Generic Graphics Accelerator

Version 1.0

User Guide

ARM 100534_0100_01_en

Generic Graphics Accelerator

User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
0100-00	29 February 2016	Non-Confidential	First release
0100-01	31 May 2016	Non-Confidential	Second release

Non-Confidential Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

http://www.arm.com

Contents

Generic Graphics Accelerator User Guide

Preface

About this book 6

Chapter 1 Introduction to the Generic Graphics Accelerator

1.1 Overview 1-9
1.2 Background 1-10
1.3 Technical implementation 1-11
1.4 Prerequisites 1-13
1.5 Limitations 1-14

Chapter 2 Getting started with an Android target

2.1 Customizing Android and Fast Models to support the Generic Graphics Accelerator 2-16

Chapter 3 Parameters and configuration

3.1 Command-line examples for booting Android 3-22
3.2 Configuration 3-24

Chapter 4 Debugging

4.1 Debugging workflow 4-27
4.2 API tracing debugging tool 4-28
4.3 Reporting bugs in the Generic Graphics Accelerator 4-29

Preface

This preface introduces the *Generic Graphics Accelerator User Guide*.

It contains the following:

- *About this book* on page 6.

About this book

This document is the user guide for the Generic Graphics Accelerator, which enables Fast Models to provide API-level support for OpenGL ES 2.0.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction to the Generic Graphics Accelerator
This chapter provides a general introduction to the Generic Graphics Accelerator.

Chapter 2 Getting started with an Android target
This section describes how to enable the Generic Graphics Accelerator to support Android on Fast Models.

Chapter 3 Parameters and configuration
This section describes the parameters that you can use to configure and run the Generic Graphics Accelerator.

Chapter 4 Debugging
This section describes the debugging workflow, debugging tools, and how to report bugs.

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.

See the *ARM Glossary* for more information.

Typographic conventions

italic
Introduces special terminology, denotes 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`
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`
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>`
Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS
Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

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 *Generic Graphics Accelerator User Guide*.
- The number ARM 100534_0100_01_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *ARM Information Center*.
- *ARM Technical Support Knowledge Articles*.
- *Support and Maintenance*.
- *ARM Glossary*.

Chapter 1
Introduction to the Generic Graphics Accelerator

This chapter provides a general introduction to the Generic Graphics Accelerator.

It contains the following sections:

- *1.1 Overview* on page 1-9.
- *1.2 Background* on page 1-10.
- *1.3 Technical implementation* on page 1-11.
- *1.4 Prerequisites* on page 1-13.
- *1.5 Limitations* on page 1-14.

1.1 Overview

The Generic Graphics Accelerator uses the GPU on the host to provide API level support for OpenGL ES 2.0 on the target, and to simulate and accelerate code execution on the host GPU.

The focus of the Generic Graphics Accelerator is to support software integration at the subsystem or SoC level. You can use the Generic Graphics Accelerator to develop and run software applications that use OpenGL ES 2.0 APIs. This document assumes that you are familiar with basic 3D software development.

Related information

OpenGL ES.

1.2 Background

The Generic Graphics Accelerator allows an OS and user-space applications that require OpenGL ES 2.0 support to run on a Fast Models platform by providing API-level support. Currently, only Android OS is supported.

The Generic Graphics Accelerator does not provide any hardware interface or replace any existing GPU driver software stack on the target. That is, it is not useful for development or integration of a driver stack.

1.3 Technical implementation

To support software development for applications using OpenGL ES APIs, the Generic Graphics Accelerator must simulate the OpenGL ES APIs on the target. The Generic Graphics Accelerator uses a similar GPU from the ARM® Mali™ GPU family to do the emulation and to map the OpenGL ES APIs from the target to the host.

The Generic Graphics Accelerator uses the Mali OpenGL ES Emulator to simulate OpenGL ES APIs, and maps the OpenGL ES APIs in the target to the OpenGL ES APIs of the OpenGL ES Emulator.

The following figure shows how the mapping is implemented in Fast Models.

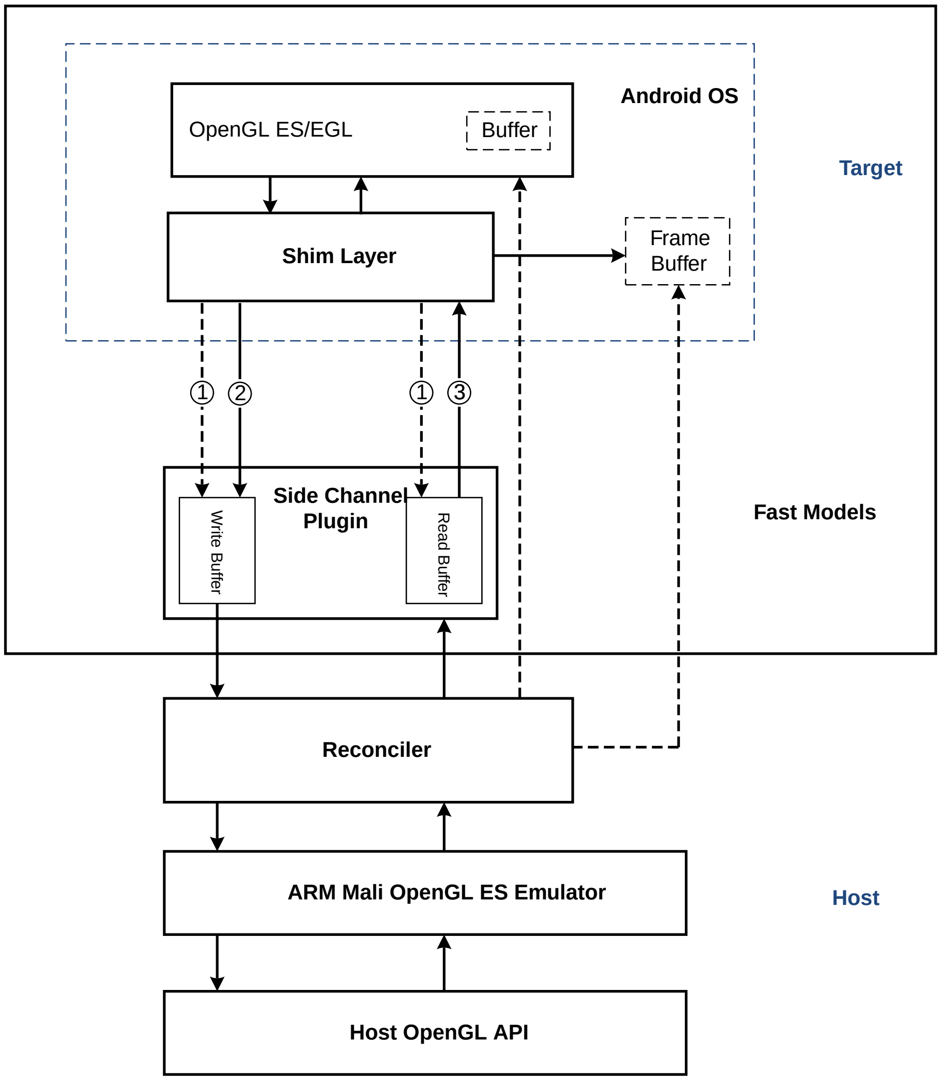

Figure 1-1 Generic Graphics Accelerator design

In this figure:
- 1 is the **SWI** signal.
- 2 is the **CADIMemRead** signal.
- 3 is the **CADIMemWrite** signal.
- The Generic Graphics Accelerator comprises the Shim Layer and the Reconciler.

The major components in *Figure 1-1 Generic Graphics Accelerator design* on page 1-11 are:

Shim Layer
Plays the role of the GPU DDK driver and communicates with the Side Channel Plugin. The Shim Layer is implemented by the Shim library.

Side Channel Plugin
Provides a channel to send messages or data between the target and the host.
The application that calls OpenGL ES APIs on the target and the OpenGL ES Emulator on the host are in different address spaces. The Side Channel Plugin is used to communicate between different address spaces.

Fast Model
Plays the role of a real hardware machine.

Reconciler
Bridges the communication between the Side Channel Plugin and the OpenGL ES Emulator.

ARM Mali OpenGL ES Emulator
Simulates the OpenGL ES APIs by using the OpenGL APIs of the graphics card in the host.

In *Figure 1-1 Generic Graphics Accelerator design* on page 1-11, the buffer inside the target covers both the usual buffer to implement OpenGL ES APIs and the frame buffer as the rendering target.

Workflow

The basic workflow is as follows:

1. An application running on the target makes an OpenGL ES API call. After the call, the application continues to run until it needs some data returned from the API.
2. The Shim Layer intercepts the OpenGL ES API call and informs the Side Channel Plugin that it needs to send messages and data to the host.
3. The Shim Layer writes the data into the Write Buffer inside the Side Channel Plugin when a software interrupt is invoked in the Fast Model.
4. The Reconciler wakes up after the buffer writing is completed.
5. The Reconciler passes the data to the ARM Mali Open GL ES Emulator on the host and waits until the API call is completed.
6. The Reconciler writes the output back to the Read Buffer of the Side Channel Plugin.
7. The Shim Layer sends another software interrupt to read the data from the Read Buffer when the application needs the data returned from the API.
8. The Side Channel Plugin reads the data from read buffer and returns directly from the interrupt.
9. The application continues to run on the target.

1.4 Prerequisites

In addition to the normal Fast Models requirements, the Generic Graphics Accelerator has the following software and hardware requirements:

- Software requirements:
  - — Android OS (32-bit and 64-bit).
  - — ARM Mali OpenGL ES Emulator.
- Hardware requirements:

  A graphics card that supports OpenGL 3.2 or above, for example NVIDIA GT730.

Related information

Requirements for Fast Models.

1.5 Limitations

The Generic Graphics Accelerator, Version 1.0 has some limitations.

API support

The Generic Graphics Accelerator supports OpenGL ES 2.0, and does not support the following APIs:

- OpenGL ES 1.0, OpenGL ES 2.1, or later versions.
- OpenVG and OpenCL.

Supported usage scenarios

The Generic Graphics Accelerator can improve Android performance and offload graphics operations from CPU models. However, it cannot be used for the following:

- Developing the Mali GPU driver.
- Verifying the SoC integration of an OS, applications, and GPU drivers.

GPU modeling

The Generic Graphics Accelerator is part of Fast Models, and cannot provide the following information for GPU modeling:

- Cycle-accurate information for a GPU.
- Performance and power estimation for a GPU.

Chapter 2
Getting started with an Android target

This section describes how to enable the Generic Graphics Accelerator to support Android on Fast Models.

It contains the following sections:

• *2.1 Customizing Android and Fast Models to support the Generic Graphics Accelerator* on page 2-16.

2.1 Customizing Android and Fast Models to support the Generic Graphics Accelerator

You must customize Android and Fast Models to support the Generic Graphics Accelerator.

Procedure

The following figure shows the relationship between the task steps and the different components. Only steps 4-7 are relevant to the Generic Graphics Accelerator. The other steps are independent of it, and can be done separately by consulting relevant documentation.

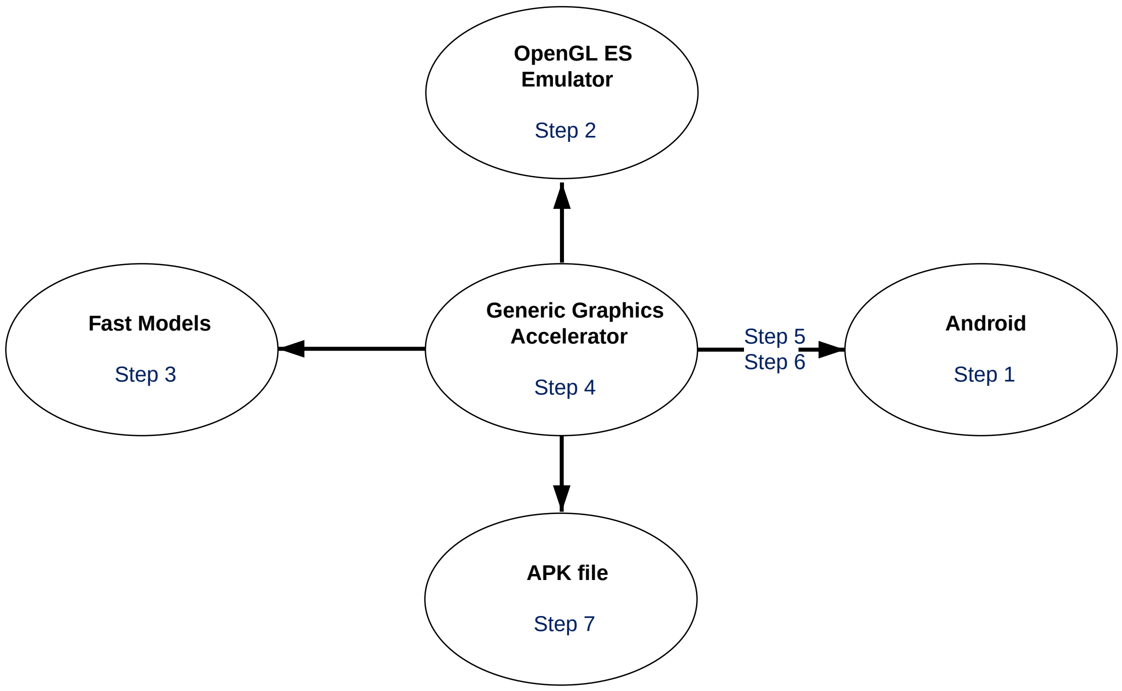

Figure 2-1 Task steps and components

1. Either rebuild Android from source, or use a pre-built binary image from Linaro.
2. Install the ARM Mali OpenGL ES Emulator.
3. Build Fast Models.
4. Boot Android by using the Generic Graphics Accelerator.
5. Push the Shim library to the target.
6. Modify Android properties and reboot Android.
7. Install and run an Android application.

This section contains the following subsections:

- *2.1.1 Building or obtaining Android* on page 2-16.
- *2.1.2 Installing the ARM Mali OpenGL ES Emulator* on page 2-17.
- *2.1.3 Building Fast Models* on page 2-17.
- *2.1.4 Booting Android* on page 2-18.
- *2.1.5 Pushing the Shim libraries to the target* on page 2-18.
- *2.1.6 Modifying Android properties and rebooting Android* on page 2-18.
- *2.1.7 Installing and running an Android application* on page 2-19.

2.1.1 Building or obtaining Android

To build Android, follow the instructions from the Linaro website.

For example, see *http://releases.linaro.org/14.12/android/lcr/fvp/*.

Alternatively, to download a pre-built Android binary image from Linaro, see the instructions at *https://community.arm.com/docs/DOC-10831*.

Note

ARM recommends that you use the latest Android source and documents.

2.1.2 Installing the ARM Mali OpenGL ES Emulator

To install the OpenGL ES Emulator, take the following steps:

Procedure

1. Download the installation package at *http://malideveloper.arm.com/develop-for-mali/tools/software-development/opengl-es-emulator/*.
2. Extract and install the package.
   For example, you can install the OpenGL ES Emulator in Linux as follows:

```
tar xvfz Mali_OpenGL_ES_Emulator-2.2-Linux-64bit.tgz

cd Mali_OpenGL_ES_Emulator-2.2-Linux-64bit/

./linux-install.sh
```

   Note

   You must install the OpenGL ES Emulator as root.

   ---

3. Set the environment variable LD_LIBRARY_PATH to specify where to load OpenGL ES Emulator libraries.
   For example:

```
export LD_LIBRARY_PATH=/opt/Mali_OpenGL_ES_Emulator-2.2-Linux-64bit/lib:$LD_LIBRARY_PATH
```

4. Verify the installation by using the `mali-cube` command to run the test program Mali Cube.
   If the OpenGL ES Emulator is installed successfully, you can see a spinning cube, as shown in the following screenshot:

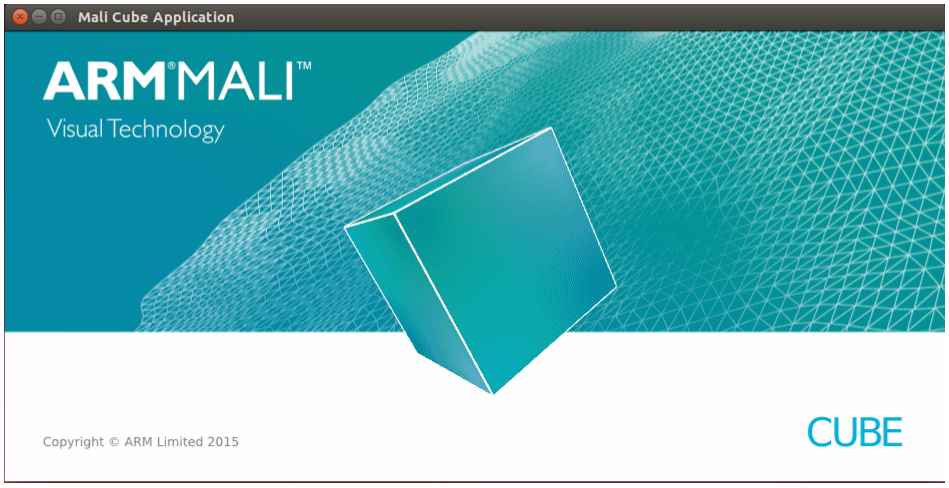

Figure 2-2 Mali Cube Application

2.1.3 Building Fast Models

You can customize Fast Models, or use sample projects to build Fast Models.

Related information

About System Generator.

2.1.4 Booting Android

To boot Android, take the following steps:

Procedure

1. Copy the `settings.ini` file from the directory *gga_installation_folder*`/GGA/reconciler/linux-x86_64/gcc-4.7.2/rel/` to your current directory.
   The Generic Graphics Accelerator requires the `settings.ini` configuration file to be in the current directory in which you enter the command to start the Generic Graphics Accelerator.
2. Boot Android with appropriate command lines.
   For details about the command lines to boot 32-bit or 64-bit Android, see *3.1 Command-line examples for booting Android* on page 3-22.

2.1.5 Pushing the Shim libraries to the target

Use `adb` to push the libraries to the target.

Procedure

1. To get `adb` to work on the Linaro pre-built image, run the following commands on the target:

```
suifconfig eth0 up
dhcptool eth0
stop adbd
setprop service.adb.tcp.port 6565
start adbd
```

   On the host, run the command:

```
adb connect localhost:5212
```

2. Go to the Generic Graphics Accelerator folder.
3. Push the Shim libraries to the target using `adb`.
   For 32-bit Android, use the following commands:

```
adb remount

adb push  GGA/shim/linux-armv7sfl/rel/libShim.so   /system/lib/libShim.so
adb push  GGA/shim/linux-armv7sfl/rel/libGLESv2.so /system/lib/egl/libGLESv2_vimpl.so
adb push  GGA/shim/linux-armv7sfl/rel/libEGL.so    /system/lib/egl/libEGL_vimpl.so

adb shell chmod 0644 /system/lib/libShim.so
adb shell chmod 0644 /system/lib/egl/libGLESv2_vimpl.so
adb shell chmod 0644 /system/lib/egl/libEGL_vimpl.so

adb shell ln -s /system/lib/egl/libGLESv2_vimpl.so /system/lib/egl/libGLESv1_CM_vimpl.so
```

   For 64-bit Android, use the following commands:

```
adb remount

adb push  GGA/shim/linux-armv7sfl/rel/libShim.so    /system/lib/libShim.so
adb push  GGA/shim/linux-armv7sfl/rel/libGLESv2.so  /system/lib/egl/libGLESv2_vimpl.so
adb push  GGA/shim/linux-armv7sfl/rel/libEGL.so     /system/lib/egl/libEGL_vimpl.so

adb push  GGA/shim/linux-armv8l_64/rel/libShim.so   /system/lib64/libShim.so
adb push  GGA/shim/linux-armv8l_64/rel/libGLESv2.so /system/lib64/egl/libGLESv2_vimpl.so
adb push  GGA/shim/linux-armv8l_64/rel/libEGL.so    /system/lib64/egl/libEGL_vimpl.so

adb shell chmod 0644 /system/lib/libShim.so
adb shell chmod 0644 /system/lib/egl/libGLESv2_vimpl.so
adb shell chmod 0644 /system/lib/egl/libEGL_vimpl.so

adb shell chmod 0644 /system/lib64/libShim.so
adb shell chmod 0644 /system/lib64/egl/libGLESv2_vimpl.so
adb shell chmod 0644 /system/lib64/egl/libEGL_vimpl.so

adb shell ln -s /system/lib/egl/libGLESv2_vimpl.so /system/lib/egl/libGLESv1_CM_vimpl.so

adb shell ln -s /system/lib64/egl/libGLESv2_vimpl.so /system/lib64/egl/libGLESv1_CM_vimpl.so
```

2.1.6 Modifying Android properties and rebooting Android

You must modify Android properties so that Android OS can run with the Generic Graphics Accelerator.

This step changes hardware rendering from `false` to `true`, and enables the libraries that were imported into Android OS by the step *2.1.5 Pushing the Shim libraries to the target* on page 2-18.

64-bit Android

To modify 64-bit Android properties, take the following steps:

Procedure

1. Edit the properties file `/system/build.prop`.
2. Add the following text to the file:

   ```
   ro.zygote.disable_gl_preload=true
   ```

3. Edit the meta-EGL implementation to disable a check governing the GPU, by using the following commands:

   ```
   adb remount
   adb shell sed -i '1,/ro.kernel.qemu/s/ro.kernel.qemu/No.kernel.qemu/' /system/lib64/
   libEGL.so
   adb shell sed -i '1,/ro.kernel.qemu/s/ro.kernel.qemu/No.kernel.qemu/' /system/lib/
   libEGL.so
   ```

4. Reboot Fast Models and Android for these changes to take effect.

2.1.7 Installing and running an Android application

Install and run an Android application to verify whether Android OS is functioning.

Note

This step should not be done until the system reboot is complete, which can take several minutes.

Procedure

1. Ensure the ADB client is available.

   ```
   export PATH=$PATH:/opt/android-sdk-linux/platform-tools/
   adb connect localhost:your_port_number
   ```

2. Install a sample application.
   For 32-bit Android:

   ```
   adb install gga_installation_folder/GGA/examples/linux-armv7sfl/Cube.apk
   ```

   For 64-bit Android:

   ```
   adb install gga_installation_folder/GGA/examples/linux-armv8l_64/Cube.apk
   ```

3. Run the sample application to verify whether Android OS is up.
   If Android OS is up, you can see a spinning cube as shown in the following screenshot:

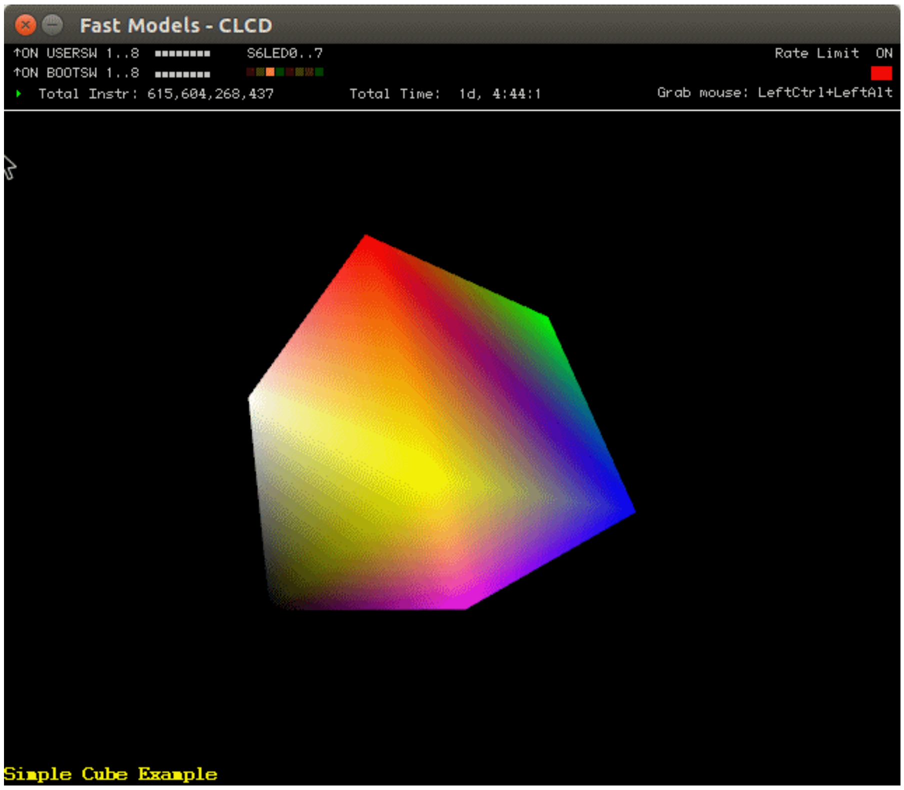

Figure 2-3 Mali cube

Chapter 3
Parameters and configuration

This section describes the parameters that you can use to configure and run the Generic Graphics Accelerator.

It contains the following sections:

- *3.1 Command-line examples for booting Android* on page 3-22.
- *3.2 Configuration* on page 3-24.

3.1 Command-line examples for booting Android

These typical command lines show you how to boot Android to use the Generic Graphics Accelerator.

Note

The name of the Sidechannel plugin in these command lines is case-sensitive.

32-bit Android for Linux

The following command-line example shows you how to boot 32-bit Android to use the Generic Graphics Accelerator in Linux.

```
../models/Linux64_GCC-4.1/FVP_VE_Cortex-A15x1 \
boot/rtsm/linux-system-semi.axf \
--plugin ../plugins/path_of_Sidechannel.so \
-C DEBUG.Sidechannel.interceptor=path_of_libReconciler.so \
-C motherboard.smsc_91c111.enabled=1 \
-C motherboard.vis.disable_visualisation=0 \
-C motherboard.hostbridge.userNetworking=1 \
-C motherboard.hostbridge.userNetPorts=5204=6565 \
-C motherboard.mmc.p_mmc_file=linaro-android-vexpress-lsk-14.10.img
```

In this example, `../models/Linux64_GCC-4.1/FVP_VE_Cortex-A15x1` is a Fast Model FVP.

The following parameters that are used in the command line shown above are relevant to the Generic Graphics Accelerator:

- The parameter `--plugin ../plugins/`*`path_of_Sidechannel.so`* instructs Fast Models to load the plugin `Sidechannel.so`, which communicates between the host and the target.
- The parameter `-C DEBUG.sidechannel.interceptor=`*`path_of_libReconciler.so`* instructs the Reconciler to use the Side Channel Plugin to intercept Fast Models.

64-bit Android for Linux

The following command line example shows you how to boot 64-bit Android to use the Generic Graphics Accelerator in Linux.

```
../../models64/Build_AEMv8A-AEMv8A/Linux64_GCC-4.1/FVP_Base_AEMv8A-AEMv8A \
--plugin ../../plugins/path_of_Sidechannel.so \
-C DEBUG.Sidechannel.interceptor=path_of_libReconciler.so \
-C pctl.startup=0.0.0.0 \
-C bp.secure_memory=0 \
-C cluster0.NUM_CORES=1 \
-C cluster1.NUM_CORES=0 \
-C cache_state_modelled=0 \
-C bp.hostbridge.userNetworking=1 \
-C bp.hostbridge.userNetPorts=5212=6565 \
-C bp.smsc_91c111.enabled=1 \
-C bp.smsc_91c111.mac_address=auto \
-C bp.pl011_uart0.untimed_fifos=1 \
-C bp.secureflashloader.fname=bl1.bin \
-C bp.flashloader0.fname=fvp_fip.bin \
-C bp.flashloader1.fname=uefi-vars.fd \
-C bp.virtioblockdevice.image_path=../linaro-android-fvp_v8-lcr-14.12_build.img
```

In this example, `../../models64/Build_AEMv8A-AEMv8A/Linux64_GCC-4.1/FVP_Base_AEMv8A-AEMv8A` is a Fast Model FVP.

The following parameters that are used in the command line shown above are relevant to the Generic Graphics Accelerator:

- The parameter `--plugin ../../plugins/`*`path_of_Sidechannel.so`* instructs Fast Models to load the plugin `Sidechannel.so`, which communicates between the host and the target.
- The parameter `-C DEBUG.sidechannel.interceptor=`*`path_of_libReconciler.so`* instructs the Reconciler to use the Side Channel Plugin to intercept Fast Models.

Related information

- For details about FVP_VE_Cortex-A15x1, see *http://releases.linaro.org/14.10/android/vexpress-lsk*.

3.2 Configuration

You can use a `settings.ini` configuration file to configure the execution speed and system debugging logs for the Generic Graphics Accelerator.

The `settings.ini` configuration file is in the directory *`installation_directory/installation_package_name`*`/GGA/reconciler/linux-x86_64/gcc-4.7.2/rel`. However, the `settings.ini` file in the current directory is used to configure the Generic Graphics Accelerator.

Note

You must copy `settings.ini` to your current directory in which you run the command to start the Generic Graphics Accelerator.

Execution speed settings

The `settings.ini` file provides the following parameters to control the execution speed of the Generic Graphics Accelerator.

`conformant`
Instructs the Generic Graphics Accelerator to execute all calls on both the host and target. When you specify `conformant`, the execution speed of the Generic Graphics Accelerator is slow because command batching is not allowed.

`fast`
Instructs the Generic Graphics Accelerator to batch commands, skip certain calls from the target, and render directly to the frame buffer. When you specify `fast`, the execution speed of the Generic Graphics Accelerator is faster. However, OpenGL ES APIs might be issued to the OpenGL ES Emulator in an order different than the order the OpenGL ES APIs are issued by the application on the Android target. As a result, the OpenGL ES API mapping between the host and the target is vague.
Do not specify `fast` if you want to check the relationship between the following OpenGL ES APIs:

- The OpenGL ES APIs called on the host graphic driver.
- The OpenGL ES APIs called by the application on the Android target.

By default, `conformant` is set.

Logging settings

The `settings.ini` file provides the `LogLevel` parameter to specify the verbosity level of logs. You can use the logs to trace the OpenGL ES API invocation sequence and debug the Generic Graphics Accelerator.

The logs can be found in the following locations:

- The logs about the interaction between the Reconciler and the OpenGL ES Emulator are printed in the current console.
- The following logs can be viewed by the `logcat` command in Android:
  — Logs about the interaction between the Shim Layer and applications.
  — Logs about the interaction between the Shim Layer and the Reconciler.

For the `LogLevel` parameter, you can specify one of the following verbosity levels, by using macros. The verbosity order goes from the least verbose level, 0 to the most verbose level, 6567.

`# LOG_LEVEL_OFF 0`
The Generic Graphics Accelerator issues no messages.

`# LOG_LEVEL_FATAL 1`
The Generic Graphics Accelerator issues fatal messages.

`# LOG_LEVEL_ERROR 2`
The Generic Graphics Accelerator issues error messages.

`# LOG_LEVEL_WARN 3`
: The Generic Graphics Accelerator issues warning messages.

`# LOG_LEVEL_INFO 6565`
: The Generic Graphics Accelerator issues information about important stages of executing APIs.

`# LOG_LEVEL_DEBUG 6566`
: The Generic Graphics Accelerator issues the names and parameters of each API that is called.

`# LOG_LEVEL_TRACE 6567`
: The Generic Graphics Accelerator issues detailed information for checking bugs in the Generic Graphics Accelerator.
Do not specify this value unless ARM requests you to provide more detailed information for debugging purpose.

The verbosity level that you choose affects the system performance. The higher the verbosity level, the slower the system runs.

The default setting is `LogLevel 2`.

If you find any abnormal situation with the Generic Graphics Accelerator, send these logs to *support-esl@arm.com* for diagnostic purposes.

Chapter 4
Debugging

This section describes the debugging workflow, debugging tools, and how to report bugs.

It contains the following sections:

- *4.1 Debugging workflow* on page 4-27.
- *4.2 API tracing debugging tool* on page 4-28.
- *4.3 Reporting bugs in the Generic Graphics Accelerator* on page 4-29.

4.1 Debugging workflow

This debugging workflow defines the steps to take to decide whether to report bugs in the Generic Graphics Accelerator.

The Generic Graphics Accelerator provides the ability to debug applications that run on it.

If your application does not run correctly on the Generic Graphics Accelerator, your application or the Generic Graphics Accelerator might contain bugs.

The following figure shows the debugging workflow:

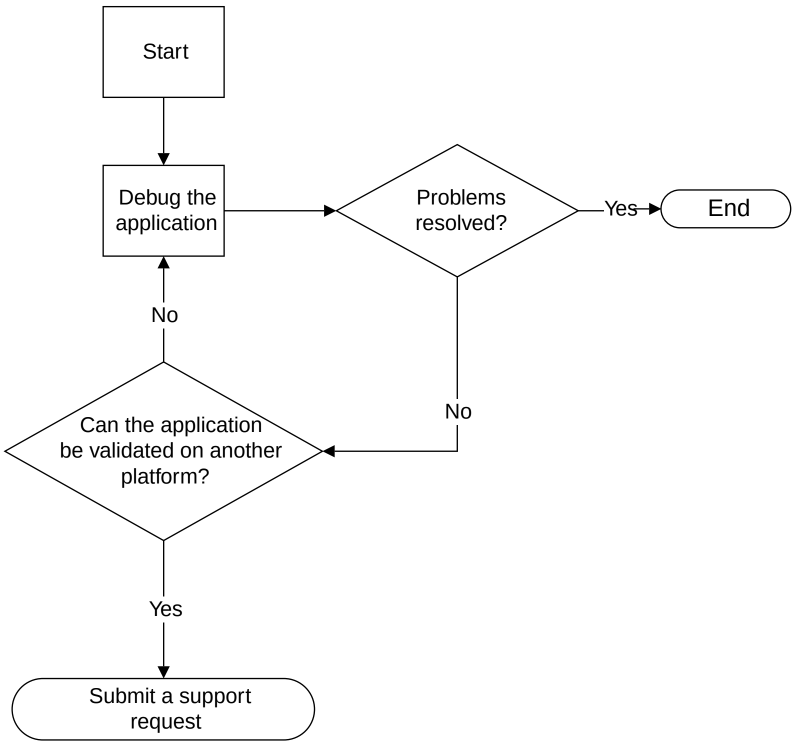

Figure 4-1 Debugging workflow

If your application does not run correctly, debug it on the Generic Graphics Accelerator. You can do this by using the logs provided by the Generic Graphics Accelerator.

If you cannot fix the problem and the application can be validated on another platform, you can submit a support request to report bugs in the Generic Graphics Accelerator.

Related references

4.3 Reporting bugs in the Generic Graphics Accelerator on page 4-29.

3.2 Configuration on page 3-24.

4.2 API tracing debugging tool

Use the API tracing debugging tool to debug your application.

The Generic Graphics Accelerator provides different verbosity levels for logs to trace APIs execution. This API tracing information is useful for debugging your application.

Specify either of the following verbosity level in the `settings.ini` file to trace APIs execution information.

# LOG_LEVEL_INFO 6565
: The Generic Graphics Accelerator issues information about important stages in executing APIs.

# LOG_LEVEL_DEBUG 6566
: The Generic Graphics Accelerator issues the names and parameters of each API that is called.

For details about the logging settings, see *3.2 Configuration* on page 3-24.

4.3 Reporting bugs in the Generic Graphics Accelerator

To report bugs, send the following information to support-esl@arm.com for diagnostic purposes:

- The specific version of Fast Models.
- The virtual platform provided by Fast Models.
- The OS of the host.
- The graphic card used in the host.
- The driver information of the graphic card.
- A brief description of the application. For example, you can include information such as whether the application is written using Java or C.
- The version of Android on the target.
- The description of the issue, with the expected output and the output you observe.
- The application that fails if possible, or a cutdown application that reproduces the issue.
- Debug logs.