Mali OpenGL ES Emulator v3.0.2
User Guide for Windows

Notice

Proprietary Notice

Contents

Introduction

The Mali OpenGL ES Emulator allows you to use the OpenGL ES 2.0, 3.0, 3.1 and 3.2 APIs to render 3D graphics content on a desktop PC.

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.

The Mali OpenGL ES Emulator is available for both Windows and Linux, and comes in both 32 and 64 bit variants.

This document describes how to install and use the Mali OpenGL ES Emulator on **Windows** systems.

Chapter 1

Installation and Configuration

Topics:

The installation package for the Mali OpenGL ES Emulator contains everything you need to get started building OpenGL ES 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 APIs.

You should have downloaded the installation package for Windows. The package should closely resemble your host environment. If you aren't sure which package to choose, the 32 bit version will work across the widest variety of platforms.

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

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

Minimum Requirements

In general, the Mali OpenGL ES Emulator will work on any system that supports at least:

- OpenGL 3.2 – when OpenGL ES 2.0 contexts are used[1]
- OpenGL 3.3 – when OpenGL ES 3.0 contexts are used
- OpenGL 4.3 – when OpenGL ES 3.1 or 3.2 contexts are used[2]

On all systems, up-to-date operating system components and graphics drivers are recommended.

We recommend using the Mali OpenGL ES Emulator with NVIDIA drivers. At the date of this release the reference driver version is 344.60 or higher.

We are aware of various issues when running the Emulator with ATI/AMD drivers and are working on improving the support. Detailed interoperability with other drivers was not tested.

Installation Using Installer

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.

Note: By default, the installer will create two extra environment variables in the current user profile: *KHRONOS_HEADERS* indicates location of include headers installed and *OPENGLES_LIBDIR* that indicates location of where OpenGL ES libraries of the emulator are installed. These might be helpful for using them within a build environment.

Installation from ZIP Package

As alternative, the Mali OpenGL ES Emulator is provided as a `.zip` package that can be unpacked to any location on disk, but in this case you will be required to manually add the libraries and headers to your build and runtime environment.

Once unpacked, you can run `mali-cube.exe` application straight from directory with the Emulator binaries.

For example, if the OpenGL ES Emulator ZIP package has been extracted to `C:\Mali_OpenGL_ES_Emulator` directory, you can use a command line to run mali-cube from within any other directory the following way:

```
> set EMULATOR_PATH=C:\Mali_OpenGL_ES_Emulator
> set PATH=%PATH%;%EMULATOR_PATH%
> %EMULATOR_PATH%\mali-cube.exe
```

The same set up applies when running custom applications linked against the Emulator.

Installation Results

Both the installer and ZIP packages provide the following file structure:

```
C:\Program Files\ARM\Mali Developer Tools\Mali OpenGL ES Emulator 3.0.2
  +-mali-cube.exe
  +-mali-checker.exe
  +-libGLESv2.dll
  +-libGLESv2.lib
  +-libEGL.dll
```

[1] `ARB_sampler_objects` also required, otherwise OpenGL 3.3.
[2] To use `GL_PRIMITIVE_RESTART_FOR_PATCHES_SUPPORTED` OpenGL 4.4 is required.

```
+-libEGL.lib
+-libMaliEmulator.dll
+-libMaliEmulator.lib
+-log4cplus.dll
+-msvcp110.dll
+-msvcp120.dll
+-msvcr110.dll
+-msvcr120.dll
+-include
|  +-EGL
|  +-GLES2
|  +-GLES3
|  +-KHR
+-openglessl
|  +-Mali-T600_r7p0-00rel0.dll
+-EULA.txt
+-LICENSES.txt
+-Mali OpenGL ES Emulator v3.0.2 User Guide for Windows.pdf
```

Verifying the Installation

Run Mali Cube application.

You can locate it in **Windows Start** > **All Programs** > **ARM** > **Mali OpenGL ES Emulator** > **Mali Cube**

You should see a Mali Cube application running indicating that the Mali OpenGL ES Emulator is correctly installed.

Figure 1: Screenshot of the Mali Cube Application

Chapter 2

Using with Your Application

Topics:

This chapter describes some basic preparations to have your OpenGL ES application working with Mali OpenGL ES Emulator.

Compiling Your Project

The following instruction assumes you are compiling your OpenGL ES application on Windows desktop using Microsoft Visual Studio 2013 (or newer) compiler [3] appropriate to the bitness of the installed version of Mali OpenGL ES Emulator.

The default OpenGL ES Emulator installation creates two environment variables `KHRONOS_HEADERS` and `OPENGLES_LIBDIR` that contain locations of include headers and emulator libraries respectively. For example, you should be able to build your application, using `cl.exe`:

```
cl.exe /I"%KHRONOS_HEADERS%" main.c "%OPENGLES_LIBDIR%/libEGL.lib"
 "%OPENGLES_LIBDIR%/libGLESv2.lib"
```

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

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

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

Choosing an EGL Configuration

The Mali OpenGL ES Emulator supports OpenGL ES 2.0, 3.0, 3.1 and 3.2 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, 3.1 or 3.2 config, use `EGL_OPENGL_ES3_BIT_KHR`.

For example:

```
EGLDisplay display;
EGLint attributes[] =
        {
            // Request OpenGL ES 2.0 configs
            EGL_RENDERABLE_TYPE, EGL_OPENGL_ES2_BIT,
            EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
            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, 3.1 or 3.2 config with no caveats, use `EGL_OPENGL_ES3_BIT_KHR`.

[3] If you are using other compiler, consult its documentation for how to add headers and libraries on your system.

- To request an OpenGL ES 2.0, 3.0, 3.1 or 3.2 config regardless of any caveats, use 0. You can also bitwise-OR the values, as described above.

Creating an EGL Context

Similarly, `eglCreateContext` can create OpenGL ES 2.0, 3.0, 3.1 or 3.2 contexts.

For ES 2.0 or 3.0 contexts, pass the `EGL_CONTEXT_CLIENT_VERSION` attribute with an appropriate value:

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

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);
```

For ES 3.1 or 3.2 contexts pass pair of `EGL_CONTEXT_MAJOR_VERSION` and `EGL_CONTEXT_MINOR_VERSION` attributes with appropriate value:

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

EGLint context_attributes[] = {     EGL_CONTEXT_MAJOR_VERSION, 3,
        EGL_CONTEXT_MINOR_VERSION, 1,
        EGL_NONE };

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

Tessellation Extension

Because of a small inconsistency in Khronos extension definitions, there is one thing worth to pay attention to when using **GL_EXT_tessellation_shader** extensions.

This extension is available with GLES 3.1 and 3.2 version contexts. However specific entry points and enums are defined in `gles2ext.h` header file. Therefore when using this extension the include section in you applications should contain both files:

```
#include <GLES2/gl2ext.h>
#include <GLES3/gl31.h>
```

Chapter 3

OpenGL ES Implementation

Topics:

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 calls are converted to OpenGL 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 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:

- OpenGL 3.2 specification plus ARB_sampler_objects extension or OpenGL 3.3 for OpenGL ES 2.0 contexts
- OpenGL 3.3 specification for OpenGL ES 3.0 contexts
- OpenGL 4.3 specification for OpenGL ES 3.1 contexts
- OpenGL 4.3 specification for OpenGL ES 3.2 contexts

OpenGL ES Extensions

The table below lists all OpenGL ES extensions supported by the OpenGL ES Emulator in various context types.

Table 1: OpenGL ES Extensions Supported by various context types.

Extension Name	2.0	3.0	3.1	3.2
GL_ARM_rgba8	X	X	X	X
GL_EXT_blend_minmax	X	X	X	X
GL_EXT_discard_framebuffer	X	X	X	X
GL_EXT_multisampled_render_to_texture	X	X	X	X
GL_EXT_occlusion_query_boolean	X	X	X	X
GL_EXT_read_format_bgra	X	X	X	X
GL_EXT_texture_format_BGRA8888	X	X	X	X
GL_EXT_texture_rg	X	X	X	X
GL_EXT_texture_storage	X	X	X	X
GL_EXT_texture_type_2_10_10_10_REV	X	X	X	X
GL_OES_compressed_ETC1_RGB8_texture	X	X	X	X
GL_OES_compressed_paletted_texture	X	X	X	X
GL_OES_depth_texture	X	X	X	X
GL_OES_depth24	X	X	X	X
GL_OES_EGL_image	X	X	X	X
GL_OES_EGL_image_external	X	X	X	X
GL_OES_element_index_uint	X	X	X	X
GL_OES_fbo_render_mipmap	X	X	X	X
GL_OES_mapbuffer	X	X	X	X
GL_OES_packed_depth_stencil	X	X	X	X
GL_OES_read_format	X	X	X	X
GL_OES_required_internalformat	X	X	X	X
GL_OES_rgb8_rgba8	X	X	X	X
GL_OES_standard_derivatives	X	X	X	X
GL_OES_texture_3D	X	X	X	X
GL_OES_texture_npot	X	X	X	X
GL_OES_vertex_array_object	X	X	X	X
GL_OES_vertex_half_float	X	X	X	X
GL_KHR_debug	X	X	X	X
GL_EXT_shader_io_blocks			X	X
GL_EXT_gpu_shader5			X	X
GL_EXT_geometry_shader			X	X
GL_EXT_tessellation_shader			X	X
GL_KHR_blend_equation_advanced			X	X

Extension Name	2.0	3.0	3.1	3.2
GL_OES_sample_variables			X	X
GL_OES_sample_shading			X	X
GL_EXT_primitive_bounding_box			X	X
GL_EXT_draw_buffers_indexed			X	X
GL_EXT_copy_image			X	X
GL_OES_texture_storage_multisample_2d_array			X	X
GL_OES_shader_multisample_interpolation			X	X
GL_OES_shader_image_atomic			X	X
GL_EXT_texture_border_clamp			X	X
GL_EXT_texture_buffer			X	X
GL_OES_texture_stencil8			X	X
GL_EXT_texture_cube_map_array			X	X

Shading Language Version

For OpenGL ES 2.0 contexts, the Mali OpenGL ES Emulator supports up to version 1.2 of the OpenGL ES Shader Language.

For OpenGL ES 3.0 contexts, OpenGL ES Shader Language version 3.0 is supported. For OpenGL ES 3.1 contexts, OpenGL ES Shader Language 3.1 is supported. For OpenGL ES 3.2 contexts, OpenGL ES Shader Language 3.2 is supported.

Compressed Texture Formats

The emulator supports (and reports support for) the following compressed texture formats for OpenGL ES 3.0, 3.1 and 3.2:

- 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

ASTC Format Support

The following ASTC formats are supported in OpenGL ES 3.0, 3.1 and 3.2:

- 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.

KHR Debug Extension

Scope

Debug messages are related with OpenGL ES emulation sub-system. EGL part is not covered by debug messages reported by implementation of this extension.

Messages

Debug messages are consistent to various error, warning and information messages the Emulator reports to the user console in the run-time. By using KHR_Debug interface, all internal messages are reported as GL_DEBUG_TYPE_ERROR_KHR type and as GL_DEBUG_SOURCE_API source. What differentiates them is their severity so that:

- FATAL ERROR level emulator messages have severity set to GL_DEBUG_SEVERITY_HIGH_KHR
- ERROR level emulator messages have severity set to GL_DEBUG_SEVERITY_MEDIUM_KHR
- WARNING level emulator messages and ERROR messages have severity set to GL_DEBUG_SEVERITY_LOW_KHR
- INFO level emulator messages and ERROR messages have severity set to GL_DEBUG_SEVERITY_NOTIFICATION_KHR

Debug Groups

OpenGL ES Emulator doesn't support more than one default debug message group. Calling PopDebugGroup() and PushDebugGroup(...) produces GL_STACK_UNDERFLOW and GL_STACK_OVERFLOW errors respectively.

Value of `GL_DEBUG_GROUP_STACK_DEPTH` is always 1.

Asynchronous and Synchronous Debug Output

Only asynchronous message callback is supported. Enabling synchronous callback with `GL_DEBUG_OUTPUT_SYNCHRONOUS` flag gives no effect and produces `GL_INVALID_OPERATION` error.

Debug Labels

At the moment there is no support for sync object labels. Calling `ObjectPtrLabel(...)` and `GetObjectPtrLabel(...)` generates `GL_INVALID_OPERATION` error.

Interaction with GL side of KHR_Debug

No interoperation with `KHR_Debug` that is provided by the underlying OpenGL driver. That means any debug message or label queries from/to GL are not reachable by the application.

Additional Considerations

Implementation dependent constants have values of:

- `GL_MAX_DEBUG_MESSAGE_LENGTH` is set to 512
- `GL_MAX_DEBUG_LOGGED_MESSAGES` is set to 2048
- `GL_MAX_LABEL_LENGTH` value will be arbitrarily set to 256.

Emulator offers `KHR_Debug` support for any context type - also those context types created without `EGL_CONTEXT_OPENGL_DEBUG_BIT_KHR` flag. `glGetInteger(...)` called with `GL_CONTEXT_FLAG_DEBUG_BIT_KHR` always returns `GL_TRUE`.

Message Identifiers

All debug messages from the Emulator have values in range 0x2000 to 0x4000.

Limitations

ES 3.1 Support

The following features are known to have limited or non-conformant behaviour:

- There is no exact support for `glMemoryBarrierByRegion` entrypoint implementation as defined in OpenGL ES 3.1 specification. The entrypoint function is available in the Emulator but acts exactly as `glMemoryBarrier` function.
- Usage of built-in mix function in shaders when uvec4 type values are used as arguments,
- `GL_TRANSFORM_FEEDBACK_VARYING` passed as programInterface parameter as per specification.
- Using built-in special fragment shader variable `gl_HelperInvocation` is not supported.
- When extension `GL_EXT_shader_io_blocks` is enabled and separable programs are used, precision qualifiers for variables in the interface blocks are omitted from checking input-output interface matching in program pipelines.

Implementation-specific Behaviour

Where the OpenGL ES 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 OpenGL ES 2.0, 3.0 and 3.1 contexts)
- `GL_MAX_UNIFORM_BLOCK_SIZE` is 16384 (for OpenGL ES 3.0 and 3.1 contexts only)

`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 drivers, 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.

`glGetShaderPrecisionFormat` Values

For OpenGL ES 3.0 and 3.1 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

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 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 emulator must convert a client-side vertex array on each draw call because the client application might modify the data between draw calls.

`glFinish` Calls Block for One Second

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

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.

The Value Of `gl_Instanceid` Is Always Zero When `gl_DrawArraysInstanced` 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

Support for 2D Depth Textures

The implementation of support for 2D depth textures have limitations and may not provide fully conformant behaviour where depth textures are used with mipmap levels greater than zero (base) level.

`GL_KHR_blend_equation_advanced` Extension Availability

Extension `GL_KHR_blend_equation_advanced` is fully conformant only on NVIDIA graphics cards.

Limited set of number of samples for `glTexStorage2DMultisample`.

The `glTexStorage2DMultisample` function will round-up number of samples passed as `samples` argument. The rounding will be to closest (not smaller) value from the set of 1,2,4 or 8 samples.

Chapter 4

EGL Implementation

Topics:

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.

Mali OpenGL ES Emulator comes with support for EGL 1.4 API version plus several EGL extensions as described later in this chapter.

The EGL implementation supports OpenGL ES API only. The EGL library does not support OpenVG. Neither graphics contexts nor surfaces created can be used 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 context version is set to 2, 3, 3.1 or 3.2 using `EGL_CONTEXT_CLIENT_VERSION` attribute or `EGL_CONTEXT_MAJOR_VERSION` / `EGL_CONTEXT_MINOR_VERSION` attribute pair respectively.

`elgGetProcAddress` function returns pointer to any EGL and GL ES entrypoint provided by the Emulator, not just an extension functions.

EGL Extensions

The Mali OpenGL ES Emulator supports the following EGL extensions:

- `EGL_KHR_create_context`
- `EGL_KHR_config_attribs`
- `EGL_KHR_image_base`
- `EGL_KHR_image`
- `EGL_KHR_image_pixmap`
- `EGL_KHR_gl_renderbuffer_image`
- `EGL_KHR_gl_texture_2D_image`
- `EGL_KHR_gl_texture_cubemap_image`

Limitations

Multiple Threads and Multiple Contexts

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

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 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 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`.

Resizing a Native Window

Resizing a native window after `glMakeCurernt` function call does not update the surface attributes.

`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.

Chapter 5

Mali Checker Application

Topics:

OpenGL ES Emulator is shipped with diagnostic application `mali-checker`. The tool can be used to diagnose configuration of the OS installed on user's machine with respect to emulator's requirements.

Performing a Check

Mali checker can be executed by launching executable:

```
mali-checker.exe
```

The application will produce and open HTML report with examination results.

Report File

The report produced by the checking application is made of three main sections:

- Overview
- Examinations
- Features and Extensions

Overview Section

This section contains information that identify the report and the operating system the examinations has been performed on.

Examinations Section

This section contains list and details of examinations that have been performed by the application.

Features and Extensions Section

This section lists all features and extensions the emulator supports.

Note: this section might not be present in the report if producing such is not possible due to blocking problems detected by examinations.

Command-line Parameters

When launched without parameters, the `mali-checker` application will create and display HTML report with all built-in checks performed.

Controlling Checks

It's possible to have a fine-grain control over number of checks the application performs with the following command-line paramters.

-a, --all

Perform all checks. Default if no individual checks are specified.

-e, --gles-and-egl

Check if libGLESv2 and libEGL are loaded and correct versions.

-E, --environment

Check the enviroment variables.

-g, --gl

Check if libGL is loaded and is suitable.

-H, --header

Check if the default headers are compatible with ours.

-m, --moc

Check if the Mali Offline Compiler is available.

Controlling Report Output

The following arguments can be used to control behaviour of the application related to producing output.

-n, --no-open

Do not open then report afterwards.

-o=<filename>, --output=<filename>

Write report to a <filename>.

Miscellaneous

Other parameters:

--, --ignore_rest

Ignores the rest of the labeled arguments following this flag.

--version

Displays version information and exits.

-h, --help

Displays usage information and exits.

Chapter 6

Maintenance

Topics:

This chapter contains topics related to maintenance, troubleshooting and support for the OpenGL ES Emulator application installed on the system.

Troubleshooting

Any erroneous situation leading to improper behaviour of installed OpenGL ES Emulator should leave error message on the application console window.

Please refer to these messages when obtaining help from technical support.

Controlling Log Verbosity

You can control level of log verbosity by setting the *MALI_EMULATOR_APP_VERBOSITY* environment variable in your system. You can set it to one of the following values:

OFF	No errors will be reported to the console.
FATAL	Will report only fatal-class error events.
ERROR	Will report regular errors and fatal error events.
WARNING	Will report warnings, regular errors and fatal error events.

If ths environment variable is unset, the emulator uses **FATAL** as the default log verbosity level.

Uninstalling Using Control Panel

Navigate to Windows Control Panel, open Uninstall a Program window and select Mali OpenGL ES Emulator on the list. Click Uninstall/Change option and follow the instructions from the wizard.

Support

For further support on this product, please visit the *ARM Connected Community*.

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.

Changes from Previous Versions

Changes In Version 3.0.2

- Missing support for GL_TEXTURE_2D_MULTISAMPLE for GL ES 3.1 added.
- GL_DITHER is an enabled state by default.
- Making glGetStringi and glGetIntegerv report extensions strings in a consitent way
- Improving performance of `glFinish()` implementation
- GL_TEXTURE_2D supported for GLES3.0 contexts

Changes In Version 3.0.1

- Fixes in accessing 3.2 API entrypoints and 3.2 specific enums.

Changes In Version 3.0

- Added support for OpenGL ES 3.2 API version.
- Improved interoperability with AMD/ATI graphics cards.

Changes In Version 2.3.0

- Mali-checker application introduced.
- Improved interoperability with AMD/ATI graphics cards.
- Installers are signed with digital certificate.

Changes In Version 2.2.1

- Using latest Mali Offline Compiler with support for OpenGL SL 310 es version of shader language.
- Fixed problem with incorrect log content on successful OpenGL SL 310 es shader compilation.
- Improving `mix(...)` built-in functions support in OpenGL SL 310 es shaders.
- Fixed problem with mali-cube window closing.

Changes In Version 2.2

- Extensions from Android Extension Pack supported.
- `eglGetProcAddress` gives pointers to any of all implemented entrypoints.

Changes In Version 2.1.1

- Fixed problem with MSAA using EGL configurations.

Changes In Version 2.1

- Added support for the following OpenGL ES extensions:
  - `GL_EXT_shader_io_blocks`
  - `GL_EXT_gpu_shader5`
  - `GL_EXT_geometry_shader/`
  - `GL_EXT_tessellation_shader`

Changes In Version 2.0

- Added support for OpenGL ES 3.1 API version.
- Extensions `GL_OES_EGL_image` and `GL_OES_EGL_image_external` are supported for GLES 3.0 contexts.

Changes In Version 1.4.1

- Buffer operations with `GL_PIXEL_UNPACK_BUFFER` binding point and compressed textures fixed.
- Implementation of `glMapBufferOES` improved.
- DEB package installation on Ubuntu 14.04 free from warning messages.
- Fixes in implementation of `EGLImage` support and `glEGLImageTargetTexture2DOES` implementation.
- Fixed problems with `glViewport` not updating resolution when re-using same context with various EGL surfaces.

Changes In Version 1.4.0

- Single library containing complete EGL/OpenGL ES emulation code for improving library loading scenarios.
- Improvements on how textures are working in various use scenarios.
- Shader source processing is reflecting extension support correctly.
- Providing mali-cube executable for installation verification.

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.

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.

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.

Changes In Version 1.0.0

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