

OpenGL ES Emulator

Version: 1.3.0

User Guide



OpenGL ES Emulator

User Guide

Copyright © 2009-2011 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.
22 July 2011	D	Non-Confidential	Updated for v1.3.0 release. Added OpenGL ES 1.1 and OpenGL ES 2.0 interoperability and added information on MESA.

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 Emulator User Guide

	Preface	
	About this book	v
	Feedback	vii
Chapter 1	Introduction	
	1.1 About the OpenGL ES Emulator	1-2
Chapter 2	Installation and Configuration on Windows	
	2.1 Installing the OpenGL ES Emulator on Windows	2-2
	2.2 Configuring the OpenGL ES Emulator on Windows	2-5
	2.3 Building the example applications on Windows	2-8
Chapter 3	Installation and Configuration on Linux	
	3.1 Installing the OpenGL ES emulator on Linux	3-2
	3.2 Configuring the OpenGL ES emulator on Linux	3-5
	3.3 Building the example applications on Linux	3-8
Chapter 4	Implementation Information	
	4.1 OpenGL ES Implementation information	4-2
	4.2 EGL implementation information on Windows	4-6
	4.3 EGL implementation information on Linux	4-10
	4.4 Using the Mesa software emulation of OpenGL	4-13

Preface

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

- *About this book on page v*
- *Feedback on page vii.*

About this book

This is the *OpenGL ES Emulator User Guide*. It provides guidelines for using the OpenGL ES 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 1.1 or 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 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 Implementation Information

Read this for information about the implementation of the OpenGL ES 2.0, OpenGL ES 1.1, and EGL APIs in the emulator.

Glossary

The *ARM Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

The *ARM Glossary* is available on the ARM Infocenter at, <http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html>.

Typographical Conventions

The typographical conventions are:

<i>italic</i>	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	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.

monospace bold	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>, <Opcod2>

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)
- *Mali GPU Mali Binary Asset Exporter User Guide* (ARM DUI 0507)
- *Mali GPU Shader Library User Guide* (ARM DUI 0510)
- *Mali GPU Application Optimization Guide* (ARM DUI 555)
- *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, contact your supplier 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. Give:

- the title
- the number, ARM DUI 0511D
- 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 Emulator, and describes how to start using the tool in your particular workflow.

It contains the following section:

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

1.1 About the OpenGL ES Emulator

The OpenGL ES Emulator is a library that maps OpenGL ES 1.1 API or 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 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 Emulator and Mali Development Studio are unable to perform syntax checking on the application shaders.

See *OpenGL ES Emulator integration on page 2-5* 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 Emulator on Microsoft Windows. It contains the following sections:

- *Installing the OpenGL ES Emulator on Windows* on page 2-2
- *Configuring the OpenGL ES Emulator on Windows* on page 2-5
- *Building the example applications on Windows* on page 2-8.

2.1 Installing the OpenGL ES 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 Emulator has been tested successfully on a 32-bit computer.

2.1.1 Supported Hardware and Software

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

- Windows XP Professional, version 2002, service pack 3

Note

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

- 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 MESA OpenGL software.

Note

The minimum appropriate extensions for MESA are:

- WGL_ARB_extensions_string
 - WGL_ARB_pixel_format
 - WGL_ARB_pbuffer
 - EXT_framebuffer_object.
-

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(TN) 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 Emulator requires approximately 5MB of disk space.

2.1.3 Installation procedure

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

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

Installing the OpenGL ES Emulator on Microsoft Windows

To install the OpenGL ES 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_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_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 Emulator is installed in:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES Emulator *vm.n.o*\

OpenGL ES Emulator content

The download package contains the OpenGL ES Emulator binaries for Windows and simple example applications, for OpenGL ES 2.0 and OpenGL ES 1.1, that runs on the OpenGL ES Emulator.

For more information see the *OpenGL ES 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 Emulator *vm.n.o*\

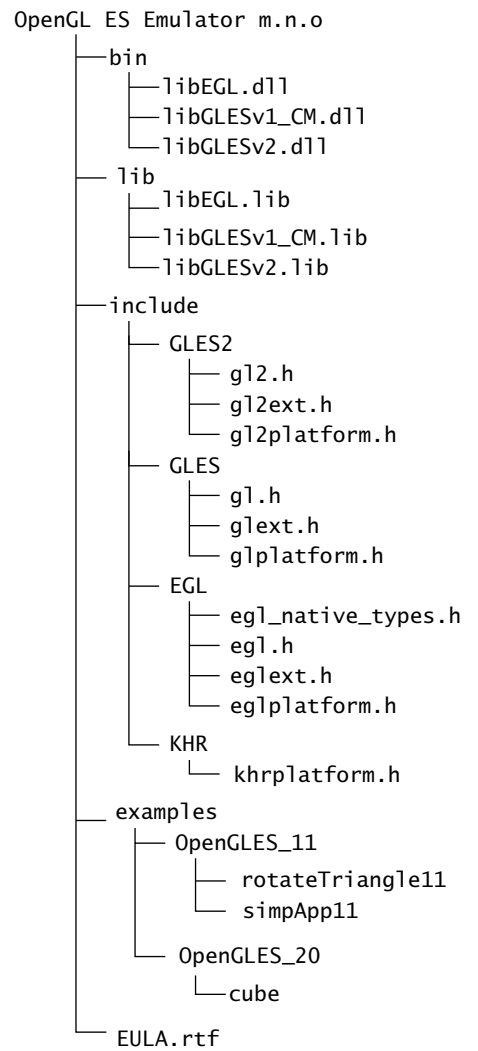


Figure 2-1 Emulator directory structure

2.2 Configuring the OpenGL ES Emulator on Windows

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

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

2.2.1 Using the OpenGL ES Emulator

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

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

To run any OpenGL ES 1.1 application on the OpenGL ES Emulator, you must link it to the OpenGL ES 1.1 libraries:

- The static libraries, `libEGL.lib` and `libGLESv1_CM.lib`, are linked while building an Open GL ES 1.1 application.
- The dynamically linked libraries, `libEGL.dll` and `libGLESv1_CM.dll`, must be provided at run-time.

2.2.2 OpenGL ES Emulator integration

This section describes:

- *OpenGL ES Emulator DLLs and libraries*
- *EGL configuration on page 2-6*
- *EGL context creation on page 2-6*
- *Shader syntax checking by the Mali GPU Offline Shader Compiler on page 2-7*
- *Limitations based on the shader language version on page 2-7.*

OpenGL ES Emulator DLLs and libraries

The OpenGL ES Emulator Library consists of DLLs, corresponding to the OpenGL ES 2.0, OpenGL ES 1.1, and EGL 1.3 APIs. For each of these DLLs, there is a corresponding import library for an OpenGL ES 2.0 or OpenGL ES 1.1 application to statically link against.

OpenGL ES 2.0 and OpenGL ES 1.1 applications must include both of their import libraries in builds to link against the OpenGL ES 2.0 and EGL 1.3 APIs or OpenGL ES 1.1 and EGL 1.3 APIs. The DLLs use the `__stdcall` calling convention.

Table 2-1 shows the files for OpenGL ES 1.1 and Open GL ES 2.0 emulation:

Table 2-1 OpenGL ES Emulator library structure

Filename	Description
<code>bin\libGLESv2.dll</code>	DLL for OpenGL ES 2.0 emulation
<code>bin\libEGL.dll</code>	DLL for EGL implementation
<code>lib\libGLESv2.lib</code>	Import library for <code>libGLESv2.dll</code>

Table 2-1 OpenGL ES Emulator library structure (continued)

Filename	Description
lib\libEGL.lib	Import library for libEGL.dll
bin\libGLESv1_CM.dll	DLL for OpenGL ES 1.1 emulation
lib\libGLESv1_CM.lib	Import library for libGLESv1_CM.dll

For convenience, the OpenGL ES Emulator also includes the files required for Open GL ES 1.1. If you are building OpenGL ES 1.1 applications, you must include libGLESv1_CM.lib instead of libGLESv2.lib.

EGL configuration

The EGL library supplied with the OpenGL ES Emulator supports OpenGL ES 1.1 and OpenGL ES 2.0. Ensure that, in the OpenGL ES 2.0 application, the attribute list passed as a parameter to `eglChooseConfig()` includes the attribute `EGL_RENDERABLE_TYPE` with the value `EGL_OPENGL_ES2_BIT` for OpenGL 2.0 applications.

Note

- Use `EGL_OPENGL_ES2_BIT|EGL_OPENGL_ES2_BIT` to request both OpenGL ES 1.1 and OpenGL ES 2.0 configs.
- The actual context used is specified in `eglCreateContext()` described in Example 2-2 on page 2-7.

Example 2-1 shows a coded section.

Example 2-1 EGL context configuration

```

EGLDisplay Display;

EGLint Attributes[] = {
    EGL_RENDERABLE_TYPE,
    EGL_OPENGL_ES2_BIT, // use EGL_OPENGL_ES2_BIT|EGL_OPENGL_ES2_BIT
                        // to return both 1.1 and 2.0 configs
    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

Example 2-2 on page 2-7 shows a coded section. Set the second element of the `ContextAttributes` array to 2 to select an OpenGL ES 2.0 context.

Note

- The OpenGL ES Emulator also works with OpenGL ES 1.1 contexts.
 - Set the second element of the ContextAttributes array to 1 to select an OpenGL ES 1.1 context
-

Example 2-2 EGL context creation

```

EGLDisplay Display;
EGLConfig Configs[1];
EGLint ContextAttributes[] = {
    EGL_CONTEXT_CLIENT_VERSION,
    2, // selects OpenGL ES 2.0, set to 1 to select OpenGL ES 1.1
    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 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 Emulator uses this compiler to check the shader syntax for the ESSL code.
2. The OpenGL ES 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.

Limitations based on the shader language version

For OpenGL ES 2.0 applications, the OpenGL ES 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 example applications on Windows

This section describes how to build the following examples:

- *Building the OpenGL ES 2.0 Cube example*
- *Building the OpenGL ES 1.1 simpApp11 example.*

2.3.1 Building the OpenGL ES 2.0 Cube example

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

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES Emulator *vm.n.o*\examples\OpenGLES_20\cube\src

To build and run this example:

1. Ensure the OpenGL ES Emulator is installed.
2. Ensure that locations of the DLLs for OpenGL ES Emulator are added to the system environment variable Path. This is described in *Using the OpenGL ES Emulator* on page 2-5.
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 Emulator *vm.n.o*\examples\OpenGLES_20\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.

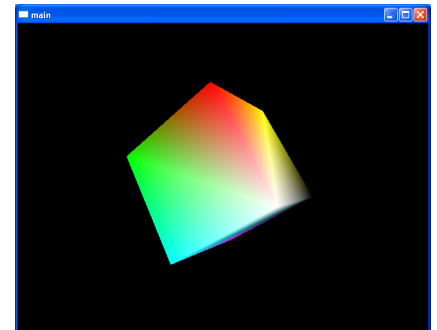


Figure 2-2 Cube example

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

2.3.2 Building the OpenGL ES 1.1 simpApp11 example

The simpApp11 example code for OpenGL ES 1.1 is included in the directory C:\Program Files\ARM\Mali Developer Tools\OpenGL ES Emulator *vm.n*\examples\OpenGLES_11\simpApp11

To build and run this example:

1. Ensure that OpenGL ES Emulator is installed.

2. The next steps involve using Microsoft Visual Studio 2005 to build and run the application:
 - a. Select **Start** → **All Programs** → **Microsoft Visual Studio 2005** → **Visual Studio Tools** → **Visual Studio 2005 Command Prompt**

- b. Change directory to where the example is present, that is:

C:\Program Files\ARM\Mali Developer Tools\Mali OpenGL ES Emulator vm.n.o\examples\OpenGL ES_11\simpApp11

- 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:

simpApp11.exe

- e. An additional window with a spinning, colored cube appears when the example application is running. See Figure 2-3:

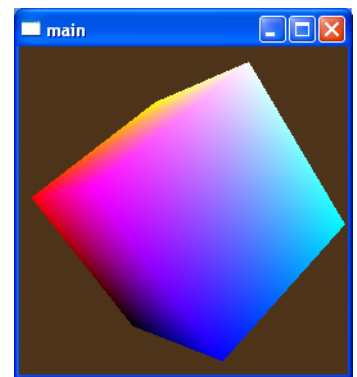


Figure 2-3 simpApp11 image

- f. To end the program, close this window and press **Ctrl+C**.

Chapter 3

Installation and Configuration on Linux

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

- *Installing the OpenGL ES emulator on Linux* on page 3-2
- *Configuring the OpenGL ES emulator on Linux* on page 3-5
- *Building the example applications on Linux* on page 3-8.

3.1 Installing the OpenGL ES 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 emulator has been tested successfully on a 32-bit computer.

3.1.1 Supported Hardware and Software

The OpenGL ES 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.

- The MESA OpenGL emulation software.

Note

- MESA was only tested for Stand-alone version 7.7.1.
 - The platform must also support GLX 1.4.
-

NVIDIA driver version

To determine the NVIDIA driver version on a Linux machine:

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

3.1.2 Disk requirements

The OpenGL ES 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 emulator*
- *OpenGL ES emulator content on page 3-3.*

Installing the OpenGL ES emulator

To install the OpenGL ES 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_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_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 emulator is installed in:

ARM/Mali_Developer_Tools/OpenGL_ES_Emulator_*m.n.o*

OpenGL ES emulator content

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

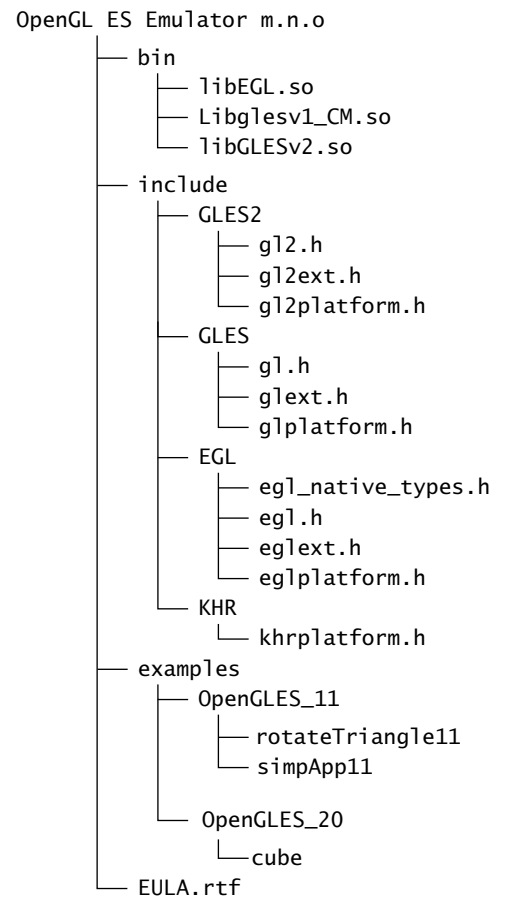


Figure 3-1 OpenGL ES emulator directory structure

3.2 Configuring the OpenGL ES emulator on Linux

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

- *Using the OpenGL ES emulator*
- *OpenGL ES emulator integration.*

3.2.1 Using the OpenGL ES emulator

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

1. To build an OpenGL ES application on the OpenGL ES 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 emulator libraries. 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 Emulator>/bin`

- The command to do this using the tcsh Linux shell is:
`setenv LD_LIBRARY_PATH <installation root directory for OpenGL ES Emulator>/bin`

For information on building the Cube example, see *Building the example applications on Linux* on page 3-8.

3.2.2 OpenGL ES emulator integration

This section describes:

- *Libraries*
- *EGL configuration* on page 3-6
- *EGL context creation* on page 3-6
- *Shader language version* on page 3-7.

Libraries

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

For convenience, the OpenGL ES emulator also includes the files required for Open GL ES 1.1. If you are building OpenGL ES 1.1 applications, you must include `libGLESv1_CM.so` instead of `libGLESv2.so`.

Table 3-1 shows the libraries for OpenGL ES 2.0 emulation:

Table 3-1 OpenGL ES emulator library structure

Filename	Description
<code>bin\libGLESv2.so</code>	Library for OpenGL ES 2.0 emulator
<code>bin\libEGL.so</code>	Library for EGL API
<code>bin\libGLESv1_CM.so</code>	Library for OpenGL ES 1.1 emulator

EGL configuration

The EGL library supplied with the OpenGL ES Emulator supports OpenGL ES 2.0 and OpenGL ES 1.1.

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, // use EGL_OPENGL_ES2_BIT|EGL_OPENGL_ES_BIT
                        // to return both 1.1 and 2.0 configs
    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 emulator supports both OpenGL ES 1.1 and 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, // selects OpenGL ES 2.0,
                                  // set to 1 to select OpenGL ES 1.1
    EGL_NONE
};

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

Shader language version

The OpenGL ES 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 example applications on Linux

This section describes how to build the following applications:

- *Building the OpenGL ES 2.0 cube example*
- *Building the OpenGL ES 1.1 simpApp11 example.*

3.3.1 Building the OpenGL ES 2.0 cube example

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

`<installation directory for OpenGL ES Emulator>/examples/OpenGL ES_20/cube`

To build and run this example:

1. Ensure that OpenGL ES 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 emulator* on page 3-5.
3. Navigate to the directory where the example is present:
`cd <installation directory for OpenGL ES Emulator>/examples/OpenGL ES_20/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.

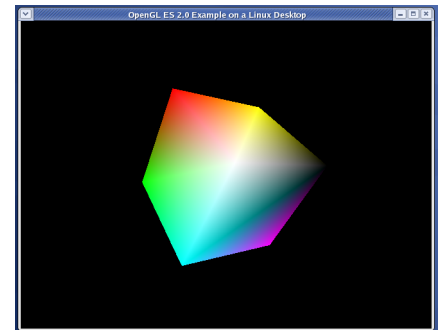


Figure 3-2 Cube image

To end the program, click anywhere on the window.

3.3.2 Building the OpenGL ES 1.1 simpApp11 example

The simpApp11 example code for OpenGL ES 1.1 is included in the directory:

`<installation directory for OpenGL ES Emulator>/examples/OpenGL ES_11/simpApp11`

To build and run this example:

1. Navigate to the directory where the example is present:
`cd <installation directory for OpenGL ES Emulator>/examples/OpenGL ES_11/simpApp11`
2. Use the `make` command to build the simpApp11 application

```
make
```

3. To run the example application, type the following command:

```
./simpApp11
```

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

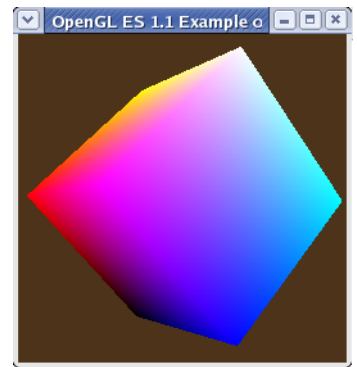


Figure 3-3 simpApp11 image

To end the program, close this window and press **Ctrl+C**

Chapter 4

Implementation Information

This chapter provides implementation information about OpenGL ES 2.0, OpenGL ES 1.1, and EGL APIs in the OpenGL ES Emulator. It contains the following sections:

- *OpenGL ES Implementation information* on page 4-2
- *EGL implementation information on Windows* on page 4-6
- *EGL implementation information on Linux* on page 4-10
- *Using the Mesa software emulation of OpenGL* on page 4-13.

4.1 OpenGL ES Implementation information

The OpenGL ES emulator converts OpenGL ES 2.0 and OpenGL ES 1.1 API calls to OpenGL 2.0 calls. These OpenGL 2.0 calls are handled by the Windows graphics drivers.

———— **Note** ————

The OpenGL ES Emulator will typically be used for OpenGL ES 2.0 applications, but it also supports OpenGL ES 1.1 applications. No additional library or DLL files are required for OpenGL ES 1.1 application emulation.

Because of the difference in specifications, OpenGL ES 2.0 or OpenGL ES 1.1 parameters are not always compatible with OpenGL 2.0. The API call conversion checks OpenGL ES 2.0 or OpenGL ES 1.1 parameters, and rejects invalid parameter values.

The OpenGL ES emulator depends on the functionality of the OpenGL 2.0 implementation provided by the graphics card drivers or MESA software. In some cases, this dependency can lead to limitations in the OpenGL ES 2.0 or OpenGL ES 1.1 implementation. This occurs when the behavior of the graphics card drivers differs from the OpenGL ES 2.0 specification.

This section describes:

- *High-performance mode*
- *General limitations*
- *NVIDIA GeForce 210 graphics card with driver version 190.45 for Windows on page 4-4.*

4.1.1 High-performance mode

In normal mode, the OpenGL ES emulator checks the setup details for all draw calls. If a configuration is found that is OpenGL 2.0 compatible, but not OpenGL ES 2.0 compatible, the draw operation returns with an error. This checking does add some overhead to the emulation process. Performance can be improved by disabling the conformance tests in `glDrawElements()` and `glDrawArrays()`.

To disable conformance checking, set the following environment variable `GLS2_NON_CONFORMANT_OPTIMIZED` to any non-zero value.

If the variable does not exist or is not set to a non-zero value, the conformance tests are performed.

4.1.2 General limitations

This section describes:

- *Implementation-specific behavior on page 4-3*
- *glShaderBinary always fails (OpenGL ES 2.0 only) on page 4-3*
- *Fixed-point data gives reduced performance on page 4-3*
- *Shader precision qualifiers are ignored (OpenGL ES 2.0 only) on page 4-3*
- *glGetShaderPrecisionFormat values (OpenGL ES 2.0 only) on page 4-3*
- *glGenerateMipmap performance (OpenGL ES 2.0 only) on page 4-3*
- *Compressed texture formats on page 4-3*
- *OpenGL GLSL Shader compiler errors reported by the OpenGL ES emulator cannot be easily mapped onto the original source code (OpenGL ES 2.0 only) on page 4-4*
- *Multiple threads and multiple contexts on page 4-4.*

Implementation-specific behavior

Where the OpenGL ES 2.0 or OpenGL ES 1.1 specifications permit 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 (on OpenGL ES 2.0)
- framebuffers.

glShaderBinary always fails (OpenGL ES 2.0 only)

Because of the incompatibility between binary formats for different graphics engines, the OpenGL ES 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 emulator converts fixed-point data and passes it to OpenGL 2.0. For the OpenGL ES 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 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 (OpenGL ES 2.0 only)

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 (OpenGL ES 2.0 only)

`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 (OpenGL ES 2.0 only)

`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 emulator cannot be easily mapped onto the original source code (OpenGL ES 2.0 only)

Shader compiler error line numbers reported from the underlying OpenGL graphics driver might not match because of the translation of ESSL to GLSL.

Due to translation of shader language from ESSL to GLSL for use by the underlying OpenGL graphics driver and the concatenation of strings input to `glShaderSource`, error line numbers may not match the original source code.

Note

- On Windows, ensure the malisc compiler is installed and on the path. The shader code is sent unmodified to malisc, so line numbers are correct. Note that separate strings submitted to `glShaderSource` are concatenated before sending to malisc.
 - On Linux, there is no workaround.
-

Multiple threads and multiple contexts

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

Vertex buffer objects performance (OpenGL ES 1.1 only)

Vertex buffer objects are supported by OpenGL ES Emulator and are implemented in software for OpenGL ES 1.1 applications. Vertex buffer objects therefore follow the behavior specified in the OpenGL ES 1.1 specification rather than the behavior of the underlying OpenGL 2.0 driver. This might affect performance.

4.1.3 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*
- *Framebuffer object with depth and stencil buffer not supported* on page 4-5
- *Framebuffer object with stencil buffer and no depth buffer not supported* on page 4-5
- *Image attachment* on page 4-5
- *Clipping to the viewport* on page 4-5
- *Link failures with attribute aliasing* on page 4-5
- *Link can succeed with undefined varying variables* on page 4-5
- *Driver adjustment of texture filtering, anti-aliasing and anisotropic filtering* on page 4-5.

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 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 4-4. However, in general, you cannot rely on the texture *Level Of Detail* (LOD) calculations or the choice between minification and magnification to be accurate.

4.2 EGL implementation information on Windows

This section describes:

- *Display initialization*
- *Default display example*
- *Window example*
- *EGL configurations* on page 4-7
- *EGL contexts* on page 4-7
- *Creation of window surface* on page 4-7
- *Creation of pixmap surfaces* on page 4-7
- *Creation of Pbuffer surfaces* on page 4-7
- *Synchronization of pixmap surfaces* on page 4-7
- *EGL limitations* on page 4-8.

4.2.1 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 Emulator. You must pass to this function either the:

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

Default display example

Example 4-1 shows a code example that uses the default display:

Example 4-1 Display initialization

```

EGLDisplay sEGLDisplay;

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

```

Window example

Example 4-2 shows a code example that uses a window display:

Example 4-2 Default display

```

EGLDisplay sEGLDisplay;

...

// Create window
sWindow = CreateWindowEx(...

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

```

4.2.2 EGL configurations

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

To get a valid configuration from `eglChooseConfig()`, set the `EGL_RENDERABLE_TYPE` in the attributes list to either:

- `EGL_OPENGL_ES2_BIT` to select OpenGL ES 2.0 configuration only
- `EGL_OPENGL_ES_BIT` to select OpenGL ES 1.1 configuration only
- `EGL_OPENGL_ES2_BIT|EGL_OPENGL_ES_BIT` to select both.

If you do not include a value for the `EGL_RENDERABLE_TYPE` attribute, `eglChooseConfig()` uses the default value which is `EGL_OPENGL_ES_BIT`.

———— **Caution** ————

If you set `EGL_RENDERABLE_TYPE` to `EGL_OPENVG_BIT`, no configurations are returned.

4.2.3 EGL contexts

The EGL implementation supports OpenGL ES 2.0 and OpenGL ES 1.1. It does not support 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. Setting the value to 2 selects OpenGL ES 2.0 support.

To obtain a valid context, set `EGL_CONTEXT_CLIENT_VERSION` to either 1 or 2 in the attributes list. Any other values result in context creation failing.

4.2.4 Creation of window surface

For an example of the code to create a window surface in an OpenGL ES 2.0 application, see the file `examples\OpenGLES_20\cube\src\main.c`.

For an example of the code to create a window surface in an OpenGL ES 1.1 application, see the file `examples\OpenGLES_11\simpApp11\simpApp11.c`.

4.2.5 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()`. The call enables access to the data bits in the bitmap.

4.2.6 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.

4.2.7 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.

Note

You must not select a native bitmap into a device context, because a native bitmap causes the the `eglWaitNative()` call to fail.

4.2.8 EGL limitations

The EGL library has sufficient functionality for the OpenGL ES 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 PC with either Windows XP or Windows 7.

The EGL library is a limited implementation of the EGL 1.3 specification. This section provides additional information about these limitations:

- *Support for OpenGL ES 1.1 and OpenGL ES 2.0 only*
- *Multiple threads and multiple contexts*
- *Window pixel format*
- *Limited bitmap support*
- *Limited results from surface queries on page 4-9*
- *No support for swap intervals on page 4-9*
- *Changing display modes does not check pbuffer lost event on page 4-9*
- *Use of displays following `eglTerminate` on page 4-9*
- *`EGL_MATCH_NATIVE_PIXMAP` attribute not supported on page 4-9*
- *Resizing a native window on page 4-9*
- *`EglChooseConfig` always sets `WGL_DOUBLE_BUFFER_ARB` true on page 4-9.*

Support for OpenGL ES 1.1 and OpenGL ES 2.0 only

The EGL library does not support graphics contexts and surfaces for use with OpenVG. No configurations are returned from `eglChooseConfig()` for values of `EGL_RENDERABLE_TYPE` other than `EGL_OPENGL_ES2_BIT` or `EGL_OPENGL_ES_BIT`.

Context creation fails unless `EGL_CONTEXT_CLIENT_VERSION` is set to 1 or 2.

Multiple threads and multiple contexts

Multiple contexts are supported, but multiple threads 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.

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 does not check pbuffer lost event

Changing display modes is not supported. A change of display mode might result in loss of Pbuffer memory. This event is not checked for. Do not change display modes while running the emulator.

———— Note —————

Pbuffers and pixmaps are supported with the `WGL_ARB_pbuffer` extension. This specifies that a `WGL_PBUFFER_LOST_ARB` query can check for loss of memory due to a display mode change.

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()`.

The EGL 1.3 specification says that the attribute `EGL_MATCH_NATIVE_PIXMAP` was introduced to make it easier to choose an `EGLConfig` to match a native pixmap. This attribute is accepted by the emulator, but is ignored other than to validate the provided handle.

Applications should work as expected even if the chosen `EGLConfig` does not match the pixmap format because rendering is done to an internal buffer and then copied to the pixmap, including any necessary pixel format conversions. If an eight bit per channel `EGLConfig` is desired (to ensure the same color precision as the native pixmap), then `EGL_RED_SIZE`, `EGL_GREEN_SIZE` and `EGL_BLUE_SIZE` should be explicitly passed to `eglChooseConfig()`.

Resizing a native window

Resizing a native window does not update the surface attributes.

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.

4.3 EGL implementation information on Linux

The EGL implementation intends to supply sufficient functionality for the OpenGL ES emulator to pass Khronos OpenGL ES 2.0 and OpenGL ES 1.1 conformance tests and to provide a platform for OpenGL ES 2.0 or OpenGL ES 1.1 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 4-11
- *Attributes for windows not supported* on page 4-11
- *Some Pbuffer attributes are not supported* on page 4-11
- *Attributes for pixmaps not supported* on page 4-11
- *Incorrect error code returned instead of EGL_BAD_MATCH* on page 4-11
- *Limited results from surface queries* on page 4-11
- *eglMakeCurrent succeeds with incompatible surface and contents* on page 4-11.

4.3.1 Unimplemented functions

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

- `eglCreatePbufferFromClientBuffer()`
- `eglSurfaceAttrib()`
- `eglSwapInterval()`
- `eglCopyBuffers()`

4.3.2 Resizing a native window

Resizing a native window does not update the surface attributes.

4.3.3 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.

4.3.4 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.
-

4.3.5 EGLConfigs not sorted

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

Applications must not rely on the configurations returned `eglChooseConfig()` by being sorted.

4.3.6 Attributes for windows not supported

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

4.3.7 Some Pbuffer attributes are not supported

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

- `EGL_VG_COLORSPACE`
- `EGL_VG_ALPHA_FORMAT`

4.3.8 Attributes for pixmaps not supported

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

4.3.9 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.

4.3.10 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`

4.3.11 eglMakeCurrent succeeds with incompatible surface and contents

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

EGL is implemented on top of GLX. The GLX layer does not detect this incompatibility on the test platform. Applications must not rely on `eglMakeCurrent()` detecting incompatibility between surface and context.

4.4 Using the Mesa software emulation of OpenGL

If your graphics card does not have drivers for OpenGL, you can use the Mesa implementation of the OpenGL specification. Mesa is available for Linux and Windows.

To use Mesa, you must direct the application to use the Mesa library instead of the vendor-supplied `opengl32.dll` or `libGL.so` library. Overwriting the standard library with the Mesa library is time consuming and affects all of your applications.

ARM recommends setting the `PATH` and `LD_LIBRARY_PATH` variables in the terminal window that you use to start the application.

4.4.1 Setting the PATH variable in Windows

To use Mesa with your Windows application:

1. Build the Mesa library and note the location of the generated `opengl32.dll` file.
2. Open a terminal window.
3. Enter the following command:

```
set PATH=path_to_mesa_dll\;%PATH%
```

———— **Note** —————

The modified path only applies to applications started in the terminal window.

4. Start your graphics application. For example, to start the cube example, enter `cube.exe`. For more information on building and running the cube example, see *Building the example applications on Windows* on page 2-8.
5. After you finish viewing the application, close the terminal window.

———— **Note** —————

This sequence will only work if there is not another `opengl32.dll` present in the system.

4.4.2 Setting the LD_LIBRARY_PATH variable in Linux

To use Mesa with your Linux application:

1. Build the Mesa library and note the location of the generated `libGL.so` file.
2. Open a terminal window.
3. If you have the `tcsh` shell, enter the following command:

```
setenv LD_LIBRARY_PATH path_to_mesa_lib/:$LD_LIBRARY_PATH
```

If you have the `bash` shell, enter the following command:

```
export LD_LIBRARY_PATH=path_to_mesa_lib/:$LD_LIBRARY_PATH
```

———— **Note** —————

The modified path only applies to applications started in the terminal window.

4. Start your graphics application. For example, to start the cube example, enter `./cube`. For more information on building and running the cube example, see *Building the example applications on Linux* on page 3-8.

5. After you finish viewing the application, close the terminal window.