Material and shader best practices for artists

Version 1.0

Non-Confidential

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

Issue 01
102576_0100_01_en

Material and shader best practices for artists

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

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	14 November 2019	Non-Confidential	Initial release

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, 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.

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 https://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.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

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

Contents

1. Material and shader best practices for artists.....6

2. Shader and materials hints and tips.....9

3. Transparency best practices..... 14

4. Other best practices and tips.....20

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

1. Material and shader best practices for artists

Welcome to the third best practice for mobile from an artists perspective. We hope you will find everything you are looking for regarding materials and shaders in this guide.

What you will learn;

- The difference between a material and a shader.
- The best practices when dealing with both materials and shaders, along with game engine best practices when dealing with both.
- Transparency best practices from an artists perspective.
- All the other small best practices that fit into the area of dealing with materials and shaders.

The difference between a shader and a material

Shader

- A shader is a small program that tell the GPU how to draw an object on the screen and all the calculations that are needed to happen on it.
- A shader needs to be attached to a material to be used.
- Some of the scripting languages commonly used to author shader are HLSL and GLSL.

Material

- A material is something that can be applied to an object or mesh to define the visual look of that object.
- A material is used to set the specific value of parameter that made available from a shader, such as the color, numeric values.

A shader and a material can be implemented differently depending on the game engine. In Unity, different shaders can be created and applied to different materials, while in Unreal engine the shader used can be based on the target platform selected.

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

Figure 1-1: Shader Material

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

Figure 1-2: Unity Shader Graph

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

2. Shader and materials hints and tips

Unity have collections of shaders targeted for mobile platforms, which are conveniently located in the mobile category. Use these shaders whenever possible for mobile game development.

- These shaders contain a lot of feature simplifications, enabling them to have a better performance on mobile based platforms.
- This means that there will be less features compared with a standard shader, such as no color tinting.

Figure 2-1: Unity Mobile Preview

Unreal Engine will use the most relevant mobile shader depending on the target platform selected. Using this mobile shader will result in a different look compared with the default Shader Model 5. You need to change the Preview Rendering Level to better visualize the final look in the device within the editor. Unreal Engine's material on mobile platforms is created using the same process with other platforms, so they generally have similar visuals and behaviors.

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

Figure 2-2: Unreal Engine Preview

Only use features that are needed. Unity and Unreal Engine compile the shader based on the material settings, and by removing features that are not needed (such as colour tinting, detail map) we will have an overall less complex shader that can potentially help with performance on mobile platforms.

Use a fewer number of texture sampler in a material

For mobile platforms, the general rule of thumb is to use as few textures as possible. This is because more textures will likely result in more texture fetches, which means more bandwidth that will most certainly affect battery life. Also, as these textures need to be saved in memory, it will increase the application size considerably.

- In Unreal Engine, the recommended maximum number of texture samplers for mobile are five.
- It is still advisable to use the least amount of textures possible, as using all five texture samplers in a material will make it considerably more expensive.
- Using texture packing will help to reduce the number of textures. Instead of using individual textures for the roughness or metallic, we can pack them into channels of a single texture.

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

Figure 2-3: Texture Packing

We can also use a numerical value instead of a texture for some parameters, such as metallic, roughness or smoothness (do this if possible and observe whether it will affect the visual quality), as this will further reduce the number of textures used.

- In Unreal Engine, this is done by adding constant to the parameter slots. In Unity, a value needs to be added within the shader. Using an unlit shader will also reduce the number of textures used, as the material is not affected by light; therefore, it will not need roughness or metallic texture.

An unlit shader is cheaper then lit

When creating a shader, we can decide how the material will react to light. Most commonly used in mobile games are either lit or unlit shaders.

Unlit

It is the fastest, cheapest shading model. Use this if targeting lower-end devices.

- Use an unlit shading model as much as possible, especially if the game is targeting lower-end devices.

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

- Unlit shading model, as the name implies, is not affected by lighting and only output emissive as color.
- As it is not affected by light, many calculations are not needed, such as specular, shadow, resulting in cheaper, faster rendering.
- Going with stylized or cartoony art direction works really well with unlit shading and should be considered when making games for mobile platforms.

Default lit

It will need extra processing compared with unlit.

- More processing needed for this compared with unlit, but will enable the surface to be affected by light and have specularity.
- For most games this is probably the shading model that is most used.

A comparison between Lit and Unlit can be seen below, same tower mesh and texture set applied using different shader. Unlit is not affected by lights and, in turn, needs less computation because of it. This will potentially result in a better gaming performance.

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

Figure 2-4: Lit and Unlit

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

3. Transparency best practices

Avoid transparency and use opaque as much as possible.

Figure 3-1: Opaque Shadows

Use an opaque material whenever possible. On mobile platforms, using transparency is best avoided unless necessary. Rendering an object with transparency will always be more expensive than opaque. Having lots of these transparent objects in games can or will affect performance, especially when these objects are rendered on top of one another many times.

This problem is known commonly as overdraw, when the same pixel is drawn multiple times. The more layers of transparency we have, the more expensive the rendering becomes. For mobile platforms, overdraw can severely affect performance; therefore, we need to be mindful when building our levels and try to keep overdraw to a minimum. Unreal Engine and Unity have a mode to visualize overdraw that can be used to track the severity of overdraw in a given scene.

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

Figure 3-2: Unreal Engine Overdraw

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

Figure 3-3: Unity Overdraw

Profile the performance and compare between the different implementations

There are different ways to do transparency in a shader. The most commonly used are alpha test and alpha blend. There are different best practices that suggest one over the other, but the result will always vary from differing use cases, so our best practice is to always profile the performance difference between the two. For the mobile platform, one of the ways to do this is using Streamline, a software or tool developed by Arm to collect the performance data of a device.

Alpha test or cutout

The alpha test implementation will make the object material look either 100% opaque or 100% transparent. We can also set the threshold of the cutout for the mask. In Unreal Engine, the term used is Masked blend mode, while in Unity this type of transparency is called Cutout.

Alpha blend

Visually alpha blend allows the material to have a range of transparency (and can make an object look partially transparent), instead of completely opaque or transparent. Unreal Engine calls this blend mode Translucent, while Unity uses the term Transparent.

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

A comparison of alpha blend and alpha test can be seen in the image below. Alpha blend allows partial transparency while alpha test will result in a sharp cutout.

Figure 3-4: Alpha Blend Alpha Test

Alpha test

- It is best to avoid cutout material (alpha test) unless it's really needed.
- Alpha test means that a material will be either 100% opaque or 100% transparent.
- Alpha test will generally not be recommended in mobile platforms as it disables some of the optimization features within the GPU. However, as always, we need to profile and compare the performance just to be sure.

Alpha blend

- For mobile platforms, Unity recommends using alpha blend over alpha test. In practice, we should always profile and compare the performance between alpha test and alpha blend, as they are usually content dependent and need to be measured.
- In general, alpha blend (all kinds of transparency) should be avoided on mobile platforms.
- If we need to have alpha blend in games, try to make the coverage of the blend area small.

Alpha test transparency might work better for foliage

In a static view, both examples below show the foliage looking perfectly fine, with alpha blend actually looking slightly better due to the soft edges compared with the sharp cut in alpha test.

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

Figure 3-5: Foliage stationary

However, in motion, alpha blend looks wrong and the leaves doesn't seem to render in the correct order. Alpha test handles the transparency and order of the leaves much better, but the edges will be harsher or aliased compared with alpha blend. Most of the time, the alpha test visual quality will be acceptable.

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

Figure 3-6: Foliage rotating

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

4. Other best practices and tips

In a situation where overdraw is unavoidable (such as when using particles) make the shader as simple as possible.

- Use the simplest shader possible, such as unlit, and avoid using unnecessary features.
- Unity has a shader designed or written specifically for particles, which can be a good implementation start.
- Try to minimize overdraw. Reducing the number and or size of particles might help with this.

Profile and visualize shader complexity

Adding more texture samplers, transparency and other features can make a shader become more complex and affect the rendering.

- Use viewmode Shader Complexity to check shader complexity inside the level. This view can give good estimates and provide an early indication of how expensive the shaders are. Green is good and red is more expensive.

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

Figure 4-1: Unreal Shader Complexity

In Unreal Engine 4, use the material stats feature to quickly profile the materials early in the process.

Figure 4-2: Unreal Material Stats

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

View mode above is a good rough guideline for any artist. To have a further look in profiling, we can use other tools, such as Streamline for Arm Mali, but they do require further graphics knowledge to understand more about how the GPU works.

Do as many operations in vertex shader as possible

The most common shader used in projects is usually a combination of vertex shader and pixel shader. Vertex shader works on every vertices and pixel shader runs on every pixel. Usually there will be more pixels rendered than there are vertices on a screen, which means there will be more computation happening for pixels than vertices. It is recommended to move computation from the pixel shader to the vertex shader whenever possible.

That being said, moving this operation to vertex shader usually means having to move the processed data to fragment (varyings). So, even though this is generally a good idea, we have to pay attention to the tiler in case it then becomes the bottleneck. As usual we need to do further profiling after working on optimizations.

A way to do this in Unreal Engine is using the Customized UVs feature, connecting nodes to them and then using a Texture Coordinate node in the pixel shader to target the customized UV. More information on this subject can be found here.

Reduce the number of expensive math operation whenever possible

We use mathematical operations within the shader to customize the look and behavior of it. Some examples of these math operations would be multiplication, addition, power, floor, logarithm.

These math operations are not equal in terms of performance cost; therefore, we need to pay attention to their usage. Some of the more costly operations are sin, pow, cos, divide, noise. Basic operations, such as additions and multiplications, are generally faster. Try to keep the number of expensive math operations as low as possible. This number particularly needs to be kept lower on older devices, such as ones with GLES 2.0.

Always do performance profiling

This is more of a general best practice, but never assume anything and always do some profiling to understand where the real bottlenecks are in an app. Further profiling is also recommended to compare the before and after effect of any optimization.

Next Steps

Have you seen our other guides? Check them out now

- Geometry Best Practices
- Texture Best Practices

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