Arm® Mali™ RenderScript Best Practices

Version 2.0

Developer Guide

101144_0200_00_en

Arm® Mali™ RenderScript Best Practices

Developer Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
0100-00	15 February 2019	Non-Confidential	First release
0200-00	12 April 2019	Non-Confidential	Second release

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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents

Arm® Mali™ RenderScript Best Practices Developer Guide

Preface

This preface introduces the *Arm® Mali™ RenderScript Best Practices Developer Guide*.

It contains the following:

- *About this book* on page 6.
- *Feedback* on page 7.

About this book

This book describes the RenderScript best practices for Arm® Mali™ GPUs.

Intended audience

This book is for developers working with RenderScript on Arm Mali™ Midgard, Bifrost, or Valhall GPUs. It is intended to be used in addition to Google RenderScript documentation, and provides Mali GPU-specific advice.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces RenderScript.

Chapter 2 RenderScript kernel best practices
This chapter describes best practices and other performance considerations for RenderScript kernels.

Appendix A Revisions
This appendix describes the changes between released issues of this book.

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.

Additional reading

This book contains information that is specific to this product. See the following documents for other relevant information.

Arm publications

- *Arm® Mali™ Bifrost and Valhall OpenCL Developer Guide* (ARM 101574).
- *Arm® Mali™ Midgard OpenCL Developer Guide* (ARM 100614).

See *https://developer.arm.com* for access to Arm documentation.

Other publications

None

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 *Arm Mali RenderScript Best Practices Developer Guide*.
- The number 101144_0200_00_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.

Chapter 1
Introduction

This chapter introduces RenderScript.

It contains the following section:

1.1 About RenderScript

RenderScript is a language and API in Android OS that enables you to use different processors for compute intensive tasks.

RenderScript applications can be written to make optimal use of the GPU acceleration capabilities offered by the Mali RenderScript GPU accelerator.

- For an overview of RenderScript, see *https://developer.android.com/guide/topics/renderscript/compute*.
- The RenderScript Java API is documented here *https://developer.android.com/reference/android/renderscript/RenderScript*.
- The RenderScript runtime API is documented here *https://developer.android.com/guide/topics/renderscript/reference/overview*.

Chapter 2
RenderScript kernel best practices

This chapter describes best practices and other performance considerations for RenderScript kernels.

It contains the following sections:

2.1 Using global variables

How to use global variables in RenderScript.

Minimize the number and size of global variables used in your kernels

To enable hardware to accelerate global variables in root functions, the global variables are copied by the RenderScript runtime. The copy operations take time, so try to minimize them. Instead of using global variables, try passing all input to a root function with the input and output allocations. If you do this in invokable helper functions, it is unrestricted because the invokable functions always execute on the application processor.

Pack the data into one allocation. For example:

```
void root(const float2* in, float* out, uint32_t x)
{
*out = in->x + in->y;
}
```

Avoid this type of construct:

```
rs_allocation array; //global allocation, same size as the input/output

void root(const float* in, float* out, uint32_t x)
{
*out = *in + rsGetElementAt_float(array, x);
}
```

Avoid writing to global variables

Writing to global variables within a kernel can degrade performance because it reduces the opportunity to apply performance optimizations. For example:

```
int global_var;

void root(const float* in, float* out)
{
//reading from the global variable is ok
int tmp = global_var;
//avoid writing to the global variable.
global_var = 1;
}
```

There is no restriction on using global variables in invokable helper functions. This is because invokable functions always execute on the application processor.

2.2 Use allocation usage flags

To ensure an allocation is supported on the GPU, include the USAGE_SCRIPT flag.

If an allocation does not have the USAGE_SCRIPT flag set, it is not supported on the GPU. If usage flags are not specified, an allocation is marked with USAGE_SCRIPT by default. However, an allocation might not have its USAGE_SCRIPT flag set, if it is explicitly marked with other flags. If a root function uses an unsupported allocation, it executes on the application processor.

If you are setting usage flags on an allocation, ensure that USAGE_SCRIPT is also added. The following example explicitly includes USAGE_SCRIPT:

```
Allocation.createTyped(
    mRS,
    typeBuilder.create(),
    MipmapControl.MIPMAP_NONE,
    //OR'd with USAGE_SCRIPT flag, acceleration enabled.
    Allocation.USAGE_GRAPHICS_TEXTURE | Allocation.USAGE_SCRIPT
);
```

The following example does not include USAGE_SCRIPT and is not usable on the GPU:

```
Allocation.createTyped(
    mRS,
    typeBuilder.create(),
    MipmapControl.MIPMAP_NONE,
    //Not marked with USAGE_SCRIPT, GPU acceleration disabled.
    Allocation.USAGE_GRAPHICS_TEXTURE
);
```

If the allocation is marked with USAGE_IO_INPUT or USAGE_IO_OUTPUT, the allocation is not supported on the GPU, even if you set USAGE_SCRIPT. For example:

```
Allocation.createTyped(
    mRS,
    typeBuilder.create(),
    MipmapControl.MIPMAP_NONE,
    //USAGE_IO_INPUT is not currently supported on the GPU, this allocation is CPU-only
    Allocation.USAGE_IO_INPUT | Allocation.USAGE_SCRIPT
);
```

2.3 Avoid casting user-defined data

Do not cast user-defined data to a different type, or reference data beyond the type referenced by the pointer.

Casting user-defined data to a different type, or reference data beyond the type referenced by the pointer, is not good practice, because it limits the scope of possible optimizations. For example:

```
typedef struct {
    int val;
} user_data_t;

void root(int * out, user_data_t * user)
{
    user_data_t * other_value = &(user[1]);
    *out = other_value->val;
}
```

2.4 Avoid recursive root functions

Do not implement root functions using direct or indirect recursion.

Recursive functions always execute on the application processor. They do not execute on the GPU.

2.5 Use RenderScript intrinsic functions

RenderScript intrinsic functions are high-performance implementations of functions.

The intrinsic functions outperform any equivalent function written in RenderScript.

Use the following intrinsic functions where possible to increase the performance of your application:

- `Blend()`
- `Blur()`
- `ColorMatrix()`
- `Convolve3x3()`
- `Convolve5x5()`
- `Histogram()`
- `LUT()`
- `YUVtoRGB()`

For more details on intrinsic functions, see *https://developer.android.com/reference/android/renderscript/ScriptIntrinsic*.

2.6 Consider numerical precision

The numerical precision of some built-in functions can vary across different functions.

The numerical precision of the following built-in functions can vary across different functions when they are accelerated on the GPU:

- cospi()
- sinpi()
- tanpi()
- dot()
- mix()

If your application is sensitive to numerical precision and uses one or more of these functions, test your script on different devices to ensure the range of results is within acceptable limits.

For more details on these built-in functions, see RenderScript runtime API documentation.

2.7 Debugging properties

RenderScript includes the debug.rs.default-CPU-driver and debug.rs.script debugging properties.

debug.rs.default-CPU-driver

Values = 0 or 1
Default value = 0
If set to 1, the *Android Open Source Project* (AOSP) implementation of RenderScript Compute is used. This does not use any GPU features.

debug.rs.script

Values = 0 or 1
Default value = 0

If set to 1, additional diagnostic information is printed in the logcat. This information includes the actual device a kernel is running on, either GPU or application processor.

If a kernel cannot be run on the GPU more detailed information is provided explaining why. For example:

```
[RS-DIAG] No support for recursive calls on GPU
```

Appendix A
Revisions

This appendix describes the changes between released issues of this book.

It contains the following section:
- *A.1 Revisions* on page Appx-A-19.

A.1 Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1 Issue 0100_00

Change	Location	Affects
First release.	-	All.

Table A-2 Issue 0200_00

Change	Location	Affects
Added Valhall applicability.	Various.	All.