OpenGL ES 3.0 Emulator
User Guide

Copyright © 2012 ARM. All rights reserved.

Release Information

The following changes have been made to this book.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 July 2012</td>
<td>A</td>
<td>Non-Confidential</td>
<td>First release for v1.0 for OpenGL ES 3.0.</td>
</tr>
</tbody>
</table>

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 3.0 Emulator User Guide

Preface
About this book ........................................................................................................ v
Feedback ........................................................................................................ vii

Chapter 1 Introduction
1.1 About the OpenGL ES 3.0 Emulator ............................................................... 1-2

Chapter 2 Installation and Configuration on Windows
2.1 Installing the OpenGL ES 3.0 Emulator on Windows ................................. 2-2
2.2 Configuring the OpenGL ES 3.0 Emulator on Windows ............................... 2-5
2.3 Building the OpenGL ES 3.0 example application on Windows .................... 2-8

Chapter 3 Implementation Information
3.1 OpenGL ES Implementation information ................................................. 3-2
3.2 EGL implementation information on Windows ........................................... 3-6
Preface

This preface introduces the OpenGL ES 3.0 Emulator User Guide. It contains the following sections:

- *About this book* on page v
About this book

This is the OpenGL ES 3.0 Emulator User Guide. It provides guidelines for using the OpenGL ES 3.0 Emulator to develop 2D and 3D graphics applications that are targeted to run on an embedded platform.

Intended audience

This guide is written for system integrators and software developers using a PC to develop OpenGL ES 2.0 or OpenGL ES 3.0 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 3.0 Emulator.

**Chapter 2 Installation and Configuration on Windows**
Read this for a description on how to install and configure the OpenGL ES 3.0 Emulator on Windows.

**Chapter 3 Implementation Information**
Read this for information about the implementation of the OpenGL ES 3.0, OpenGL ES 2.0, and EGL APIs in the OpenGL ES 3.0 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.


Typographical Conventions

The typographical conventions are:

- **italic**: 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.
- **monospace** **italic**: Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
- **monospace** **bold**: 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.
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](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:

- **OpenGL ES Emulator** (ARM DUI 0587)
- **Mali™ Software Development Kit for Android** (ARM DUI 0511)
- **Mali™ Software Development Kit for Linux on ARM** (ARM DUI 0607)
- **Mali™ Performance Analysis Tool User Guide** (ARM DUI 0502)
- **Mali™ Texture Compression Tool User Guide** (ARM DUI 0503)
- **Mali™ Shader Developer Studio User Guide** (ARM DUI 0504)
- **Mali™ User Interface Engine User Guide** (ARM DUI 0505)
- **Mali™ Mali Binary Asset Exporter User Guide** (ARM DUI 0507)
- **Mali™ Shader Library User Guide** (ARM DUI 0510)
- **Mali™ Application Optimization Guide** (ARM DUI 555)

Other publications

This section lists relevant documents published by third parties:

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 0668A
- 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 3.0 Emulator, and describes how to start using the tool in your particular workflow.

It contains the following section:

• *About the OpenGL ES 3.0 Emulator on page 1-2.*
1.1  About the OpenGL ES 3.0 Emulator

The OpenGL ES 3.0 Emulator is a library that maps OpenGL ES API calls to the OpenGL API. This allows embedded applications that target an SoC supporting OpenGL ES to run on a desktop PC.
Chapter 2
Installation and Configuration on Windows

This chapter provides information about installing and configuring the Mali GPU OpenGL ES 3.0 Emulator on Microsoft Windows. It contains the following sections:

- Installing the OpenGL ES 3.0 Emulator on Windows on page 2-2
- Configuring the OpenGL ES 3.0 Emulator on Windows on page 2-5
- Building the OpenGL ES 3.0 example application on Windows on page 2-8.
2.1 Installing the OpenGL ES 3.0 Emulator on Windows

This section describes the installation on Microsoft Windows.

--- Note ---

The OpenGL ES 3.0 Emulator has been tested successfully on 32-bit computers.

---

2.1.1 Supported Hardware and Software

The OpenGL ES 3.0 Emulator, Windows version, has been tested with the following software and hardware:

- 32-bit Windows 7 Professional SP1 and NVIDIA GeForce GTX 550 graphics card.
- 32-bit Windows 7 Professional SP1 and AMD/ATI Radeon HD 4850 graphics card.

There are the following restrictions:

- The graphics card versions are recommendations. The OpenGL ES 3.0 Emulator typically also works with other graphics cards and driver versions provided they support OpenGL 3.3 or above with appropriate extensions. Wherever possible, update your drivers to the latest version.

  Older graphics cards might work if you are only emulating OpenGL ES 2.0 applications.

- The minimum driver extension is `WGL_ARB_transform_feedback2` which is required for pause and resume transform feedback for OpenGL ES 3.0. The OpenGL ES 3.0 Emulator will run as normal, but crash if the user tries to pause or resume transform feedback.

- The following OpenGL ES 3.0 features are not available in OpenGL 3.3 and are emulated in software:
  - Immutable-Format Textures, but not emulated if the `ARB_texture_storage` GL extension is present.
  - Internal Format Queries, but not emulated if `ARB_internalformat_query` GL extension is present.
  - Transform Feedback Objects, but not emulated if `ARB_transform_feedback2` GL extension is present.
  - Some ES3.0 specific functions such as `ReleaseShaderCompiler()`, `GetShaderPrecisionFormat()`, `DepthRangef()`, and `ClearDepthf()`, are handled by the `ARB_ES2_compatibility` GL extension if it is present.

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 3.0 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 3.0 Emulator on Microsoft Windows
• OpenGL ES 3.0 Emulator content.

Installing the OpenGL ES 3.0 Emulator on Microsoft Windows

To install the OpenGL ES 3.0 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_3_0_Emulator_m.n.o.p_Win32.exe
   where:
   m identifies the major version
   n identifies the minor version.
   o.p identifies the part and build version.
3. Run the OpenGL_ES_3_0_Emulator_m.n.o.p_Win32.exe file by double clicking on it.
4. Select the installation options and click Finish to complete the installation.

By default, the OpenGL ES 3.0 Emulator is installed in:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 3.0 Emulator v\m.n.o\n
OpenGL ES 3.0 Emulator content

The download package contains the OpenGL ES 3.0 Emulator binaries for Windows and simple sample applications, for OpenGL ES 2.0 and OpenGL ES 3.0, that run on the OpenGL ES 3.0 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 OpenGL ES 3.0 Emulator. The default installation directory is:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 3.0 Emulator v\m.n.o\
Figure 2-1 OpenGL ES 3.0 Emulator directory structure
2.2 Configuring the OpenGL ES 3.0 Emulator on Windows

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

- **OpenGL ES 3.0 Emulator DLLs and libraries**
- **EGL configuration**
- **EGL context creation on page 2-6**
- **Limitations based on the shader language version on page 2-7.**

### 2.2.1 OpenGL ES 3.0 Emulator DLLs and libraries

The OpenGL ES 3.0 Emulator Library consists of DLLs corresponding to the OpenGL ES 3.0 and EGL 1.3 APIs.

- OpenGL ES 3.0 applications must include the libEGL.lib and libGLES2.lib libraries in builds to link against the OpenGL ES 3.0 API.
  
The dynamically linked libraries, libEGL.dll and libGLESv2.dll, must be provided at run-time. The DLLs use the _stdcall calling convention.

- OpenGL ES 2.0 applications must include the libGLESv2.lib and libEGL.lib libraries in builds to link against the OpenGL ES 2.0 API.
  
The dynamically linked libraries, libEGL.dll and libGLESv2.dll, must be provided at run-time. The DLLs use the _stdcall calling convention.

**Caution**

- The OpenGL ES 3.0 libraries are backward compatible with OpenGL ES 2.0, but not OpenGL ES 1.1.

- If you require OpenGL ES 1.1 or strict OpenGL ES 2.0 compliance, use a version 1.2 of the OpenGL ES 3.0 Emulator.

Table 2-1 shows the files for emulation:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin\libGLESv2.dll</td>
<td>DLL for OpenGL ES 3.0 emulation</td>
</tr>
<tr>
<td>1ib\libGLESv2.lib</td>
<td>Import library for libGLESv3.dll</td>
</tr>
<tr>
<td>bin\libEGL.dll</td>
<td>DLL for EGL implementation</td>
</tr>
<tr>
<td>1ib\libEGL.lib</td>
<td>Import library for libEGL.dll</td>
</tr>
</tbody>
</table>

### 2.2.2 EGL configuration

The EGL library supplied with the OpenGL ES 3.0 Emulator supports OpenGL ES 2.0, and OpenGL ES 3.0.

Ensure that, in the OpenGL ES application, the attribute list passed as a parameter to eglChooseConfig() includes the attribute EGL_RENDERABLE_TYPE with an appropriate value of EGL_OPAQUE_ES2_BIT for the OpenGL ES version:

- Use EGL_OPAQUE_ES3_BIT to request OpenGL ES 3.0 configs.
- Use EGL_OPAQUE_ES2_BIT to request OpenGL ES 2.0 configs.
• OR the constants together to request more than one context. For example, use
EGL_OPENGL_ES3_BIT|EGL_OPENGL_ES2_BIT to request both OpenGL ES 3.0 and OpenGL ES
2.0 configs.

--- Note ---
The actual context used is specified in eglCreateContext() described in Example 2-2.

Example 2-1 shows a coded section.

Example 2-1 EGL context configuration

```c
EGLDisplay Display;

EGLint Attributes[] = {
    EGL_RENDERABLE_TYPE,
    EGL_OPENGL_ES3_BIT, // use EGL_OPENGL_ES3_BIT|EGL_OPENGL_ES2_BIT
                        // to return both 3.0 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);
```

2.2.3 EGL context creation

Example 2-2 shows a coded section. Set the second element of the ContextAttributes array to
select an OpenGL ES context:
• Set the element to 3 to select OpenGL ES 3.0 contexts.
• Set the element to 2 to select OpenGL ES 2.0 contexts.

Example 2-2 EGL context creation

```c
EGLDisplay Display;
EGLConfig Configs[1];
EGLint ContextAttributes[] = {
    EGL_CONTEXT_CLIENT_VERSION,
    3,      // selects OpenGL ES 3.0, set to 2 to select OpenGL ES 2.0
    EGL_NONE
};
...
Context = eglCreateContext(Display, Configs[0], EGL_NO_CONTEXT,
                           ContextAttributes);
```
2.2.4 Limitations based on the shader language version

The OpenGL ES 3.0 Emulator uses version 3.30 of the OpenGL Shading Language.
2.3 Building the OpenGL ES 3.0 example application on Windows

The cube example code for OpenGL ES 3.0 is included in the following directory:

C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 3.0 Emulator v.m.n.o\examples\OpenGLES_30\cube\src

To build and run this example:

1. Ensure the OpenGL ES 3.0 Emulator is installed.
2. Ensure that locations of the DLLs for OpenGL ES 3.0 Emulator are added to the system environment variable Path. This is described in Configuring the OpenGL ES 3.0 Emulator on Windows on page 2-5.
3. The next steps use Microsoft Visual Studio 2008 to build and run the application:
   b. Change directory to the following directory that contains the example:
C:\Program Files\ARM\Mali Developer Tools\OpenGL ES 3.0 Emulator v.m.n.o\examples\OpenGLES_30\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.

f. To end the program, close this window.
Chapter 3

Implementation Information

This chapter provides implementation information about OpenGL ES 3.0 and EGL APIs in the OpenGL ES 3.0 Emulator. It contains the following sections:

- *OpenGL ES Implementation information* on page 3-2
- *EGL implementation information on Windows* on page 3-6.
### 3.1 OpenGL ES Implementation information

The OpenGL ES 3.0 Emulator converts OpenGL ES calls to OpenGL calls:

- OpenGL ES 3.0 API calls are converted to OpenGL 3.3 calls. These OpenGL 3.3 calls are handled by the platform graphics drivers.
- OpenGL ES 2.0 API calls are converted to OpenGL 2.0 calls. These OpenGL 2.0 calls are handled by the platform graphics drivers.

**Note**

- No additional library or DLL files are required for OpenGL ES 2.0 application emulation.
- Version 1.0 of the OpenGL ES 3.0 Emulator does not support OpenGL ES 1.1 applications.

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

The OpenGL ES 3.0 Emulator depends on the functionality of the OpenGL implementation provided by the graphics card drivers. In some cases, this dependency can lead to limitations in the OpenGL ES implementation. This occurs when the behavior of the graphics card drivers differs from the OpenGL 3.3 specification for OpenGL ES 3.0 applications (or OpenGL 2.0 for OpenGL ES 2.0 applications).

This section describes:

- General limitations
- NVIDIA GeForce 210 graphics card with driver version 190.45 for Windows on page 3-4.

#### 3.1.1 General limitations

This section describes:

- Implementation-specific behavior
- `glShaderBinary` always fails on page 3-3
- Fixed-point data gives reduced performance on page 3-3
- Shader precision qualifiers are ignored on page 3-3
- Compressed texture formats on page 3-3
- Multiple threads and multiple contexts on page 3-4.

**Implementation-specific behavior**

Where the OpenGL ES 3.0 or OpenGL ES 2.0 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.
glProgramBinary always fails

The emulator does not support any program binary formats. The call glProgramBinary() always returns GL_INVALID_OPERATION.

glShaderBinary always fails

Because of the incompatibility between binary formats for different graphics engines, the OpenGL ES 3.0 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 3.0 does not provide support for fixed-point data, but this is required by the OpenGL ES 3.0 specification. The OpenGL ES 3.0 Emulator converts fixed-point data and passes it to OpenGL. For the OpenGL ES 3.0 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 3.0 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 3.0 Shading Language (ESSL) and OpenGL ES 3.0 Shading Language have no equivalents in the OpenGL 3.3 Shading Language (GLSL) and OpenGL 2.0 Shading Language, and are removed. Because precision of shader variables is implementation dependent in OpenGL, shader variables might not have the minimum range or precision required by the ESSL specification.

glGetShaderPrecisionFormat values

For OpenGL ES 3.0 applications, glGetShaderPrecisionFormat() forwards the call to the underlying GL implementation if the ARB_ES2 compatibility extension is present. If the extension is not present, the following range and precision information is reported:

- **GL_LOW_FLOAT / GL_MEDIUM_FLOAT / GL_HIGH_FLOAT** precision type:
  - Min range: 127
  - Max range: 127
  - Precision: 23

- **GL_LOW_INT, GL_MEDIUM_INT, GL_HIGH_INT** precision type:
  - Min range: 31
  - Max range: 30
  - Precision: 0

Compressed texture formats

The emulator supports (and reports support for) the following compressed texture formats for OpenGL ES 3.0:

- GL_COMPRESSED_R11_EAC
- GL_COMPRESSED_RG11_EAC
- GL_COMPRESSED_RGB8_ETC2
- GL_COMPRESSED_RGB8_PUNCHTHROUGH_ALPHA1_ETC2
• GL_COMPRESSED_RGBA8_ETC2_EAC
• GL_COMPRESSED_SIGNED_R11_EAC
• GL_COMPRESSED_SIGNED_RG11_EAC
• GL_COMPRESSED_SRGB8_ALPHA8_ETC2_EAC
• GL_COMPRESSED_SRGB8_PUNCHTHROUGH_ALPHA1_ETC2
• GL_COMPRESSED_SRGB8_ETC2.

Internally, the textures are decompressed to unsigned byte format (amount of components depends on the compression algorithm in question). Due to that, no performance boost should be expected when using compressed textures with the emulator.

Multiple threads and multiple contexts

For OpenGL ES 3.0 applications, Multiple contexts and multiple threads are supported.

3.1.2 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
• Framebuffer object with depth and stencil buffer not supported
• Image attachment on page 3-5
• Clipping to the viewport on page 3-5
• Link failures with attribute aliasing on page 3-5
• Link can succeed with undefined varying variables on page 3-5
• Driver adjustment of texture filtering, anti-aliasing and anisotropic filtering on page 3-5.

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.
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 3-4. However, in general, you cannot rely on the texture Level Of Detail (LOD) calculations or the choice between minification and magnification to be accurate.
3.2  **EGL implementation information on Windows**

This section describes:

- Display initialization
- Default display example
- Window example
- EGL configurations
- EGL contexts on page 3-7
- Creation of pixmap surfaces on page 3-7
- Creation of Pbuffer surfaces on page 3-7
- Synchronization of pixmap surfaces on page 3-7
- EGL limitations on page 3-8.

### 3.2.1 Display initialization

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

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

**Default display example**

*Example 3-1* shows a code example that uses the default display:

```c
Example 3-1 Display initialization

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

**Window example**

*Example 3-2* shows a code example that uses a window display:

```c
Example 3-2 Default display

EGLDisplay sEGLDisplay;

...  
// Create windowsWindow = CreateWindowEx(...
// EGL init.
sEGLDisplay = eglGetDisplay(GetDC(sWindow));
eglInitialize(sEGLDisplay, NULL, NULL);
```

### 3.2.2 EGL configurations

The EGL implementation supports OpenGL ES 3.0 and OpenGL ES 2.0.

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_ES3_BIT to select OpenGL ES 3.0 configuration only
- EGL_OPENGL_ES2_BIT to select OpenGL ES 2.0 configuration only
- OR values to select multiple configurations. For example, EGL_OPENGL_ES3_BIT|EGL_OPENGL_ES3_BIT selects the OpenGL ES 3.0 and OpenGL ES 2.0 configuration.

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_OPENGL_BIT, no configurations are returned.

### 3.2.3 EGL contexts

The EGL implementation supports OpenGL ES 3.0 and OpenGL ES 2.0. 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, and setting the value to 3 selects OpenGL ES 3.0 support.

**Note**
- To obtain a valid context, set EGL_CONTEXT_CLIENT_VERSION to the either 3 or 2 in the attributes list.
- Any other values result in context creation failing.

### 3.2.4 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.

### 3.2.5 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.

### 3.2.6 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 `eglWaitNative()` call to fail.

### 3.2.7 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 2.0 and OpenGL ES 3.0 only**
- **Multiple threads and multiple contexts**
- **Window pixel format**
- **Limited bitmap support**
- **Limited results from surface queries**
- **No support for swap intervals on page 3-9**
- **Changing display modes does not check pbuffer lost event on page 3-9**
- **Use of displays following eglTerminate on page 3-9**
- **EGL_MATCH_NATIVE_PIXMAP attribute not supported on page 3-9**
- **Resizing a native window on page 3-9**
- **EglChooseConfig always sets WGL_DOUBLE_BUFFER_ARB true on page 3-9.**

#### Support for OpenGL ES 2.0 and OpenGL ES 3.0 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 2 or 3.

#### 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.