



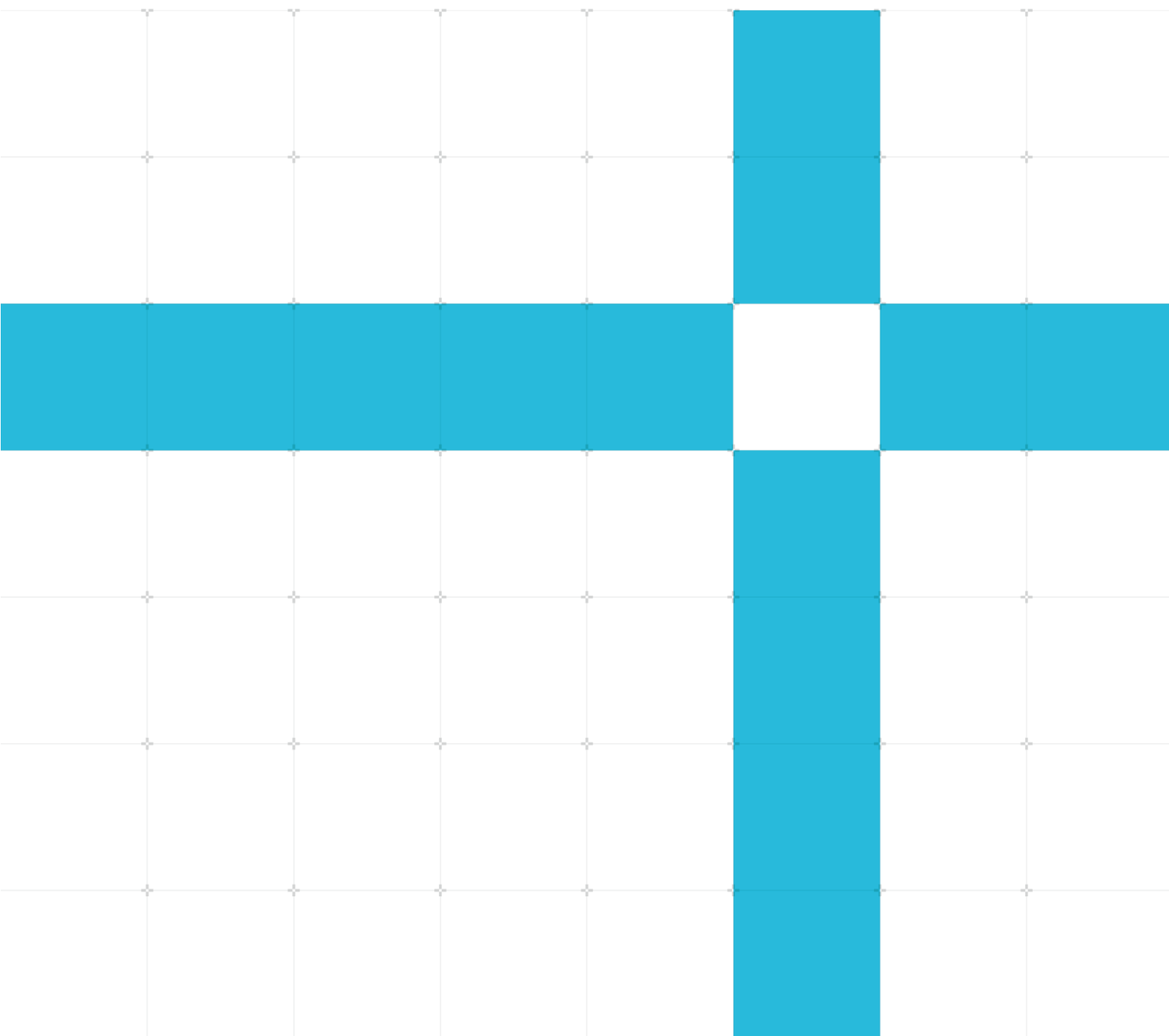
# Material and Shader Best Practices for Unreal Engine

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).  
All rights reserved.

**Issue 01**

102676



## Unreal Engine

### Material and Shader Best Practices for Unreal Engine

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

#### Release information

#### Document history

Issue	Date	Confidentiality	Change
01	October 13, 2021	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, 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 affiliates) 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 © 2021 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

[www.arm.com](http://www.arm.com)

# Contents

<b>1 Overview.....</b>	<b>5</b>
<b>2 What is a material and a shader? .....</b>	<b>6</b>
<b>3 Shaders for mobile platforms.....</b>	<b>7</b>
3.1 Texture samplers .....	8
3.2 Unlit and lit shaders .....	9
<b>4 Transparency considerations.....</b>	<b>10</b>
4.1 Alpha test.....	11
4.2 Alpha blend .....	11
4.3 Alpha test transparency for foliage .....	12
<b>5 Profile and visualize shader complexity.....</b>	<b>13</b>
5.1 Do operations in vertex shader.....	14
5.2 Reduce the number of math operations .....	14
5.3 Performance profiling.....	15
<b>6 Related information .....</b>	<b>16</b>
<b>7 Next steps .....</b>	<b>17</b>

# 1 Overview

In this guide you will learn about how to use materials and shaders accurately and effectively in Epic's Unreal Engine game engine. Everything in this guide is from the perspective of a technical artist targeting a mobile platform.

By the end of this guide, you will understand the following:

- The difference between a material and a shader
- The best practices and game engine best practices for materials and shaders
- Transparency best practices for artists
- Reducing shader complexity

## 2 What is a material and a shader?

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 value of a parameter that is available from a shader, such as the color, numeric values, and so on.

A shader is a small program that tells the GPU how to draw an object on the screen and the calculations needed to happen on the object.

Shaders have the following prerequisites:

- A shader needs to be attached to a material used
- A scripting language used to author shaders, such as HLSL and GLSL

A shader and a material can be used in different ways, depending on the game engine. In Unreal Engine, the shader used can be based on the target platform selected. The following image shows a shader and material in Unreal Engine:

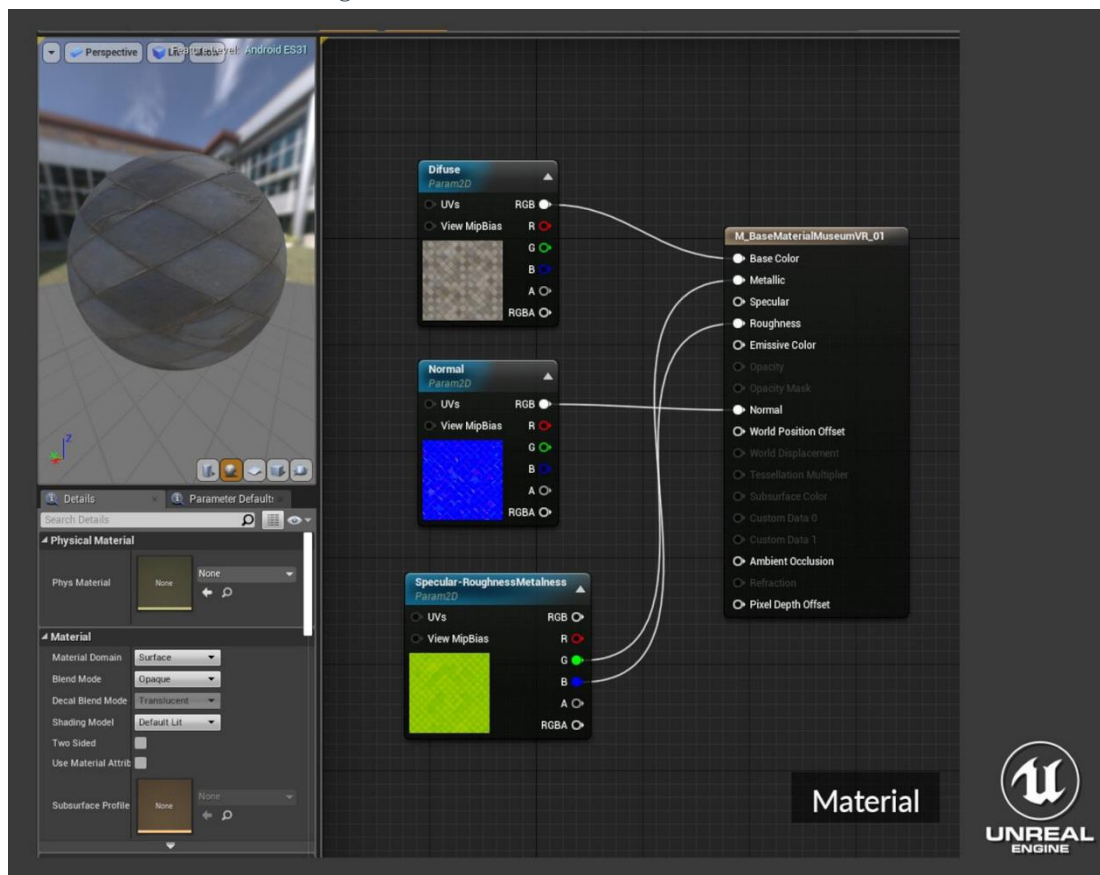
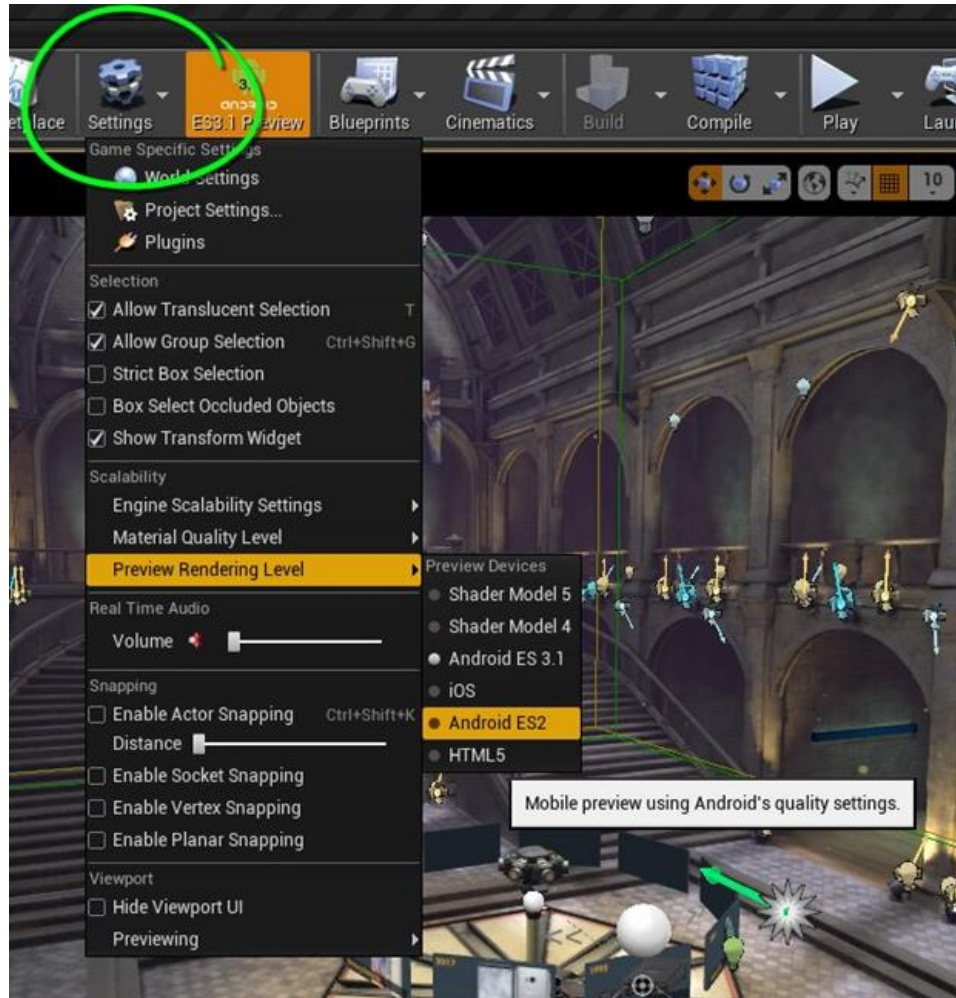


Figure 1: Shader and Material screen

## 3 Shaders for mobile platforms

Unreal Engine uses the most relevant mobile shader depending on the target platform selected. Using this mobile shader results in a different look compared to the default Shader Model 5. To better visualize the final look in the device within the editor, change the Preview Rendering Level, as shown in the following screenshot:



**Figure 2: Preview Rendering Level menu**

Unreal Engine's material on mobile platforms is created using the same process as other platforms, so they have similar visuals and behaviors.

Unreal Engine compiles the shader based on the material settings. By removing features that are not needed such as color tinting and detail maps, the shader will be less complex overall. Removing unused features can help with performance on mobile platforms.

## 3.1 Texture samplers

For mobile platforms, it is recommended that you use as few textures as possible. This is because more textures can result in more texture fetches, and the extra bandwidth will affect battery life. Also, because these textures are saved in memory, the application size increases.

The following are suggestions for using texture samplers on mobile devices:

- In Unreal Engine, the recommended maximum number of texture samplers for mobile is five. For more information, see the Unreal Engine [Materials for Mobile Platforms](#) documentation.
- If possible, use the least number of textures. It is considerably more expensive to use all five texture samplers in a material.
- Use texture packing to reduce the number of textures. For example, instead of using individual textures for roughness or metallic, pack the textures into channels of a single texture. An example of texture packing is shown in the following screenshot:

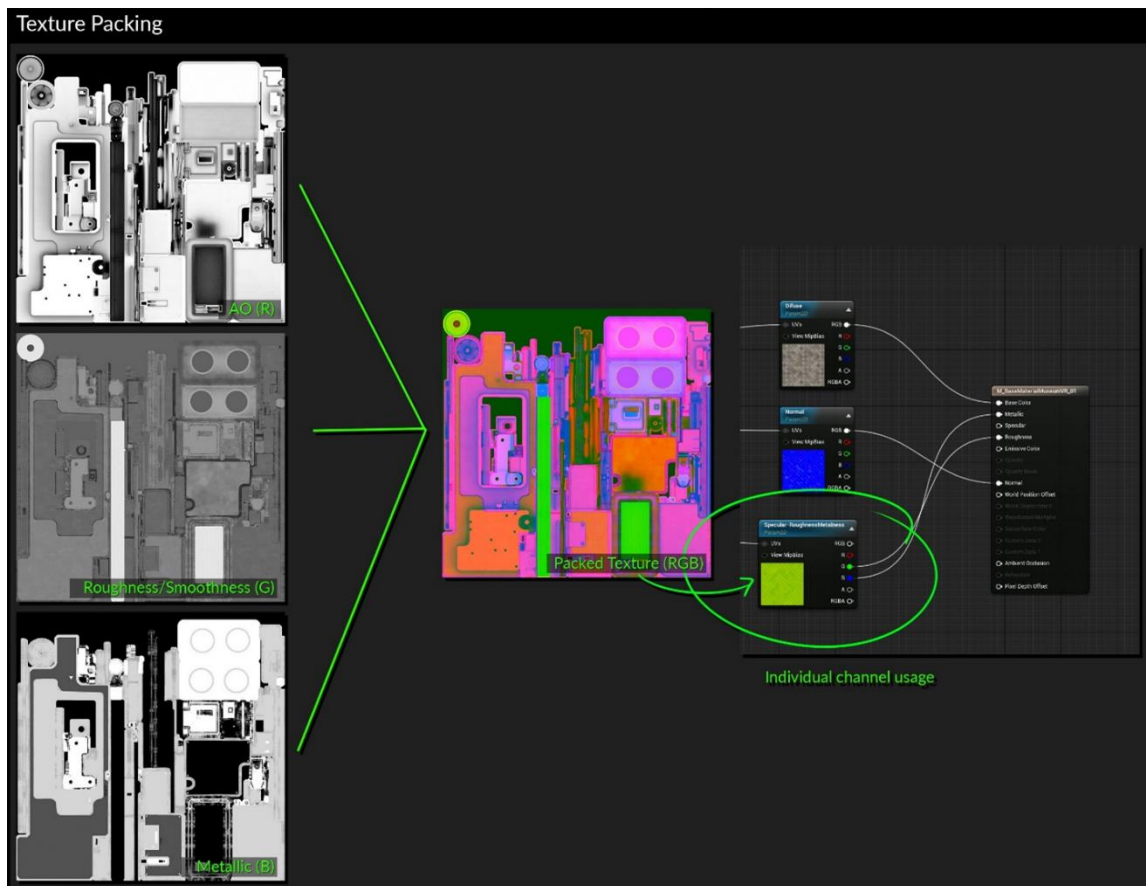


Figure 3: Texture Packing screenshot

We can also use a numerical value instead of a texture for some parameters such as metallic, roughness, or smoothness. Using a numerical value will further reduce the number of textures used. In Unreal Engine, add constant to the parameter slots to use numerical values.



## 3.2 Unlit and lit shaders

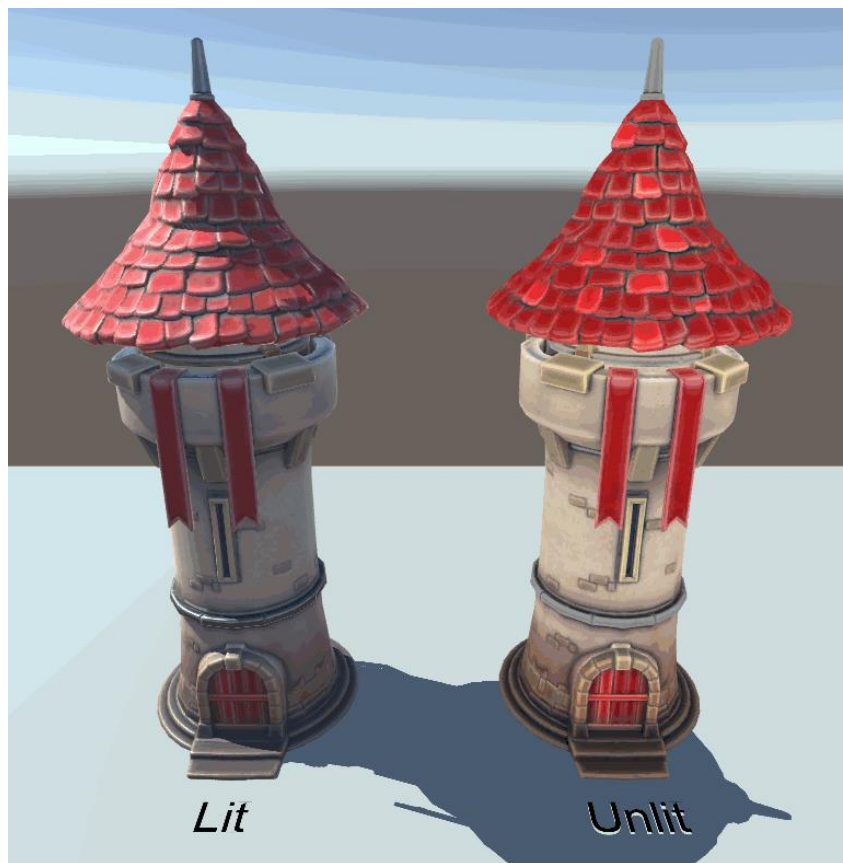
When creating a shader, we can decide how the material reacts to light. The most used shaders in mobile games are lit or unlit shaders. Use an unlit shader to reduce the number of textures used, because the material is not affected by light and will not need roughness or metallic texture.

Unlit is the fastest and cheapest shading model. Use unlit shaders if you are targeting lower-end devices.

The following tips will help you get the most benefits from shaders:

- Use an unlit shading model as much as possible, especially if the game is targeting lower-end devices.
- An unlit shading model is not affected by lighting and only outputs emissive as color.
- Because an unlit shader is not affected by light, many calculations are not needed such as specular, shadow, and so on. This reduction in calculations results in cheaper and faster rendering.
- Use stylized or cartoon art with unlit shading when making games for mobile platforms.

Default lit shaders need extra processing, compared to unlit shaders. More processing is needed for lit shaders, however the surface can be affected by light and have specularity. For most games, lit shaders is the shading model that is used, as shown in the following image:



**Figure 4: Lit shaders example**

## 4 Transparency considerations

We recommend that you avoid transparency and use an opaque material when possible. On mobile platforms, only use transparency if necessary because transparency will be more expensive than opaque.

Having many transparent objects in games affects performance when objects are rendered on top of another several times. This rendering problem is known 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, be careful when building levels and keep overdraw to a minimum. In a situation where overdraw is unavoidable (such as when using particles) make the shader as simple as possible.

Unreal Engine can visualize overdraw track the severity of overdraw in each scene, as shown in the following screenshot:

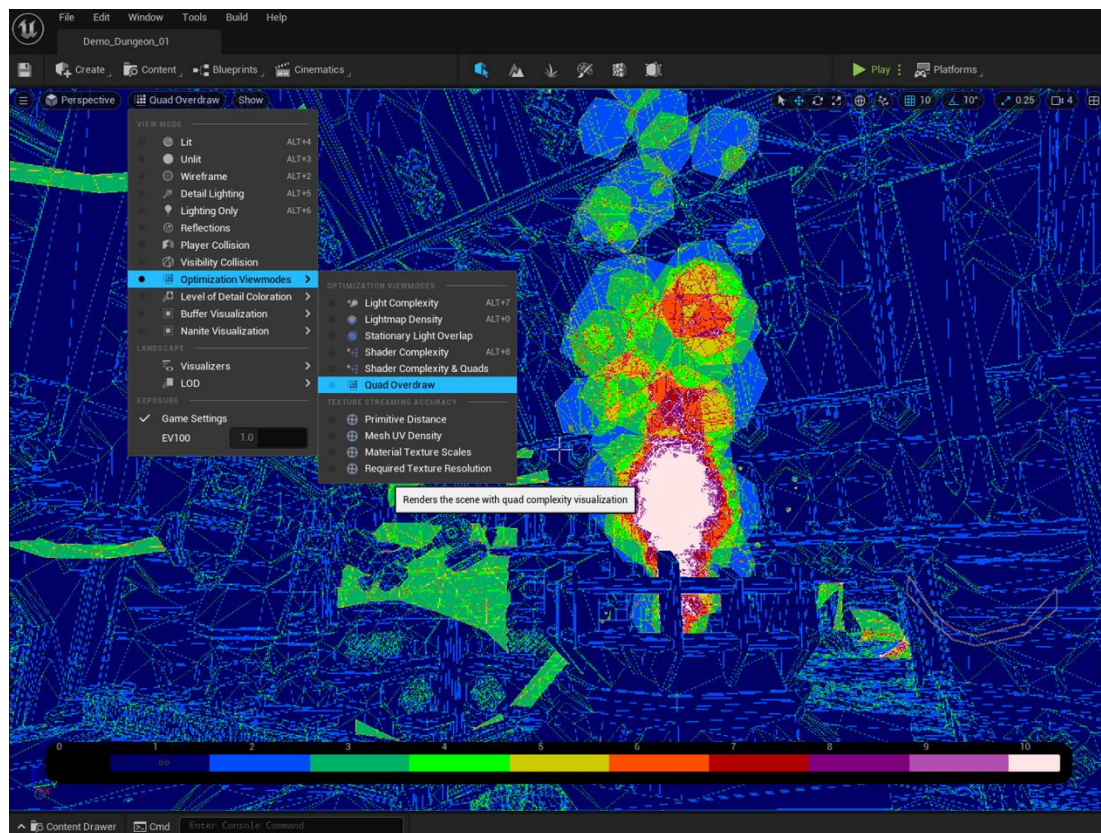


Figure 5: Overdraw example

This image visualizes overdraw. Using the key on the bottom of the image, you can see hot spots of over draw. Areas that colors are on the right side of the key need optimization.

If we use transparency in a game, profile the performance and compare between the different implementations.

The most common ways to do transparency in a shader are alpha test and alpha blend. There are best practices for each method, but the result will vary for different use cases. The best practice is to always profile the performance difference between the two methods. For mobile platforms, you can use Streamline, a tool developed by Arm to collect the performance data of a device.

## 4.1 Alpha test

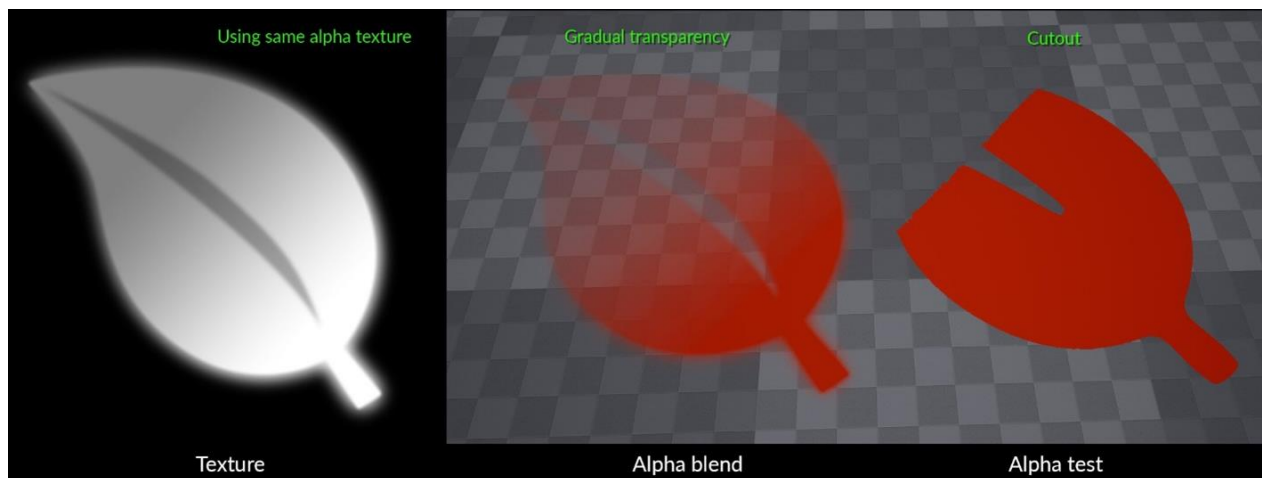
The alpha test implementation makes the object material look either 100% opaque or 100% transparent. We can also set the threshold of the cutout for the mask. In Unreal Engine, the term used is Masked blend mode. The following is a list of best practices when using alpha test:

- Avoid Masked blend mode material unless necessary. Using Masked blend mode contributes to overdraw and using this mode too much can negatively influence performance.
- Alpha test is recommended in mobile platforms because it disables some of the optimization features in the GPU. However, you can profile and compare the performance to test performance.

## 4.2 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.

A comparison of alpha blend and alpha test is shown in the following example:



**Figure 6: Alpha blend and alpha test**

Alpha blend allows partial transparency while alpha test results in a sharp cutout.

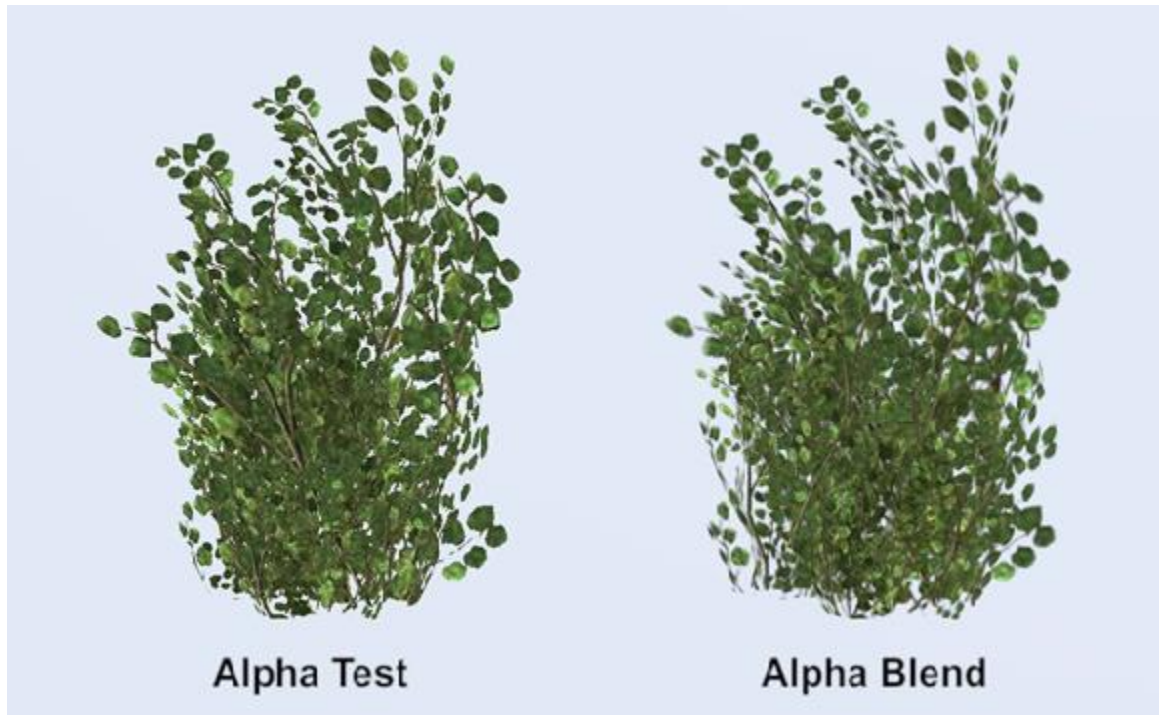
The following is a list of best practices when using alpha blend:

- In general, alpha blend and all translucencies should be avoided on mobile platforms. Using translucency contributes to overdraw, using too much can negatively influence performance.
- Avoiding adding unneeded overdraw to maintain performance.

- If we use alpha blend in games, make the coverage of the blend area small. The more area that translucency covers; the more pixels need to be redrawn.

## 4.3 Alpha test transparency for foliage

The following image shows alpha test and alpha blend transparency in foliage:



**Figure 7: Alpha test and alpha blend for foliage**

The alpha blend looks slightly better due to the soft edges compared to the sharp cut in alpha test.

However, in motion, alpha blend leaves do not seem to render in the correct order. Alpha test handles the transparency and order of the leaves better, but the edges are harsher and aliased. Generally, the alpha test visual quality will be acceptable.



## 5 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 Viewmodes in Shader Complexity to check shader complexity inside the level. This view gives estimates and provides an early indication of how expensive the shaders are. In the following example screenshot, green is in the good range and red is more complex:

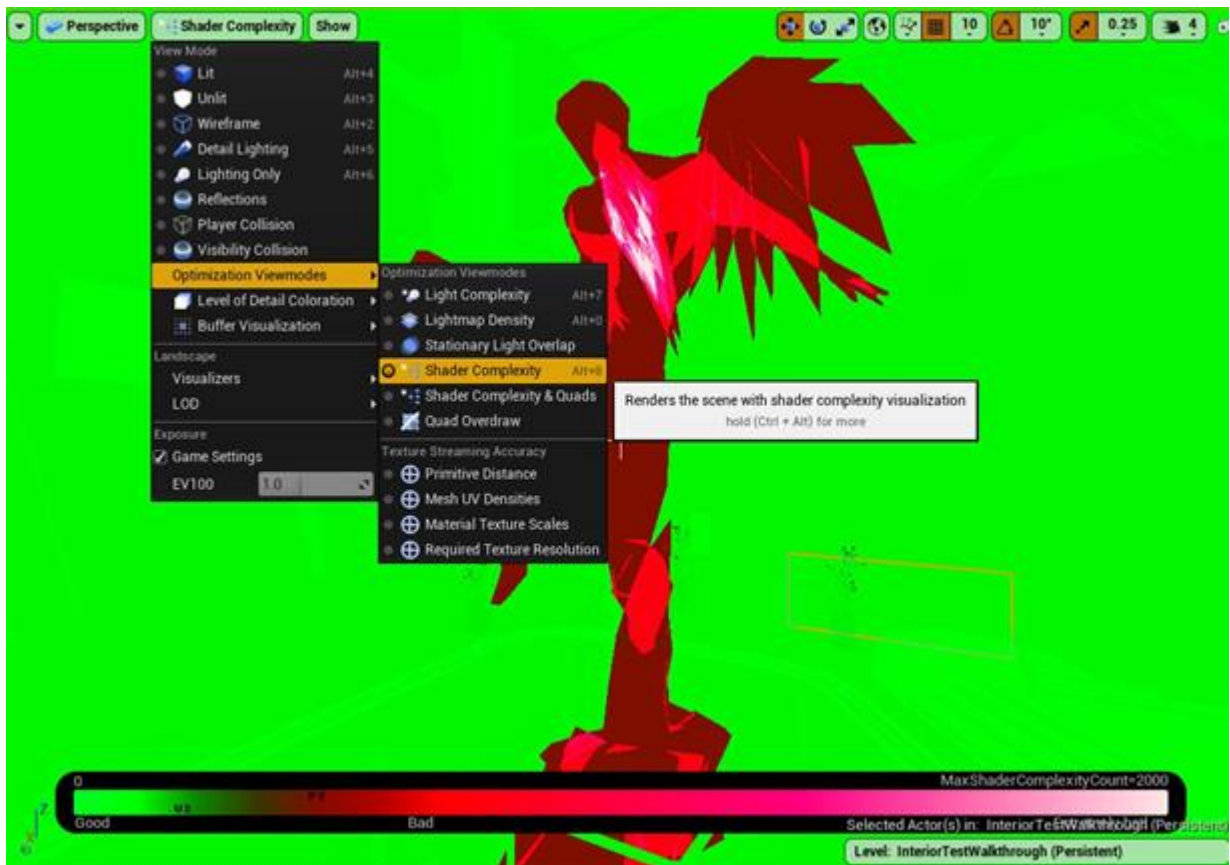


Figure 8: Shader Complexity menu

In Unreal Engine, use the material stats feature to quickly profile the materials early in the process, as shown in the following screenshot:

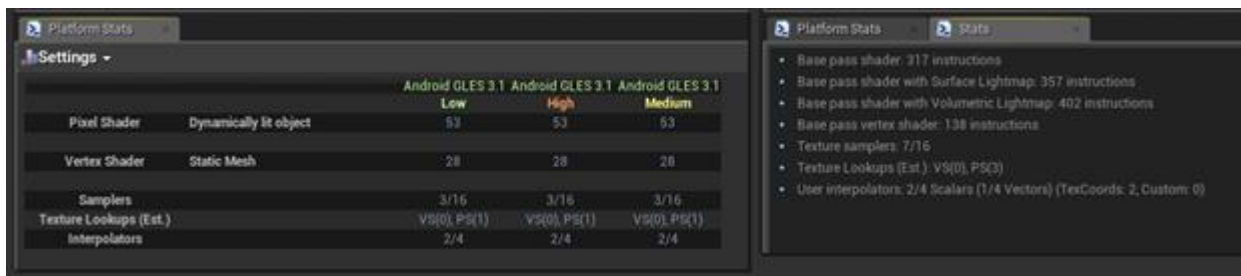


Figure 9: Material stats example

This view mode is a good guideline for artists. To look further into profiling, you can use tools such as Streamline for Arm Mali. These tools require further graphics knowledge to understand more about how the GPU works.

## 5.1 Do operations in vertex shader

The most common shader used in projects is a combination of vertex shader and pixel shader. Vertex shader works on every vertex 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 where possible.

When you move this operation to vertex shader, you also move the processed data to fragment. This data shared between the vertex and fragment shaders are called varyings. We must pay attention to the tiler in case it then becomes an obstruction.

We need to do further profiling after working on optimizations using the Customized UVs feature in Unreal Engine. Then we connect nodes and use a Texture Coordinate node in the pixel shader to target the customized UV. For more information, see the Unreal Engine documentation on [Customized UVs](#).

## 5.2 Reduce the number of math operations

We use mathematical operations in the shader to customize the look and behavior. For example, multiplication, addition, power, floor, and 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, and 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 devices with GLES 2.0.

## 5.3 Performance profiling

As a general best practice, never make assumption and do some profiling to understand where the obstructions are in an app. Further profiling is also recommended to compare the effects of any optimizations.

## 6 Related information

The following is related to material in this guide:

- [Unreal Engine documentation](#)
- [Use Streamline to Optimize Applications for Mali GPUs](#)



# 7 Next steps

This guide has introduced you to using materials and shaders in the Unreal Engine game engine. You learned about shaders for mobile platforms, the difference between transparency methods, and how to reduce shader complexity.

After reading this guide, you can use these best practices to optimize the performance of your apps on mobile devices that use Unreal Engine.