Mali OpenGL® ES Emulator v1.3.1

User Guide

ARM® MALI™

Visual Computing

malideveloper.arm.com

Contents

Introduction

The Mali OpenGL® ES Emulator allows you to use the OpenGL ES 2.0 and 3.0 APIs to render 3D graphics content on a desktop PC running Windows or Linux. You can use it to try out ideas and develop content without having access to an embedded or mobile device, or to allow your cross-platform application to target both mobile and desktop GPUs without having to modify the graphics layer.

This document describes how to install and use the Mali OpenGL ES Emulator for Windows and Linux.

1 Minimum Requirements

In general, the Mali OpenGL ES Emulator will work on any system that supports at least OpenGL 3.2 (for OpenGL ES 2.0 contexts only) or OpenGL 3.3 (for OpenGL ES 3.0 contexs). Detailed requirements are shown below. On all systems, up-to-date operating system components and graphics drivers are recommended.

1.1 Linux

In order to build and run the supplied example code, a GNU toolchain is required. To install the Mali OpenGL ES Emulator to a system-wide location, root access may be required.

1.2 Windows

In order to build and run the supplied example code, Microsoft Visual Studio 2005 or higher is required.

The Microsoft Visual C++ 2008 Redistributable Package is required and can be downloaded from the Microsoft website.

2 Installation and Configuration

The installation package for the Mali OpenGL ES Emulator contains everything you need to get started building OpenGL ES 2.0 and 3.0 applications on a desktop computer. It includes header files and shared libraries to which you can link your application in order to render 3D graphics using the OpenGL ES 2.0 and 3.0 APIs.

You should have downloaded the installation package appropriate to your platform. The Mali OpenGL ES Emulator is available for both Windows and Linux, and comes in both 32 and 64 bit variants. Typically, you should download the package that most closely resembles your host environment. If you aren't sure which package to choose, the 32 bit version will work across the widest variety of platforms.

Advanced usage: It's possible to install both the 32 and 64 bit variants of the Mali OpenGL ES Emulator on the same system, but the correct management of paths is left to the user. The installer for both Windows and Linux will set up the system to use the emulator headers and libraries that were installed most recently.

Warning: On Linux, both the 32 and 64 bit installers will attempt to copy the package content to the same directories under `/usr/local`; running both installation scripts will overwrite the previous installation! Users of multiple architectures should copy the package content manually to avoid any overwriting.

The installation package contains three main components: the emulator libraries, the header files and a simple demonstration program.

2.1 Linux

On Linux, the Mali OpenGL ES Emulator is provided as a `.tgz` package that can be extracted to any location on disk. This package can be extracted using any modern version of GNU `tar`:

```
tar xvzf Mali_OpenGL_ES_Emulator_v1.3.1.<build>_Linux_<arch>.tgz
```

This will extract the headers, libraries and example code to a folder in the current working directory. The emulator can be used as-is in this state, but if you wish you can install it into a system-wide location, which may simplify adding the libraries and headers into your project. To do this, run

```
sudo ./install.sh
```

from within the extracted directory.

This will:

a) copy the header files required for developing OpenGL ES applications to `/usr/local/include`,
b) in `/usr/local/lib` directory install the following libraries:
    a. `libGLESv2.so`
    b. `libEGL.so`
    c. `libGLESv2.so.2.0.0`
    d. `libEGL.so.1.4.0`
    e. `libMaliT6xxSC.so`
c) in `/usr/local/bin` directory install:
    a. `mali_essl_checker` executable

The example code will not be copied.

Note: if you wish to use the Mali OpenGL ES Emulator without installing to a system-wide location, you will be required to manually remove version number in libraries by changing libraries names to `libGLESv2.so` and `libEGL.so` respectively and add the libraries and headers to your build environment.

2.2 Windows

On Windows, the Mali OpenGL ES Emulator is provided as an executable installer package. To install the software, run the installer and follow the instructions on screen. The installer will prompt you to specify where to install the libraries and headers, and also prompt for a location into which the example code will be installed.

When installation is complete, the following files are installed in destination location:

a) within `bin` sub-directory:
   a. `libGLESv2.dll`
   b. `libEGL.dll`
   c. `libMaliT6xxSC.dll`
   d. `mali_essl_checker.exe`

b) within `lib` sub-directory:
   a. `libEGL.lib`
   b. `libGLESv2.lib`
   c. within `include` sub-directory header files required for developing OpenGL ES applications

Note: By default the location for the headers and libraries and the location for the example code differ.

3 Getting Started with the Mali OpenGL ES Emulator

3.1 Verifying the Installation is Working

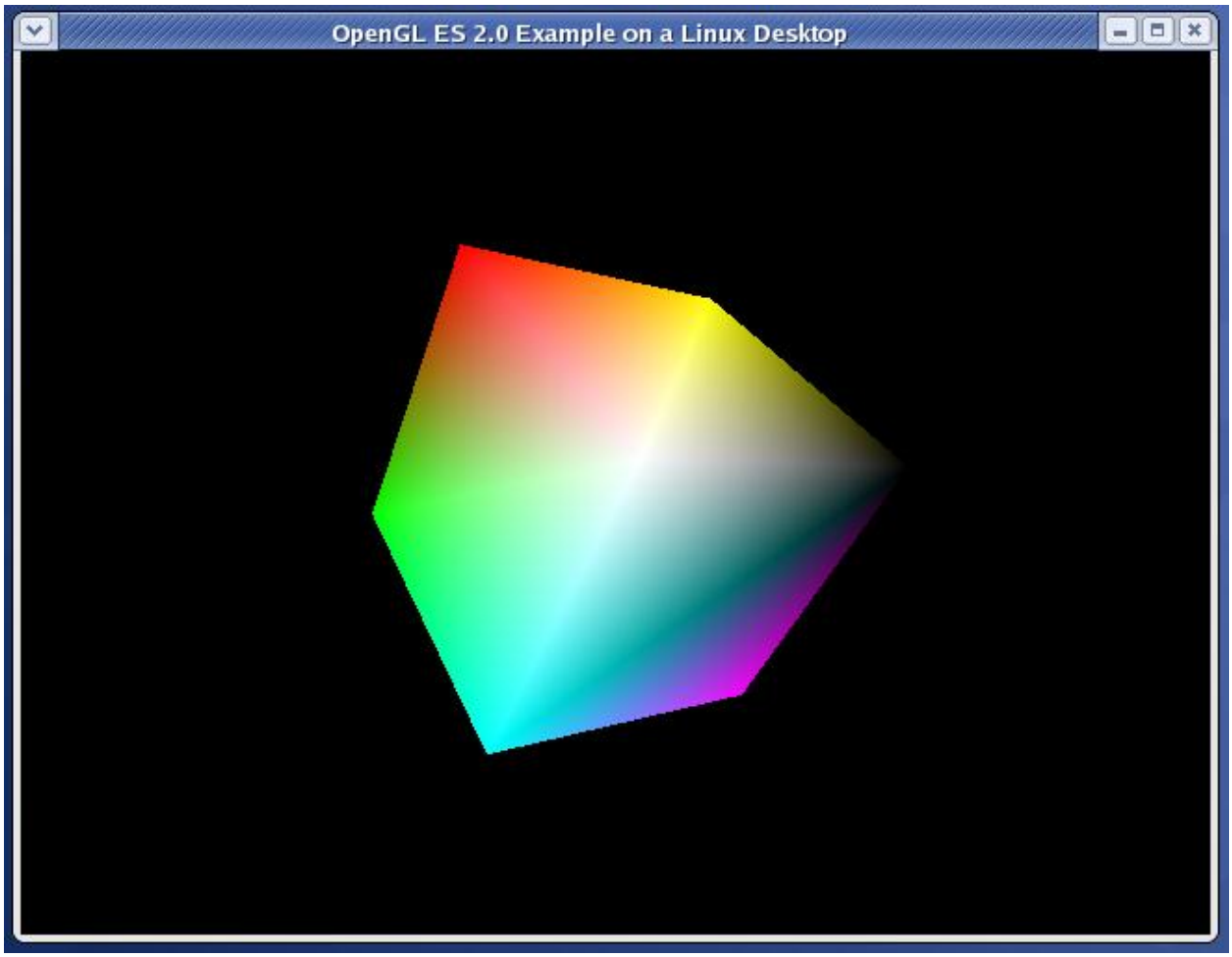
In order to verify the Mali OpenGL ES Emulator is correctly installed, you can use the supplied example project.

3.1.1 Linux

Assuming you have installed the headers and libraries to a system location, you should be able to build and run the example application by changing to the `examples/cube` directory and running `make`. Compilation should be successful, and the `cube` binary should be produced. If so, run the example application by executing:

```
./cube
```

You should see a spinning cube on the screen. If you do, the Mali OpenGL ES Emulator is correctly installed.



OpenGL ES Cube Application on Linux

3.1.2 Windows

3.1.2.1 Building example for 32-bit version of the Emulator

- Open a Visual Studio Command Prompt
- Change to the example code directory
- Run `nmake`

3.1.2.2 Building example for 64-bit version of the Emulator

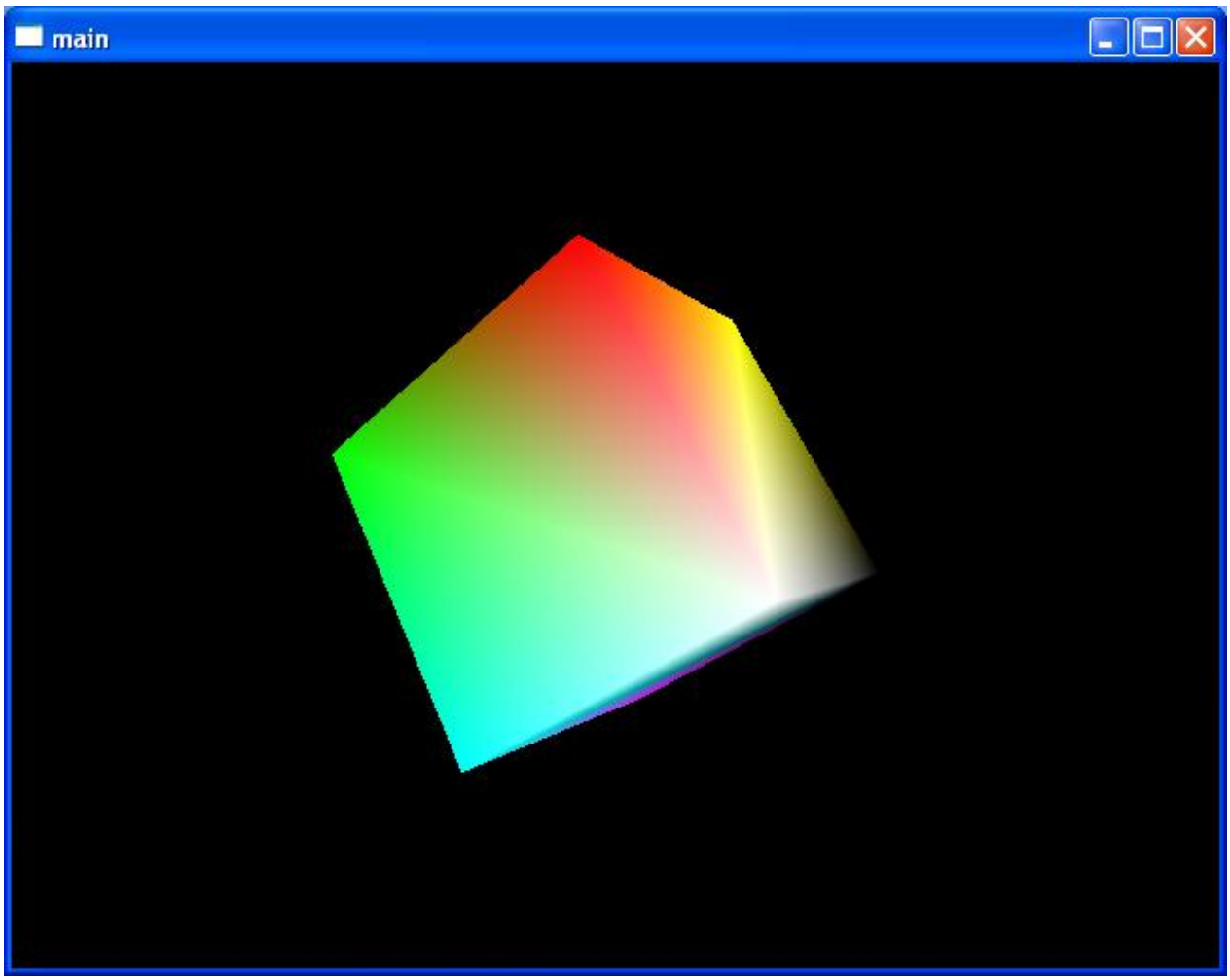
- Open command prompt `nmake`
- Execute script `C:\Program Files\Microsoft SDKs\Windows\<SDK_VERSION>\Bin\SetEnv.Cmd /x64`
- Change to the example code directory
- Run `nmake`

Note: `<SDK_VERSION>` indicates the exact version of Microsoft Windows SDK installed on your computer.

3.1.2.3 Running the example

A `cube.exe` file should be produced.

If you run this binary, you should see a spinning cube on the screen indicating that the Mali OpenGL ES Emulator is correctly installed.



OpenGL ES Cube Application on Windows

3.2 Using the Mali OpenGL ES Emulator in Your Application

3.2.1 Adding the Headers and Libraries to Your Project

To use the Mali OpenGL ES Emulator in your project, you must add the header files from the include directory (`EGL`, `GLES` etc) to the include path of your build. The headers expect to be referenced in relation to the `include` directory itself, so you should only add the `include` directory.

Similarly, you should add the library location to your build and specify the `libEGL` and `libGLESv2` libraries when linking. For example, using gcc:

```
gcc -o main -I/path/to/emulator/include -L/path/to/emulator/bin -lEGL -
lGLESv2 main.c
```

and in your code, you should reference the headers as follows:

```
#include <EGL/egl.h>
#include <GLES3/gl3.h>
```

Consult your compiler documentation for how to add headers and libraries on your system.

Note: `libEGL` and `libGLES` use the `__stdcall` calling convention.

3.2.2 Choosing an EGL Configuration

The Mali OpenGL ES Emulator supports both OpenGL ES 2.0 and 3.0 EGL configs. You should ensure that your application requests the correct type of config by passing the `EGL_RENDERABLE_TYPE` attribute to `eglChooseConfig`:

- To request an OpenGL ES 2.0 config, use `EGL_OPENGL_ES2_BIT`.
- To request an OpenGL ES 3.0 config, use `EGL_OPENGL_ES3_BIT_KHR`.

For example:

```
EGLDisplay display;

EGLint attributes[] = {
    EGL_RENDERABLE_TYPE, EGL_OPENGL_ES2_BIT,  // Request OpenGL ES 2.0
configs
    EGL_RED_SIZE, 8,
    EGL_GREEN_SIZE, 8,
    EGL_BLUE_SIZE, 8,
    EGL_NONE
};

EGLConfig configs[1];
EGLint num_configs;

eglChooseConfig(display, attributes, configs, 1, &num_configs);
```

You can also request that both types of configs are returned by bitwise-ORing the values: `EGL_OPENGL_ES2_BIT | EGL_OPENGL_ES3_BIT_KHR`.

It is possible to request EGL config either with no caveats (neither `EGL_SLOW_CONFIG` or `EGL_NON_CONFORMANT_CONFIG`) or regardless of any caveats by passing the `EGL_CONFORMANT` attribute to `eglChooseConfig`:

- To request an OpenGL ES 2.0 config with no caveats, use `EGL_OPENGL_ES2_BIT`.
- To request an OpenGL ES 3.0 config with no caveats, use `EGL_OPENGL_ES3_BIT_KHR`.
- To request an OpenGL ES 2.0 or 3.0 config regardless of any caveats, use `0`.

You can also bitwise-OR the values, as described above.

3.2.3 Creating an EGL Context

Similarly, `eglCreateContext` can create OpenGL ES 2.0 or 3.0 contexts. Pass the `EGL_CONTEXT_CLIENT_VERSION` attribute with an appropriate value:

```
EGLDisplay display;
EGLConfig  configs[1];
```

```
EGLint context_attributes[] = {
    EGL_CONTEXT_CLIENT_VERSION, 2,  // Select an OpenGL ES 2.0 context
    EGL_NONE
};

context = eglCreateContext(display, configs[0], EGL_NO_CONTEXT,
context_attributes);
```

4 About the OpenGL ES Implementation

The Mali OpenGL ES Emulator works by transparently converting all OpenGL ES API calls to appropriate sequences of OpenGL calls. These OpenGL calls are then handled by the native platform's graphics driver. OpenGL ES 3.0 calls are converted to OpenGL 3.3 calls, and OpenGL ES 2.0 calls are converted to OpenGL 2.0 calls.

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 behaviour of the graphics card driver differs from the OpenGL 3.3 specification for OpenGL ES 3.0 applications (or OpenGL 2.0 for OpenGL ES 2.0 applications).

4.1 OpenGL ES Conformance

This product is based on a published Khronos Specification, and is expected to pass the Khronos Conformance Testing Process when available. Current status can be found at www.khronos.org/conformance.

4.2 Support for OpenGL ES Extensions

For OpenGL ES 2.0 contexts, the Mali OpenGL ES Emulator supports the following extensions:

- `GL_ARM_rgba8`
- `GL_EXT_blend_minmax`
- `GL_EXT_discard_framebuffer`
- `GL_EXT_multisampled_render_to_texture`
- `GL_EXT_occlusion_query_boolean`
- `GL_EXT_read_format_bgra`
- `GL_EXT_texture_format_BGRA8888`
- `GL_EXT_texture_rg`
- `GL_EXT_texture_storage`
- `GL_EXT_texture_type_2_10_10_10_REV`
- `GL_OES_compressed_ETC1_RGB8_texture`
- `GL_OES_compressed_paletted_texture`
- `GL_OES_depth_texture`
- `GL_OES_depth24`
- `GL_OES_EGL_image`
- `GL_OES_EGL_image_external`
- `GL_OES_element_index_uint`
- `GL_OES_fbo_render_mipmap`
- `GL_OES_mapbuffer`
- `GL_OES_packed_depth_stencil`
- `GL_OES_read_format`
- `GL_OES_required_internalformat`
- `GL_OES_rgb8_rgba8`
- `GL_OES_standard_derivatives`
- `GL_OES_texture_3D`
- `GL_OES_texture_npot`
- `GL_OES_vertex_array_object`
- `GL_OES_vertex_half_float`

For OpenGL ES 3.0 contexts, the Mali OpenGL ES Emulator supports the following extensions:

- `GL_ARM_rgba8`

- GL_EXT_blend_minmax
- GL_EXT_discard_framebuffer
- GL_EXT_multisampled_render_to_texture
- GL_EXT_occlusion_query_boolean
- GL_EXT_read_format_bgra
- GL_EXT_texture_format_BGRA8888
- GL_EXT_texture_rg
- GL_EXT_texture_storage
- GL_EXT_texture_type_2_10_10_10_REV
- GL_KHR_texture_compression_astc_ldr
- GL_OES_compressed_ETC1_RGB8_texture
- GL_OES_compressed_paletted_texture
- GL_OES_depth_texture
- GL_OES_depth24
- GL_OES_element_index_uint
- GL_OES_mapbuffer
- GL_OES_read_format
- GL_OES_required_internalformat
- GL_OES_rgb8_rgba8
- GL_OES_texture_3D
- GL_OES_texture_npot
- GL_OES_vertex_array_object
- GL_OES_vertex_half_float

4.3 Shading Language Version

For OpenGL ES 2.0 contexts, the Mali OpenGL ES Emulator supports up to version 1.2 of the OpenGL Shader Language. For OpenGL ES 3.0 Contexts, OpenGL Shader Language version 3.3 is supported.

4.4 Limitations

4.4.1 Implementation-specific behaviour

Where the OpenGL ES 3.0 or OpenGL ES 2.0 specifications permit implementation-specific behaviour, the behaviour is usually determined by the underlying driver. The behaviour of the graphics card drivers can differ from the behaviour of Mali drivers and hardware. This includes implementation-dependent limits, for example:

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

The following properties are enforced to mimic Mali driver behaviour:

- GL_MAX_RENDERBUFFER_SIZE is 4096 (for both OpenGL ES 2.0 and 3.0)
- GL_MAX_UNIFORM_BLOCK_SIZE is 16384 (for OpenGL ES 3.0 contexts only)

4.4.2 `glProgramBinary` always fails

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

4.4.3 `glShaderBinary` always fails

Because of the incompatibility between binary formats for different graphics drivers, 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.

4.4.4 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 Mali OpenGL ES Emulator converts fixed-point data and passes it to OpenGL. For the Mali 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 Mali 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.

4.4.5 `gIGetShaderPrecisionFormat` 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

4.4.6 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`
- `GL_ETC1_RGB8_OES`

4.4.7 ASTC format support

The following ASTC formats are supported in OpenGL ES 3.0:

- `GL_COMPRESSED_RGBA_ASTC_4x4_KHR`
- `GL_COMPRESSED_RGBA_ASTC_5x4_KHR`
- `GL_COMPRESSED_RGBA_ASTC_5x5_KHR`
- `GL_COMPRESSED_RGBA_ASTC_6x5_KHR`
- `GL_COMPRESSED_RGBA_ASTC_6x6_KHR`

- `GL_COMPRESSED_RGBA_ASTC_8x5_KHR`
- `GL_COMPRESSED_RGBA_ASTC_8x6_KHR`
- `GL_COMPRESSED_RGBA_ASTC_8x8_KHR`
- `GL_COMPRESSED_RGBA_ASTC_10x5_KHR`
- `GL_COMPRESSED_RGBA_ASTC_10x6_KHR`
- `GL_COMPRESSED_RGBA_ASTC_10x8_KHR`
- `GL_COMPRESSED_RGBA_ASTC_10x10_KHR`
- `GL_COMPRESSED_RGBA_ASTC_12x10_KHR`
- `GL_COMPRESSED_RGBA_ASTC_12x12_KHR`

which are stored in `GL_RGBA + GL_RGBA + GL_FLOAT` internal format and

- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_4x4_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_5x4_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_5x5_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_6x5_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_6x6_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_8x5_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_8x6_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_8x8_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_10x5_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_10x6_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_10x8_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_10x10_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_12x10_KHR`
- `GL_COMPRESSED_SRGB8_ALPHA8_ASTC_12x12_KHR`

which are stored in `GL_RGBA + GL_SRGB_ALPHA + GL_FLOAT` internal format.

Internally, the textures are decompressed to unsigned byte format (the number of components depends on the compression algorithm in question) or floating-point format.

5 About the EGL Implementation

The EGL implementation supports both OpenGL ES 2.0 and 3.0. OpenVG is not supported.

5.1 Synchronisation of pixmap surfaces

Pixmap surfaces are supported through the use of graphics driver pbuffers. You must use the appropriate EGL synchronisation calls to get OpenGL ES 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 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 renders to the pbuffer.

5.2 Limitations

The EGL library is a limited implementation of EGL that suffices to allow the Mali OpenGL ES Emulator to pass the Khronos Conformance Test Suite. As such, there are some limitations.

5.2.1 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_ES3_BIT_KHR`.

Context creation fails unless `EGL_CONTEXT_CLIENT_VERSION` is set to 2 or 3.

5.2.2 Multiple threads and multiple contexts

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

5.2.3 Window pixel format

You must set pixel format only through `eglCreateWindowSurface`.

5.2.4 Limited bitmap support

Bitmap rendering only works correctly for uncompressed, bottom-up, 32-bit RGB bitmaps.

5.2.5 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`.

5.2.6 No support for swap intervals

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

5.2.7 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.

5.2.8 Use of displays following `eglTerminate`

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

5.2.9 `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 EGL config 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 EGL config is desired (to ensure the same colour precision as the native pixmap), then `EGL_RED_SIZE`, `EGL_GREEN_SIZE` and `EGL_BLUE_SIZE` should be explicitly passed to `eglChooseConfig`.

5.2.10 Resizing a native window

Resizing a native window does not update the surface attributes.

5.2.11 `eglChooseConfig` always selects double-buffered configs

The EGL attribute list is translated to an attribute list for the underlying window system. This attribute list always has the double-buffering attribute set to true. This means that some available matching configurations might not be returned.

6 Known Issues and Limitations

6.1 Mac OS X and Mesa3D are not supported

Neither Mac OS X nor Mesa3D support OpenGL 3.3 and hence are not compatible with the Mali OpenGL ES Emulator.

6.2 `glFinish` calls block for one second

Calls to `glFinish` will block the calling thread for one second and are discouraged.

6.3 Support for non-NVIDIA drivers is preliminary

ARM recommends using the Mali OpenGL ES Emulator with NVIDIA drivers. We are aware of many issues when running the Emulator with ATI/AMD drivers and are working on improving the support. Other drivers are not tested.

6.4 The value of `gl_InstanceID` is always zero when `glDrawArraysInstanced` is called from within a transform feedback block

On certain models of host GPU the value of `gl_InstanceID` will always be zero rather than the ID of the instance currently being drawn. This is due to a driver issue. The known affected GPUs and drivers are:

- NVIDIA GeForce 210, driver version 301.42
- NVIDIA GeForce 210, beta driver 304.79
- NVIDIA NVS 300, beta driver 304.79

6.5 Installation with non-admin user rights on Windows

When OpenGL ES 3.0 Emulator is installed on Windows using non-admin user rights account, the operating system asks for providing administrator account credentials. When such are given installation process continues with installing examples code within given administrator’s account Documents folder instead of original user’s one.

7 Changes from Previous Versions

7.1 Changes in version 1.3.1

- Various improvements and bug fixes.

7.2 Changes in version 1.3.0

- Support for additional OpenGL ES extensions:
- Improved support of existing OpenGL ES extensions under OpenGL ES 2.0 and 3.0 contexts.
- Improved handling of depth textures.
- Various improvements and bug fixes.

7.3 Changes in version 1.2.0

- Support for `GL_KHR_texture_compression_astc_ldr`.
- Support for several new OpenGL ES extensions.
- `GL_OES_read_format` support is no longer reported as it defines a core functionality of both OpenGL ES 2.0 and OpenGL ES 3.0 implementations.
- Shader verification was improved under Linux.
- Fixed a problem occurring under specific conditions that caused console windows to quickly become visible and disappear when compiling shaders under Windows.
- Fixed `glRenderbufferStorageMultisample/glTexImage2D/glVertexAttribPointer` problem occurring with an AMD driver.

7.4 Changes in version 1.1.0

- Support for context sharing.
- Support for ETC1 textures.
- The OpenGL ES 3.0 Emulator is now called the Mali OpenGL ES Emulator and supports both OpenGL ES 3.0 and OpenGL ES 2.0 contexts.
- Improved ESSL error detection.
- Improved texture format/internal format/type support.
- Improved generate-upon-binding support.
- Fixes for various memory and resource leaks.
- Various other bug-fixes and improvements.

7.5 Changes in version 1.0.0

- This version is the first release of the Mali OpenGL ES Emulator.

8 Support

For support on this product, please visit the Mali Developer Centre.

8.1 Continued Support

It should be noted that continuing support of the product will only be provided by ARM if such support is covered by a current contract with the recipient.

9 Legal

9.1 Proprietary Notice