Tampere University of Applied Sciences

Efficient Texturing Techniques for Game Environment Art

Emilia Aaltonen

BACHELOR'S THESIS
April 2022

Business and Media
Interactive media

ABSTRACT

Tampereen ammattikorkeakoulu

Tampere University of Applied Sciences

Degree Programme in Media and Arts

Interactive Media

AALTONEN, EMILIA:

Efficient Texturing Techniques for Game Environment Art

Bachelor's thesis 50 pages

April 2022

An environment artist’s role in the game industry requires them to combine their artistic skills with technical knowledge about the best performance practices and to do this under quick turnaround times and deadlines. This thesis covers six techniques that are used by industry professionals to both make their games perform better and to make the development process itself more efficient. The theoretical part of the thesis covers each of the techniques one by one, examining their advantages and disadvantages. While the techniques are common and effective, not every game or development team needs to use all of them which is why the thesis also covers in which cases they might be the most useful.

A 3D environment scene was created to accompany the theoretical part of the thesis. It shows the various techniques discussed in the thesis in practice. The project section covers the 3D environment art creation process phase by phase, starting with the planning and ending with a finished scene in a game engine.

The project provided a thorough understanding of the skills and technical requirements needed to work with the efficient techniques as well as a clear view into the benefits of using them. Most of the techniques were easy to implement but a few required slightly more technical knowledge from the artist. The visual quality achieved with the techniques and combined with the gains in performance proved that the techniques are very profitable tools to use in game development.

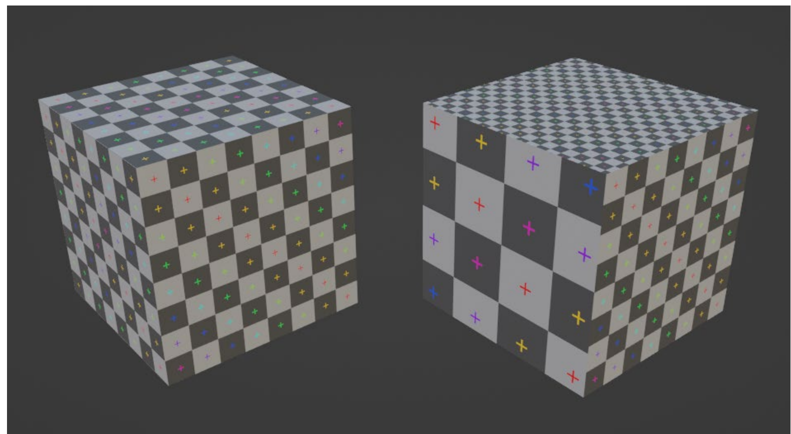
Key words: 3d, game art, texturing, techniques, environment art

CONTENTS

1 INTRODUCTION 6
2 TECHNIQUES FOR EFFICIENT TEXTURING.................................... 7
2.1 Texture atlasing.. 7
2.2 Tiling 8
2.3 Trim sheets 10
2.3.1 The “Ultimate Trim” technique 12
2.4 Decals 15
2.5 Channel packing 19
2.6 Texture blending 22
3 PROJECT 29
3.1 Goals and scope 29
3.2 Planning 29
3.3 Production 32
3.3.1 Creating the textures 32
3.3.2 Modelling the assets 39
3.3.3 Unreal Engine 41
3.4 Result 43
4 DISCUSSION 47
REFERENCES 49

ABBREVIATIONS AND TERMS

AAA games	Games made by large, established publishers, typically with high budgets.
Alpha	Transparency
CPU	Central processing unit, primary computer component that processes instructions
GPU	Graphics processing unit, a computer component specialized in rendering images rapidly
LOD	Level of Detail. A performance optimization technique where objects further from the player are switched out to lower geometry versions of the same object.
Node based	A visual way to represent code.
Normal map	An image that can be used to modify the direction light bounces off of a surface. They are usually used to create the illusion of more detail on low geometry models.
Open-world games	Usually refers to games with large worlds that the player can freely travel across at any point in the game.
Shader	A set of rules that define how a 3D object is rendered. A shader takes in data, such as meshes, textures and light, and uses it to calculate how the pixels on the screen are rendered.
Texel density	The resolution of a texture of a game object in comparison to the size of the game object. Sticking to the same texel density for every object in the game keeps the game looking consistent (picture 1).



	PICTURE 1. Visualization of even and uneven texel density.
UVs	Texture coordinates. A flat representation of a 3D model that serves as a connection between the 2D texture and the 3D model.
UV unwrapping	The process of creating the UVs by creating cuts in the 3d model’s geometry to flatten it and converting it to 2D space

1 INTRODUCTION

The current trend seems to be that every new generation of games needs to be more detailed than the last to keep up with the players' increasing demands. Reaching this detail level pushes on the limits of both the hardware running the game and the people making it. This is why game developers need to make their games with efficiency in mind. This is especially true in games that thrive for the hyperrealistic style seen in many AAA games, but even with more simplified art styles, actions can be taken to make the game perform better especially on lower-end platforms.

Efficiency in the context of video game production and within this thesis measures two things: in-game efficiency and production efficiency. In-game efficiency considers how well the game performs. This includes many things, such as lagging, load times, and how much the game takes up memory. Production efficiency considers the people and companies working on the game. In game production time is limited, and any techniques to speed up workflows are incredibly helpful.

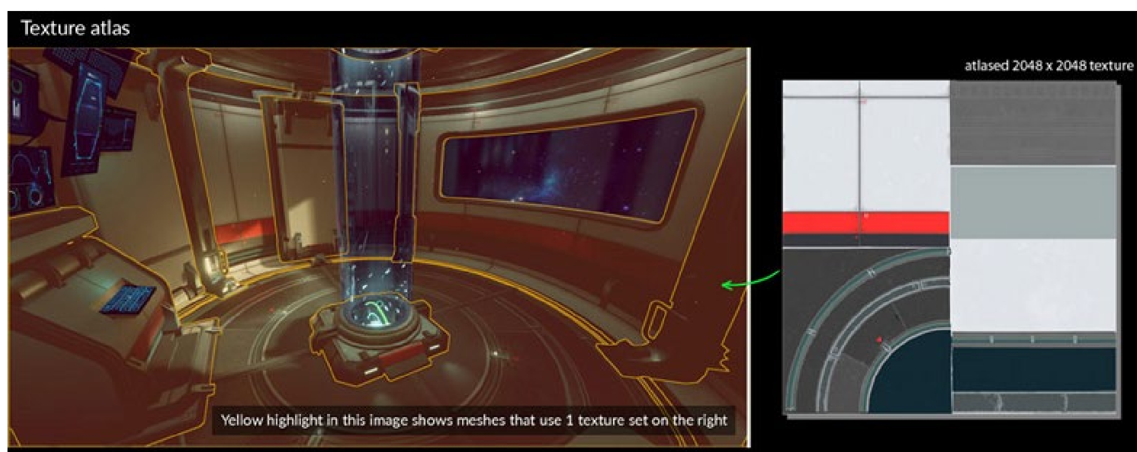
This thesis explores multiple different texturing techniques and examines what effects each of them will have on either the production speed or the performance of the game. The techniques discussed are texture atlases, tiling textures, trim sheets, decals, channel packing, and texture blending. The thesis will also go over the possible negative consequences of choosing to use the techniques.

This thesis was made to give the reader an idea of what different approaches they could implement to their workflow to improve the development of their games. It was written with the expectation that the reader has a basic understanding of 3D modelling and texturing. The thesis is aimed at people interested in video game art, but mainly game development students who would like to expand their knowledge of the techniques used in game development today.

2 TECHNIQUES FOR EFFICIENT TEXTURING

2.1 Texture atlasing

Texture atlases are one of the most common techniques that artists will utilize to make games run more efficiently. They are textures with several smaller textures fitted into one image. The atlased textures can be the textures of only one object, or with some planning, one texture atlas could have the textures for multiple different objects (picture 2). This is preferable as objects that share a texture will only make one draw call. (Arm Developer 2020).



PICTURE 2. Several objects using the same texture (Arm Developer 2020).

A draw call is a set of instructions from the CPU telling the GPU what to render on the screen. Having many draw calls slows down the game, because it takes time for the GPU to receive the rendering instructions. Therefore, it is faster for the computer to do one draw call for a bigger texture than many draw calls for multiple small textures. According to Hider (2017), on modern day consoles and PCs 2000-3000 draw calls per frame is reasonable, but in mobile games that number is only a few hundred.

Objects sharing the same texture can be batched together. This essentially means that the engine will see all the batched objects as one, and therefore only require one draw call for them. Batching works only on static objects. One issue with batching is that since the game sees the batched objects as one, it will aways

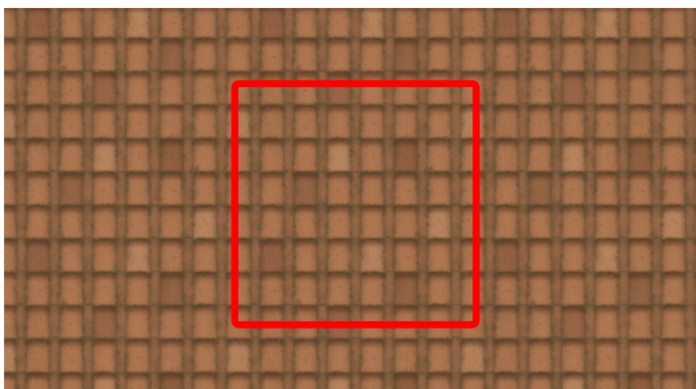
render them all at the same time. This means that even objects that are not currently visible to the player will get rendered. (King 2021)

Using texture atlases well requires some planning. Generally, objects that will always be seen together can be safely atlased (King 2021). A good example could be if a game that has a level set in a grocery store and all the fruit textures of the produce isle are atlased together. Alone that would work very well, but should the artist then need an apple for the teacher desk for a level set in a school, the game would need to load in the whole big texture with all the fruit just for one apple.

A good fix for this issue would be to save a tiny portion of each texture atlas for cases like this (King 2021). In this example the classroom would have its own texture atlas with textures for chalk boards, pencil sharpeners, and such, and then an empty portion where the textures of the apple could be copy-pasted into.

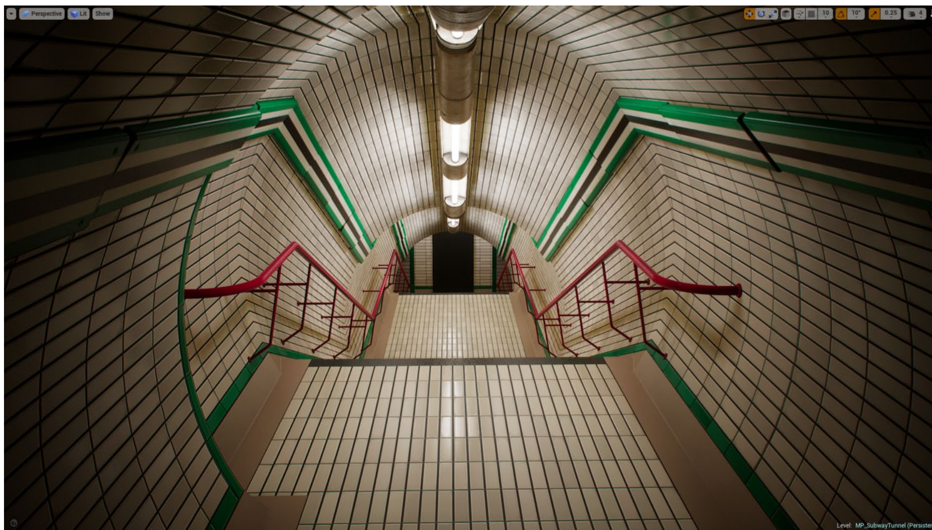
2.2 Tiling

Tiling textures have been the basis of game art for a long time. They are textures that have no visible border when laid side by side in every direction (picture 3). They are by nature very repetitive, which is why a lot of the techniques discussed later in this thesis revolve around improving upon and working around tiling textures.



PICTURE 3. Tiling roof tile texture with the texture borders of the texture image highlighted.

The idea with tiling textures is that one small texture can be used to tile over a larger space (picture 4). They save on texture memory and provide a consistently high texel density. Big surfaces, such as walls, floors, and terrains usually use tiling textures, because uniquely texturing all the walls of just one building would require massive texture sizes for a good result. Using tiling textures also saves on production time, as they are often faster to iterate upon than texture atlases and one tiling texture can easily be switched out for another one to get a different effect. (Kronenberger 2020)



PICTURE 4. A subway tunnel scene gets a lot out of one tiling texture (Hurtubise 2018).

If an object needs more than one texture, tiling or not, it will make multiple draw calls. This will affect performance. It is the game artists' job to balance out when the performance loss from making multiple draw calls is more worth it than the possible texture memory loss from making unique textures. If high texel density is not that important, for example the camera is very far, texture atlases might be preferable. In most modern first-person camera games and VR games texel density is more important because the player can get close to examine the textures. In these cases, tiling textures should be used when possible.

When working with tiling textures it is important to remember that humans are excellent at noticing patterns (picture 5). The textures should generally be quite even, and no part should stand out. If a player notices unnatural repetition of

textures or other game assets their immersion is broken. This might lead to shorter playtimes and worse player retention. (Greuter & Nash 2014)

PICTURE 5. Eye-catching tiling in the water of the game Pokémon Legends: Arceus (Gray 2022).

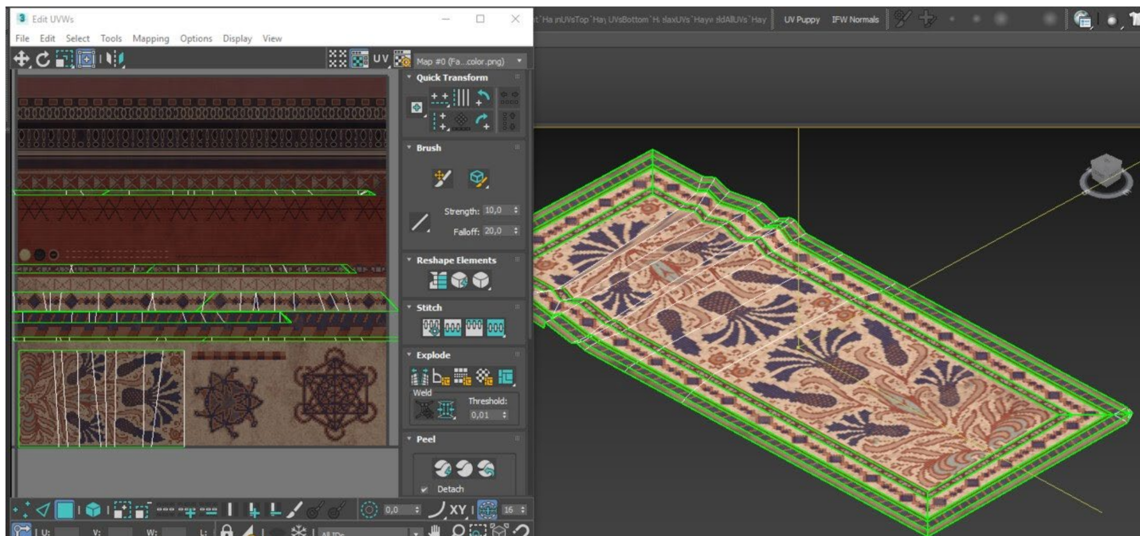
With the repetitive nature also comes the issue of detailing. Because the tiling textures often must be unspecific and subtle to hide the tiling, the environment artists cannot get the necessary detail into the game environment with them alone. These issues are where newer techniques such as decals and texture blending are very helpful. They will be discussed later in the thesis.

It is also important to be mindful of UV seams. Good seam placement is always important, but even more so when the item will not have unique textures. This is because the artist cannot simply paint out the issues in the texture if for example the texture does not line up on both sides of a UV seam. (Kronenberger 2020)

2.3 Trim sheets

Trim sheets are like tiling textures, except that they only tile on one axis. They are often used to texture long pieces of geometry, for example crown molding, but they can be used for much more when planned well. Usually, they are made

as texture atlases, so that the trims form multiple thin strips on one texture sheet (picture 6). Any extra space can be used for other small details like screws or handles to get even more out of the same texture. (Kronenberger 2020)



PICTURE 6. Example of trim sheet usage (Kronenberger 2020).

Working with trim sheets requires more planning from the artist than a traditional workflow. This is because the technique requires a reversed workflow, where the textures are created first, and the unwrapping is done last. When creating the trims, it is important to keep in mind how big each of them needs to be, so that the final models do not have drastically uneven texel density. After the trim sheets are done the artist can then UV unwrap the model and layout the UV islands along the trims. Any curved UV islands can be straightened to fit the trims - most UV editors have a feature that does this. The UV islands can be rescaled a little to correctly fit the trims, but the texel density should not drop noticeably. (Kronenberger 2020)

Because the trims are often thin, the textured asset might need more divisions in the mesh so that it can be separated into thin enough parts. Using a couple of extra edge loops in the geometry is still considered cheaper than having to make a unique texture for every item. (Kronenberger 2020)

Using the same trim sheet to texture multiple similar items or even using one to texture the whole environment helps with keeping a consistent look throughout the game while still allowing for interesting variance. An example of this can be

seen in picture 7. It also saves space as well as production time if used efficiently, since fewer textures need to be made. (Derozier & Fleau, 2020)

PICTURE 7. A temple textured with trim sheets in Assassin's Creed Odyssey (Derozier & Fleau, 2020).

Trims sheets have the same drawbacks as tiling textures. They lack the fine control that using unique textures has. The trims cannot have any large details like stains or scratches or anything that would be big enough to cross over from one trim to another. That would require the trims to be lined up very carefully every time they are used, or otherwise there would be a noticeable cut-off. That would go against the whole point of using trim sheets as individual modular pieces. Any eye-catching detail will also cause obvious tiling. A good general guideline is that objects of natural origin should have little to no visible repetition, but man-made objects, for example ornate vases or industrially made metal railings, can have visible repetition. Repetition is an intrinsic part of urban environments and should not be avoided when it would be appropriate.

2.3.1 The "Ultimate Trim" technique

A variation of the trim sheet technique called the "Ultimate Trim" technique was developed for Insomniac Games' Sunset Overdrive by their Principal environ-

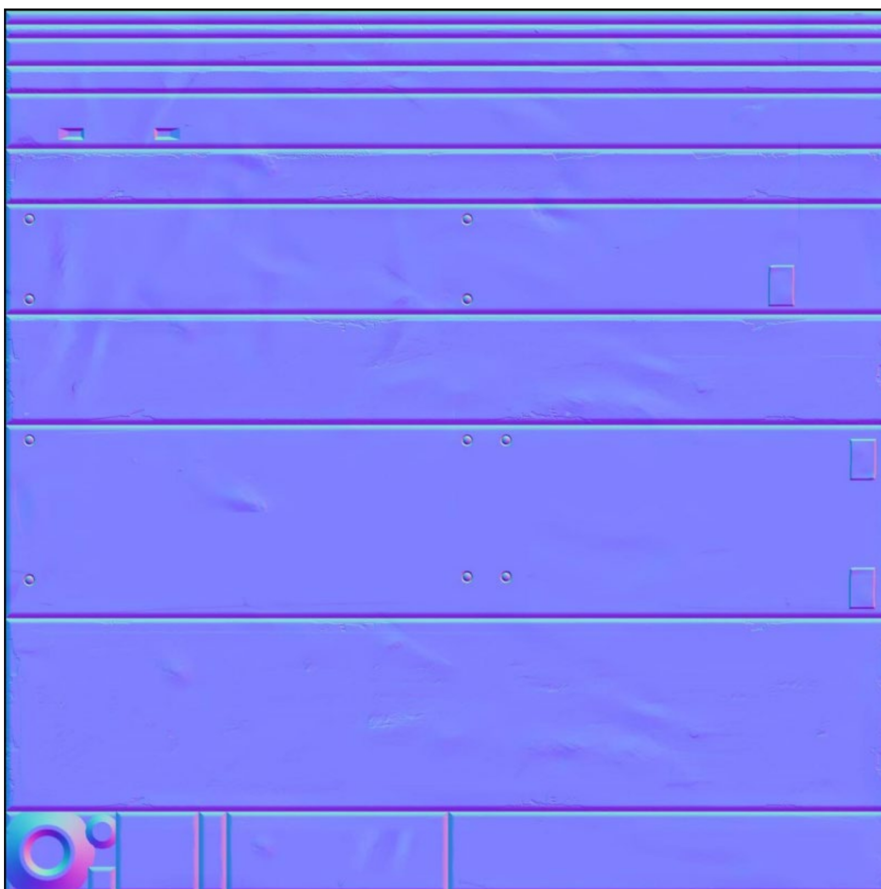
ment artist Morten Olsen and his team. This technique has since gained popularity and has been used in other Insomniac games with large open worlds, such as Marvel's Spider-Man. (Olsen 2015)

The basis of this technique is that all the textures in the game are made to follow a standardized layout, as seen in picture 8. All the items in the game environment from small signs to entire buildings are then UV unwrapped into trims and mapped to follow the trim layout. The trim layout has different sizes of trims to map to different sized parts to keep texture density uniform. All sizes of trims have two different versions, allowing for more variance while saving on draw calls. At the bottom of every texture is a free space reserved for any special details that any models might need, like buttons, stickers, or special endcaps. (Olsen 2015)

PICTURE 8. The standardized trim layout used in Sunset Overdrive (Olsen 2015).

Because all the items in the game are textured following the same layout, the artist can switch out textures between different objects without needing to alter the UVs. This is why Ultimate trim allows for extremely quick iterating and therefore saves a ton of precious production time. According to Olsen (2015), the technique was developed because the environment art team had to prioritize speed over all else since they had a large open-world environment to texture with a relatively small team for a AAA game. Ultimate trim also saves a ton of memory since the textures are created to be reused.

In addition to the standardized trim layout, Ultimate trim also involves a specific way of handling normal maps. Every single trim has a 45-degree angle painted along its edges in the normal map as seen in picture 9. This creates the illusion of a clean beveled edge when used on two parallel faces (picture 10). When used on a large scale the performance savings that come from faking higher poly geometry add up. (Olsen 2015)



PICTURE 9. Angled normals on all trims (Olsen 2015).

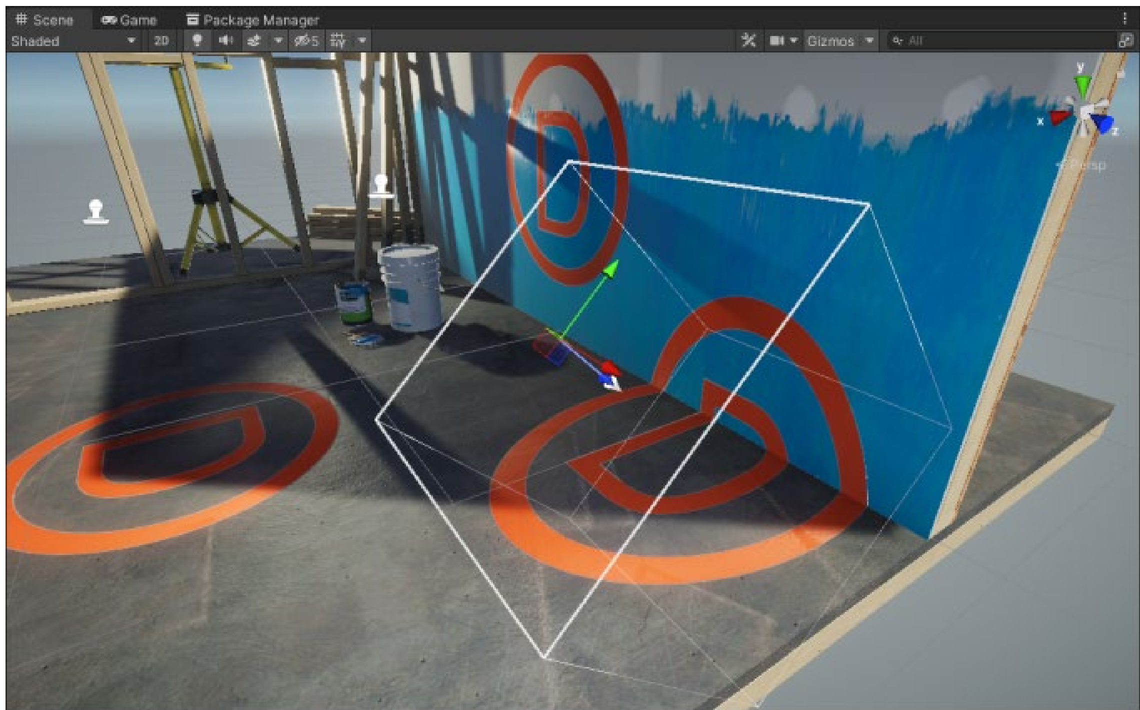
PICTURE 10. 45-degree normal bevels create the illusion of more geometry on the armrest (Olsen 2015).

2.4 Decals

Decals are polygons or small meshes with texture that are laid on top of a larger mesh, creating the effect that the decal is part of the other mesh. Decals usually have transparency, but it is not necessary. Decals are used to add variety and storytelling aspects to scenes. They are a good way to hide the repetitiveness of tiling textures or modular assets with relatively small textures. Common use cases for them are damage, such as cracks or scratches, dirt, and various sign-age (picture 11).

PICTURE 11. Different decals in a subway station (Hurtubise 2018).

There are two different types of decals, which are used for different purposes. The first kind is projected decals, which are created in-engine. Both Unity and Unreal Engine have built-in systems for them. In both, the decal projector is shown as a wireframe box with an arrow pointing to the direction of the projection (picture 12). Any mesh that overlaps the box and is in front of the arrow will have the decal shown on top. As seen in picture 12, projected decals can be projected over multiple elements. Because of that and their easy adjustability in engine, they are a quick and user-friendly way to bring detail and believability into the scene. Projected decals do however work best on flat surfaces, because the more parallel a surface is to the direction of the decal projection, the more the decal will stretch (Unreal Engine 4.27 Documentation).

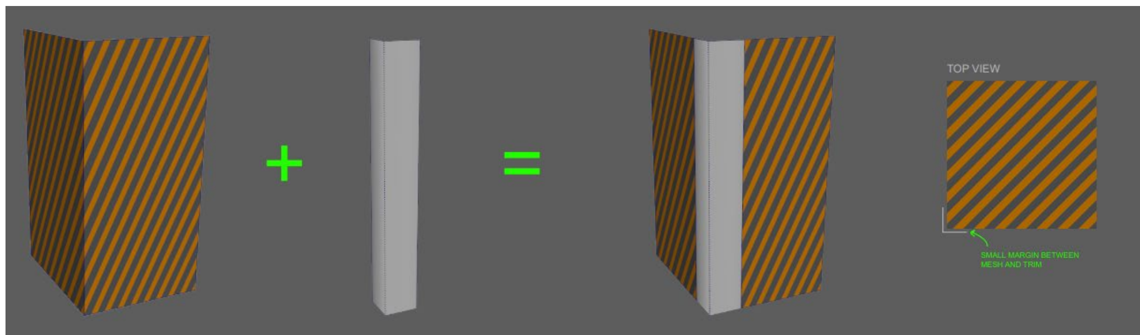


PICTURE 12. Example of decal projection on multiple surfaces (Unity 2020).

Because projection-based decals are created in-engine, they can also be created in the game while it is playing. The most common examples of these are bullet holes and other damage caused by the player. Many multiplayer games such as Fortnite and Overwatch also allow the player to place decals themselves in the form of stickers or graffiti.

Projected decals are considered lightweight as they do not have a notable effect on performance. They can start to have an effect if the decal starts to take up a large portion of the screen space. (Unreal Engine 4.27 Documentation)

Another method of creating decals uses a small amount of geometry to hold the decal texture. It is essentially a small piece of extra geometry that lays very close to the main mesh but not quite overlapping (picture 13). This method requires the artist to place the decals already in the 3D editor. Therefore, it is more time-consuming to make adjustments to it. (Lezzi 2019)

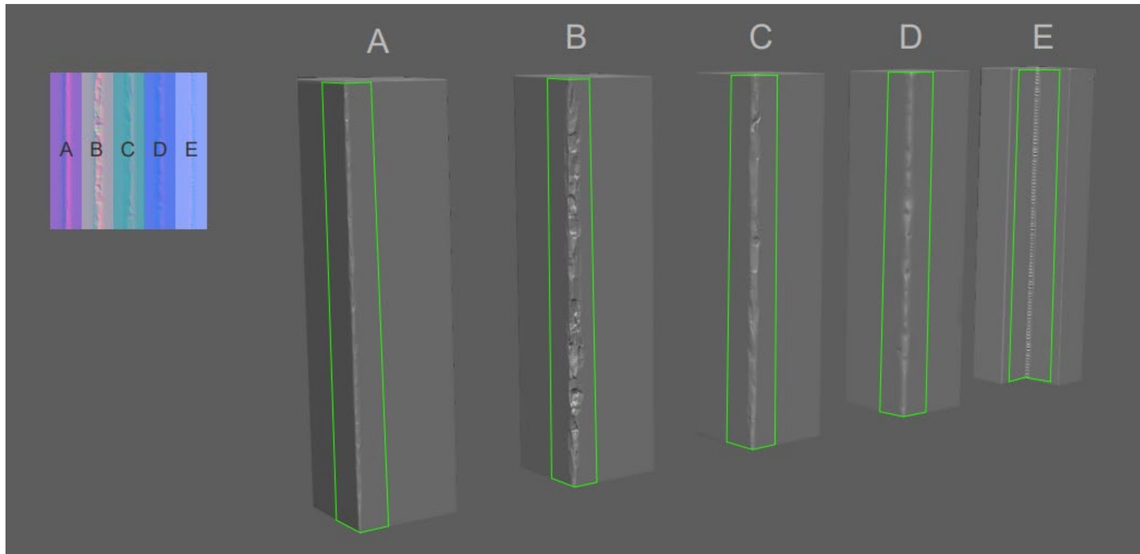


PICTURE 13. How an edge decal lays on top of the main mesh (Lezzi 2019).

Unlike projected decals, mesh-based decals work just fine on non-flat surfaces and can easily be used to wrap around objects. That is why this method is most often used for edge detailing, such as dents, weld marks, and worn-off paint (picture 14). It creates high detail and realism while leaving the underlying tiling textures untouched. Edge damage decals are usually made as trim sheets. (Lezzi 2019)

PICTURE 14. Edge damage decals used on the concrete windowsill and the wood trim on the floor (Lezzi 2019).

Whereas a decal projector always projects the whole texture given to it, texture-based decals are UV mapped, meaning that multiple decals can be stored on one sheet and used separately as needed (picture 15). Using the same texture also allows the decals to be batched together to improve performance.



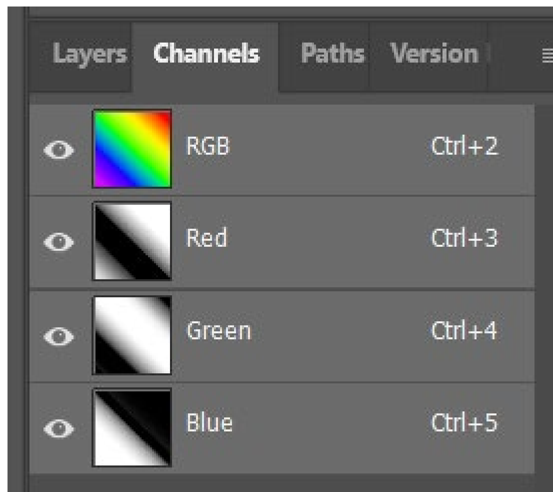
PICTURE 15. Different kinds of edge damage decals stored on one normal map (Lezzi 2019).

2.5 Channel packing

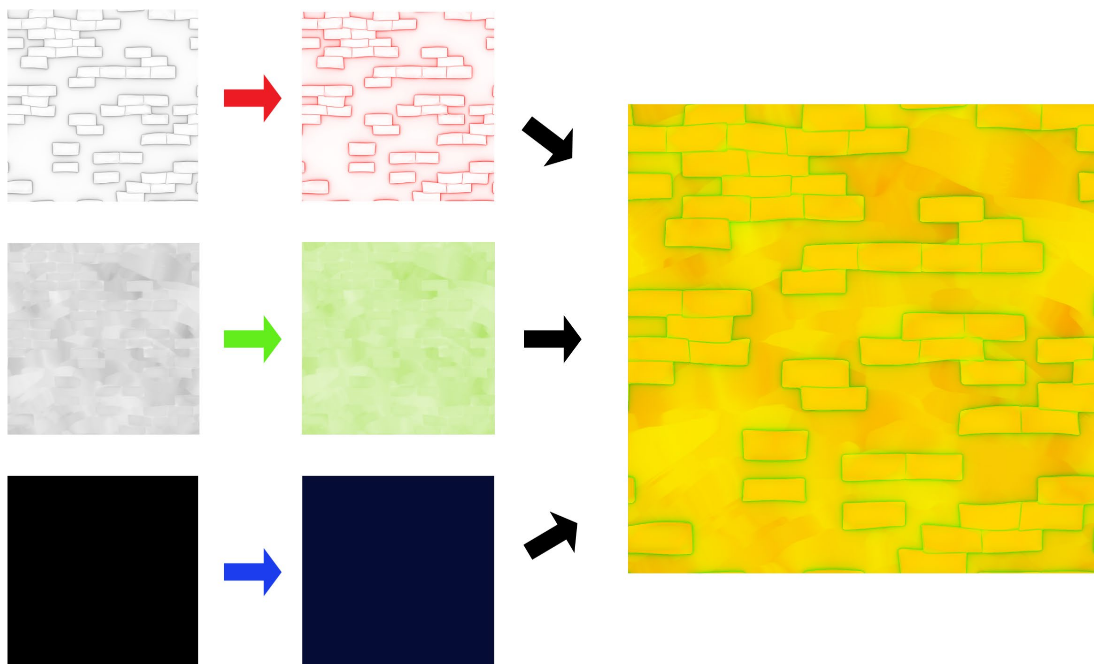
A lot of the textures used in games are grayscale images. Take for example a roughness map: it defines which areas of the textured object are rough. The roughness is defined on a scale from zero to one, zero being the most reflective and one being the roughest. On a roughness map the value of one is shown as white and the value of zero as black with different shades of gray for the values in between.

A color image when exported out of an image editor or a texturing software is an RGB or an RGBA image. This means that it has individual channels that store the images red, green, blue, and sometimes alpha values each separately on a zero to one, black and white scale as seen in picture 16. The forementioned roughness map only needs one of these channels to hold all the information it can provide. This is the basis of a practice called channel packing. It is a process where multiple different grayscale images are inserted into each of the RGB and

sometimes alpha channels of one image, instead of all being separate images (picture 17). Some of the maps that can be channel packed include metallic, roughness, or occlusion maps and different kinds of masks (Arm Developer 2020)



PICTURE 16. The RGB channels as they are shown in Photoshop.



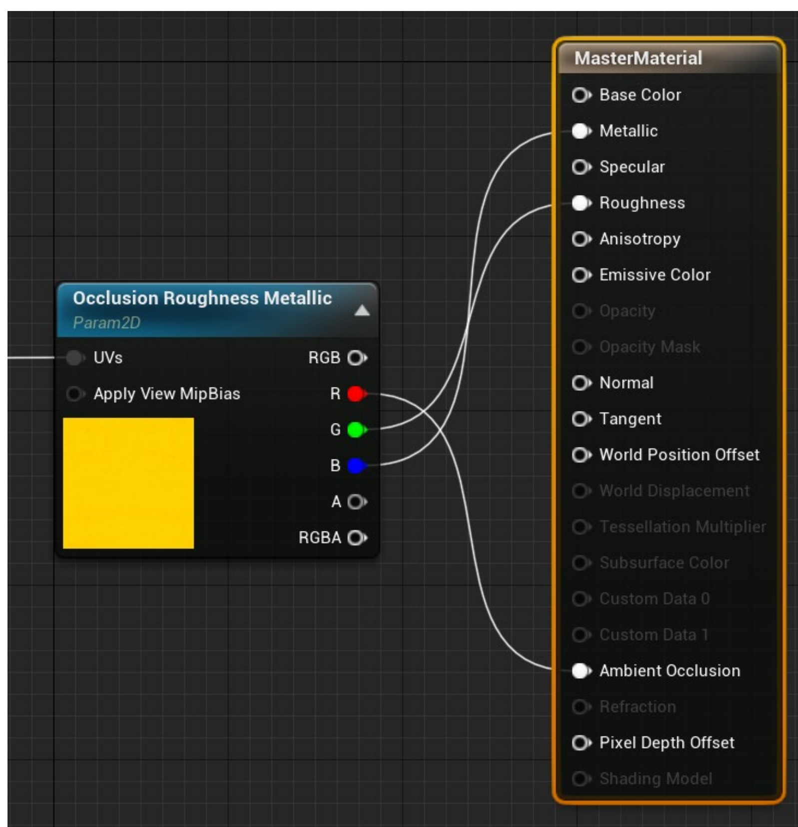
PICTURE 17. How the occlusion, roughness, and metallic maps of a wall texture are packed into one.

Channel packing is done because one color image takes up one third of the memory that three of them does. Reducing the amount of memory needed for textures is one of the key things to consider when texturing for video games. It also means that the game only has to load in one image instead of three. Channel

packing has been used in the industry for years. There really are no drawbacks to using it, which is why it is a well-established and accepted technique.

Channel packing can be done manually in an image editor, or it can be automated. The industry standard texturing programs, such as Substance Painter know which textures to pack into which channels automatically based on a selected texturing pre-set. Other programs such as Blender, Substance Designer, and 3D coat need the artist to specify which textures to pack and which channel to pack them into.

Most game engines are well set up for channel-packed textures. The different color channels are accessible without any extra steps, so they can simply be connected to their respective inputs as seen in picture 18.



PICTURE 18. How the RGB channels are separated inside a shader.

Depending on the color depth of the image the green channel of an image sometimes can have a few more bits. This is because human eyes are very sensitive to the color green. As a result of this, the most important grayscale texture map

should be packed into the green channel. The blue channel has the least bits and should have the least important map. (Arm Developer 2020)

One stumbling block with channel packing is that different game engines have slightly different methods of working with the textures. Unity’s standard shader uses a smoothness map instead of a roughness map to set how rough or smooth a material is. A smoothness map is simply a roughness map with the values inverted. It is important to keep in mind that textures made and packed for one engine are not necessarily interchangeable for another. (Hegedüs 2019)

2.6 Texture blending

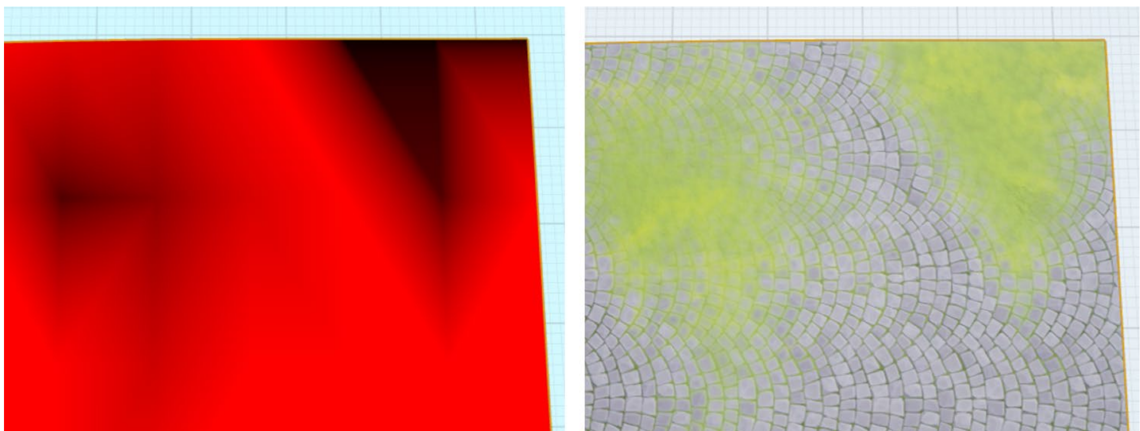
One of the most common ways to get more variety in environments is by using a technique called texture blending. Texture blending allows the artist to blend multiple different materials or textures together (picture 19). This brings more detail and life into the environment as well as helps in keeping a consistent look throughout the environment. Texture blending can also be used to hide obvious tiling when used in conjunction with tiling textures. There are a few different ways of doing it, with their own advantages and disadvantages.

PICTURE 19. Moss texture is blended together with a brick texture (Świerad 2016).

Vertex painting is a type of texture blending, where RGB data values are stored in each vertex of a mesh and then used to blend between different materials. Vertices can be thought of simply as points in space that form 3d meshes, but they can store in themselves many different values other than position. Some of the most common ones are normals, texture coordinates, and color. (Unity documentation 2022)

Vertex painting works similarly to painting in a 3D texturing program but instead of editing texture maps, the artist edits the color values stored in the Vertices. There are three available color values to paint: red, green, and blue (RGB).

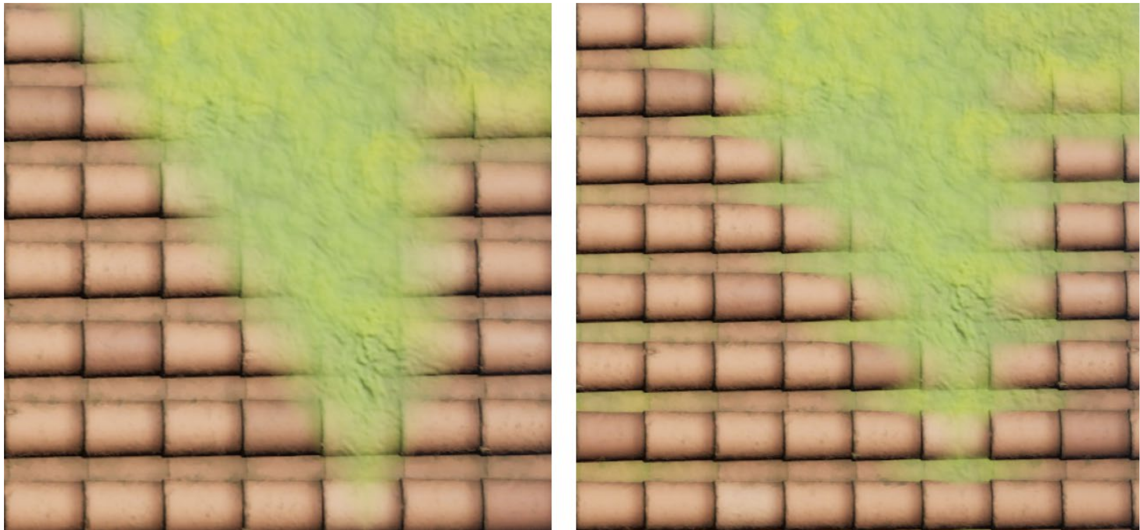
For the vertex painting to work, the active shader must be set up in a specific way. Most commonly it is done with a lerp (linear interpolate) operation. It can be set up to recognize a Vertices color value. If that value is zero, it will display the first texture, if it is one it will display another. If it is in between the numbers, it will blend between the two textures (picture 20). The basic lerp operation is one of the cheapest operations a shader can have performance wise. (Świerad 2016)



PICTURE 20. In Unreal Engine 4 black areas in vertex painting mode show where the red vertex data is the highest.

The area where the two textures blend into each other is an even gradient, and it can look very unnatural, especially if the mesh has low geometry. To obfuscate this, the shader can be set up to blend between the textures based on a greyscale map. If a heightmap is used, a convincing effect of the second texture gathering in the low points of the second texture is achieved. (Świerad 2016) This is very

effective on different tile or brick surfaces that would have moss or dirt in their crevices and an example of this can be seen in picture 21.



PICTURE 21. Vertex painting with and without a heightmap.

Vertex painting is a very fast way to get a lot of detail into the environment. For the environment artist vertex painting is very easy, since it is fully editable in engine, so if they would like a little more moss on a rock or another puddle somewhere, there is no need to edit, export and import any files. There is also no need neatly layout the UV islands to perfectly fit the UV space if all the textures used are tiling textures.

Another reason vertex painting is used very often is because it is very cheap. It saves a ton of texture memory since every item does not need a unique texture. For example, if a scene has multiple pillars, they could all have the same base pillar texture, but in order to keep the pillars from looking too similar, the environment artist can use vertex painting to add dirt and moss to each of the pillars separately. To reach the same effect without vertex painting, each of the pillars would have had to have its own texture, or risk looking very repetitive to the player. It is also considered very cheap from a performance point of view since it is inherently just some numbers and not additional files (Unity documentation 2022).

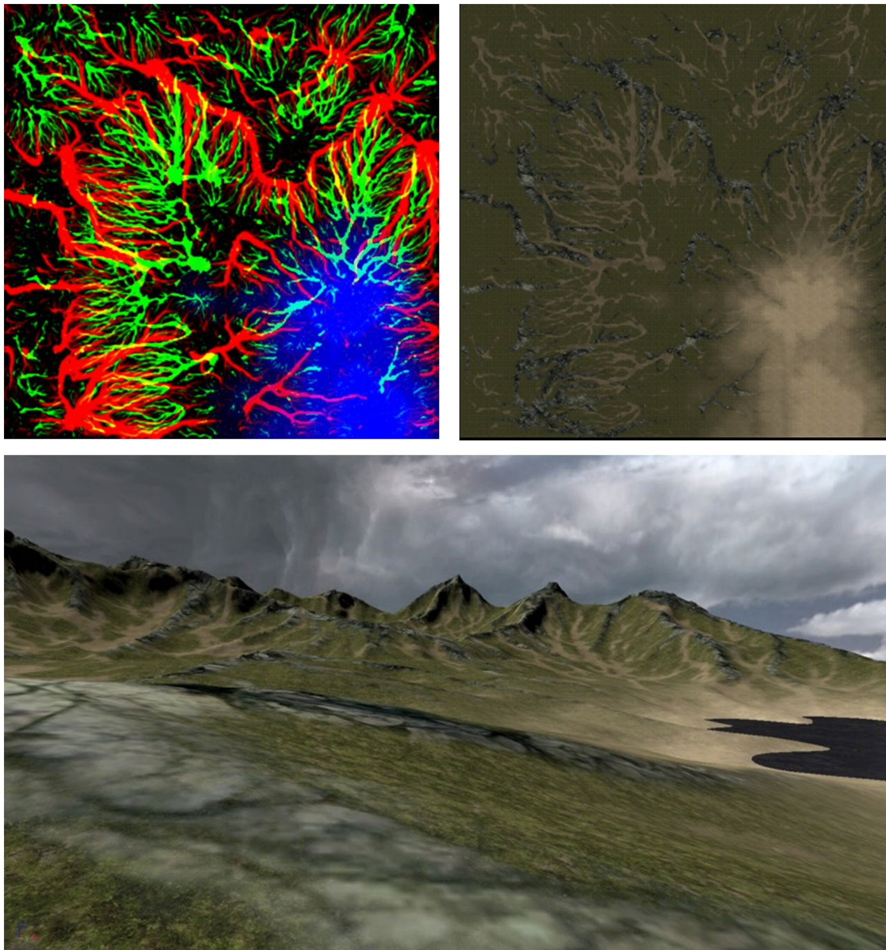
Vertex color data can also be used for many other useful things in games, such as shader-based animation, or controlling transparency. However, these are not

usually done through vertex painting, but instead through assigning the vertex color in a 3D program. (Glad 2020) This method is not as fast as editing in the engine because of the need to reimport the asset on every iteration, but it is more precise.

The two biggest shortcomings of vertex painting are at its very core because it needs to have enough Vertices to look good. This need can cause issues with lower level of detail (LOD) models. If the vertex paint is obvious on the screen on the highest level of LOD model and the Vertices holding the vertex paint data are no longer there when switching to a lower LOD, the player will see the textures suddenly disappear.

Another issue is that the blend between the textures always happens as the mid-point of the two Vertices and there is no way to change that. A heightmap only affects the shape of the blended edge. Hence there is no way to get a precise result. The edited mesh needs to have a sufficient number of Vertices for a good result. This need might result in having to add additional geometry where it would otherwise be unnecessary. A small amount of extra geometry can often still be cheaper than having unique textures. Regardless, because of this need for geometry, it is important to keep in mind that there are alternative ways to do texture blending.

One of these alternatives is to use a blend map. Blend maps are bitmaps that have up to four grayscale maps channel packed in them. These grayscale maps are then used as masks to define where a specific texture is to be rendered on a mesh (picture 22).



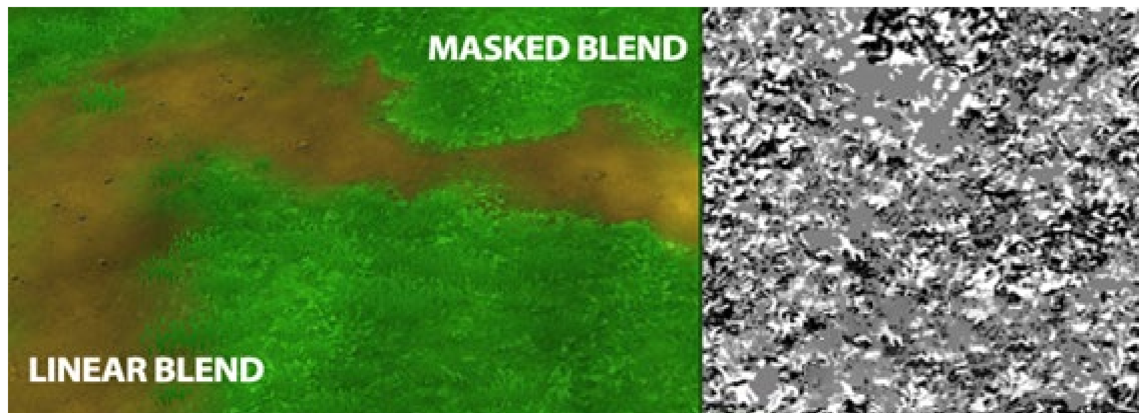
PICTURE 22. How a blend map can be used to define textures over a large environment (Unreal Engine 3 documentation 2012).

Blend maps are most commonly used with tiling textures over very large surfaces, such as terrain in open-world games or very large levels in for example driving games. These are cases where vertex painting would not be the best option because the terrains would need to have incredibly high geometry. They can also be used on smaller objects when more control is required for the blending. When used on smaller objects, it is often reserved of the most important "hero" assets.

Using a blend map gives more control to the artist to define exactly how the textures blend. The artist is not limited by the Vertices of the mesh and can control the shape of the blend with more precision. It also does not have the same gradient artifacts that come from vertex painting.

If the blend map does not have a large enough resolution the blending areas will become blurry. To combat this, a blend mask can be used to fake a higher resolution on the edges of the blend (picture 13). It works similarly to the heightmap

used in vertex painting. The map can be a separate image or packed into one of the channels of the blend map. (McGuire 2010)



PICTURE 23. A blend mask (right) being used to enhance the blending point between two textures (Yang 2013).

When vertex painting the vertex color data is stored in the Vertices of each individual mesh. This allows different objects to use the same material because the vertex data of the objects are separate. When using a blend map this does not work, because the color data is stored in a bitmap image that directly corresponds to the UVs of a specific object. As a result, each object would need its own blend map and material. Additionally, when vertex painting the position of the UVs does not really matter, but when using a blend map, the UVs of the mesh need to be carefully laid out, like when using unique textures. These are some of the reasons why blend maps are usually used only on large unique surfaces, such as terrain.

Blend masks are also not as lightweight and flexible as vertex painting, as you need the images, which take space and cannot be edited in-engine. They can be quite large, and they add up, and even just one of them takes up much more memory than vertex color.

Out of these two options, vertex painting is generally preferred for its ease of use since it is editable in-engine. A practical example could be an environment artist creating a village scene. They might set the terrain to have a grass texture as the base and then for vertex painting set up the shader to have a dirt path texture, a soil texture, and a puddle texture all corresponding to the red, green and blue vertex data values. The artist can then easily paint directly on the terrain mesh where they want the paths, farmland, and puddles to be. If they need to move a

building into another place, they can simply erase and redraw the vertex paint of the path leading to the house. If the terrain textures were made using a blend map, there would be a lot more guesswork involved, since the artist cannot see the texture in relation to its environment when editing it in an image editor. This can lead to longer iteration times.

3 PROJECT

3.1 Goals and scope

A 3D scene was created to accompany the theoretical part of the thesis. The intent was that it would provide a practical viewpoint to how the texturing techniques discussed in this thesis are used and a deeper understanding of the different benefits and drawbacks of using these techniques.

Previous students' thesis projects had shown that the scope and scale of the project was something to consider carefully or else it might grow unmanageable for the timeframe. This is why the project was planned with three separate levels of completion in mind. The smallest would have been with only one of the houses finished, medium one with a row of houses finished and the largest with the whole street finished. The middle scope with the one row of houses was chosen as the primary goal and it ended up being a realistic decision as that is what was finished in time for the thesis.

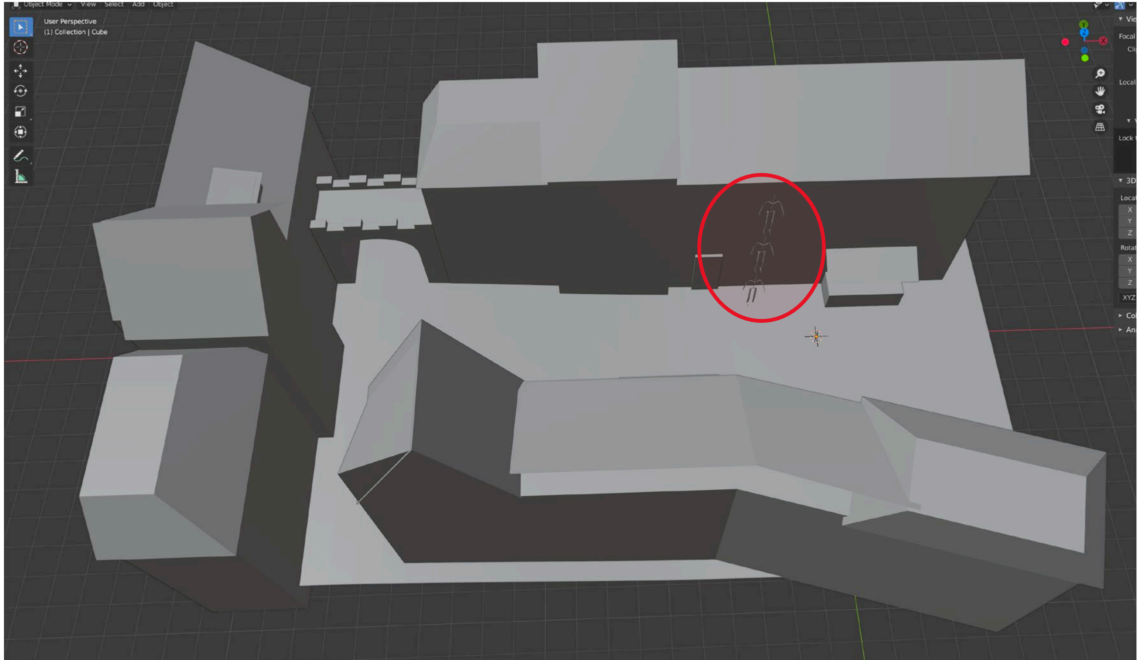
Unreal Engine 4 (UE4) was chosen as the game engine for the project. This was a good opportunity to learn it and a lot of the learning material referenced in the thesis is for UE4. The target platform for the environment was PC and consoles.

A medieval, German inspired theme with stone and half-timbered buildings was selected. The setting was chosen because those types of houses lend themselves well to using trim sheets and tiling textures. The scene was decided to be a portion of a street from a moderately wealthy part of town with townhouses on both sides and one somewhat fancier building at the end of the street.

3.2 Planning

A block-out was created to help plan the scene and to gauge how big of a task it will be. The block out was done for the largest scale version of the project with all the buildings included, even though only one side of it ended up being used. 1.8m tall human figures were placed in the scene to visualize how big everything needs

to realistically be (picture 24). 1.8m is considered the standard height for this purpose. Comparing doorways and floor heights to realistic people's height took out a lot of the guesswork. The layout of the block out carried through to the end mostly unchanged. The main difference is that two of the roofs were changed out for gable roofs.



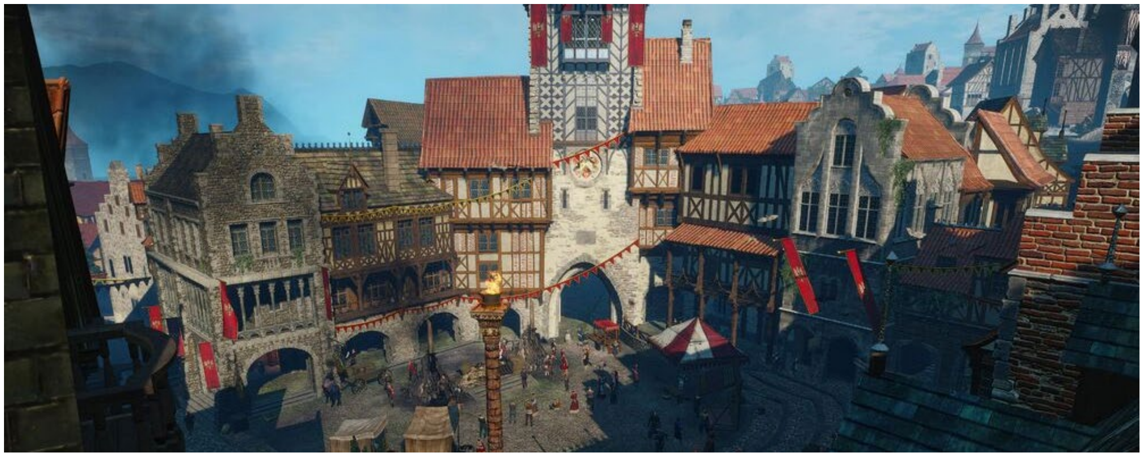
PICTURE 24. Block out with the human size references circled.

The next step was to figure out the textures. A rudimentary plan was made in Photoshop to help visualize the task ahead (picture 25). The plan included one texture atlas, that would have the trim sheets as well as any other necessary small textures like windows and door handles. Just one atlas texture was used to save on draw calls, save texture space and simply to see how much detail could be put into the scene with this limitation. The plan also included seven tiling textures, five larger ones for the large surfaces and two smaller ones for vertex painting variety on to the larger textures. One of the smaller ones, the dirt texture, eloquently referred to as gunk in picture 25, turned out to be unnecessary in the end. It did not match the vision as vertex painting it on the buildings made the scene look too murky. The grass texture also changed to a moss texture because the moss texture could also be used on the roof and not just on the ground.

PICTURE 25. Early plan for all the necessary textures

A texel density of 10.24 was selected as this was said to be quite standard in modern games these days by professionals in the industry (Crumpler 2021). This meant that the textures would have 1024 pixels per meter. It was decided that the larger textures would be 2048 by 2048 pixels, as they need to cover very large areas. With the selected texel density a 2048 texture covers two meters, and this in part keeps the tiling from being too repetitive. The smaller textures are 1024 by 1024 pixels big but looking back they could have been smaller. They are mainly used for vertex painting small irregularly shaped areas, so any repetition would not have been very noticeable.

Style-wise the games The Witcher 3: Wild Hunt and Overwatch served as the main sources of inspiration (pictures 26 & 27). The Witcher 3's worlds were a great source for architectural ideas whereas Overwatch was referred to more for color and textures. The Witcher's worlds are much grimier and semi-realistic in style, whereas for this project a stylized, bright, welcoming atmosphere was desired. Having two good sources for reference allowed for picking and choosing the best parts of both. For many of the smaller architectural and decorative details, real life German and Tudor buildings were referenced.



PICTURE 26. Hierarch Square in the game The Witcher 3: Wild Hunt (Witcher Wiki 2016).

PICTURE 27. Eichenwalde map in the game Overwatch (Overwatch Wiki 2016).

3.3 Production

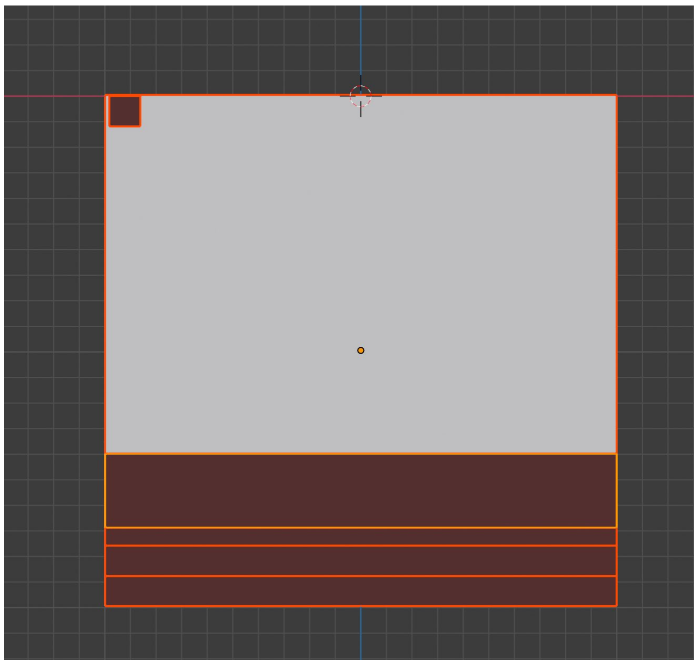
Having a clear plan for the textures and a block out for the scene made it easy to move on to the production phases. The textures were created first.

3.3.1 Creating the textures

The first textures made were the trims. The trim sheet has trims for wood beams in two different sizes, an end cap for the wood beam and a row of five stone bricks. Having two of the same size wood trim made the wood part of the trim-sheet wider, which allowed larger wood surfaces, such as doors, to be textured

in fewer pieces. The trimsheet has a lot of empty space left over, which later had the rest of the atlas textures added into it.

To create the trims, some base meshes had to be created first. The base meshes need to be as wide as the final textures need to be and as long as the final texture is wide so that the trims tile. An easy way to make sure that the measurements match up, is to first create a plane as big as the texture. The base mesh can be created by adding cuts and extruding the plane. In this project's case that was a 2x2 meter plane (picture 28).



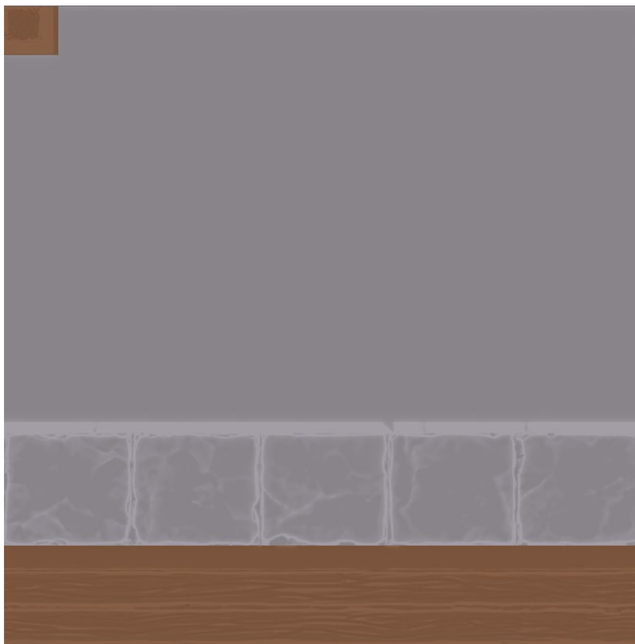
PICTURE 28. Trim base meshes (brown) on top of the 2x2 meter plane they were later baked on (white).

The base meshes were then exported into ZBrush where they had some wood grain and stone details sculpted on to them. It was important to be mindful of the detail level, so that any repetition would not be easily noticeable. To help create the seamless tiling effect, ZBrush has a setting called wrap mode in the Curve menu. When it is turned on, sculpting on one edge of the mesh also affects the other side as if they were connected.

A 45-degree angle was sculpted on the sides of both the wood and stone trims. This mimics the Ultimate trim technique discussed in section 2.3.1, and fakes a

higher poly look on the objects that use the trim. A remarkable amount of geometry was saved even on the wood beams alone since there are so many of them in the scene.

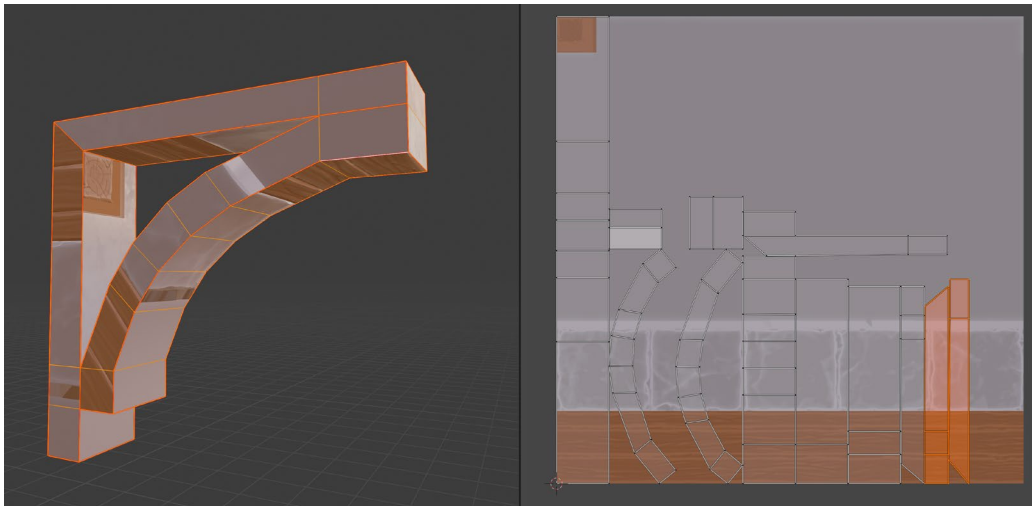
After the sculpting was done, the trims were brought to Substance Painter which is a texturing program that allows for fast texturing utilizing the data from the high poly versions of the 3D models. In there a normal map was baked from the sculpts to the 2x2 meter plane. The rest of the texture maps were created from the bake using Substance Painter's different generators (picture 29).



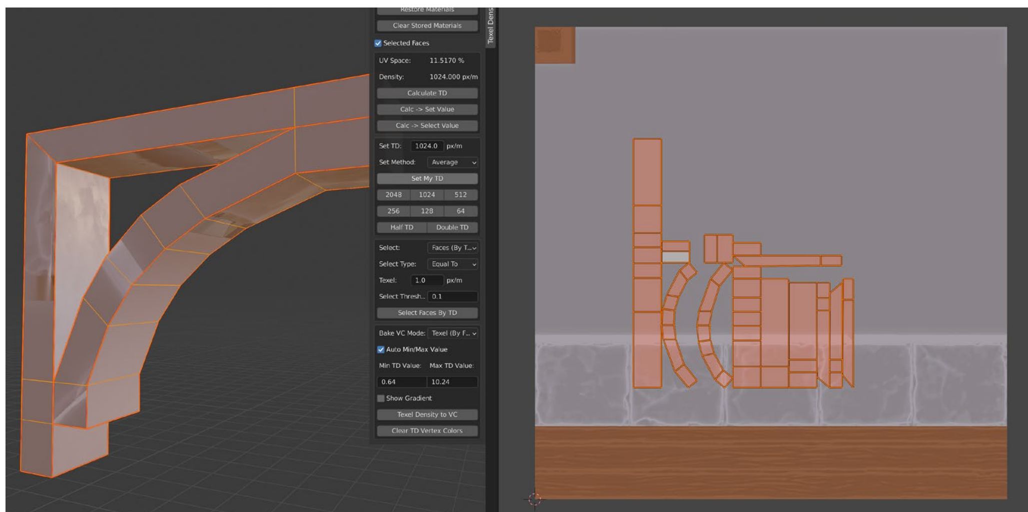
PICTURE 29. Color map of the finished trims.

It should be noted that Blender and Unreal Engine use different methods for handling their normal maps. Blender uses OpenGL and Unreal Engine uses DirectX. The difference between the two is that the green channel is inverted on the Y axis. The textures were created to go into Unreal Engine and so use DirectX. That is why the following screenshots taken in Blender may look odd and different from the final screenshots taken in Unreal Engine.

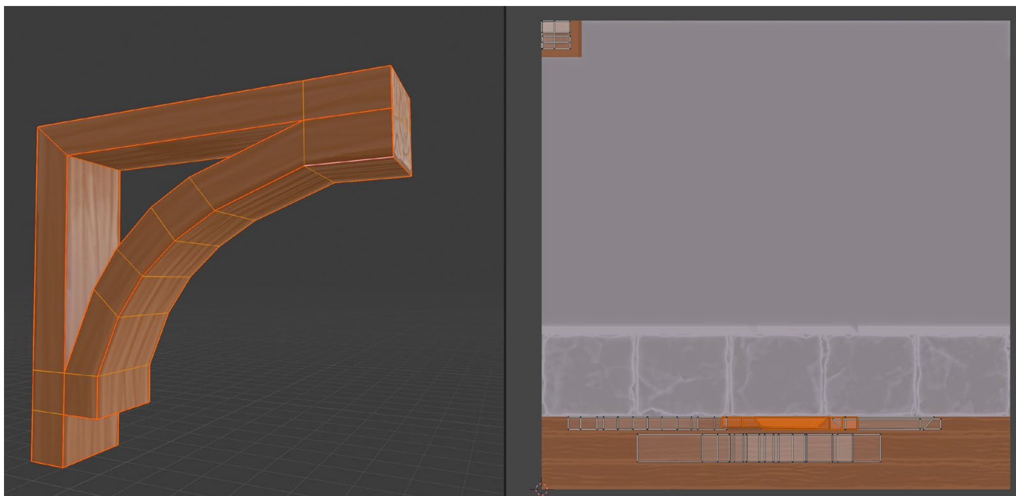
Texturing with the trim sheet was very straight forward and fast. A Blender addon called The Texel Density Checker was used to keep consistent texel density through every object in this project. Pictures 30, 31 and 32 describe the workflow of using trims textures with the addon.



PICTURE 30. First the model is UV unwrapped as usual.



PICTURE 31. The Texel Density Checker -addon rescales the UV islands to match the set texel density.

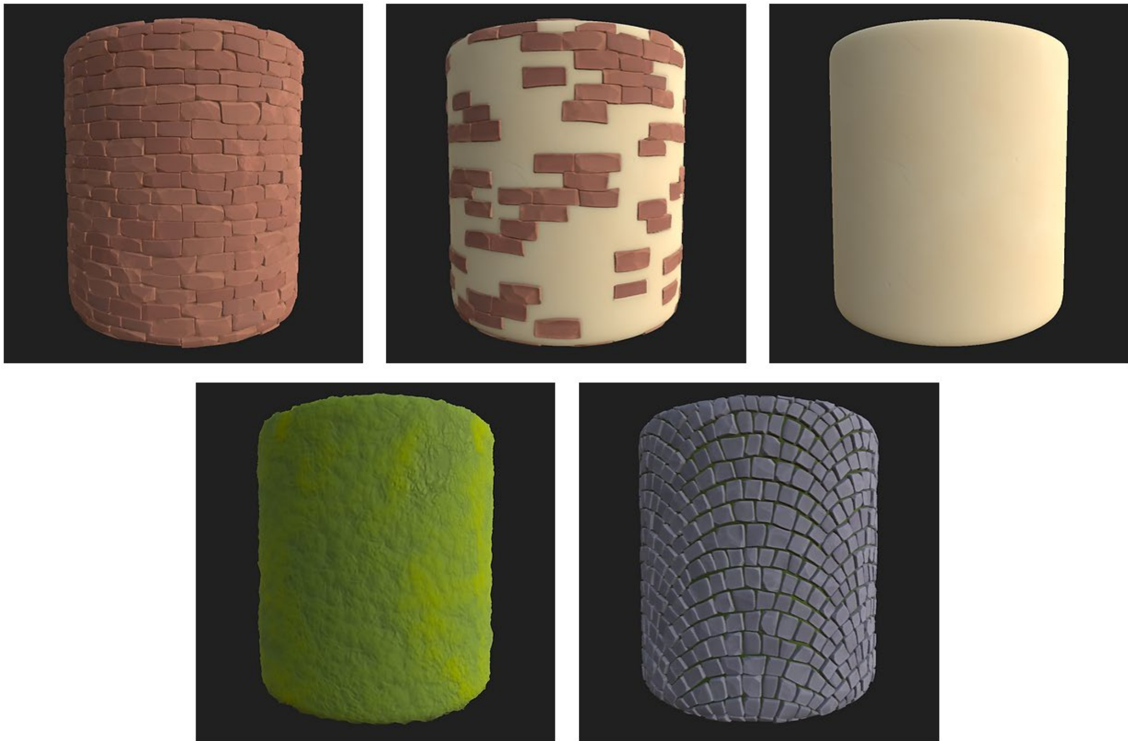


PICTURE 32. The correctly sized UV islands are placed along the trims. Any curved UV islands need to be straightened first.

Piecing together all the individual small textures to the texture atlas was one of the last things done for the project. Having the individual pieces as their own textures and materials allowed for much faster iteration on them which is why it did not make sense to do it any earlier. The texture atlas was left with some empty spaces as this project is expected to continue after the thesis is done (picture 33).

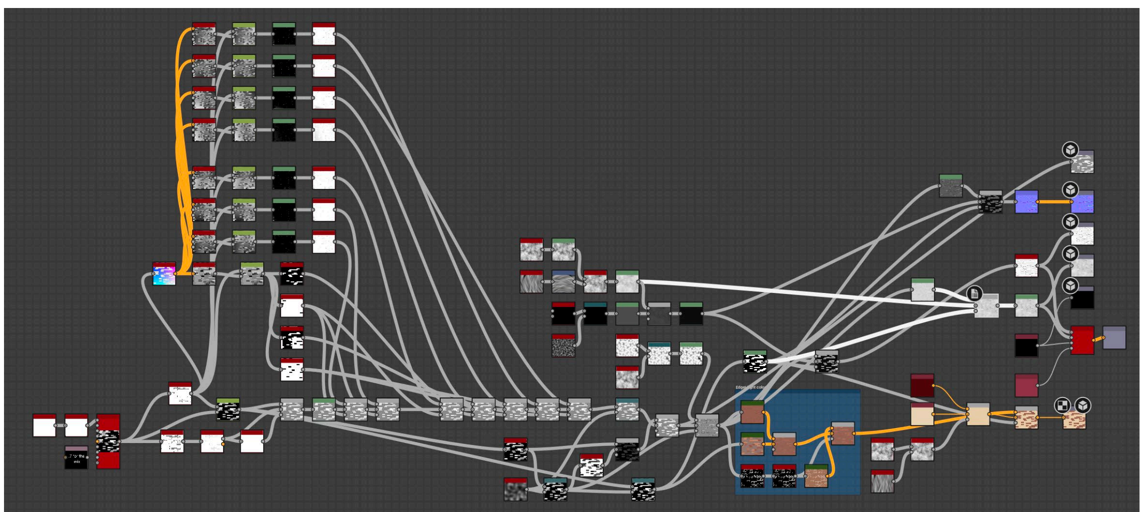
PICTURE 33. Color map of the final texture atlas material.

Most of the tiling textures were made in Substance designer. Substance Designer was a new program to learn, which slowed down the work considerably. It was worth it since it is a very useful program. It has a node-based interface which lends itself for procedural generation of materials and nondestructive workflows. Materials out of Substance Designer also tile perfectly by default, so it was nice to not have to worry about the seams. The textures done in Substance Designer were the three wall textures as well as the cobblestone and moss textures seen in picture 34.



PICTURE 34. Materials made in Substance Designer.

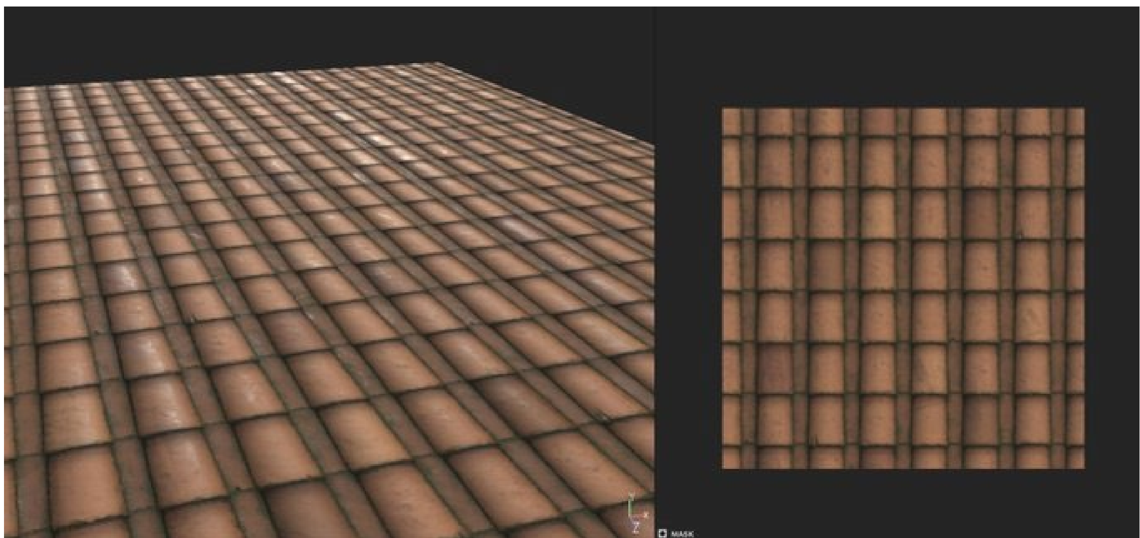
The three wall textures were all made using the same node tree seen in picture 35. The basis of it is made following a brick wall tutorial by YouTuber 3DEx with some alterations. The biggest addition was the ability to decrease the number of bricks on the material and so showing more and more of the plaster material from underneath. This is done in one of the first nodes with a simple slider. Having the three materials of the brick wall, plaster wall and the mix of them was key in achieving a natural blend for the vertex painted wall on the middlemost building.



PICTURE 35. Substance Designer node tree for the three wall textures.

The tiling texture for the roof was made with a somewhat similar workflow as the trim sheet. First a base mesh of just two rows of tiles was created. This mesh had some wear and tear sculpted on in ZBrush, after which it was brought back to Blender. In Blender the bricks were duplicated and randomly placed so that they covered a two-by-two-meter area. The bottom and top rows and the rows on the far sides had to be the same sets of tiles for the tiling effect to be seamless.

This large set of sculpted roof tiles was then textured in Substance Painter (picture 36). An old, weathered look was desired, while still aiming to keep the roof from being off-puttingly dirty. Some algae was placed in the crevices, which makes the roof look convincingly old but also helps to emphasize the shape of the tiles. Some tiles were randomly colored to be a slightly different tint. This was done by painting a mask for one tile and then using a transform effect to change its position. This layer was duplicated and moved again to get the natural color variation that old terracotta tiles have. Should something similar need to be done again, a different technique would be used, as this one made the Substance Painter file be almost impossible to re-open and it crashed the computer several times, unless the folder containing the tile variation layers was hidden.



PICTURE 36. The terracotta roof texture in Substance Painter.

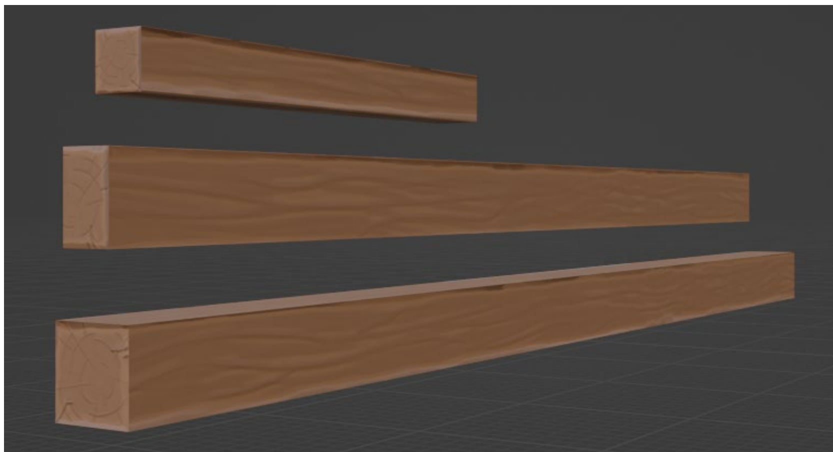
All the ambient occlusion, roughness, and metallic texture maps made for the project were channel packed to save space. Substance Painter did this automatically, but for Substance Designer channel packing had to be manually set up. This was done by adding a RGBA merge node and connecting the desired texture

node outputs to it. The merge node had to then be given its own output node. These can be seen in picture 35 as the red and gray nodes furthest on the left.

3.3.2 Modelling the assets

The base for the buildings and terrain was done in the block out phase, so it was easy to start working off that. All the assets were made in Blender using very basic modelling techniques.

A lot of the design process used for modelling was trying to think of different ways to use the trim textures. The most frequent use for the trims was on the wood beams that are very prevalent throughout the scene (picture 37). Creating a couple of different sizes of the beams at the start made it quick to duplicate them as needed. Because of the trimsheet, the beams could be changed to be any length without having to do anything to the textures or UVs. The trims were also used on stairs, window trims and to cover the underside of roof eaves to name a few.



PICTURE 37. Different shapes of wood beams used in the scene.

Using just one atlas texture did at times feel a little limiting, but not being able to texture every object uniquely meant having to get creative with what was already there. The market stall is made by using the wood trims for all the wooden parts and the wall texture without the normal map for the fabric. The doors of the buildings are also a good example of what can be done with trim sheets. They all have a unique look to them even though they are made with the same exact textures (picture 38).

PICTURE 38. Three different doors made by using the same trim textures.

One additional thing to remember while modelling was the usage of vertex paint, which was discussed in section 2.6. Areas that would later in the engine receive vertex paint needed to have higher geometry. This meant that large flat surfaces that could have easily been just one polygon had to be divided into smaller parts, as seen on one of the walls in picture 39. The same thing was done on the roofs to make the texture blending of the moss look nice.

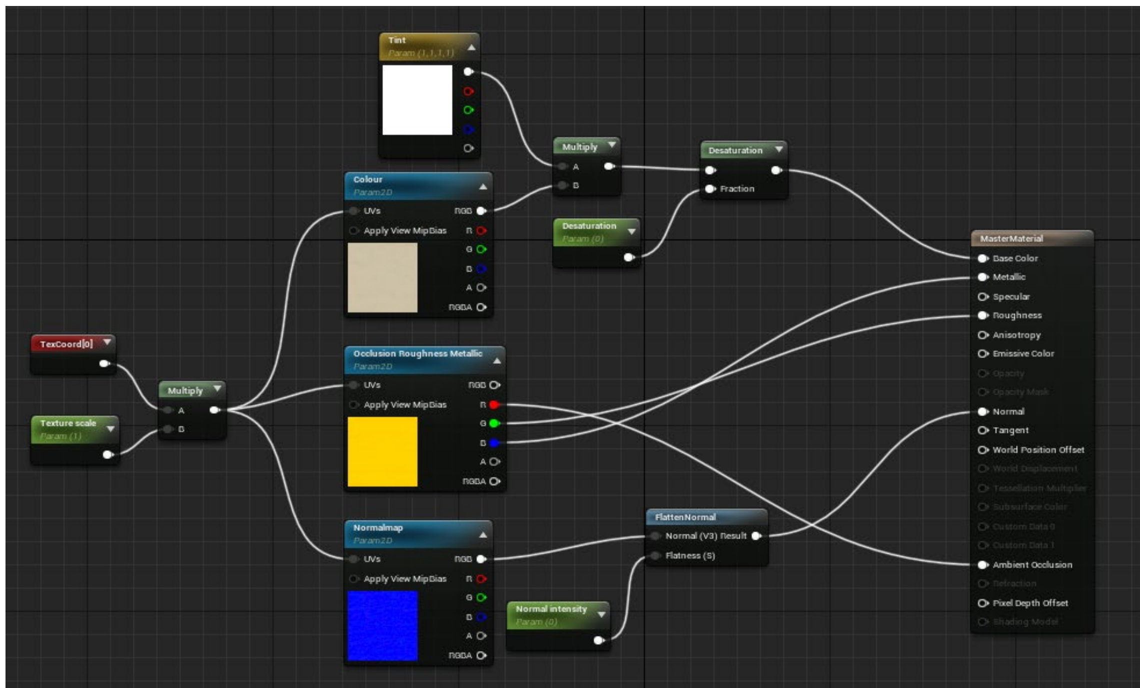
PICTURE 39. The house in the middle needed extra geometry for the brick vertex paint to work properly.

3.3.3 Unreal Engine

The finalizing work was done in the game engine. The first step was to create the shaders. Unreal Engine 4 has a node-based shader system, which has a definite learning curve, but makes the shaders easy to create and adjust for those who do not know how to code them.

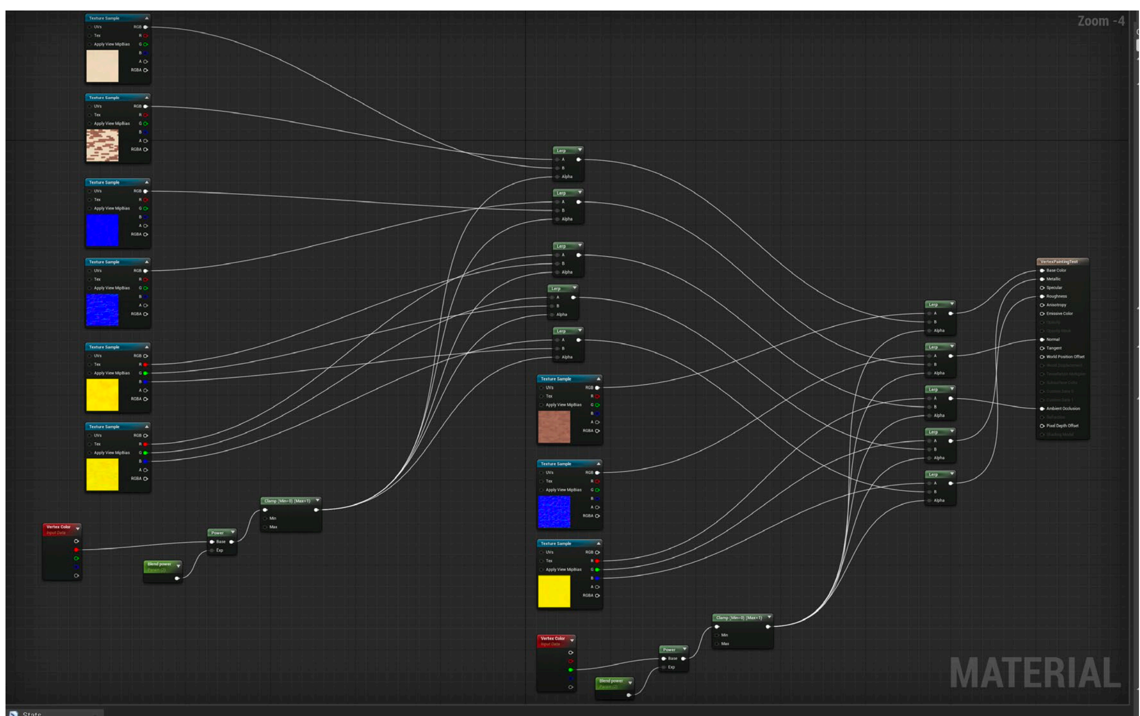
Two main shaders were created for this project. In Unreal Engine these are referred to as master materials. A master material acts as a template or a parent from which child material instances can be made. The material instances contain the same parameters set in the master material allowing the artist to alter the parameters in the material instance while not affecting the master material. This allows the same master material to be used for many material instances. Having the materials as instances keeps them easy to edit, since adding or removing parameters in the master material does the same for all the material instances. It also keeps them consistent for the artists to use, since all material instances parameters are set up in the exact same way.

The first master material created was a rather basic one (picture 40). Most of the items use this shader. It contains parameters for the color map, normal map and the channel packed ambient occlusion, roughness, and metallic maps. It also allows the artist to control the intensity of the normal map and tint the color. The shader also has the option to change the scale of the textures, but that did not prove to be very useful in this project.



PICTURE 40. Basic master material.

The other master material made was made for vertex painting (picture 41). The roof, cobblestone and the uneven brick wall use this shader. The shader takes the vertex paint and uses that to mix between the different materials. Vertex painting was one of the most fun parts of creating this scene. It felt really playful, and it was nice to see immediately how the scene came to life.



PICTURE 41. Master vertex painting shader.

Another part of the finalizing process was adding decals. Some cracks and dents were added to the plaster to create the effect that the houses have history and have been there for a long time (picture 42). An interesting discovery made while working with Unreal Engine’s decal system is that the decals are by default very large, roughly 5 meters in every direction. This led to some unwanted effects with the shadows when scaling it back down by hand as the scale easily went into negative numbers. Scaling by typing in the desired scale value resulted in more consistent results.

PICTURE 42. Damage decal.

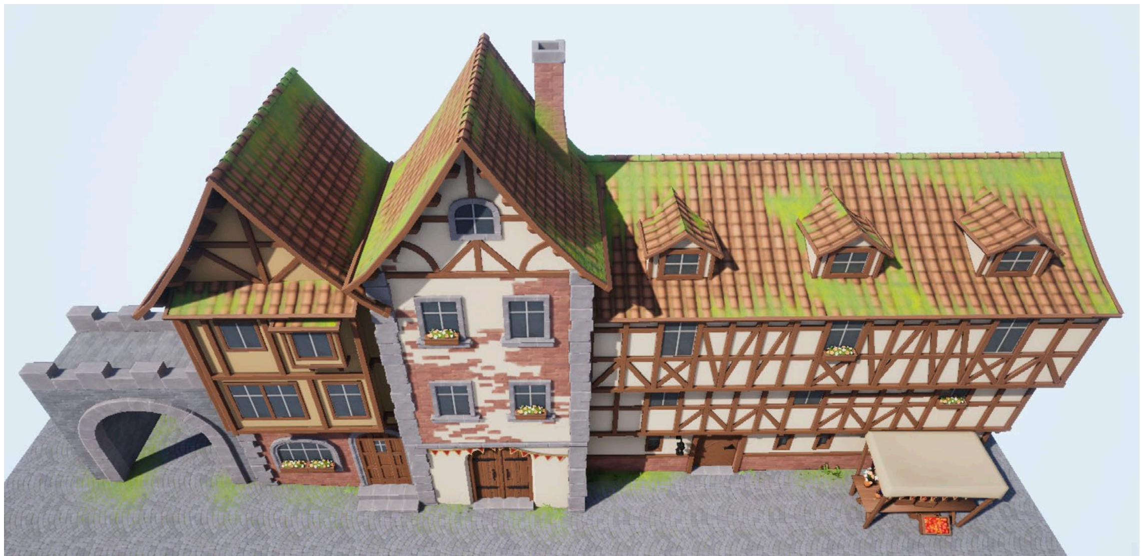
3.4 Result

Creating a scene using the efficient techniques that the thesis covers was quite fast. Not texturing every little corner uniquely saved a ton of time and texture memory. In addition to the various techniques, timewise having a solid plan from the beginning was also key for efficiency while working on the project. The techniques also brought a lot of life and detail to the scene, namely vertex painting and decals.

The current final scene includes only the buildings on one side of the road. This was expected from the beginning and work on this project will continue after the thesis is finished. The biggest hindrance for the project was learning the new programs involved, especially Substance Designer. As the tiling textures and most of the texture atlas are already done, finishing the rest of the block out to a similar detail level should not take nearly as long.

All in all, the scene as it is currently met the goals set in the beginning. It shows the different techniques in practice, it uses the textures well and visually it has a pleasant and bright atmosphere. Below are several screenshots of the scene as it is currently in Unreal Engine 4 (pictures 43-47).

PICTURE 43. Result.



PICTURE 44. Result.

PICTURE 45. Result.

PICTURE 46. Result.

PICTURE 47. Result.

4 DISCUSSION

Using the various efficient texturing techniques made creating a visually pleasing game environment with good in-game performance easy and quick. The tiling textures in combination with the trimsheet did most of the heavy lifting, they saved a lot of texture memory and production time, and even without the other textures, texturing with them alone would already make for a decent scene. Vertex painting and using the decals to add detail on top of the base tiling textures brings detail, realism, and visual interest to the scene. They also distract the viewer from the tiling effect. Channel packing the grayscale maps of all the materials into one took the amount of texture maps needed for one material down to three from the original six. This cut down on texture memory needed and had no real effect on the workflow, as getting channel packed materials out form the programs used was either instantaneous or very fast to set up.

The project demonstrated how using the efficient texturing techniques benefits the artist and how it would benefit the team when used on a larger scale. Most of the techniques should be included in the game development process from the beginning as going back to implement them afterwards wastes production time. Using the efficient texturing techniques can require the artists to expand their knowledge to the more technical side of things, such as shaders. If the artists do not know these techniques and the programs necessary yet, it can cause delays.

The thesis and project also covered the performance gains that come from using the various techniques. They include savings in texture memory space, draw calls. and in loading times. These are instrumental in videogame production and therefore the techniques are valuable tools for developers.

There are a couple of things that could be looked into as development suggestions and improvements to the project. One interesting one would be how the use of LODs affects the use of vertex painting, and what changes would need to be made to the workflow to account for the lower geometry models. Another one would be to try to utilize the Ultimate Trim technique discussed in 2.3.1., although the results of that technique would likely work the best on a considerably larger

scale project. The real results of the different texturing techniques on the project currently may also be seen better as the scene continues to grow.

In conclusion, the project successfully showed the results and gains of using each of the aforementioned techniques. They benefited both the workflow and performance as expected. It is clear now that adopting these techniques has many rewards that would compensate for the time it takes to learn them.

REFERENCES

Arm Developer. 2020. Real-time 3D Art Best Practices: Texturing. Read on 06.03.2022. https://developer.arm.com/documentation/102449/0100/Texture-channel-packing

Crumpler, C. ArtStation Learning. 2021. Environment Production - Part 6 - Production-Efficient Asset Creation. Watched on 02.02.2022. https://www.artstation.com/learning/courses/yV8/production-efficient-asset-creation/chapters/7jGO/trim-creation

Derozier, V. & Fleau, P. GDC Vault. 2019. Texturing the World of Assassin's Creed Odyssey. Watched on 03.04.2022.
https://www.gdcvault.com/play/1026145/Texturing-the-World-of-Assassin

Glad, A. Side FX. 2020. Vertex color based animation basics. Released on March 10, 2020. Read on 01.04.2022. https://www.sidefx.com/tutorials/vertex-color-based-animation-basics/

Gray, K. Nintendo Life. 2022. Soapbox: Pokémon Legends: Arceus Raises The Question - How Much Do Janky Graphics Matter?. Published on 28.01.2022. Read on 23.03.2022. https://www.nintendolife.com/features/soapbox-pokemon-legends-arceus-raises-the-question-how-much-do-janky-graphics-matter

Greuter, S. & Nash, A. 2014. Game Asset Repetition. Proceedings of the 2014 Conference on Interactive Entertainment. Association for Computing Machinery. Read on 03.04.2022.

Hegedüs, D. Runemark. 2019. PBR Materials for Unity. Published on 18.03.2019. Read on 16.4.2022. https://runemarkstudio.com/3d/pbr-materials-for-unity/

Hider, J. Jess's UE4 Tutorials. 2017. Realtime Rendering for Artists. Read on 24.03.2022. https://jesshiderue4.wordpress.com/real-time-rendering-an-overview-for-artists/

Hurtubise, F. 80LV. 2018. Using Tileable Textures in Game Environments. Published on 13.02.2018. Read on 04.04.2022. https://80.lv/articles/using-tileable-textures-in-game-environments

King, D. 2021. Twitter thread. Read on 20.03.2022. https://twitter.com/delaneykingrox/status/1504747059009953795

Kronenberger, L. Beyond Extent. 2020. Balancing Modularity and Uniqueness in Environment Art. Published on 14.09.2020. Read on 06.03.2022.

Lezzi, L. Normal Edge Decal Tutorial. 2019. Read on 04.04.2022. Requires access right. https://leonano.com/store

McGuire, M. Max's Programming Blog. 2010. Blending Terrain Textures. Read on 25.03.2022 http://web.archive.org/web/20140809193644/http://www.m4x0r.com/blog/2010/05/blending-terrain-textures/

Olsen, M. GDC Vault. 2015. The Ultimate Trim: Texturing Techniques of Sunset Overdrive. Watched on 3.4.2022. https://www.gdcvault.com/play/1022323/The-Ultimate-Trim-Texturing-Techniques

Overwatch Wiki. 2016. Read on 10.4.2022. https://overwatch-archive.fandom.com/wiki/Eichenwalde?file=Eichenwalde_screenshot_3.png

Świerad, O. Tech Art Aid. 2016. UE4: Detailed Texture Blending with Height-Lerp and Vertex Painting. Video. Watched on 24.03.2022. https://www.youtube.com/watch?v=dghCetkArJI

Unity Documentation. 2020. Decal Renderer Feature. Manual. Read on 05.04.2022. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@12.0/manual/renderer-feature-decal.html

Unity Documentation. 2022. Providing vertex data to vertex programs. Manual. Released on 19.03.2022. Read on 24.03.2022. https://docs.unity3d.com/Manual/SL-VertexProgramInputs.html

Unreal Engine 3 Documentation. Terrain Advanced Textures. Manual. Read on 24.03.2022. https://docs.unrealengine.com/udk/Three/TerrainAdvancedTextures.html

Unreal Engine 4.27 Documentation. Decals. Manual. Read on 05.04.2022. https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/Decals/

Witcher Wiki. 2016. Read on 10.4.2022. https://witcher-games.fandom.com/wiki/Hierarch_Square?direction=next&oldid=332860

Yang, R. Radiator. 2013. Hacking blend transition masks into the Unity terrain shader. Publised on 10.09.2013. Read on 17.04.2022. https://www.blog.radiator.debacle.us/2013/09/hacking-blend-transition-masks-into.html