

Arm® Mali™ RenderScript Best Practices

Version 1.0

Developer Guide



Arm® Mali™ RenderScript Best Practices

Developer Guide

Copyright © 2019 Arm Limited or its affiliates. All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
0100-00	15 February 2019	Non-Confidential	First release

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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

About this book	6
Feedback	8

Chapter 1

Introduction

1.1 About RenderScript	1-10
------------------------------	------

Chapter 2

RenderScript kernel best practices

2.1 Using global variables	2-12
2.2 Use allocation usage flags	2-13
2.3 Avoid casting user-defined data	2-14
2.4 Avoid recursive root functions	2-15
2.5 Use RenderScript intrinsic functions	2-16
2.6 Consider numerical precision	2-17
2.7 Debugging properties	2-18

Appendix A

Revisions

A.1 Revisions	Appx-A-20
---------------------	-----------

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 8.](#)

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 or Bifrost 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.

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.

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 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_0100_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 on page 1-10.](#)

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 on page 2-12.*
- *2.2 Use allocation usage flags on page 2-13.*
- *2.3 Avoid casting user-defined data on page 2-14.*
- *2.4 Avoid recursive root functions on page 2-15.*
- *2.5 Use RenderScript intrinsic functions on page 2-16.*
- *2.6 Consider numerical precision on page 2-17.*
- *2.7 Debugging properties on page 2-18.*

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-20.](#)

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