



Unreal Engine

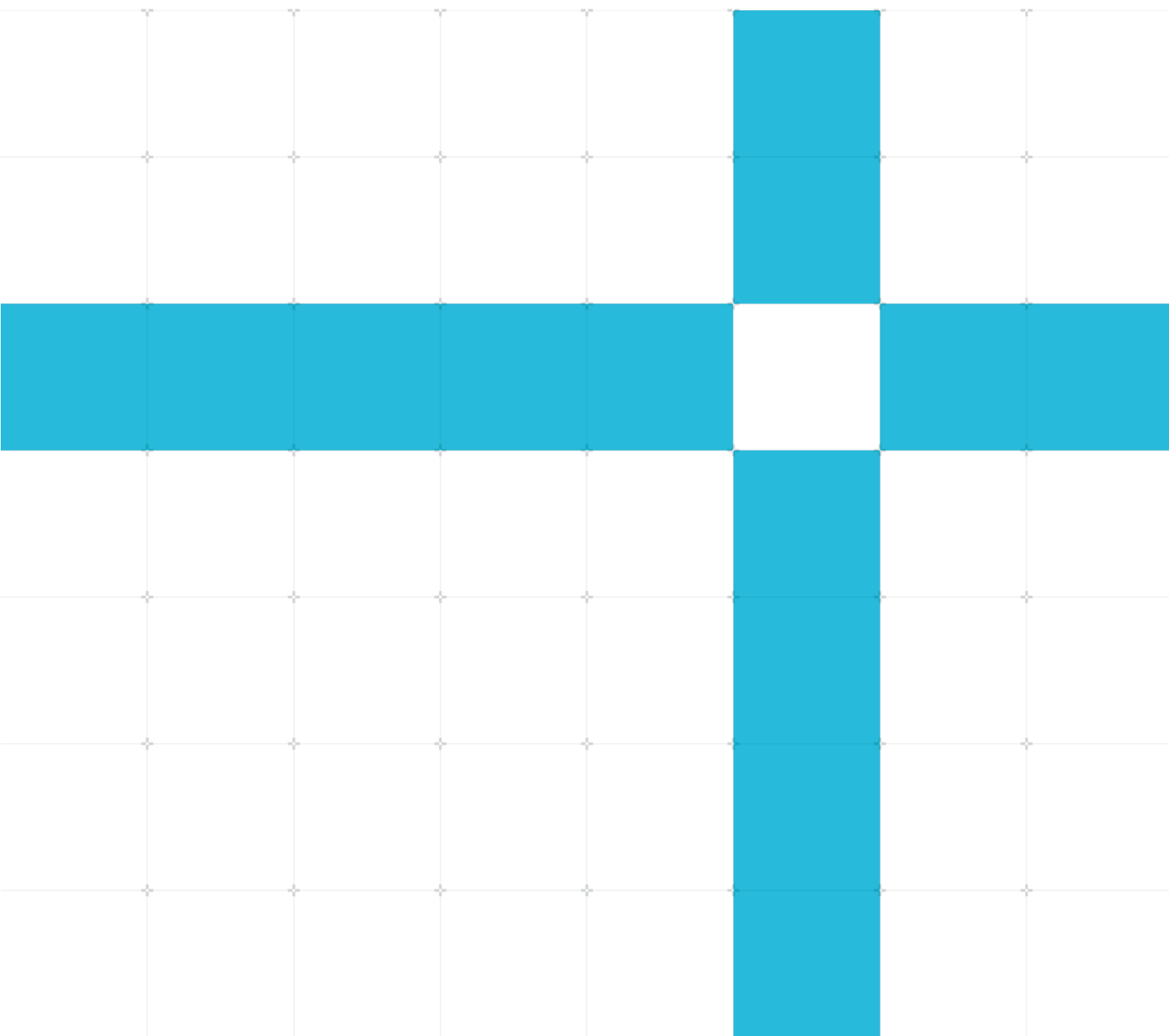
Textures best practices for Unreal Engine

Non-Confidential

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

Issue 01

102696



Unreal Engine

Textures best practices for Unreal Engine

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

Release information

Document history

Issue	Date	Confidentiality	Change
01	27-Sept-2021	Non-Confidential	First issue

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, 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.

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 <https://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

developer.arm.com

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.

Contents

1	Overview	5
2	Texture atlasing	6
3	Texture filtering	7
3.1	Texture filtering best practices.....	8
4	Texture mipmapping	10
4.1	Mipmapping best practices	10
5	Texture size, color space, and compression.....	12
5.1	Texture size.....	12
5.2	Texture color space	12
5.3	Texture compression	14
5.4	Texture compression best practices	14
6	UV unwrapping.....	17
7	Visual impact	19
8	Texture channel packing	22
9	Alpha channels.....	24
10	Normal map best practice.....	25
11	Normal map baking best practice	26
12	Related information.....	29
13	Next steps.....	30

1 Overview

Textures are an integral part of a game. It is an area that artists have direct control to improve the performance of games. This best practice guide describes several texture optimizations that can help your games to run more smoothly and look better.

The overall goal of the best practices series of guides is to help you make your games perform better on mobile based platforms.

The contents of this guide include the following:

- Texture atlasing
- Texture filtering
- Texture mipmap
- Texture size
- Texture color space
- Texture compression
- UV unwrap
- UV visual
- UV channel packing
- Other best practices related to textures.

2 Texture atlasing

A texture atlas is an image that contains data from several smaller images that have been packed together. Instead of having one texture for one mesh, you have a larger texture that several meshes share.

A texture atlas can be created before making the asset, which means that the asset is [UV unwrapped](#) according to the texture atlas. This requires some early planning when creating the texture.

The texture atlas can also be created after the asset is finished by merging textures in painting software. However, this also means that the UV islands must be rearranged according to the texture.

A UV island is a connected group of polygons in a texture map.

The following image highlights several 3D objects using one texture set:

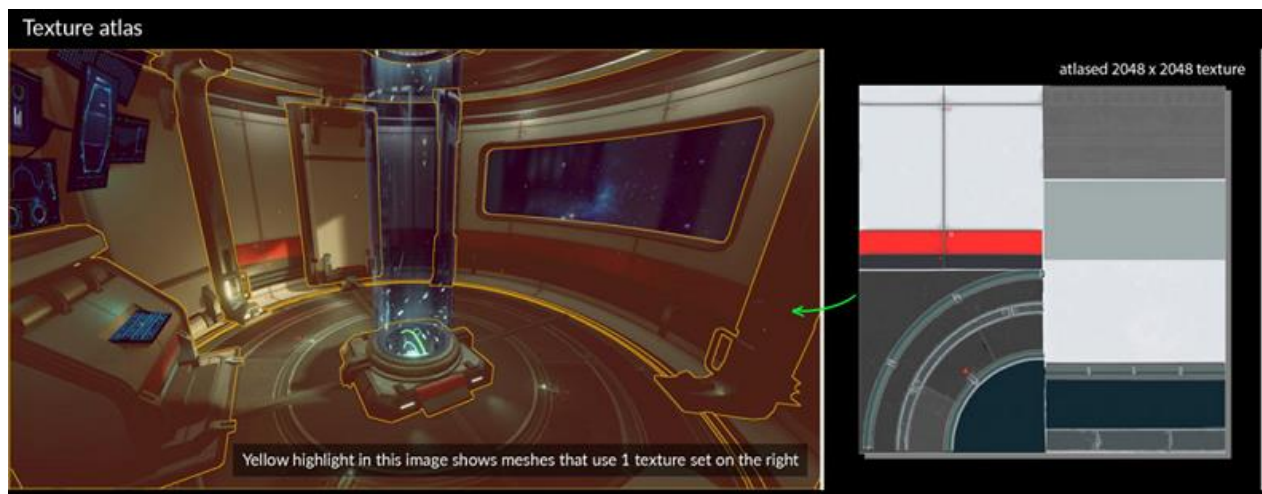


Figure 1 Texture atlas

Texture atlasing enables batching for several static objects that share this texture atlas and the same material. Batching reduces the number of draw calls. Fewer draw calls results in better performance when the game is CPU-bound.

In Unreal Engine, batching needs to be done manually. You can perform batching either by merging objects in a 3D software tool, or by using the UE4 Actor Merging tool. The UE4 Actor Merging tool also creates the texture atlas automatically. Further information can be found in [Actor Merging](#) in the Unreal Engine documentation.

Texture atlasing also requires fewer textures inside the game as they are packed together. With compression, this helps to keep the memory cost of textures down.

3 Texture filtering

Texture filtering is a method that is used to improve the texture quality in a scene. Without texture filtering, artifacts like aliasing generally look worse.

Texture filtering makes textures look better and less blocky. Usually, this makes the game look better.

However, texture filtering can also degrade performance. This is because better quality often means that more processing is required. Finding a good balance between performance and visual quality is important.

Texture filtering can account for up to half of the GPU energy consumption. Choosing simpler texture filters can reduce the energy demands of an application.

There are several options available in popular game engines for texture filtering:

- Nearest, or Point filtering.

When seen up close, nearest filtering makes the texture look blocky. This is the simplest and cheapest texture filtering.

- Bilinear filtering.

The texture is blurrier up close with bilinear filtering. The four nearest texels are sampled and then averaged to color the main pixel. Unlike nearest filtering, bilinear filtering results in less blocky pixels as the pixels have a smooth gradient, as shown in the following image:

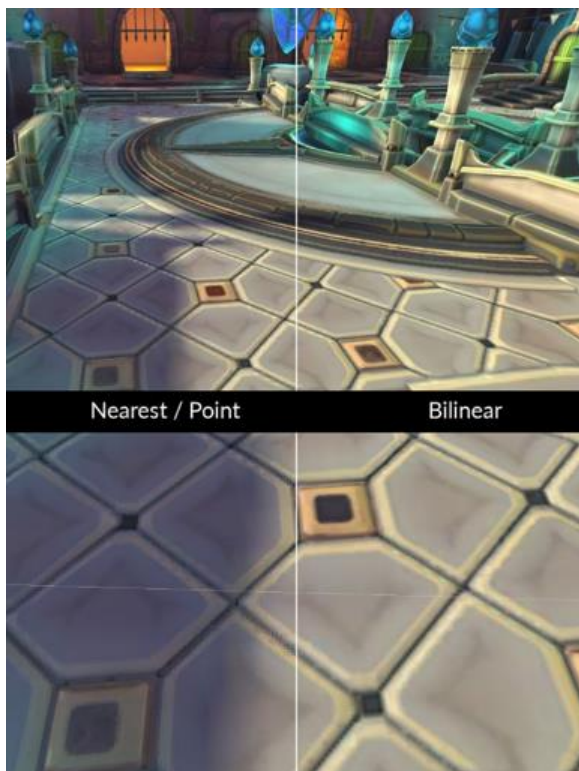
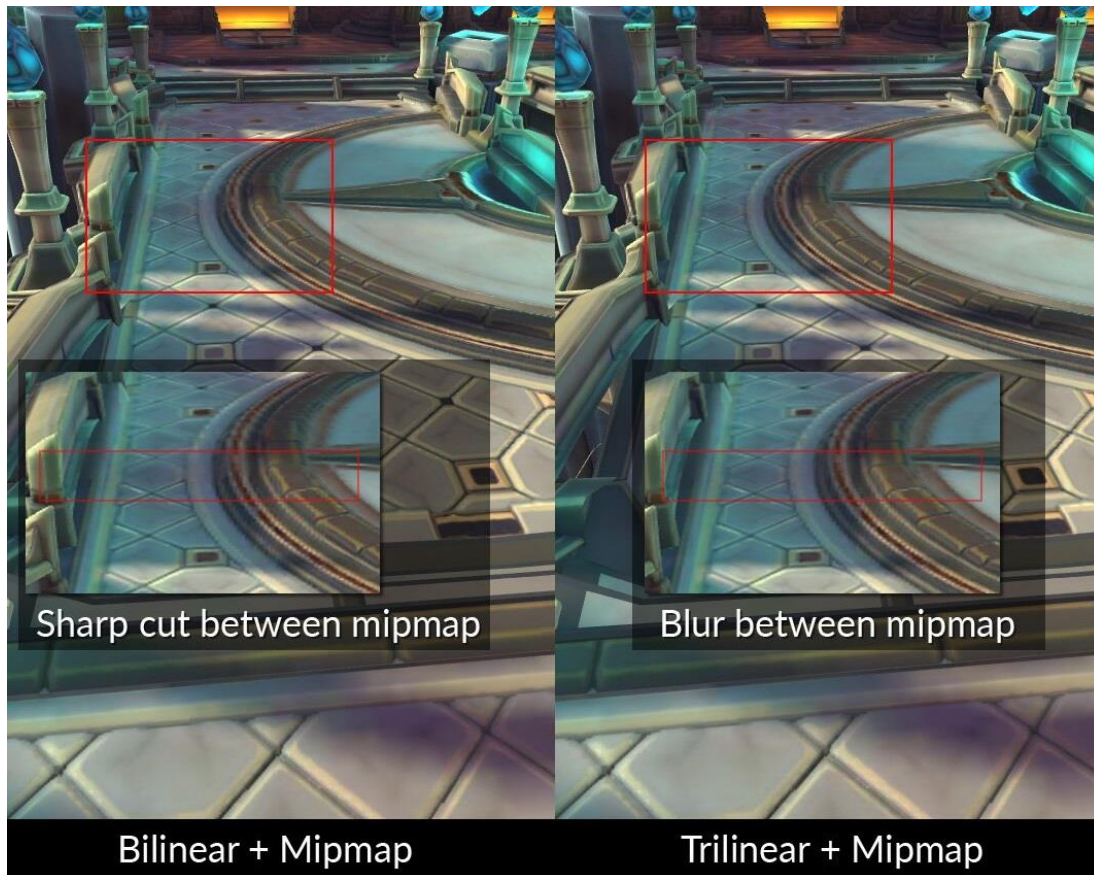


Figure 2 Texture filtering – bilinear filtering

- Trilinear filtering.

Trilinear filtering is like bilinear filtering, but with added blend between mipmap levels. Trilinear filtering removes noticeable changes between mipmaps by smoothing the transition between mipmaps, as shown in the following image:

**Figure 1 Texture filtering – bilinear compared to trilinear**

- Anisotropic filtering

Anisotropic filtering makes textures look better when viewed at an angle. It is good for ground level textures, for example.

3.1 Texture filtering best practices

Arm recommends that you try the following texture filtering tips:

- Use bilinear filtering for a balance between performance and visual quality.
- Use trilinear filtering selectively. This is because trilinear filtering requires more memory bandwidth than bilinear filtering.

- A combination of filters may look and perform better than using trilinear filtering.

4 Texture mipmapping

Mipmaps are copies of the original texture that are saved at lower resolutions. You can think of mipmapping like the equivalent of Level of Detail (LOD), but for textures.

Based on how much texture-space a fragment occupies, you can select an appropriate level for sampling. When an object is further from the camera, a lower resolution texture is applied. A higher resolution texture is applied when an object is closer the camera. The following image shows a mipmap collection containing the same texture at different resolutions:

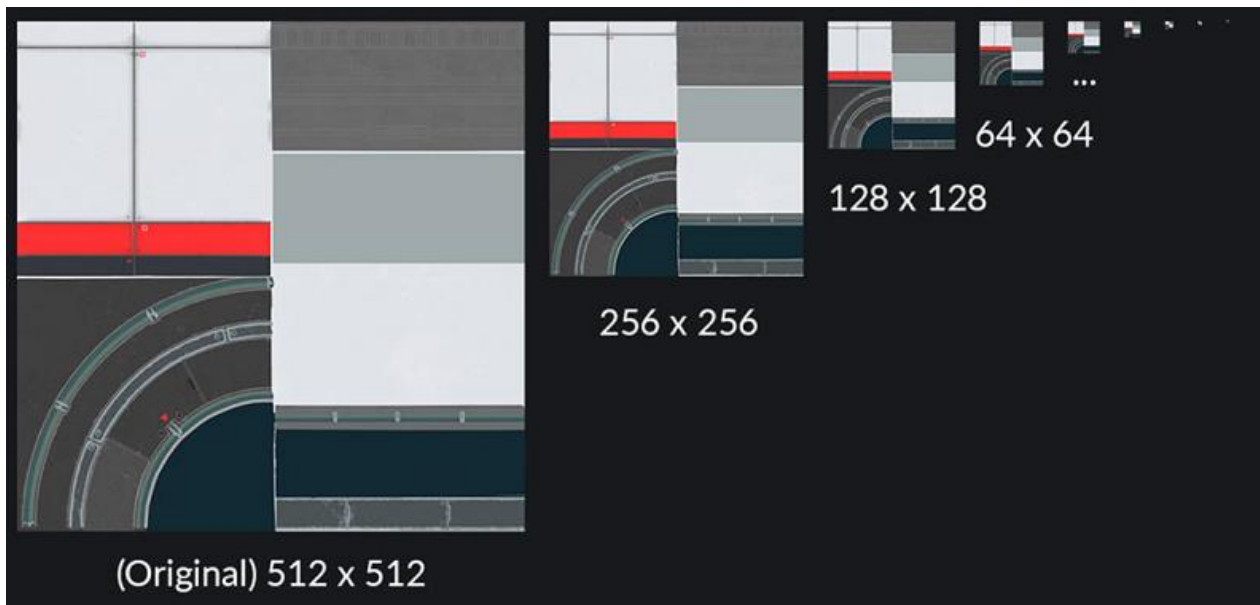
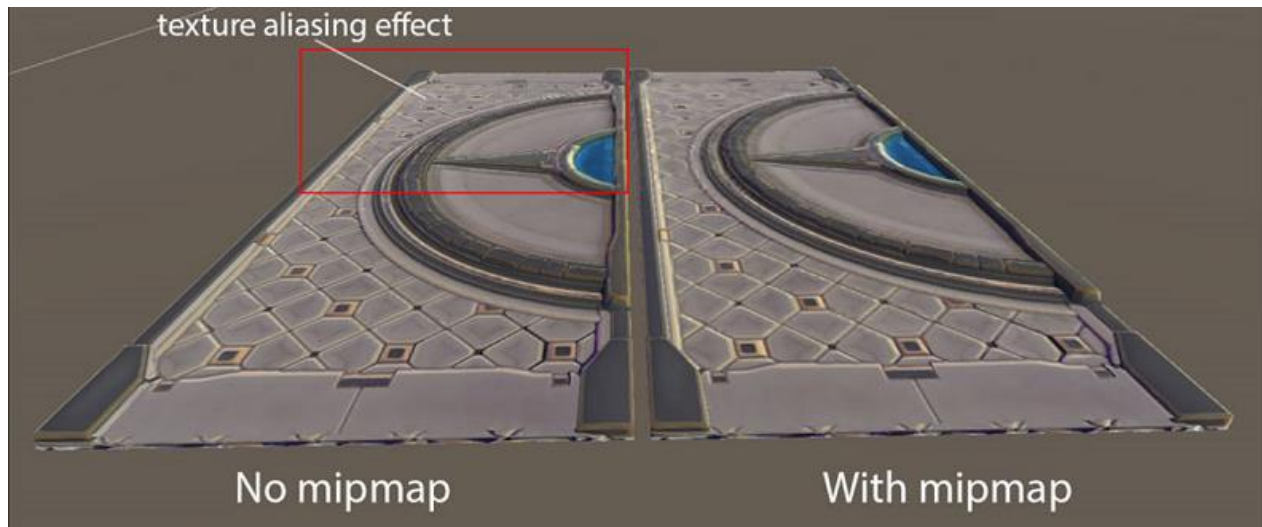


Figure 2 Mipmap chain

4.1 Mipmapping best practices

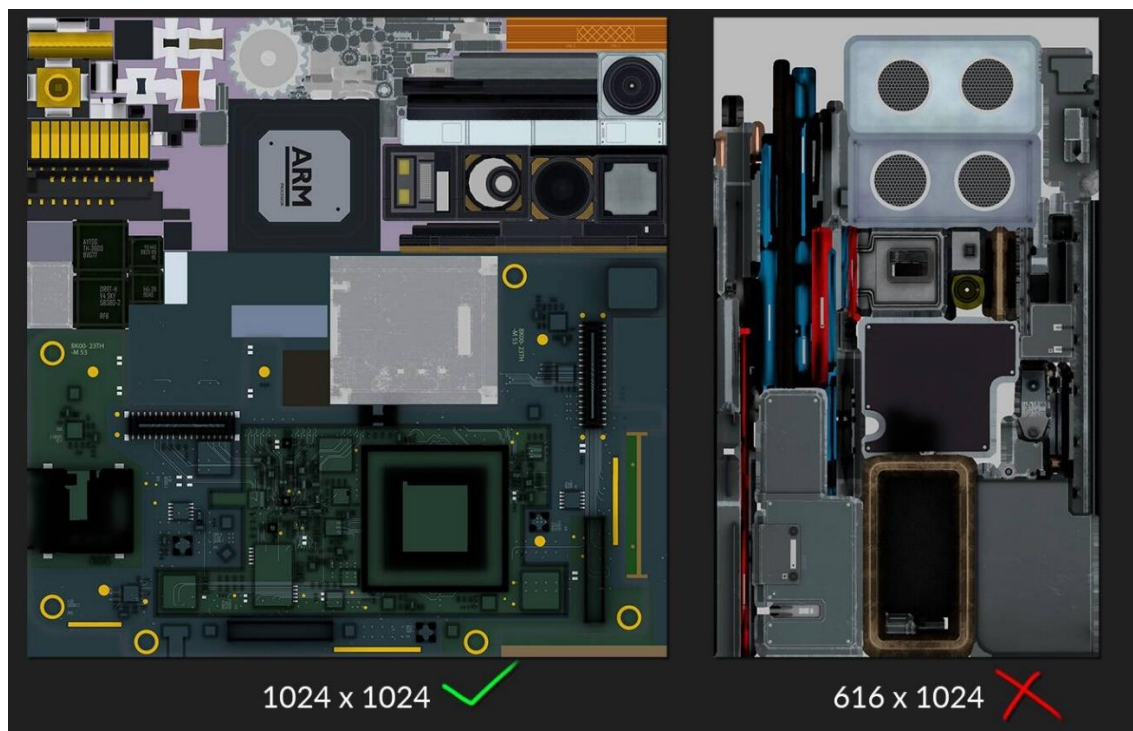
Make sure that you use mipmapping. This is because mipmapping improves both performance and quality. Mipmapping improves the performance of the GPU. This is because the GPU does not need to render the full resolution texture on an object that is far away from the camera.

Mipmapping also reduces the texture aliasing and improves the final image quality. Texture aliasing causes a flickering effect on areas that are further from the camera. The following image shows texture aliasing on the same texture both with and without mipmapping:

**Figure 3 Mipmap comparison**

In Unreal Engine, to generate a mipmap use a texture ratio with a power of 2, for example 512x1024, 128x128, 2048x2048, and so on. Unreal Engine does not generate a mipmap chain when texture ratios are not a power of 2. Textures do not need to be square, for example Unreal Engine will generate a mipmap for a 512 x 1024 texture.

The following image shows how a mipmap is generated for a 1024 x 1024 texture, but not for a 616 x 1024 texture:

**Figure 4 Texture with power of 2 ratio for Unreal**

5 Texture size, color space, and compression

Texture size, color space, and compression can all have an impact on the performance of your game.

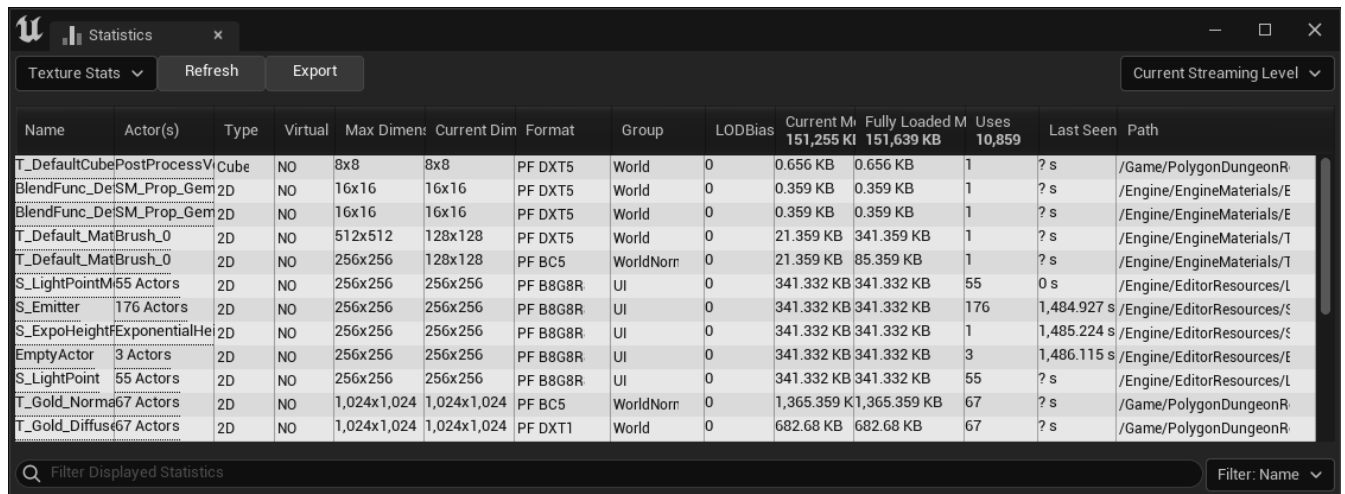
5.1 Texture size

Only create textures that are large enough to meet the required quality, but no bigger. It is also best practice to use a large-size texture atlas that contains several textures that are shared between many meshes.

Textures can be different sizes. Reducing the size of certain textures that require less detail helps to reduce bandwidth. For example, a diffuse texture might be 1024x1024 pixels while a roughness, or metallic, map might be 512x512 pixels.

Selectively reduce the texture size and check whether visual impact has been degraded afterwards.

Unreal Engine provides the Statistics tool which can help you manage your textures sizes. The Statistics tool shows data about the different assets in your project. This helps you check that your assets stay within budget, and quickly identify assets that are not within budget. The following image shows the Statistics tool:



The screenshot shows the Unreal Engine Statistics tool window. It has a title bar with the Unreal logo and a close button. Below the title bar is a toolbar with 'Texture Stats', 'Refresh', and 'Export' buttons. On the right, there is a 'Current Streaming Level' dropdown menu. The main area is a table with the following columns: Name, Actor(s), Type, Virtual, Max Dimen, Current Dim, Format, Group, LODBias, Current M, Fully Loaded M, Uses, Last Seen, and Path. The table lists various textures and materials, including 'T_DefaultCubePostProcessV', 'BlendFunc_DeSM_Prop_Gem', 'T_Default_MatBrush_0', 'S_LightPointM55 Actors', 'S_Emitter', 'S_ExpoHeightExponentialHe', 'EmptyActor', 'S_LightPoint', 'T_Gold_Norma67 Actors', and 'T_Gold_Diffus67 Actors'. Each row shows details about the texture, such as its dimensions, format, and usage statistics.

Name	Actor(s)	Type	Virtual	Max Dimen	Current Dim	Format	Group	LODBias	Current M	Fully Loaded M	Uses	Last Seen	Path
T_DefaultCubePostProcessV		Cube	NO	8x8	8x8	PF DXT5	World	0	0.656 KB	0.656 KB	1	? s	/Game/PolygonDungeonR
BlendFunc_DeSM_Prop_Gem		2D	NO	16x16	16x16	PF DXT5	World	0	0.359 KB	0.359 KB	1	? s	/Engine/EngineMaterials/E
BlendFunc_DeSM_Prop_Gem		2D	NO	16x16	16x16	PF DXT5	World	0	0.359 KB	0.359 KB	1	? s	/Engine/EngineMaterials/E
T_Default_MatBrush_0		2D	NO	512x512	128x128	PF DXT5	World	0	21.359 KB	341.359 KB	1	? s	/Engine/EngineMaterials/T
T_Default_MatBrush_0		2D	NO	256x256	128x128	PF BC5	WorldNorr	0	21.359 KB	85.359 KB	1	? s	/Engine/EngineMaterials/T
S_LightPointM55 Actors		2D	NO	256x256	256x256	PF B8G8R	UI	0	341.332 KB	341.332 KB	55	0 s	/Engine/EditorResources/L
S_Emitter	176 Actors	2D	NO	256x256	256x256	PF B8G8R	UI	0	341.332 KB	341.332 KB	176	1,484.927 s	/Engine/EditorResources/€
S_ExpoHeightExponentialHe		2D	NO	256x256	256x256	PF B8G8R	UI	0	341.332 KB	341.332 KB	1	1,485.224 s	/Engine/EditorResources/€
EmptyActor	3 Actors	2D	NO	256x256	256x256	PF B8G8R	UI	0	341.332 KB	341.332 KB	3	1,486.115 s	/Engine/EditorResources/€
S_LightPoint	55 Actors	2D	NO	256x256	256x256	PF B8G8R	UI	0	341.332 KB	341.332 KB	55	? s	/Engine/EditorResources/L
T_Gold_Norma67 Actors		2D	NO	1,024x1,024	1,024x1,024	PF BC5	WorldNorr	0	1,365.359 KB	1,365.359 KB	67	? s	/Game/PolygonDungeonR
T_Gold_Diffus67 Actors		2D	NO	1,024x1,024	1,024x1,024	PF DXT1	World	0	682.68 KB	682.68 KB	67	? s	/Game/PolygonDungeonR

Figure 5 Unreal Engine Statistics tool

5.2 Texture color space

Most texturing software, for example Adobe Photoshop or Substance Painter, works and exports using the sRGB color space.

We recommend that you try the following:

- Use diffuse textures in the sRGB color space.
- Textures that are not processed as color must not be in the sRGB color space. Examples include metallic, roughness, and normal maps. This is because of the following reasons:
 - ◆ Maps are used as data, and are not used as color.
 - ◆ Using sRGB in these maps results in the wrong look, or visual, on the material.



The **sRGB (Color Texture)** setting in the **Inspector** window for the texture must not be ticked for roughness, specular, normal maps, or similar, as shown in the following image.

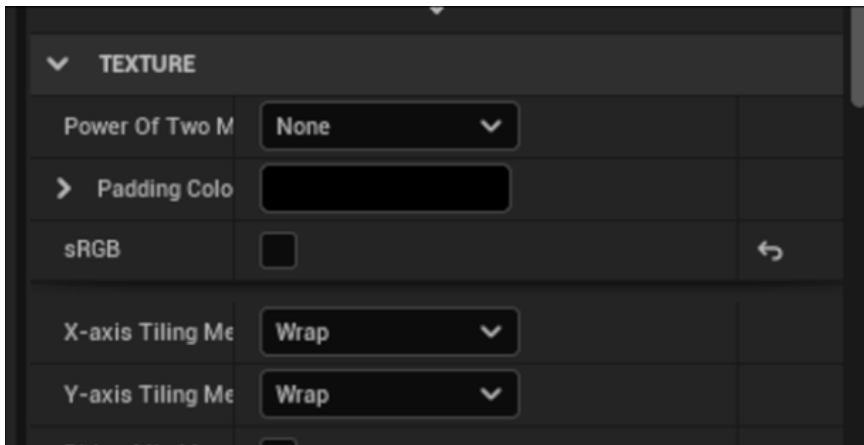


Figure 6 The sRGB (Color Texture) setting in the Inspector window

The following screenshot shows you what happens when sRGB is incorrectly applied to such a texture:

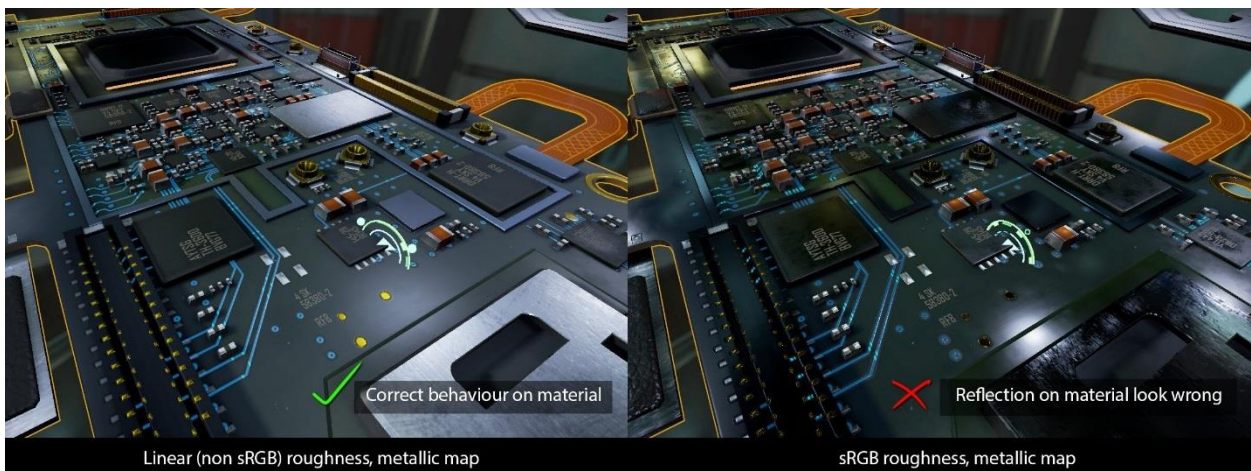


Figure 7 sRGB application on texture comparison

5.3 Texture compression

Texture compression is an image compression that is applied to reduce texture data size, while minimizing the loss in visual quality. In development, we export textures using a common format, like TGA, or PNG. These formats are more convenient to use, and major image software programs support them.

These formats must not be used in final rendering. This is because they are slower to both access and sample, when compared to specialized image formats. For Android, there are several options, like Adaptive Scalable Texture Compression (ASTC) and Ericsson Texture Compression (ETC) 2.

5.4 Texture compression best practices

We recommend that you use the ASTC technology that Arm created. Here several reasons to use ASTC:

- ASTC gives better quality with the same memory size as ETC.
- ASTC produces the same quality with a smaller memory size than ETC.
- ASTC takes longer to encode than ETC. This means that game packaging process can take more time. If this is an issue, then it is better to use ASTC on the final packaging of the game.
- ASTC allows for more control in terms of quality, because ASTC allows the block size to be set. While there is no single best default for block size, setting the block size to 5x5 or 6x6 pixels is a good starting point.

Sometimes, it might be faster to use ETC for development if you have to quickly deploy your game on a device. You can use ASTC with fast compression settings to get around the increase in deployment time. When encoding, there are options to trade off speed against both quality and size. For the final build, ASTC is the best option in terms of balance between visual quality and file size.

The game engine handles the texture compression when you package the game. However, you can choose which compression technique to use. You must choose the format to use, so it is difficult to skip this step.

Unreal Engine provides a feature to package the game with all texture formats, then choose at run time which format to use based on the device. This results in larger packaged files, but avoids compatibility issues.

To use ASTC in Unreal Engine, use the **Platforms** settings. Under **Texture Format Properties** set **ASTC texture format priority** to a high value such as 0.8 or 1. This ensures your project uses ASTC over ETC if it is supported on devices. The following screenshot shows how to select ASTC when building an Android package in Unreal Engine:

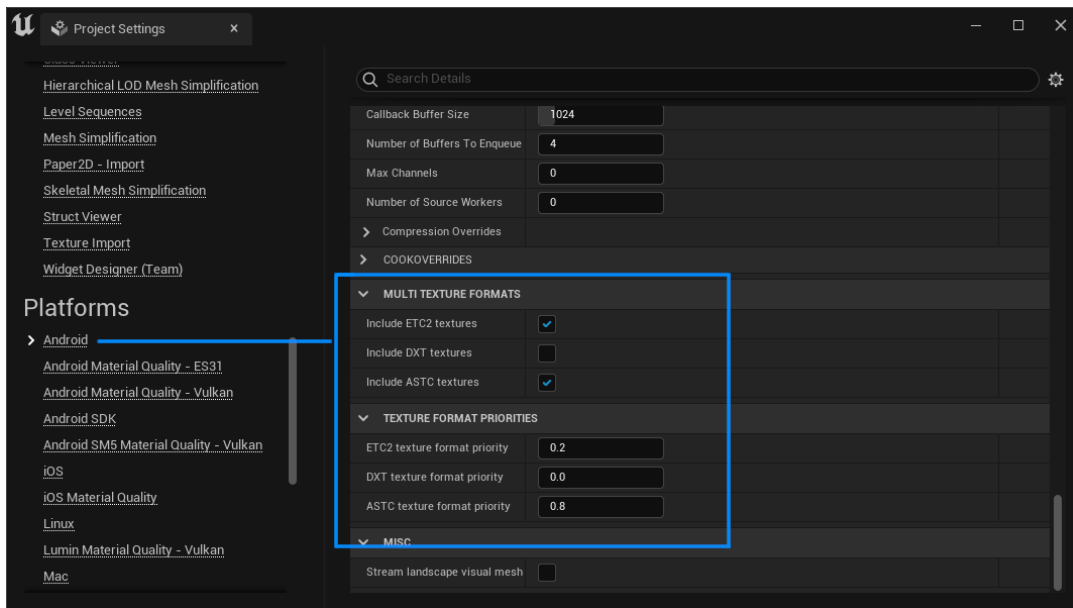


Figure 8 ASTC texture format priority setting

For control over the ASTC block sizes, go to the **Cooker** settings in the **Project** settings. On the **TEXTURES** tab, configure the **Speed** and **Size** settings for your project's compression. We suggest starting with 1 for **Speed** and 3 for **Size**, but experiment to discover what works best for your project.

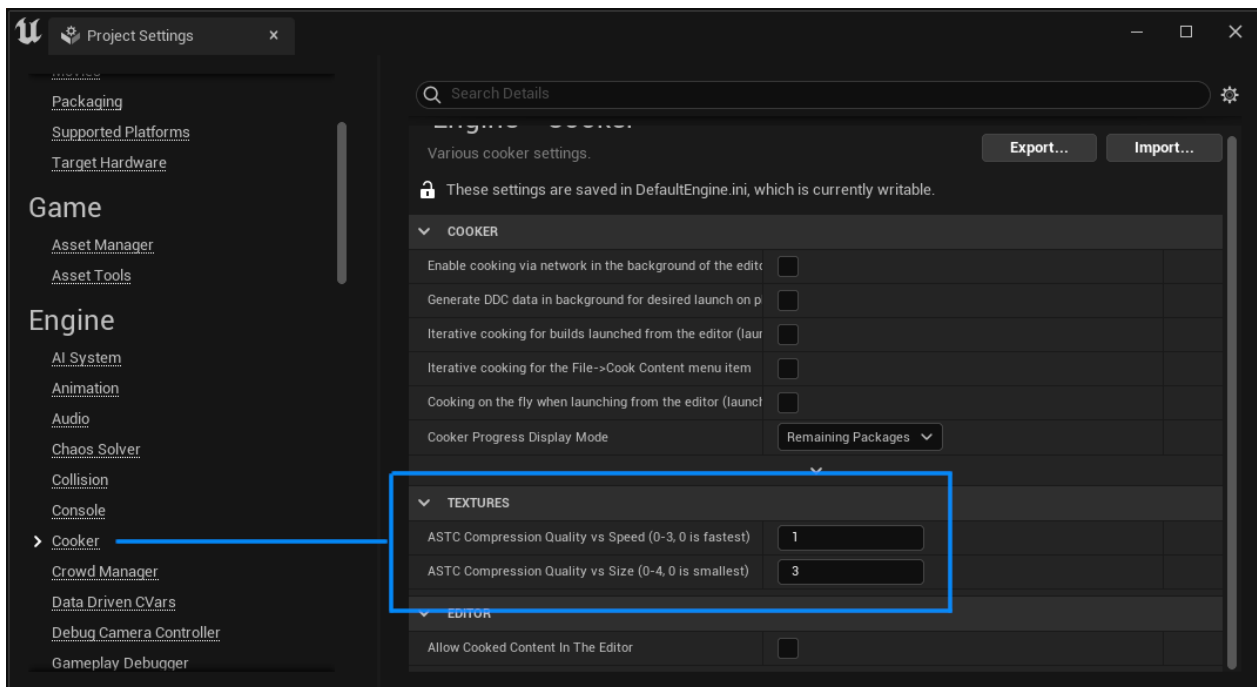


Figure 9 ASTC compression quality settings

The following image shows both the difference in quality, and the respective file size between ETC and ASTC compression:

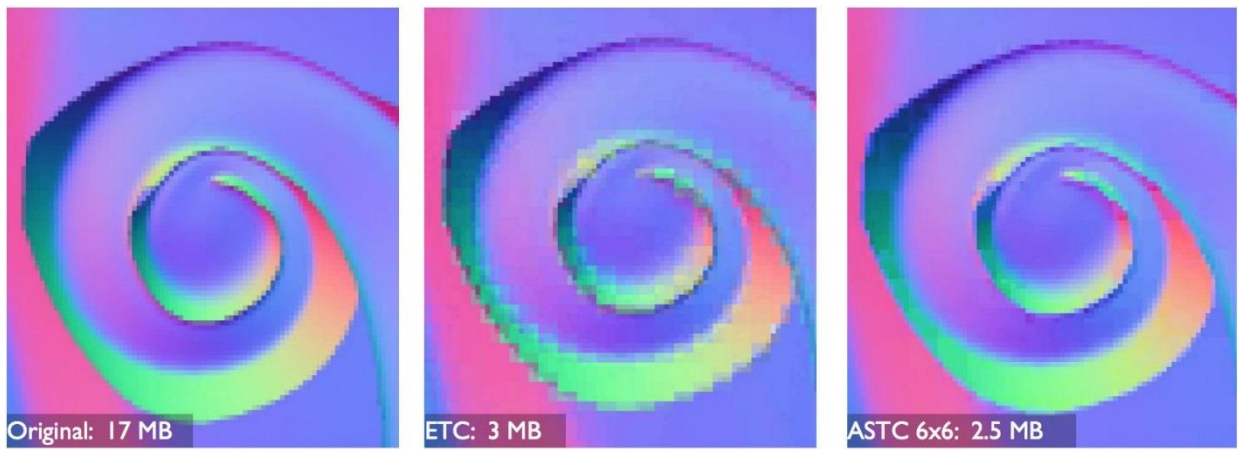


Figure 10 Image compression comparison between ETC and ASTC

6 UV unwrapping

UV unwrapping is the process of creating a UV map. A UV map projects 2D textures onto the surface of a 3D model.

It is best practice to keep UV islands as straight as possible.



A UV island is a connected group of polygons in a texture map.

The reasons for keeping UV islands as straight as possible are:

- It makes packing UV islands easier and less space is wasted.
- A straight UV helps to reduce the [staircase effect](#) happening on textures.
- On mobile platforms, texture space is limited. This is because the texture size is usually smaller on a mobile platform than on a games console or a PC. Good UV packing ensures that you get the best resolution from your texture.
- You might consider having a slightly distorted UV by keeping the UV straight, in order to have better quality texture overall.

Place UV seams in places that make them not too visible, for visual quality purposes. This is because the texture seam can look bad on a model. Therefore, split UV islands where the edges are sharp and have a small space between the UV islands. This helps later on to create better normal maps through the baking process.

The following image shows an example of how to use UV unwrapping to maximize texture space:

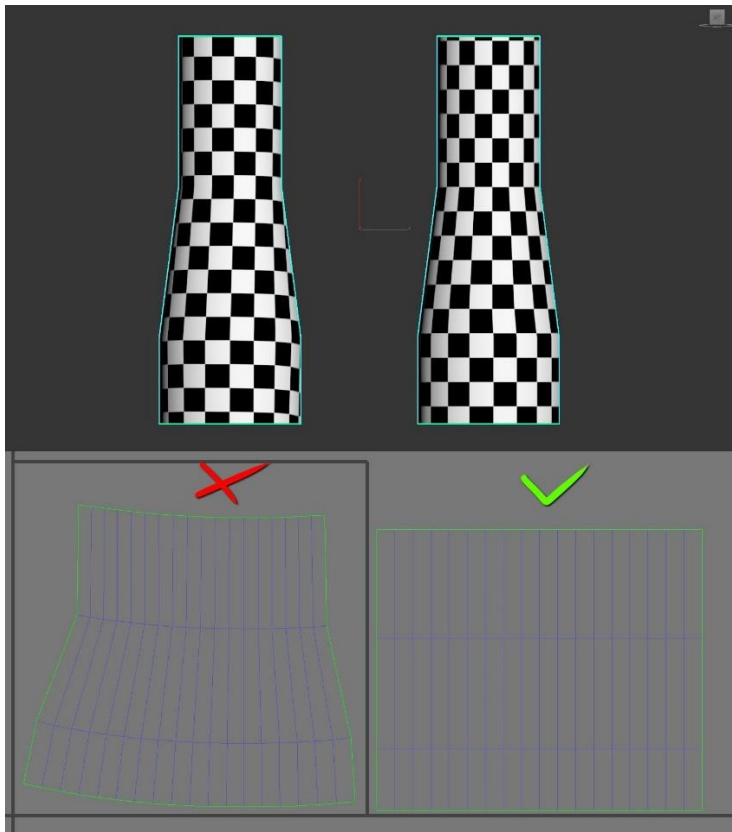


Figure 11 UV unwrapping to maximize texture space

7 Visual impact

The detail you include when creating textures should be proportionate to the visual impact of that detail. Make sure that you only create details that can be seen.

Phone screens are small, therefore fine-grained detail is not visible. Take this into account when creating textures. For example, you do not need a 4K texture with lots of details for a chair that is barely visible in the corner of the room.

The following screenshot shows an example. The small 256x256 pixel texture on the left has a low level of detail to be used on soldier characters. The larger image on the right shows the entire scene, and shows that the low level of texture detail is sufficient for the required amount of detail:



Figure 12 Small texture with no excessive small detail

In certain cases, you need to exaggerate and highlight edges and shading to improve shape readability. Because mobile platforms generally use smaller textures, it might be hard to capture all of the detail that is needed within this small texture.

Use fewer textures and bake in any extra details into one texture. This is important because:

- Phone screens are small, and some details are better to be baked onto the diffuse texture itself to make sure that those details are visible.
- Elements like ambient occlusion and small specular highlights can be baked in and then added to the diffuse texture.

This approach means that you do not have to rely too much on shader and engine features to get specular and ambient occlusion.

The following screenshot shows an example of details that have been baked into a texture, for example the highlights on roof tiles:



Figure 13 Details baked into texture

When possible, use grayscale textures that allow color tinting in the shader. This saves texture memory at the cost of creating a custom shader to perform the tinting.

Be selective with this technique, because not all objects look good using this method. It is easier to apply this technique to an object that has a uniform, or similar, color.

You can also use RGB masks and then apply textures that are based on the color range of the mask to achieve this effect.

The following image shows an example of a grayscale texture that is being applied to a tinted pillar:

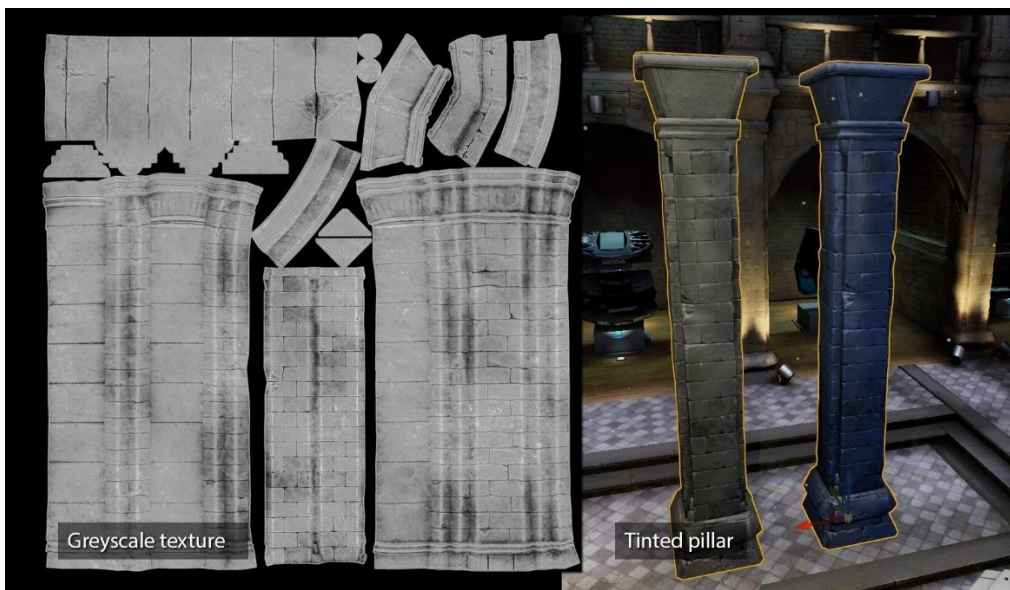


Figure 14 Grayscale texture applied to a tinted pillar

8 Texture channel packing

Use texture channels to pack multiple textures into one.

Arm recommends texture channel packing because packing helps to save texture memory. Packing saves memory because you can get three maps into one texture using this technique. This means that fewer texture samplers are required.

This texture packing technique is commonly used to pack roughness, or smoothness, and metallic into one texture. But it can be applied for any texture map.

Use the green channel to store the more important mask. The green channel usually has more bits. This is because our eyes are more sensitive to green and less sensitive to blue. The roughness/smoothness map will usually have more detail than metallic and should be placed in the green channel. Set the texture to linear/RGB instead of sRGB color space for these maps.

The following image shows an example of texture packing:

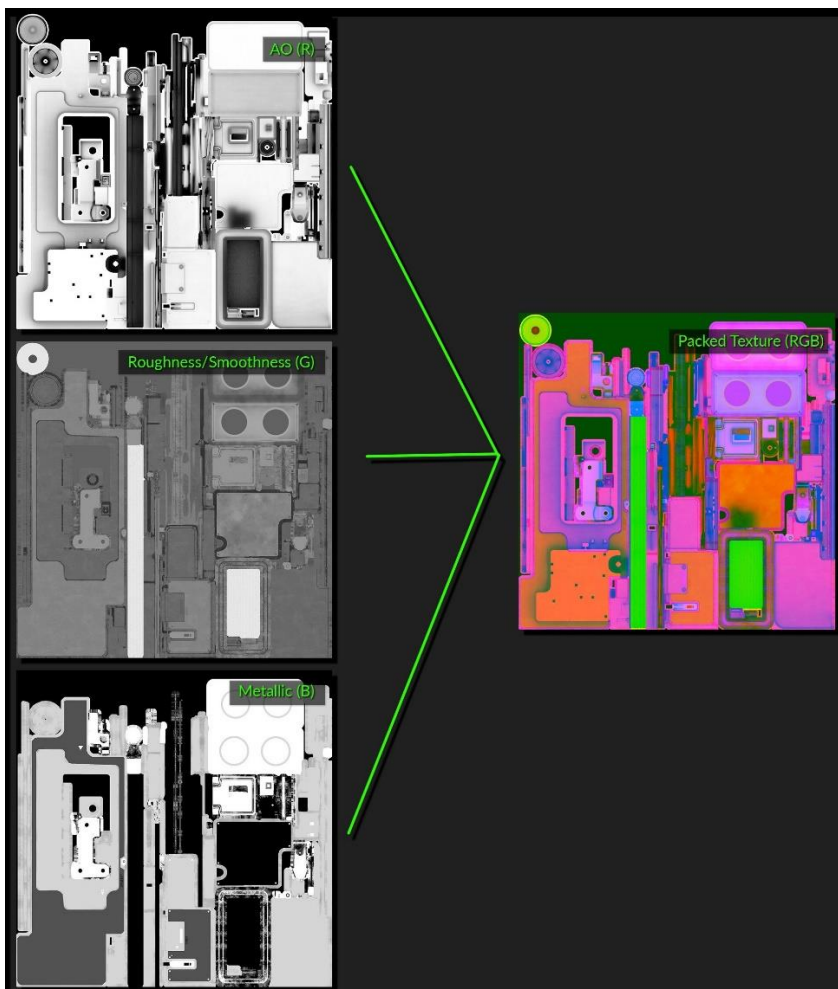


Figure 15 Texture channel packing

Further reading on the color sensitive cones [can be found here](#).

9 Alpha channels

Be selective when adding an alpha channel to a texture. Adding transparency often makes the texture larger in file size because images are converted to a 32-bit format, increasing the overall bandwidth use.

Another way to store an alpha channel is by using an extra channel in roughness, or metallic textures instead of adding an alpha channel to the diffuse texture. In Unreal, these textures usually only use two channels out of three: roughness (G) and metallic (B), leaving the (R) channel free. The ambient occlusion map can usually be baked in the diffuse map.

By using the free channel to store the alpha mask, you can keep the diffuse texture at 16-bit, halving the file size.

The following image shows an example of how you could store an opacity map in the red channel:

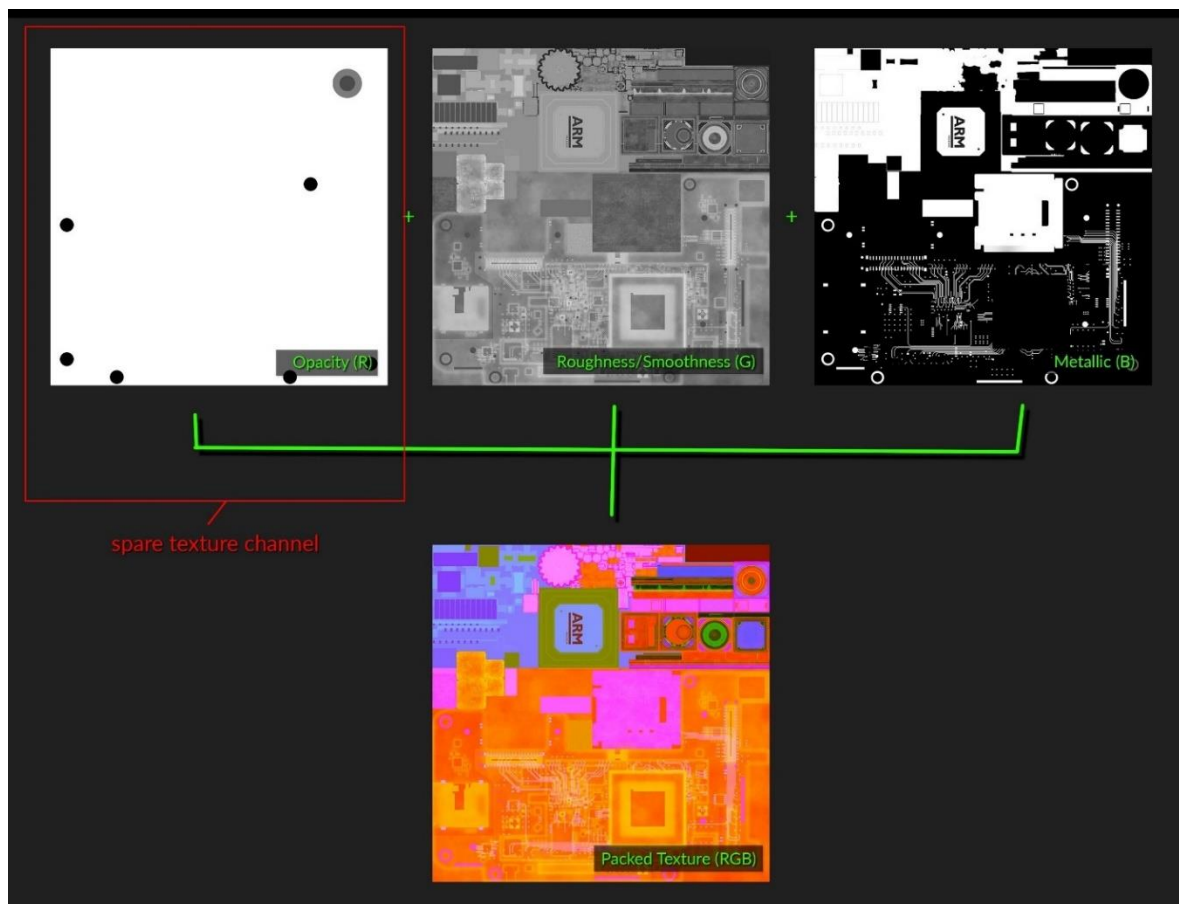


Figure 16 Storing the opacity map in the red channel

10 Normal map best practice

A normal map is a good way to make a 3D object appear to have more detail. Normal mapping is best used to add smaller details like wrinkles, bolts, and other details that need lots of triangles to model.

The usage of normal mapping can depend on the type and art direction of a game.

In most of our internal projects, we use normal mapping with no noticeable degradation in performance. Because we target high-end devices for most of our demos, low-end devices might have different results.

Using normal mapping does come with a cost, even if the cost is small. Remember:

A normal map is an extra texture. This means more texture fetches, which results in more bandwidth being used. Use normal maps sparingly when targeting lower-end devices.

The following image shows an example of how you could use a normal map and textures for the smaller details:

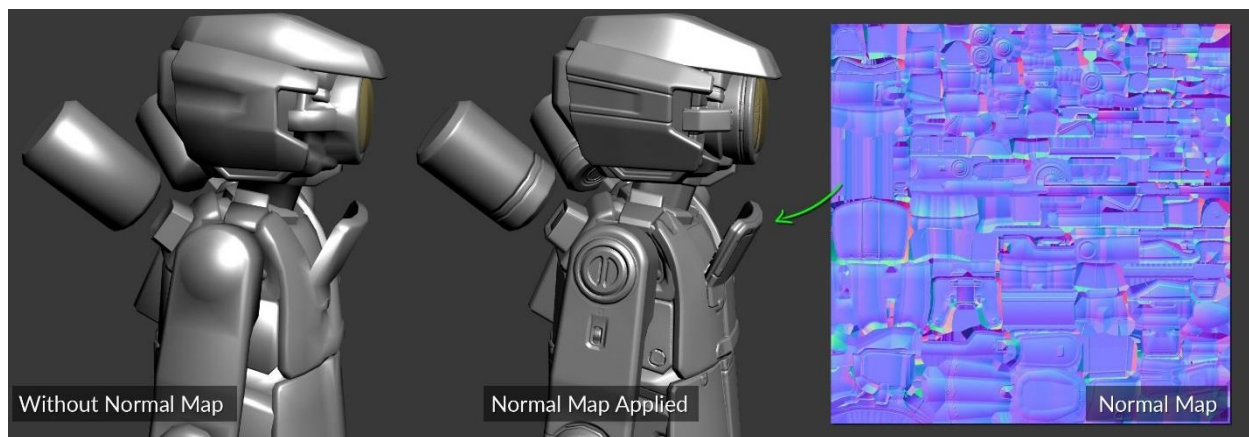


Figure 17 Using a normal map and textures for smaller details

11 Normal map baking best practice

Using a cage is a great way to get a high-quality normal map, regardless of the surface that you are baking.

Most normal mapping software can make your cage automatically. However, you can make one manually by copying your low polygon model and then increasing its scale slightly.

The purpose of using the cage is for the program to change the direction that is used to calculate the normal when baking. This produces far better results on split-normal and hard edges, as you can see in the following image:

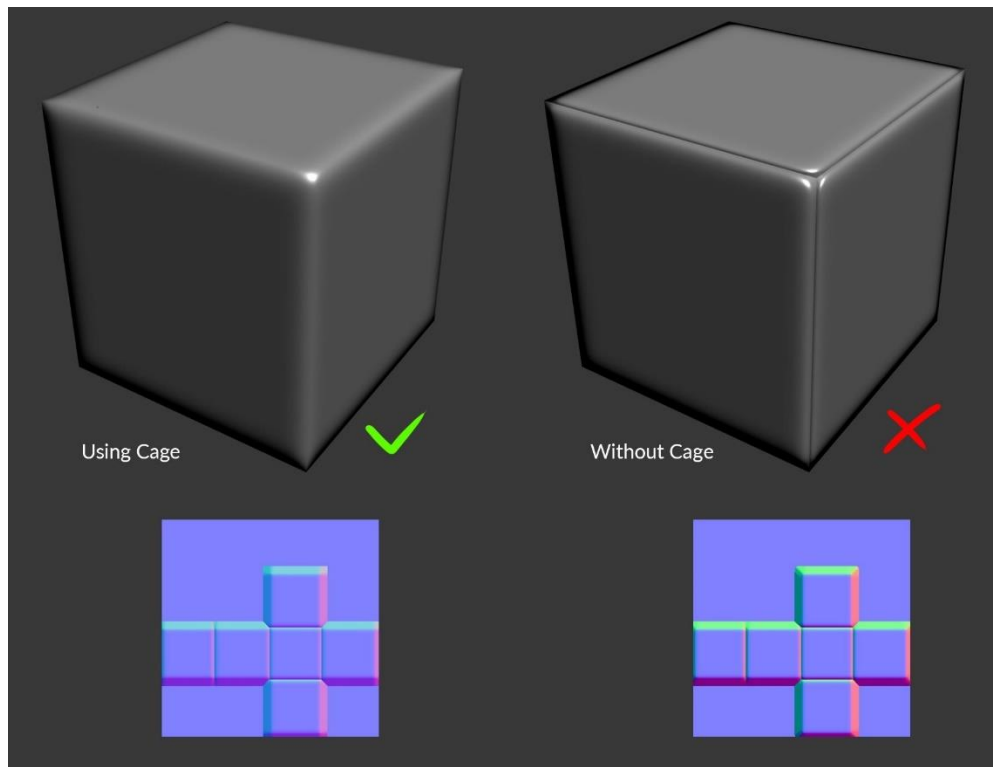


Figure 18 Cage use comparison

The cage is basically a larger, or pushed out, version of a low polygon count model. Encompass the high polygon count model for the baking to work well.

A mesh cage is used to limit the ray cast distance that is used during normal map baking. A cage can also solve problems with split-normal seams on the normal map.

If the baking software supports it, bake by matching the mesh names to mitigate the problem of creating a wrong normal map projection. When objects are too close to each other, they can unexpectedly project the normal map onto the wrong face. Matching the mesh names ensures that baking is only done on the right surface, with a matching name.

Further information on matching meshes by name is available on the [substance3d website](#) along with the [Marmoset Toolbag tutorial](#).

If mesh name matching is not possible, explode the mesh. Mesh exploding means moving parts away from each other, so that the normal maps do not project to unwanted surfaces. This also helps with incorrect normal map projection. A separate bake for ambient occlusion can sometimes be required with this solution. The following image shows an exploded mesh:

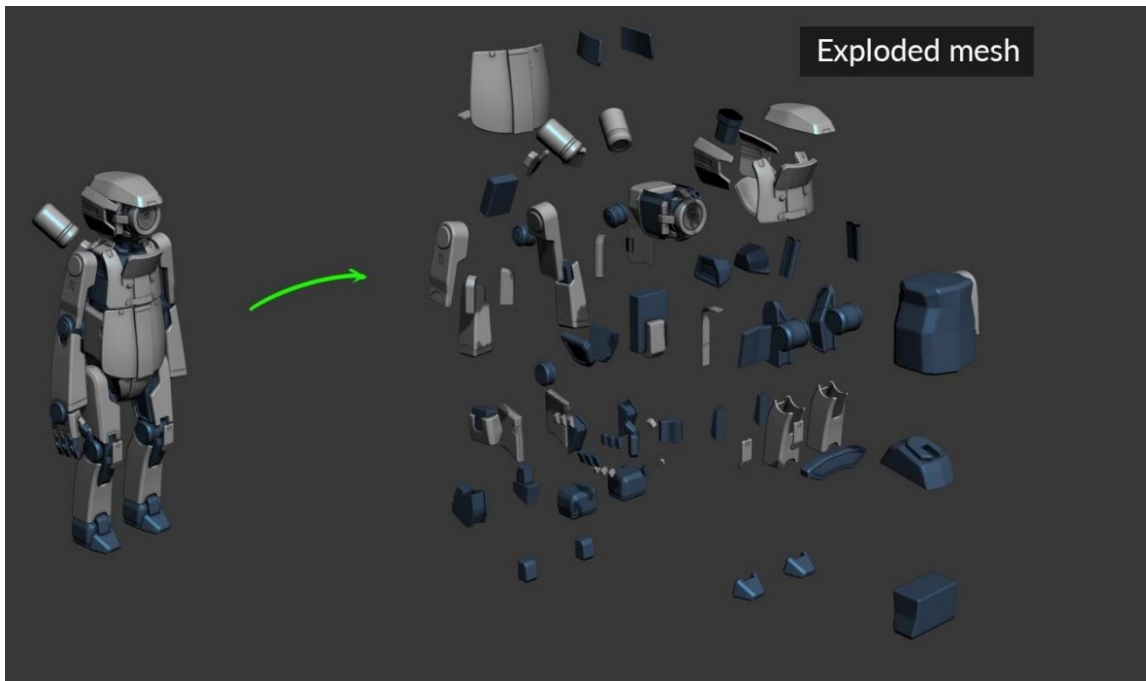


Figure 19 Exploded mesh

Split UVs on hard edges, because a continuous UV on hard edges causes visible seams. The general rule is to keep the angle less than 90 degrees, or set it as a different smoothing group. Coincide UV seams with different smoothing groups on the triangles.

The following image shows an example of how breaking the UV on hard edges can improve the appearance of seams:

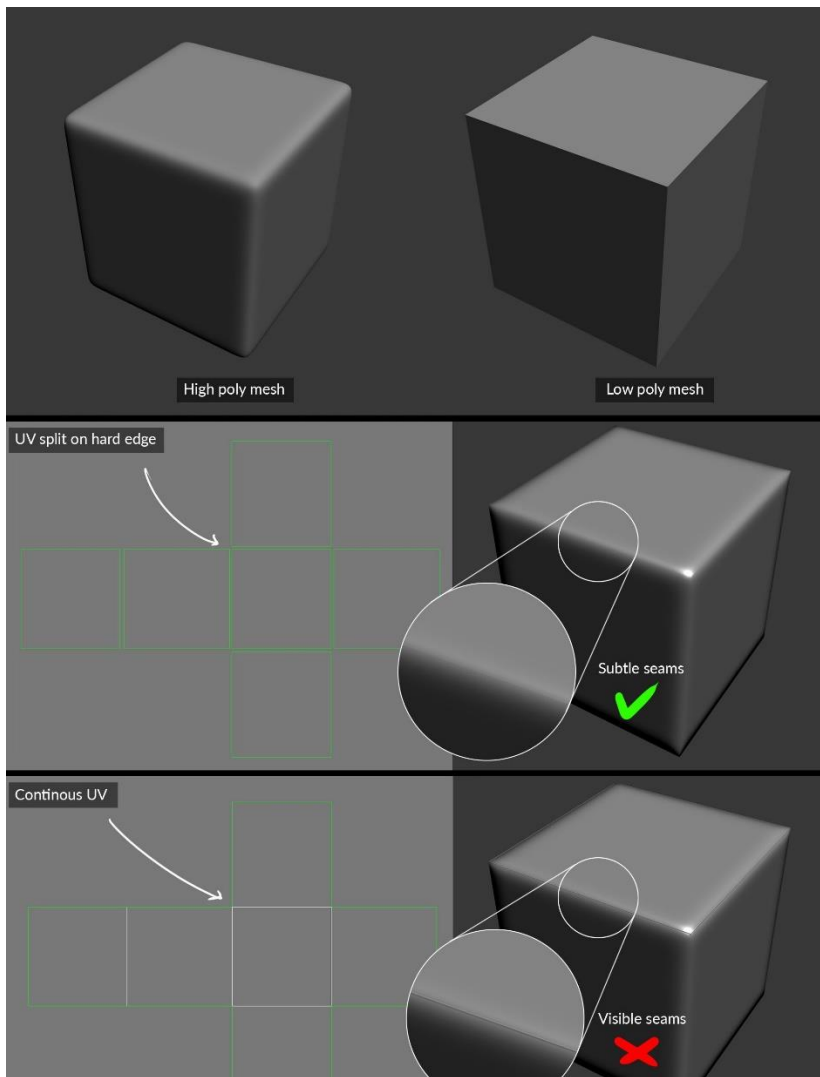


Figure 20 Breaking UVs on hard edges

12 Related information

Here are some resources related to material in this guide:

- [Arm Mali GPU OpenGL ES Application Optimization Guide: Bandwidth Optimizations](#)
- [Marmoset: The Toolbag Baking Tutorial](#)
- [Substance Bakers: Matching by Name](#)

13 Next steps

This guide covers texture optimizations that can help your games to run more smoothly and look better.

You can continue learning about best practices for game artists and how to get the best out of your game on mobile by reading the other guides in our series:

- [Real-time 3D Art Best Practices: Geometry](#)
- [Real-time 3D Art Best Practices: Materials and Shaders](#)