENT`, an error is generated.

### Clipping to the viewport

Where the line or point size is greater than 1, the generated fragments are clipped to the viewport.

### Link failures with attribute aliasing

Attribute aliasing with `glBindAttribLocation` causes link failures, even when there is no path through the vertex shader that references both attributes.

### Link can succeed with undefined varying variables

Linking can succeed even if the fragment shader uses a varying variable that is not defined by the vertex shader. `glLinkProgram()` returns `GL_LINK_STATUS` as `GL_TRUE`. This can cause unpredictable behavior of the application.

**Driver adjustment of texture filtering, anti-aliasing and anisotropic filtering**

NVIDIA drivers make adjustments to texture filtering, anti-aliasing, and anisotropic filtering in an attempt to improve game play experience. Some of these adjustments can be disabled in the NVIDIA Control Panel, see *Driver settings* on page 5-4. However, in general, you cannot rely on the texture *Level Of Detail* (LOD) calculations or the choice between minification and magnification to be accurate.

## 5.2 EGL implementation information on Windows

This section describes:

- *EGL native types*
- *Display initialization*
- *Default display example*
- *Window example*
- *EGL configurations* on page 5-7
- *EGL contexts* on page 5-7
- *Creation of window surface* on page 5-7
- *Creation of pixmap surfaces* on page 5-7
- *Creation of Pbuffer surfaces* on page 5-7
- *Synchronization of pixmap surfaces* on page 5-8
- *EGL limitations* on page 5-8.

### 5.2.1 EGL native types

The EGL implementation defines the platform specific native types as follows:

```
typedef HDC NativeDisplayType;
```

```
typedef HBITMAP NativePixmapType;
```

```
typedef HWND NativeWindowType;
```

These are defined in the file `include\EGL\egl_native_types.h`

### 5.2.2 Display initialization

In an OpenGL ES 2.0 application, use the `eglGetDisplay` call to create a window that displays the rendered output from the Open GL ES 2.0 Emulator. You must pass to this function either the:

- value `EGL_DEFAULT_DISPLAY`
- *Handle of the Device Context (HDC).*

#### Default display example

Example 5-1 shows a code example that uses the default display:

#### Example 5-1

---

```
EGLDisplay sEGLDisplay;

// EGL init.
sEGLDisplay = eglGetDisplay((EGLNativeDisplayType) EGL_DEFAULT_DISPLAY);
eglInitialize(sEGLDisplay, NULL, NULL);
```

---

#### Window example

Example 5-2 on page 5-7 shows a code example that uses a window display:

**Example 5-2**


---

```

EGLDisplay sEGLDisplay;

...

// Create window
sWindow = CreateWindowEx(...

// EGL init.
sEGLDisplay = eglGetDisplay(GetDC(sWindow));
eglInitialize(sEGLDisplay, NULL, NULL);

```

---

**5.2.3 EGL configurations**

The EGL implementation supports OpenGL ES 2.0 only. It does not support OpenGL ES 1.1 or OpenVG configurations.

To get a valid configuration from `eglChooseConfig()`, you must set the `EGL_RENDERABLE_TYPE` in the attributes list to the value `EGL_OPENGL_ES2_BIT`. If you use values of `EGL_OPENGL_ES_BIT` or `EGL_OPENVG_BIT`, no configurations are returned. If you do not include a value for the `EGL_RENDERABLE_TYPE` attribute, `eglChooseConfig()` uses the default value. The EGL 1.3 default value for this attribute is `EGL_OPENGL_ES_BIT`. This means that no configurations are returned.

**5.2.4 EGL contexts**

The EGL implementation supports OpenGL ES 2.0 only. It does not support OpenGL ES 1.1 or OpenVG contexts.

The EGL 1.3 specification defines the default for attribute `EGL_CONTEXT_CLIENT_VERSION` to be the value 1. This implies EGL 1.3 is requesting a configuration for OpenGL ES 1.x support.

If there is no value for `EGL_CONTEXT_CLIENT_VERSION` in the attribute list, the default value is used when the list is passed to `eglCreateContext`. A default value of 1 results in context creation failing because OpenGL ES 1.x contexts are not supported.

To obtain a valid context You must set `EGL_CONTEXT_CLIENT_VERSION` to the value of 2 in the attributes list. Any other values result in context creation failing.

**5.2.5 Creation of window surface**

For an example of the code to create a window surface, see file `example\cube\main.c`.

**5.2.6 Creation of pixmap surfaces**

To access data bits of a Windows bitmap in the EGL API, you must pass the bitmap to EGL as a native pixmap. You must create this pixmap with the Windows API call `CreateDIBSection()`. For an example of the code to create a pixmap surface, see file `example\pixmap\triangle.c`.

**5.2.7 Creation of Pbuffer surfaces**

A Pbuffer has no associated native structure, and is created through the specification of attributes to `eglCreatePbufferSurface`. No platform specific code is required.

## 5.2.8 Synchronization of pixmap surfaces

Pixmap surfaces are supported through the use of graphics driver Pbuffers. You must use the appropriate EGL synchronization calls to get OpenGL ES 2.0 to render on to the native pixmap. This corresponds to the expected use of these calls in the EGL 1.3 specification.

The call `eglWaitNative(EGL_CORE_NATIVE_ENGINE)` copies bitmap data from the native bitmap to the graphics driver Pbuffer before the OpenGL ES 2.0 API calls are made to render to the Pbuffer. The calls `eglWaitClient()`, `eglWaitGL()` and `glFinish()` copy data back from the graphics driver Pbuffer to the native pixmap after OpenGL ES 2.0 renders to the Pbuffer. Example code for this synchronization is given in the file `example\pixmap\triangle.c`.

---

### Note

---

You must not select a native bitmap into a device context, else the `eglWaitNative()` call fails.

---

## 5.2.9 EGL limitations

The EGL library has sufficient functionality for the OpenGL ES 2.0 Emulator to pass Khronos OpenGL ES 2.0 conformance tests and to provide a platform for OpenGL ES 2.0 applications to be run on a Windows XP PC. The EGL library is a limited implementation of the EGL 1.3 specification. In particular, because the OpenGL ES 2.0 Emulator only provides support for the OpenGL ES 2.0 API, the EGL library does not provide support for graphics contexts and surfaces for use with OpenVG or OpenGL ES 1.1. This section provides additional information about these limitations:

- *Supports OpenGL ES 2.0 only*
- *Multiple threads and multiple contexts*
- *Window pixel format*
- *Limited bitmap support on page 5-9*
- *Limited results from surface queries on page 5-9*
- *No support for swap intervals on page 5-9*
- *Changing display modes on page 5-9*
- *Use of displays following `eglTerminate` on page 5-9*
- *`EGL_MATCH_NATIVE_PIXMAP` attribute not supported on page 5-9*
- *Resizing a native window on page 5-9*
- *Pbuffer lost events are not checked on page 5-9*
- *`EglChooseConfig` always sets `WGL_DOUBLE_BUFFER_ARB` true on page 5-9.*

### Supports OpenGL ES 2.0 only

The EGL library does not support graphics contexts and surfaces for use with OpenVG or OpenGL ES 1.1. No configurations are returned from `eglChooseConfig` for values of `EGL_RENDERABLE_TYPE` other than `EGL_OPENGL_ES2_BIT`. This means that the default value does not return a configuration.

Context creation fails unless `EGL_CONTEXT_CLIENT_VERSION` is set to 2.

### Multiple threads and multiple contexts

Multiple threads and contexts are not supported and might lead to unpredictable behavior.

### Window pixel format

You must set pixel format only through `eglCreateWindowSurface()`.

### Limited bitmap support

Bitmap rendering only works correctly for uncompressed, bottom-up, 32-bit RGB bitmaps. An example of creating such a bitmap is provided in `example\pixmap\triangle.c`.

### Limited results from surface queries

All parameters to `eglQuerySurface` are implemented, but those specific to OpenVG, and those that depend on the physical properties of the display, for example `EGL_HORIZONTAL_RESOLUTION`, return arbitrary values or `EGL_UNKNOWN`.

### No support for swap intervals

The `eglSwapInterval` function has no effect and always succeeds. The swap interval depends on the OpenGL 2.0 driver.

### Changing display modes

Changing display modes is not supported. For example, this can cause a Pbuffer lost event.

### Pbuffer lost events are not checked

A change of display mode might result in loss of Pbuffer memory. This event is not checked for. This defect is documented in the OpenGL ES 2.0 Emulator Errata.

### Use of displays following eglTerminate

Displays are destroyed in `eglTerminate`. Later calls treat the display as invalid.

### EGL\_MATCH\_NATIVE\_PIXMAP attribute not supported

The attribute `EGL_MATCH_NATIVE_PIXMAP` is not supported by `eglChooseConfig`. This defect is documented in the *OpenGL ES 2.0 Emulator Errata*.

### Resizing a native window

Resizing a native window does not update the surface attributes. This defect is documented in the OpenGL ES 2.0 Emulator Errata.

### EglChooseConfig always sets WGL\_DOUBLE\_BUFFER\_ARB true

The EGL attribute list is translated to an attribute list for WGL. This WGL attribute list always has `WGL_DOUBLE_BUFFER_ARB` set to true. This means that some available matching WGL configurations might not be returned. This defect is documented in the *OpenGL ES 2.0 Emulator Errata*.

## 5.3 EGL implementation information on Linux

The EGL implementation intends to supply sufficient functionality for the OpenGL ES 2.0 Emulator to pass Khronos OpenGL ES 2.0 conformance tests and to provide a platform for OpenGL ES 2.0 applications to be run on a Linux PC. The EGL library is a limited implementation of the EGL 1.3 specification. This section provides additional information about these limitations:

- *Unimplemented functions*
- *Resizing a native window*
- *eglChooseConfig always selects configurations that use the back buffer*
- *Some EGLConfig attributes are not supported*
- *EGLConfigs not sorted* on page 5-11
- *Attributes for windows not supported* on page 5-11
- *Some Pbuffer attributes are not supported* on page 5-11
- *Attributes for pixmaps not supported* on page 5-11
- *Incorrect error code returned instead of EGL\_BAD\_MATCH* on page 5-11
- *Limited results from surface queries* on page 5-11
- *eglMakeCurrent succeeds with incompatible surface and contents* on page 5-12.

### 5.3.1 Unimplemented functions

There are 34 functions in the EGL 1.3 specification, the following six functions are not implemented:

- `eglCreatePbufferFromClientBuffer`
- `eglSurfaceAttrib`
- `eglBindTexImage`
- `eglReleaseTexImage`
- `eglSwapInterval`
- `eglCopyBuffers`

### 5.3.2 Resizing a native window

Resizing a native window does not update the surface attributes. This defect is documented in the *OpenGL ES 2.0 Emulator Errata*.

### 5.3.3 eglChooseConfig always selects configurations that use the back buffer

The EGL specification enables you to specify whether to use the back buffer or not in the attribute list passed to `eglCreateWindowSurface`. The GLX function for window surface creation does not permit this. The GLX function for choosing configurations lets you specify whether you want to use a back buffer or not. The EGL implementation uses this function to select only those configurations that enable use of the back buffer. As a side effect of this, applications cannot disable use of the back buffer. This defect is documented in the *OpenGL ES 2.0 Emulator Errata*.

### 5.3.4 Some EGLConfig attributes are not supported

The following EGLConfig attributes are not supported:

- `EGL_LUMINANCE_SIZE`
- `EGL_ALPHA_MASK_SIZE`
- `EGL_BIND_TO_TEXTURE_RGB`
- `EGL_BIND_TO_TEXTURE_RGBA`

- EGL\_COLOR\_BUFFER\_TYPE
- EGL\_MAX\_SWAP\_INTERVAL
- EGL\_MATCH\_NATIVE\_PIXMAP

---

**Note**

---

- `eglChooseConfig` returns an error if any of these attributes is specified in the attribute list
  - `eglGetConfigAttrib` returns an error if any of these attributes is queried.
- 

### 5.3.5 EGLConfigs not sorted

The list of configurations returned by `eglChooseConfig` is not sorted. This is because EGL and GLX have different sorting criteria.

### 5.3.6 Attributes for windows not supported

Attributes for windows are not supported. The attribute list passed to `eglCreateWindowSurface` must be NULL or empty.

### 5.3.7 Some Pbuffer attributes are not supported

`eglCreatePbufferSurface` returns an error if any of the following attributes are specified in the attribute list:

- EGL\_TEXTURE\_FORMAT
- EGL\_TEXTURE\_TARGET
- EGL\_MIPMAP\_TEXTURE
- EGL\_VG\_COLORSPACE
- EGL\_VG\_ALPHA\_FORMAT

### 5.3.8 Attributes for pixmaps not supported

Attributes for pixmaps are not supported. The attribute list passed to `eglCreatePixmapSurface` must be NULL or empty.

### 5.3.9 Incorrect error code returned instead of EGL\_BAD\_MATCH

Sometimes, instead of `EGL_BAD_MATCH`, EGL returns an incorrect error code. This happens because GLX does not have an error code corresponding to `EGL_BAD_MATCH` and EGL is not always able to detect the real cause of the error.

### 5.3.10 Limited results from surface queries

`eglQuerySurface` returns an error if any of the following attributes are queried:

- EGL\_VG\_ALPHA\_FORMAT
- EGL\_VG\_COLORSPACE
- EGL\_HORIZONTAL\_RESOLUTION
- EGL\_MIPMAP\_TEXTURE
- EGL\_MIPMAP\_LEVEL
- EGL\_PIXEL\_ASPECT\_RATIO
- EGL\_RENDER\_BUFFER
- EGL\_TEXTURE\_FORMAT
- EGL\_TEXTURE\_TARGET

- EGL\_VERTICAL\_RESOLUTION

### 5.3.11 **eglMakeCurrent succeeds with incompatible surface and contents**

On some platforms, `eglMakeCurrent` succeeds with incompatible surface and context. This might not happen on other platforms.

# Appendix A

## **Adding registry settings in Windows**

This appendix describes how to enable offline instrumentation on Windows. It contains the following section:

- *Enabling instrumentation* on page A-2

## A.1 Enabling instrumentation

Instrumentation is turned on by the Windows registry settings. These settings are not created during installation, so you must create them manually.

———— **Note** ————

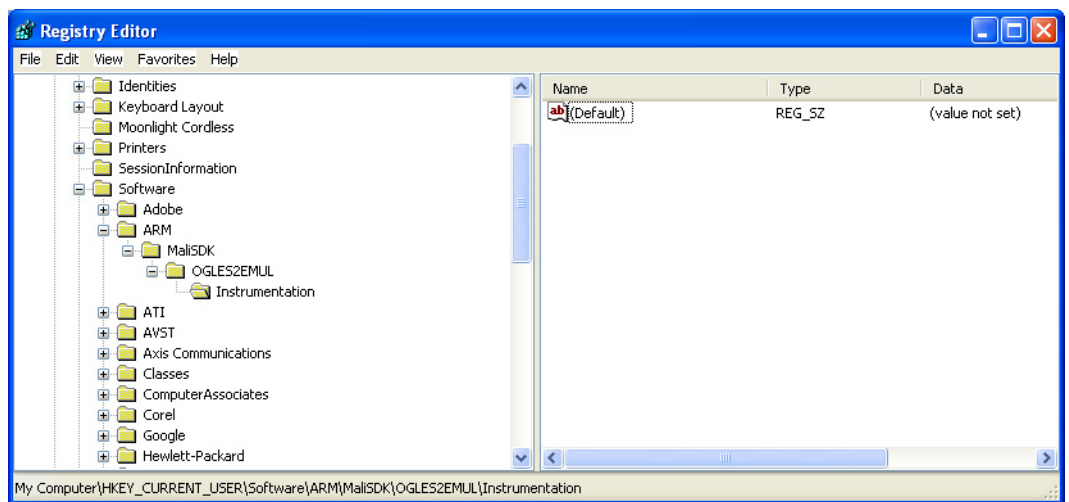
The LIBs and DLLs are described in *DLLs and libraries* on page 2-6.

To activate Instrumentation:

1. Run regedit from a command prompt
2. Create the following key inside the key HKEY\_CURRENT\_USER:

Software\ARM\MaliSDK\OGLES2EMUL\Instrumentation

Figure A-1 shows the key created.



**Figure A-1** Registry editor key creation

3. Under this entry, add the values given in Table A-1:

**Table A-1** Registry file values

Name	Type	Data
MALI_DUMP_FILE	REG_SZ	MaliCount.ds2
MALI_DUMP_FRAMEBUFFER	REG_SZ	FALSE
MALI_DUMP_NAME	REG_SZ	OGLES20EMUL
MALI_DUMP_PRINT	REG_SZ	FALSE
MALI_RECORD_ALL	REG_SZ	TRUE

To add these values:

- a. Right click to display the context menu and Select **New** → **String Value**

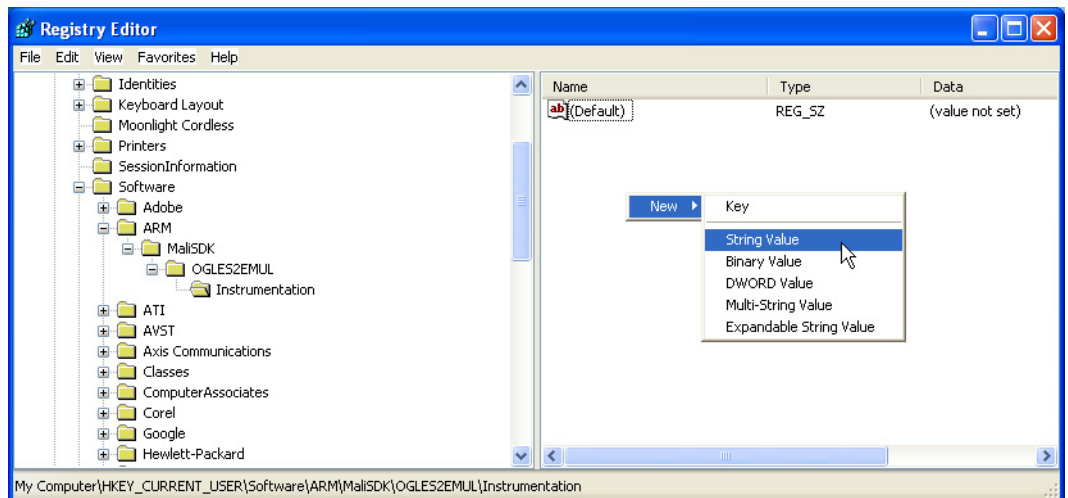


Figure A-2 Creating a String Value

- b. Type in the **Name** as shown in Figure A-3.

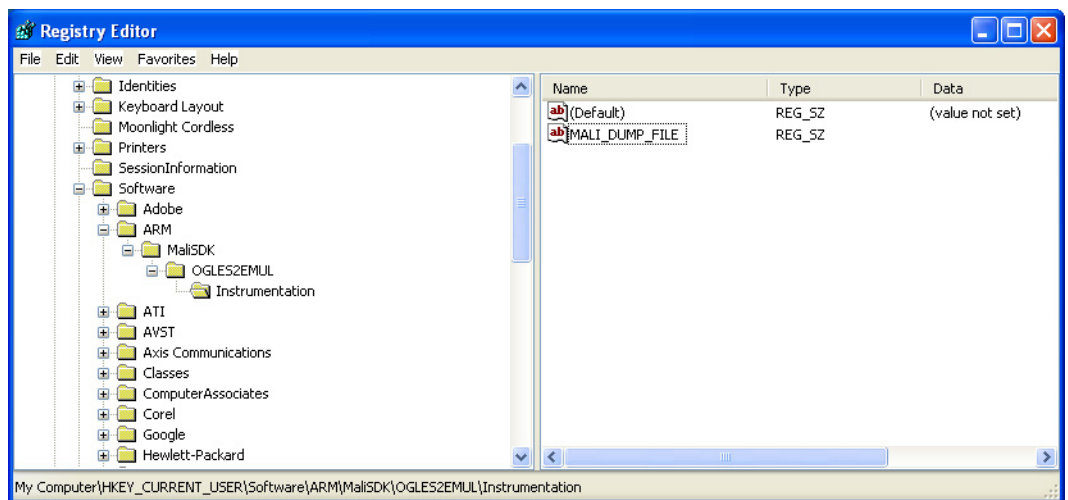


Figure A-3 Created value

- c. Double click the String Value to obtain the Edit String dialog box as shown in Figure A-4. Enter the required value data and click **OK**.

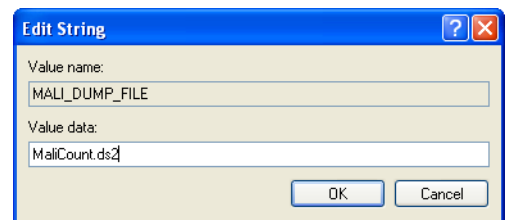


Figure A-4 Edit String dialog box

- d. Repeat the procedure until you have all six values as shown in Figure A-5 on page A-4.

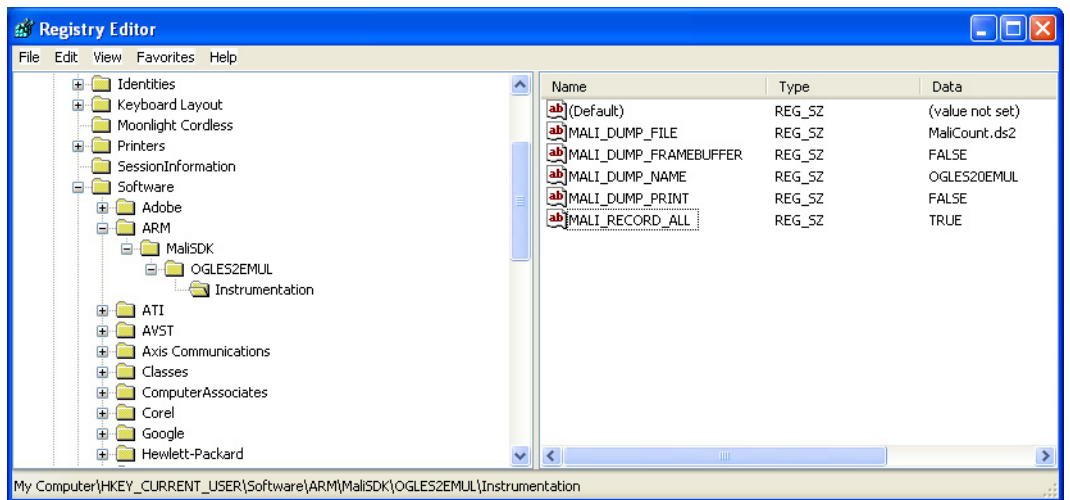


Figure A-5 Registry file values

# Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

<b>API</b>	<i>See</i> Application Programming Interface.
<b>Application Programming Interface (API)</b>	A specification for a set of procedures, functions, data structures, and constants that are used to interface two or more software components together. For example, an API between an operating system and the application programs that use it might specify exactly how to read data from a file.
<b>Attribute aliasing</b>	The processes whereby applications are permitted to bind more than one vertex shader attribute name to the same generic vertex attribute index.
<b>Debugger</b>	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
<b>Display subsystem</b>	The display that a final image is viewed on and the associated software that controls the operation of the display.
<b>Ericsson Texture Compression (ETC)</b>	A 4 <i>bit-per-pixel</i> (bpp) texture compression algorithm.
<b>ESSL</b>	<i>See</i> OpenGL ES Shading Language.
<b>ETC</b>	<i>See</i> Ericsson Texture Compression.
<b>Fragment shader</b>	A program running on the pixel processor that calculates the color and other characteristics of each fragment.
<b>Framebuffer</b>	A memory buffer containing a complete frame of graphical data.
<b>GPU</b>	<i>See</i> Graphics Processor Unit.

<b>Graphics application</b>	A custom program that runs on the platform CPU and uses the Mali GPU to display graphics.
<b>Graphics driver</b>	A software library implementing OpenGL ES or OpenVG, using graphics accelerator hardware. See also OpenGL ES driver.
<b>Graphics Processor Unit (GPU)</b>	A hardware accelerator for graphics systems using OpenGL ES and OpenVG. The hardware accelerator comprises of an optional geometry processor and a pixel processor together with memory management. Mali programmable GPUs, such as the Mali-200 and Mali-400 MP GPUs, consist of a geometry processor and at least one pixel processor. Mali fixed-function GPUs, such as the Mali-55 GPU consist of a pixel processor only.
<b>Instrumented drivers</b>	Alternative graphics drivers that are used with the Mali GPU. The Instrumented drivers include additional functionality such as error logging and recording performance data files for use by the Performance Analysis Tool.  <i>See also</i> Performance Analysis Tool.
<b>Mali</b>	A name given to graphics software and hardware products from ARM that aid 2D and 3D acceleration.
<b>Mali Developer Tools</b>	A set of development programs that enables software developers to create graphic applications.
<b>Offline Shader Compiler</b>	A command line tool that translates vertex shaders and fragment shaders written in the ESSL into binary vertex shaders and binary fragment shaders that you can link and run on the GPU. In the context of the emulator, the Offline Shader Compiler is used to check validity of the shader programs.
<b>OpenGL ES driver</b>	On graphics systems that use the OpenGL ES API, the OpenGL ES driver is a specialized driver that controls the graphics hardware.
<b>OpenGL ES Shading Language (ESSL)</b>	A programming language used to create custom shader programs that can be used within a programmable pipeline, on graphics hardware. You can also use pre-defined library shaders, written in ESSL.
<b>Performance Analysis Tool</b>	A fully-customizable GUI tool that displays and analyzes performance data files produced by the Instrumented drivers, together with framebuffer information.  <i>See also</i> Instrumented drivers, Performance data file.
<b>Pixel</b>	A pixel is a discrete element that forms part of an image on a display. The word pixel is derived from the term Picture Element.
<b>Performance counter</b>	A register in the processor that is incremented when a certain state is detected in the processor. Performance counter data is represented as performance variables that are displayed in the Performance Analysis Tool.
<b>Performance data file</b>	Files that contain a description of the performance counters, together with the performance counter data in the form of a series of values and images. Performance data files are saved in .ds2 format and can be loaded directly into the Performance Analysis Tool.

**Performance variable**

Data produced by the instrumented OpenGL ES 2.0 Emulator, that can be displayed and analyzed as statistical information in the Performance Analysis Tool.

**Red Green Blue Alpha (RGBA)**

An implementation of the RGB color model, that includes an Alpha value to indicate opacity. If a fragment has an Alpha value of 0%, it is fully transparent, making it invisible. An Alpha value of 100% results in a fully opaque pixel.

**Shader**

A program, usually an application program, running on the GPU, that calculates some aspect of the graphical output. See fragment shader and vertex shader.

**Shader Library**

A set of shader examples, tutorials, and other information, designed to assist with developing shader programs for the Mali GPU. The Shader Library is a component of the Mali GPU Developer Tools.

**Vertex shader**

A program running on the geometry processor, that calculates the position and other characteristics, such as color and texture coordinates, for each vertex.