Benjamin Lindquist

Game Environment Texturing

Texture Blending and Other Texturing Techniques

Metropolia Ammattikorkeakoulu

Medianomi (AMK)

Viestintä

Opinnäytetyö

30.4.2015

Tekijä Otsikko Sivumäärä Aika	Benjamin Lindquist Peliympäristöjen teksturointi – Tekstuuriblendaus ja muut teksturointitekniikat 63 sivua + 1 liite 30.4.2015
Tutkinto	Medianomi
Koulutusohjelma	Viestintä
Suuntautumisvaihtoehto	3D-animointi ja -visualisointi
Ohjaaja	Lehtori Kristian Simolin

Opinnäytetyön tavoitteena on osoittaa, miten reaaliaikaisia peliympäristöjä voidaan teksturoida sujuvasti ja tehokkaasti sekä millaisia menetelmiä tähän tarkoitukseen on olemassa. Havainnollistan perinteisten teksturointimenetelmien rajoituksia ja tarjoan sujuvampia vaihtoehtoja näiden tilalle. Olen tutkinut monia teksturointimenetelmiä kattavasti lukemalla lukuisia lähteitä ja käyttämällä useita niistä sekä työssä että vapaa-ajalla. Käyn läpi näiden eri menetelmien ominaisuuksia sekä niiden hyviä ja huonoja puolia esimerkkien kautta havainnollistaen. Pääasiallisena aiheena käyn läpi tutkimustani blendkarttojen käytettävyydestä reaaliaikaisen teksturoinnin yhteydessä, sekä paneudun syvemmin siihen, miten kehitin reaaliaikaisen blendkarttoja hyödyntävän varjostinohjelman Blenderissä Bugbear Entertainment Ltd:n Environment Art -tiimille. Lopuksi esittelen myös muutamia muita vaihtoehtoisia teksturointimenetelmiä, kuten esimerkiksi modulaarista teksturointia, joita olen tutkinut ja käyttänyt työelämässäni. Esittelen myös joitakin lupaavia menetelmiä, joita olen tutkinut vapaa-ajallani, vaikka en vielä ole itse niitä käytännössä soveltanut.

Monet opinnäytetyössä esittämäni menetelmät ovat tulleet tutuiksi työskennellessäni Bugbear Entertainment- ja Remedy Entertainment peliyhtiöissä. Käytän pääasiassa tietoa, jota olen oppinut Bugbear Entertainmentin Next Car Game: Wreckfestin sekä Remedy Entertainmentin julkaisemattoman mobiilipelin kehityksen aikana. Opinnäytetyössäni käyttämäni blendmateriaalitekniikat perustuvat pääasiassa työhöni Bugbearin Next Car Game: Wreckfest -pelin parissa ja modulaariset teksturointi- ja mallintamistyötapani ovat kehittyneet Remedy Entertainmentin mobiilipeliprojektin tuotannon aikana. Opinnäytetyössäni hyödynnän muun muassa tätä työkokemustani ja saamaani opetusta muilta alan ammattilaisilta vahvistaen tietojani eri lähteiden avulla.

Uskon tutkimustyöni eri teksturointimenetelmistä olevan hyötyä kenelle tahansa reaaliaikaisesta grafiikasta kiinnostuneelle. Olisin halunnut jatkaa varjostimeni kehitystyötä, jotta se olisi vielä yhtenäisempi Bugbearin blendvarjostimeen, mutta uskon, että tässäkin kehitysvaiheessa siitä voi olla suurta apua. Kehittämäni varjostin on vapaasti ladattavissa internetistä ja sitä voi käyttää Blender-ohjelmassa rajoituksetta. Siitä huolimatta, että tekstuuriblendaus on kehittynyt monella tapaa ja monia uusia menetelmiä on ilmestynyt, perinteinen tekstuuriblendaus on yhä varteenotettava menetelmä modernissa teksturoinnissa.

Avainsanat	3D, teksturointi, peli, reaaliaikainen, grafiikka, blendkartta, blendvarjostin, splat map, modulaarinen, proseduraalinen, bugbear, remedy

Author Title Number of Pages Date	Benjamin Lindquist Game Environment Texturing – Texture Blending and Other Texturing Techniques 63 pages + 1 appendix 30 April 2015
Degree	Bachelor of Culture and Arts
Degree Programme	Media
Specialisation option	3D Animation and Visualization
Supervisor	Kristian Simolin, Senior Lecturer

This thesis demonstrates how real-time game environments can be textured smoothly and efficiently and to pinpoint what kind of methods can be used for this purpose. I point out the limitations of traditional texturing methods and propose more efficient alternatives to them. I have gained a thorough understanding of many texturing methods by studying several sources and by using the methods both at work and in my free time. I discuss the properties as well as the pros and cons of these methods with reference to real-world examples.

The thesis reports on the research I did regarding the usage of blend maps in real-time texturing as well as an in-depth look into how I developed a real-time blend shader in Blender, to be used for this purpose, for the Bugbear Entertainment Environment Art team. Finally I present some other alternative texturing methods, such as modular texturing, that I have researched and used in my line of work. I also highlight some promising methods that I have studied during my free time but which I have not tried out myself yet.

I have gained an insight into the methods I discuss in the thesis through my work experience at the Bugbear Entertainment and Remedy Entertainment game companies. I mainly use knowledge I have gained during the development of Bugbear Entertainment's Next Car Game: Wreckfest and an unreleased Remedy Entertainment mobile game. The blend material techniques I have used in the thesis are primarily based on the work I did for Bugbear's Next Car Game: Wreckfest and the modular texturing and modeling techniques have mainly been acquired during the development of the mobile game project for Remedy Entertainment. I rely on this work experience and the information I have received from other industry professionals, while simultaneously backing this knowledge with references to several sources.

I believe that my research into different texturing methods can be useful for anyone interested in real-time graphics. I would have liked to develop my shader further to be even more in line with the one used at Bugbear, but even at the state it is currently in, I feel that it can be of great help. The shader is freely downloadable from the internet and it can be used inside Blender without restriction. Traditional texture blending is still a viable method for modern texturing even though texture blending has evolved a lot in several ways and a lot of new alternative texturing methods have arisen.

Keywords	3D, texturing, game, real-time, graphics, blend map, blend shader, splat map, modular, procedural, bugbear, remedy

Table of Contents

1 Introduction 1

2 Terminology 5

3 Basics of texturing 8

4 Texture blending 12

4.1 Alpha maps 12

4.2 Blend maps 16

4.3 Blend mask 21

4.4 Vertex blending 23

5 Blend map workflow improvement 25

5.1 Blend map creation workflow at Bugbear 25

5.2 Blend shader development 27

5.3 Blend shader dissection 29

6 Other environment texturing techniques 39

6.1 Modularity 39

6.2 Decals 44

6.3 Virtual Texturing 47

6.4 Procedural texturing 48

6.5 Layered materials 51

6.6 Photogrammetry 52

7 Conclusion 56

8 Sources 58

Appendices

Appendix 1. Blend shader

1 Introduction

This thesis examines different aspects of texturing real-time game environments and demonstrates the challenges I have faced while creating content, such as large environments, while working in the game development industry, as well as methods to solve these challenges and texturing tasks in general. This thesis is mainly targeted at 3D artists, with the assumption that the readers have a basic understanding of texturing, modeling and math and they know the terminology related to these fields. I demonstrate the rigidity of traditional texturing methods and introduce different techniques to do texturing more efficiently. To a great extent, I rely on my several years of experience in the field of 3D graphics in general as well as my professional experience in the game development industry at companies such as Bugbear Entertainment and Remedy Entertainment. I will also refer to a select few sources.

Image 1. The Bugbear Entertainment logo (Bugbear Entertainment 2015a).

Bugbear Entertainment is one of the leading developers of destruction themed car racing games and it is well-known for its high quality products. Bugbear Entertainment's most notable accomplishments include such games as Rally Trophy, the FlatOut series, Ridge Racer Unbounded and their most recent game, Next Car Game: Wreckfest. Even though Bugbear's games strive to be believable they often compromise realism in order to keep the complete chaos of the race fun and exciting. My experience at Bugbear has mostly been related to an upcoming game called Next Car Game: Wreckfest, which is a spiritual successor to the popular FlatOut series.

Image 2. A screenshot from Bugbear's upcoming game, Next Car Game: Wreckfest (Next Car Game: Wreckfest 2015a).

The rights to the FlatOut name are not a property of Bugbear Entertainment and the third installment in the series was actually developed by another studio. The third game was not very well received and was generally quite harshly critiqued by the players and reviewers alike. An actual FlatOut sequel by Bugbear is quite unlikely but Next Car Game: Wreckfest means to appeal to the same target audience. Next Car Game: Wreckfest is available, at the moment of writing, through Early Access on Steam and features much of the same mayhem that made the FlatOut series so popular.

Image 3. The Remedy Entertainment logo (Remedy Entertainment 2015).

Remedy Entertainment is currently the largest AAA game development company in Finland and one of the largest game development companies overall in Finland in regards to revenue and number of employees. The company is known across the world for its high quality games. With a very well received backlog of games, Remedy has managed to be highly successful in the AAA action game market. Remedy mainly focuses on immersive story-driven action games and it is the creator of several successful games such as Max Payne and Alan Wake. The games have been extremely popular with over 10 million copies sold worldwide. Remedy has previously mainly concentrated on the AAA production of games, but in recent years it has also started developing games for other platforms such as mobile phones. The most recent mobile release, Agents of Storm, was Remedy's first in-house developed mobile project. The second mobile project, which I was initially working on when joining Remedy, eventually got canceled.

One of my main points of focus in this thesis concerns a flexible and powerful way of texturing large environments. The technique is called splat mapping or blend mapping. I will refer to this method as blend mapping throughout the thesis as that is the term I am used to. I will demonstrate the flexibility and efficiency of this process as well as discuss other methods to achieve similar end results. There are several more recent methods that are more flexible than blend mapping and I will pinpoint and walk through them as well. I will elaborate on the techniques used at Bugbear in the process of creating blend maps and I will reflect on the problems of working this way as well as techniques on how to further improve the workflow.

A part of my thesis will address a real-time texturing shader I developed for the purpose of improving the workflow of creating blend maps during my time at Bugbear Entertainment. I have learned a lot about different techniques regarding development of convincing environments while working at Bugbear and several of the methods demonstrated in this thesis have been put to the test during the development of Bugbear Entertainment's upcoming game Next Car Game: Wreckfest.

Having worked for nearly two years at Bugbear, I was ready for new challenges. When a former colleague asked me to join the ranks of a mobile team at Remedy Entertainment, I felt this change of scenery was the right step to take at the time. At Remedy Entertainment, I was also tasked with creating environments, but the restrictions and techniques used during the development of the project were quite significantly different from what I had gotten used to at Bugbear. I will go through these techniques and address their benefits and drawbacks as well. One of the main methods of environment creation in the game project I was working on when I joined the company, was modular modeling and texturing for mobile. This approach to creating larger environments efficiently was quite different as it mainly relied on the diversity of the way I was able to use limited texture atlases to my advantage with a clever texturing layout and modeling workflow.

2 Terminology

An **alpha map**, **mask** or **stencil** in the context of this thesis is an image commonly containing gray-scale color information that defines where a specific texture or material is rendered (3D Buzz 2008; Blender Wiki 2011).

A **blend map** - also known as a splat map - commonly refers to an RGB(A) image texture containing masking information in each color channel that are used to define where different textures or materials are rendered. Image formats that support an alpha channel allow for even more textures or materials to be blended. Simply put a blend map is a group of alpha maps packed into one image file. (Glasser 2005; L3DT documentation 2013; Shader Forge Wiki 2014; Wikipedia 2013.)

Blend mapping refers to the method of using blend maps, vertex colors or some other form of masking to texture terrains or assets. In the context of this thesis blend mapping refers to RGB image based blending. (Glasser 2005; L3DT documentation 2013; Shader Forge Wiki 2014; Wikipedia 2013.)

A **blend shader** is what I call the shader I developed to visualize the painting of blend maps.

The **clipping planes** define the distance from the in-engine camera at which the z-buffer is calculated. The clipping plane has a near and a far value which defines the volume in the camera frustum that is rendered as well as the accuracy of the z-buffer. (Baker; GitHub 2013; Microsoft Developer Network 2015a.)

The camera or view **frustum** is the volume that is defined by the near and far clipping planes. (Baker; GitHub 2013; Microsoft Developer Network 2015a.)

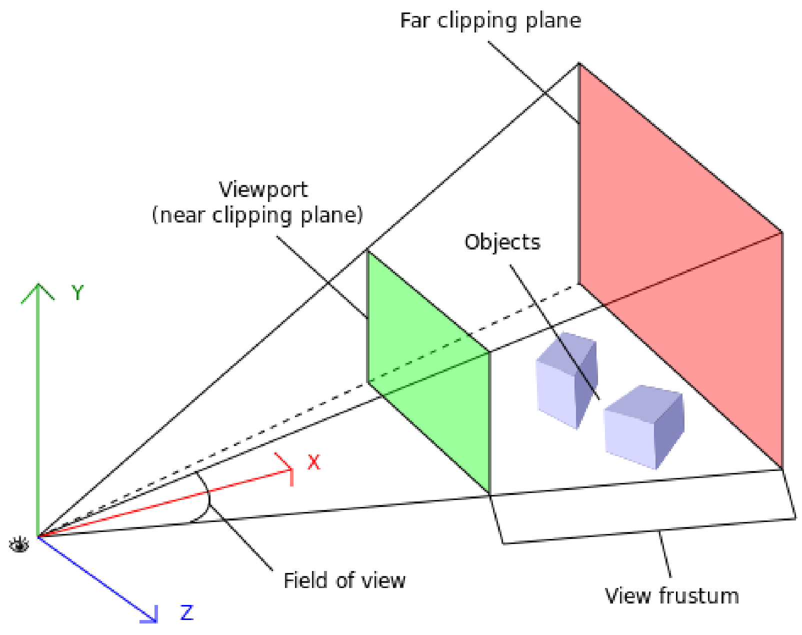


Image 4. View frustum is the volume defined by the near and the far clipping planes (GitHub 2013).

Mip mapping refers to the use of mip maps which are pre-calculated smaller versions of the main texture that are used for texture filtering in order to avoid artifacts like moire patterns (Computer Graphics Laboratory).

A **node** in the context of this thesis refers to a specific operation within a shader, based on a visual scripting system (Blender Wiki 2014).

A **node network** - also sometimes called a noodle - is a group of nodes that lead to the desired end result and can contain several different operations (Blender Wiki 2014).

A **shader** is a small computer program used to do shading. It describes the properties of a vertex or pixel. The program tells the computer to draw something in a specific way. (DirectX.com.)

A **texel** is a basic unit to describe a pixel in a texture image. **Texel density** refers to the resolution or the amount of pixels that are mapped onto a specific area on a 3D model. (Tamakuwala 2013.)
Texture streaming refers to a technique where a texture's highest mip level is being loaded into memory when it is needed while other textures are unloaded to free up memory accordingly (Hastings 2007).

A **tileable texture** is a seamless image that does not contain any visible interruptions in the flow of the image. This texture can be used to create large surfaces by repeating it multiple times, in other words, the texture is tiled. (Polycount Wiki 2014.)

UV coordinates refer to the X and Y coordinates in 2D texture space. A 2D image is mapped onto a 3D model through the unwrapping or flattening of the 3D model into 2D space and is assigned UV coordinates. (Polycount Wiki 2015a.)

Vertex blending refers to a method of using vertex colors for blending textures and the **vertex color** refers to the color data each vertex in a 3D mesh can contain (Polycount Wiki 2011).

Viewport in the medium of 3D computer graphics refers to the rectangular window where the 3D objects can be viewed. Objects within the view frustum are projected to the viewport and can be seen on screen. (GitHub 2013.)

A **z-buffer** determines which object or surface is rendered on top of another in 3D space. A low bit depth or great distances can cause a problem called z-fighting. The z-buffer is nonlinear and has a greater accuracy at close distances. The distance at which objects or surfaces are rendered are defined by the clipping planes. (Baker; Microsoft Developer Network 2015b.)

Z-fighting means that the calculations that determine which surface to render on top of another are non-conclusive and that it can not be determined which object or surface should be rendered in front of another. This results in one of the surfaces or objects randomly showing through the other one and a visual flickering occurs. (Baker.)

3 Basics of texturing

In the early days of 3D graphics, different colors for surfaces could only be achieved by having separate objects or surfaces that were assigned specific colors each. Edwin Catmull invented the method of wrapping a 2D image onto a 3D object. This method was called texture mapping and it allowed an artist to add a great deal of visual surface variation without the need for multiple objects or surfaces. The image could either be wrapped across the whole model, adding unique detail everywhere, or repeated multiple times across the surface. (Animation Supplement 2012.)

Using a single large image to texture assets is still quite common if a large texture budget is available or when creating smaller objects, but it can be a resource intensive way of working and it is especially problematic when facing large scale texturing tasks. Having large areas covered by a single gigantic texture is rarely a good idea, especially in game development, as it generally means a great deal of memory and processing power is needed to deal with that information. (Panda3D Manual.)

Image 5. A work-in-progress environment created by the user rouncer on gamedev.net. The whole environment is a single large image painted by hand. (rouncer 2013.)

Such software as World Machine allow one-texture landscape generation which is great, if one can afford it in the texture budget. Despite the resource heaviness, one of

the advantages of a single image method is that it allows for a much greater control for the artists, as they can paint any detail where they want and are not restricted by anything else than the resolution of the image in question.
(World Machine Software LLC 2013.)

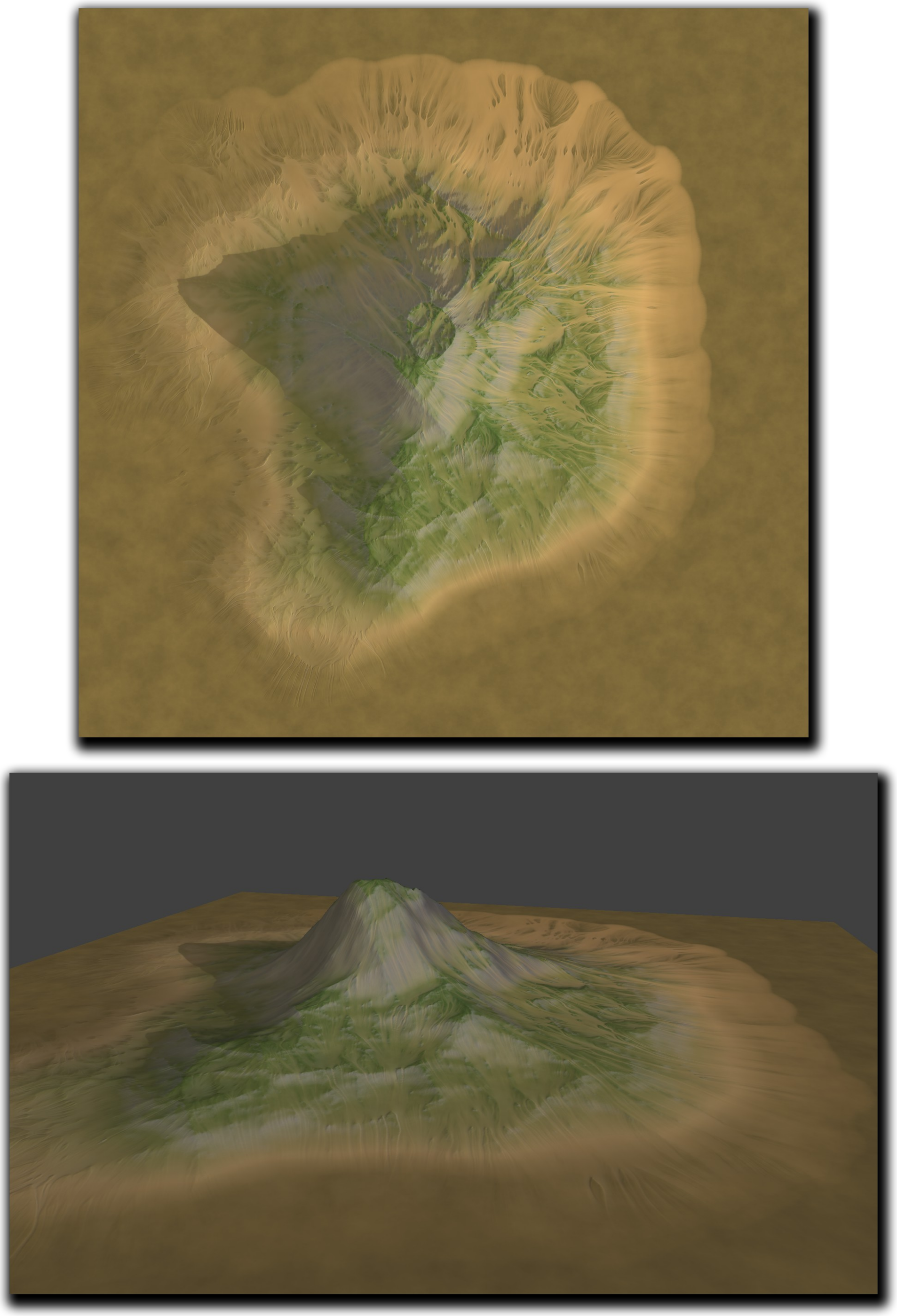


Image 6. World Machine provides one-texture landscape generation.

Even though this kind of texturing has been something to use sparingly before, it has been revisited in recent years as new kinds of texture streaming methods have been developed. Methods like id Software's megatexturing allows for a lot of finer control by the artist without having an enormous texture constantly stored in memory. This topic will be covered in greater depth in chapter 6.3. (id Software 2009.)

Another quite common solution is to have a tileable texture cover the whole object or terrain, which allows for a fairly high resolution as the image is repeated numerous times over the surface. This method restricts the artist from larger scale control, however, and oftentimes a distinctly visible repeating pattern can destroy the illusion of a believable unified surface. (Animation Supplement 2012.)

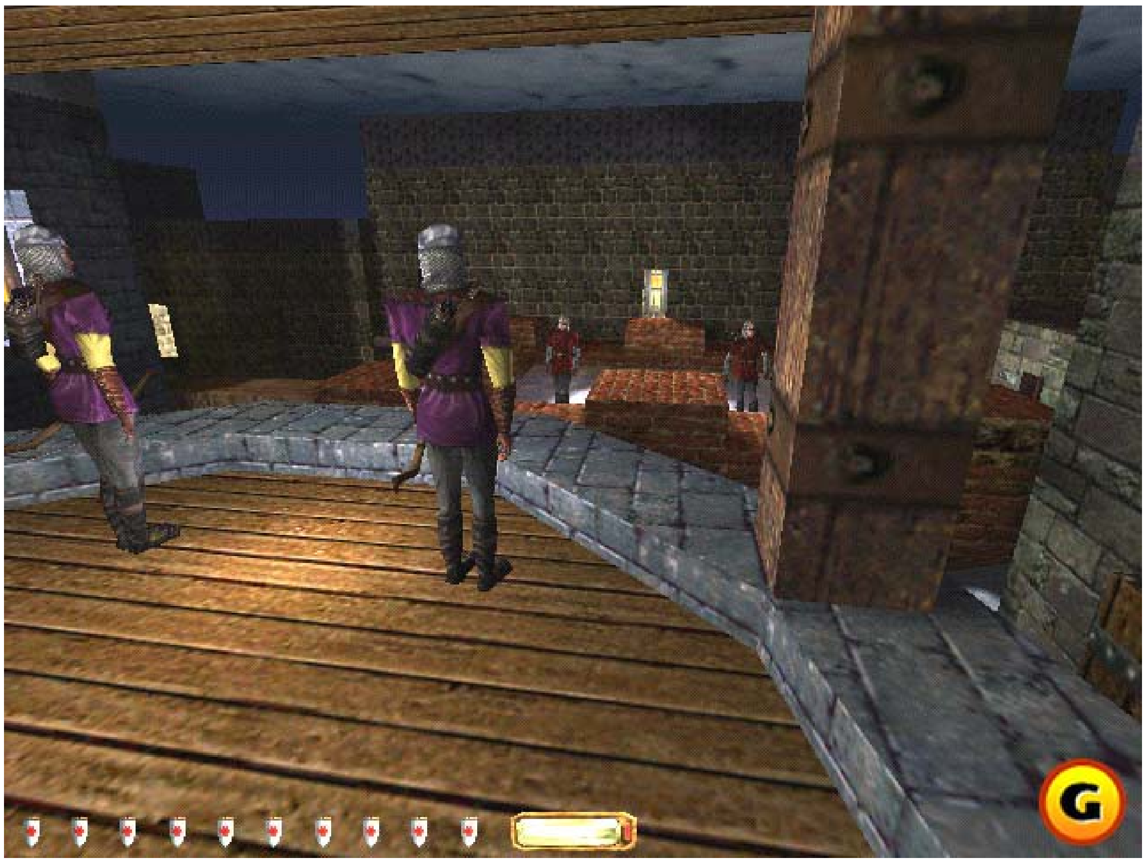


Image 7. A screenshot from Thief II: The Metal Age. The repeating patterns of the tileable textures are extremely noticeable. (Thief II: The Metal Age 2000.)

Table 1. The pros and cons of traditional texturing methods.

Texture type	Pros	Cons
Large uniquely mapped texture	Can paint specific details anywhere	Requires a lot of memory
Tileable texture	Small memory footprint	Lack of fine control

In order to have the flexibility of the large single image texturing method and the high definition of the tileable texture method, one could try to combine these two methods in some way. This is where blend mapping comes in.

4 Texture blending

Blending different textures or materials together to create large diverse surfaces is quite an old method that is used extensively in the game industry even today. There are different ways of blending textures or materials together, but they all strive to add variety to surfaces without the need to create a single large texture that contains all the required detail. (Glasser 2005; L3DT documentation 2013; Shader Forge Wiki 2014; Wikipedia 2013.)

4.1 Alpha maps

Before more complex multi-channel blend maps are analyzed, a simple two-texture blend will be looked at to make the principle easier to understand for anyone unfamiliar with the subject. This simple version of a blend map is a single black and white image called an alpha map, stencil map or mask. This black and white image is used to define where two different specified textures or materials are rendered. For the sake of simplicity, the technique will be explained in the context of a simple two-texture blend. (3D Buzz 2008; Blender Wiki 2011.)

Image 8. An alpha map used to blend two different textures together.

One texture or material is defined to be rendered wherever white occurs in the alpha map. The other texture is defined to be rendered wherever there is black. All the values between black and white result in a blend of the two specified textures or materials. Taking advantage of the gray values will make the transition from one texture to another a lot smoother than simply using either fully black or white colors, so this should be kept in mind when making organic transitions between different surfaces. As an example, an artist might want to create a large field of grass with a path crossing it. Using an alpha map, it could be defined that wherever a value of one occurs, a path texture is rendered and wherever a value of zero occurs, a grass texture will be rendered. This way the artist would not have to create a single large texture with both the path and grass in the same texture file as in the case of the one-texture method. Instead two separate textures could be set, one for the path and one for the grass. These textures are then placed according to the information contained in the alpha map. A bit of sand could also easily be blended in from the path over the grass areas near the transition to the path to make the switch of textures more smooth and believable. Using tileable textures allows repeating detail such as grass multiple times over the surface resulting in a high definition with a relatively low memory hit. The alpha map in itself does not necessarily have to be of a very high resolution as it is simply used for the blending of the textures. There are, however, also some tricks for adding additional detail to the blending if needed, which will be looked at in greater detail in chapter 4.3. (3D Buzz 2008; Blender Wiki 2011.)

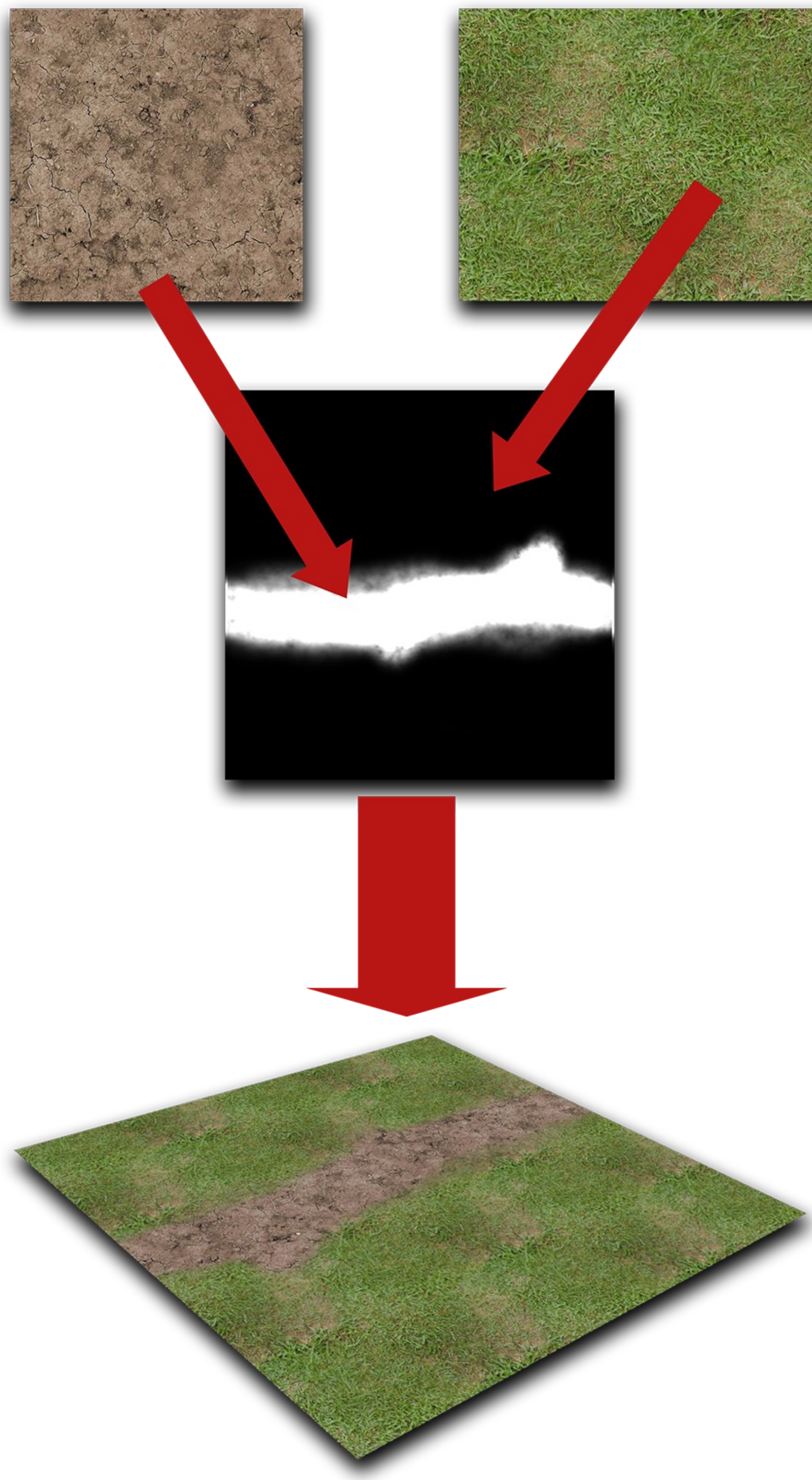


Image 9. An example breakdown of a simple blend of two different textures.

This method is a significant time saver and allows for a great deal of flexibility as only the alpha map needs to be modified, if changes to the location or shape of the path are required. Often the mesh geometry will contain detail for the path which would have to be changed, but for prototyping and testing, just modifying an alpha map makes for immensely fast iteration times. Color, density or type of material could also easily be changed by simply modifying or changing out the grass color texture and the blended terrain will update accordingly. (3D Buzz 2008; Blender Wiki 2011.)

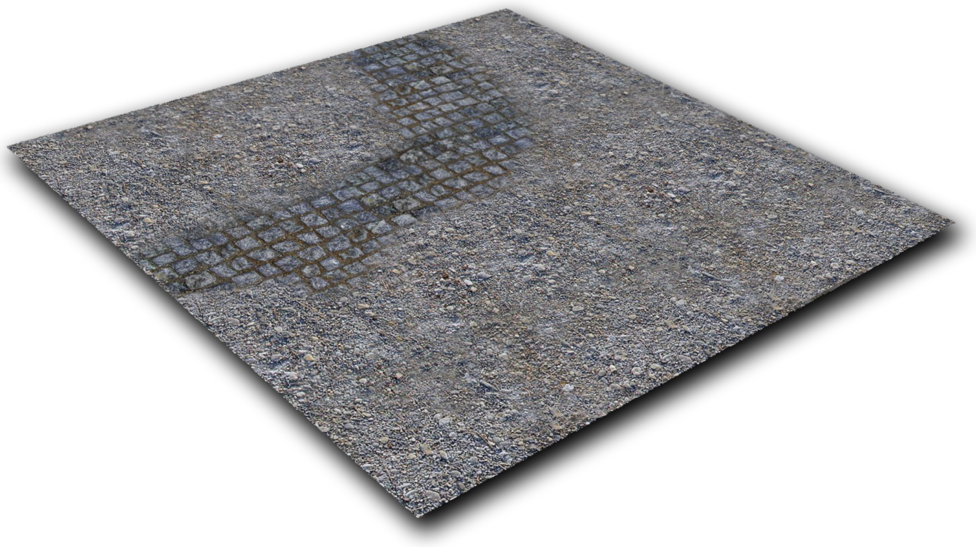


Image 10. An example of how easily a completely different look can be created by changing the ground textures and modifying the alpha map to take a bend instead of going in a straight line.

As can be seen from the last example even though the path takes a turn the tileable cobblestones will not follow the direction of the bend. This is something that would be countered with additional geometry and some clever unwrapping that will be covered in chapter 5.3.

4.2 Blend maps

Blend maps are basically a collection of several alpha maps packed into a full RGB color map. Each channel of the blend map acts as an individual alpha map. Any information in a specific color channel will blend in the texture related to this channel at that point and everything else will be specified by the other channels. These color channels can be utilized to blend different textures or materials together to form a larger unified surface, as in the previous example of the simple alpha map. Most commonly, blend maps are used together with different tileable textures to achieve a large texel density as well as variation. Having many different tileable textures blending together will naturally add a lot more variation than could be achieved with just a single tileable texture. The power of blending different textures together is that a very high resolution result can be achieved while avoiding most of the visually noticeable tiling due to the fact that variation is being added by blending in different textures. For even more textures to blend between a 32-bit image that contains an additional channel, the alpha channel, could be used. (Glasser 2005; L3DT documentation 2013; Shader Forge Wiki 2014; Wikipedia 2013.)

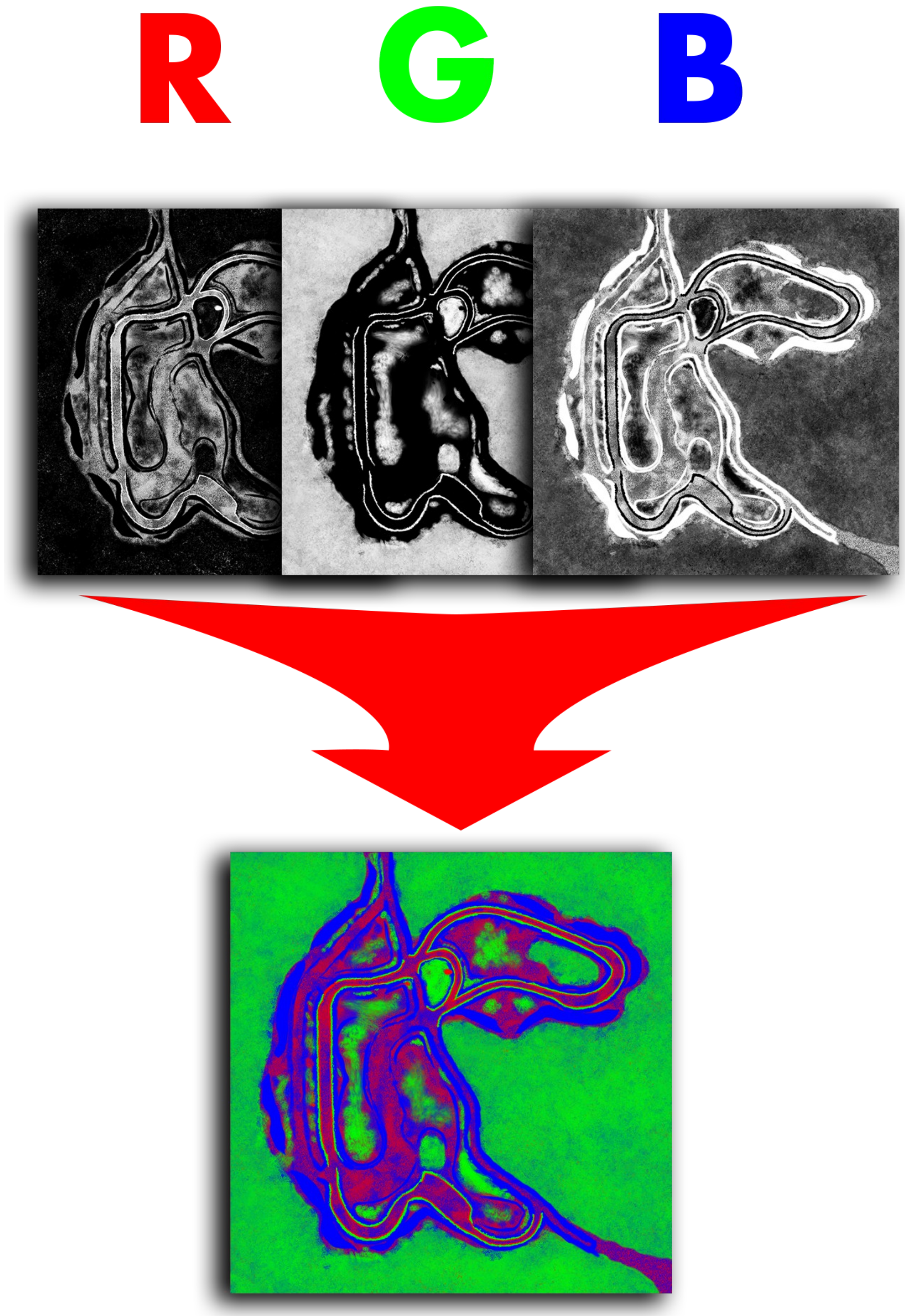


Image 11. The different color channels of this blend map can be thought of as separate alpha maps to make it easier to understand (Bugbear Entertainment 2013a).

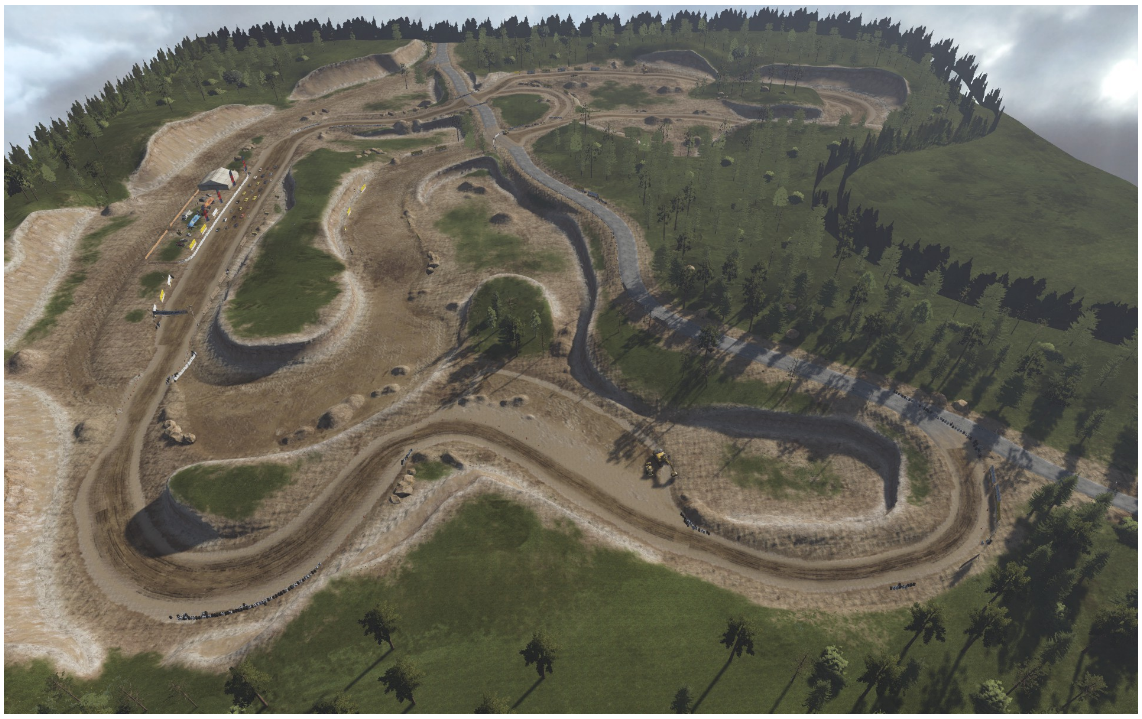


Image 12. The final result of the blend map in the previous example used in Next Car Game: Wreckfest (Next Car Game: Wreckfest 2015b).

There are some differences in how a game engine uses the blend map input. Some shaders, such as the one used at Bugbear, normalize values that are not a pure red, green or blue never to add up to more than the maximum value of a single color channel. If some kind of normalizing did not take place, a pixel with the maximum value in both the red and green channel, for example, would lead to the textures represented by both channels being added together at maximum opacity. Adding both textures together at maximum opacity would lead to undesirable, overblown results. Therefore, a maximum value existing in two channels (255, 255, 0) will instead mix the two with 50% opacity. All three color channels maxed out (255, 255, 255), a combined value resulting in white, would instead mix the three by 33%. The Bugbear shader does, however, aim to only show one specific material at any given point, so in most cases the materials are not blended linearly as in the previous two examples. A threshold set in the shader determines, how sensitive the texture blending is. What this means in practice is that a low value in a color channel will be ignored, if there is a large value in another channel at the same point. So a specific point with the color value 10 for red, 10 for green and 255 for blue would only show the blue, as the other values are low enough to be ignored. Had the value for the green channel been, say, 230, it would probably have been slightly blended in, depending on the threshold set in the shader. To simplify further, values that are close to each other are blended according to the threshold set in the shader while much lower values will be ignored. In practice this order-independent technique means that the order in which the textures are combined does not matter for the final result. Pure values of a single color channel will naturally only show the texture or material linked to that channel. (Shader Forge Wiki 2014; Unreal Engine Docs 2014b.)

This is very similar, for example, to how landscape painting works in the Unreal Engine when using weight based blending. Blending can be defined by a weight value that always normalizes all the textures' blending values to a total value of one. Increasing a material's weight value will decrease the weights of the other materials at that point. This is basically the same technique as adding different color channels together and normalizing the result. Blending textures by a weight value is also order-independent, which means that the order of the materials does not matter for the final result. (Shader Forge Wiki 2014; Unreal Engine Docs 2014b.)

There are other ways of blending textures that require for the textures to be ordered in a specific manner to get the intended results. In the case of using blend maps with this technique, a single pixel's total combined value can also be greater than the maximum value of a single color channel as every channel is treated separately and layered on top of each other instead of being combined. (Shader Forge Wiki 2014; Unreal Engine Docs 2014b.)

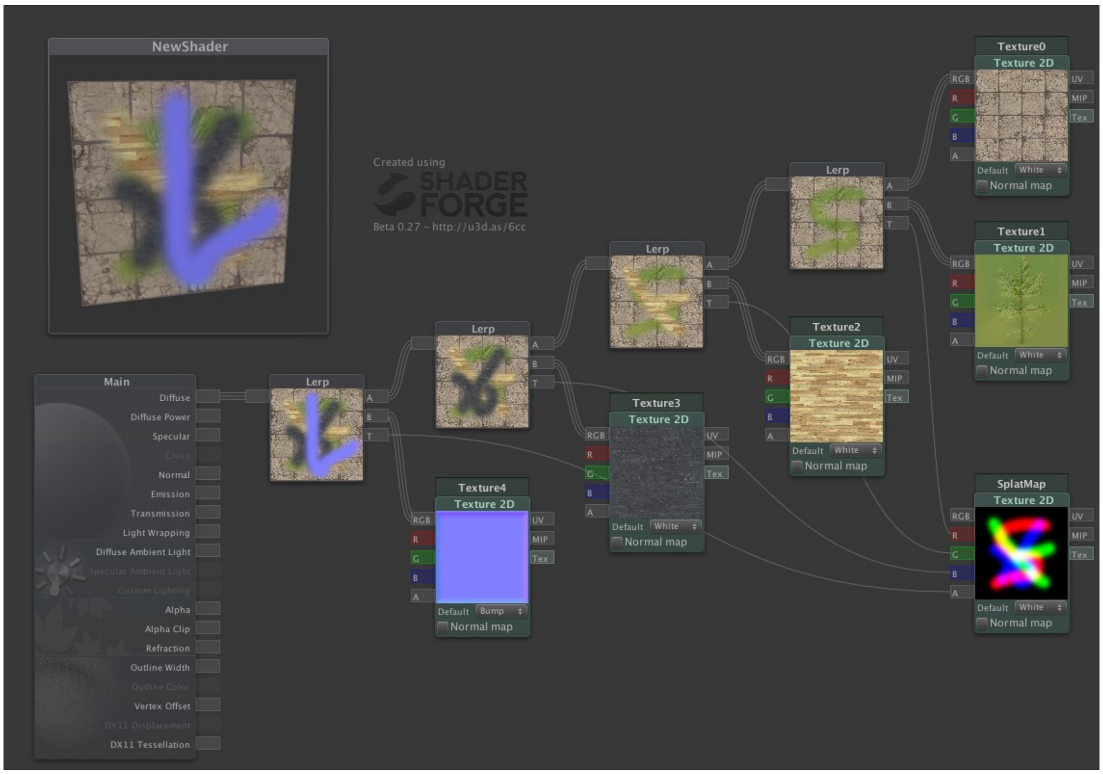


Image 13. As can be seen in this image, the order of the textures matters when using this technique (Shader Forge Wiki 2014).

In the previous example image, the color channels are painted separately and blended in a specific order to get a layering functionality. As can be seen in the image, the combined value where information exists in all channels looks overblown, but it does not matter for the final result, as every channel is treated separately and no normalizing is required. The layering functionality means that removing the green in this example would reveal the red underneath as the red channel is defined to be underneath the green. This method can be very useful when, for example, making terrain with snow on top. Removing the snow would automatically show the underlying terrain texture instead of having to paint it there manually. A drawback of this method is naturally that painting more red on a specific area would not make the linked texture show, if it was covered by green. Both methods have their benefits and it is up to the user to decide which method works best for each situation. This is also why in the Unreal Engine, for example, the user is allowed to choose which blending method to use. (Shader Forge Wiki 2014; Unreal Engine Docs 2014b.)

4.3 Blend mask

Sometimes the transitions between the different textures or materials can look a little too blurry or pixelated, if the resolution of the blend map is too low. A simple but effective trick to avoid this lack of detail in the blend map can be to add something called a blend mask. The blend mask aims to affect the transitions of the blend map. The blend mask can be a small tileable texture with, for example, a noise pattern and can either be a separate texture or stored in the tileable texture's alpha channel. The mask texture does not have to look good in this case; the noise pattern simply acts to break up the transition areas in a greater detail and hide the low resolution pixelation of the blend map. As this blend mask is tileable, it can be repeated numerous times over the terrain, which allows the artist to specify a higher resolution transition than what might be realistically achievable by a single large uniquely mapped blend map. This way of breaking up the transitions is a “one size fits all” method that won't take into account the nature of the different materials or textures that are being blended. (Erdőkövy 2013; McGuire 2010.)

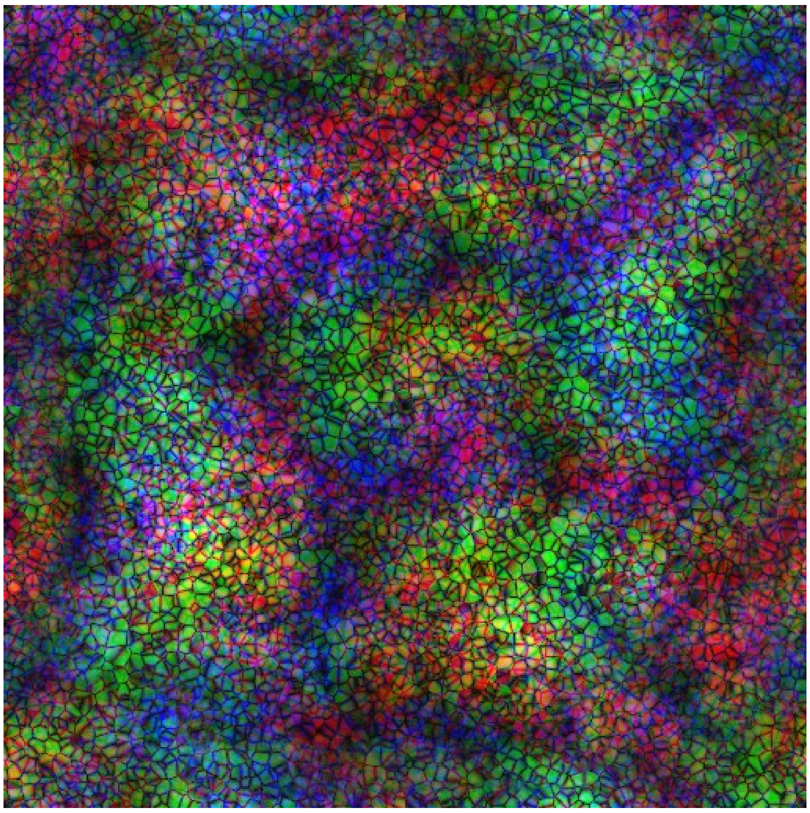


Image 14. A blend mask that breaks up the blend map in transition areas (Bugbear Entertainment 2013b).

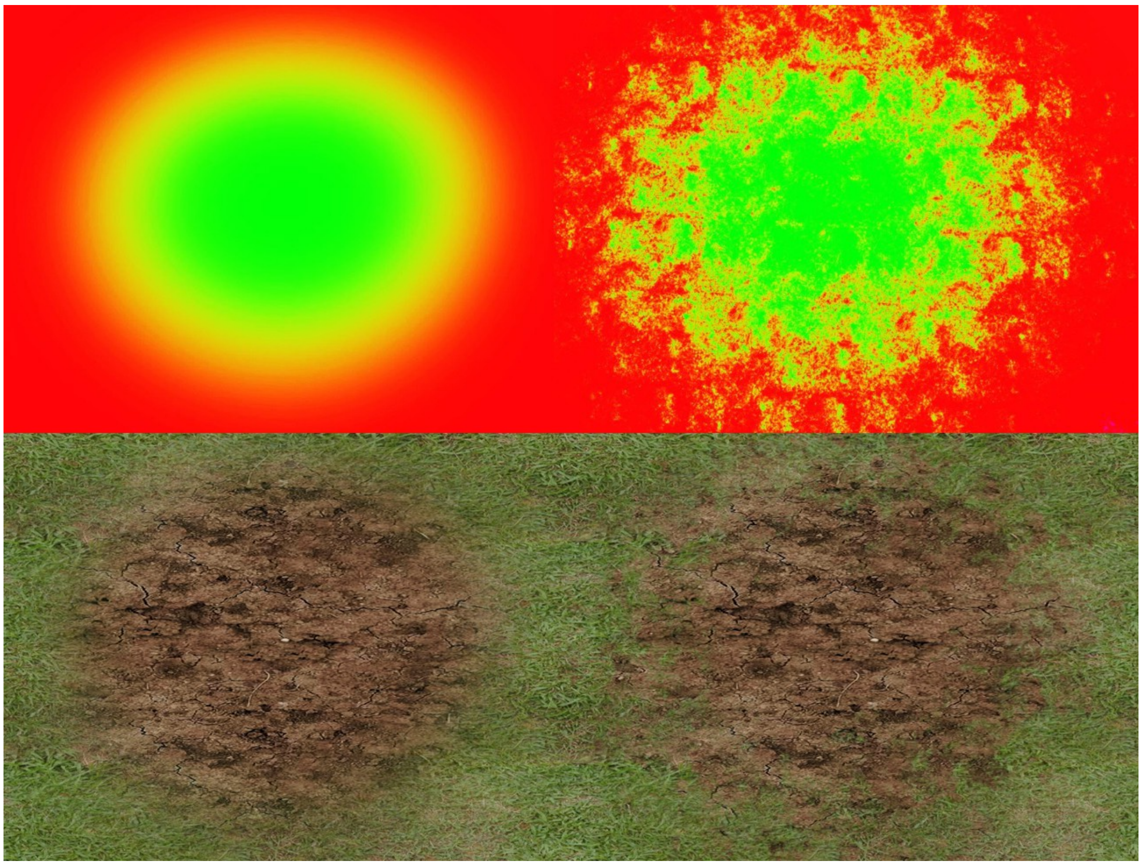


Image 15. An example of how the blend mask affects the transition areas of the blend map (Bugbear Entertainment 2014a).

The blend transitions might also look odd due to an unconvincing transition that does not make physical sense. In this case just ramping up the resolution will not fix the problem. For example, a brick wall transitioning smoothly into hanging foliage on the wall does not make sense as the foliage would follow the shape of the bricks and gather in the crevices before it covers the actual brick surface. A more sophisticated way of using blend masks is to have a material-specific mask that is tailored to make the transitions look convincing. In the case of a brick wall to foliage transition, a height map could be used to affect the blending in such a way that the crevices of the bricks get blended before the topmost surface of the bricks. (McGuire 2010; Yang 2013; Cryengine Manual 2014a; Cryengine Manual 2014b.)

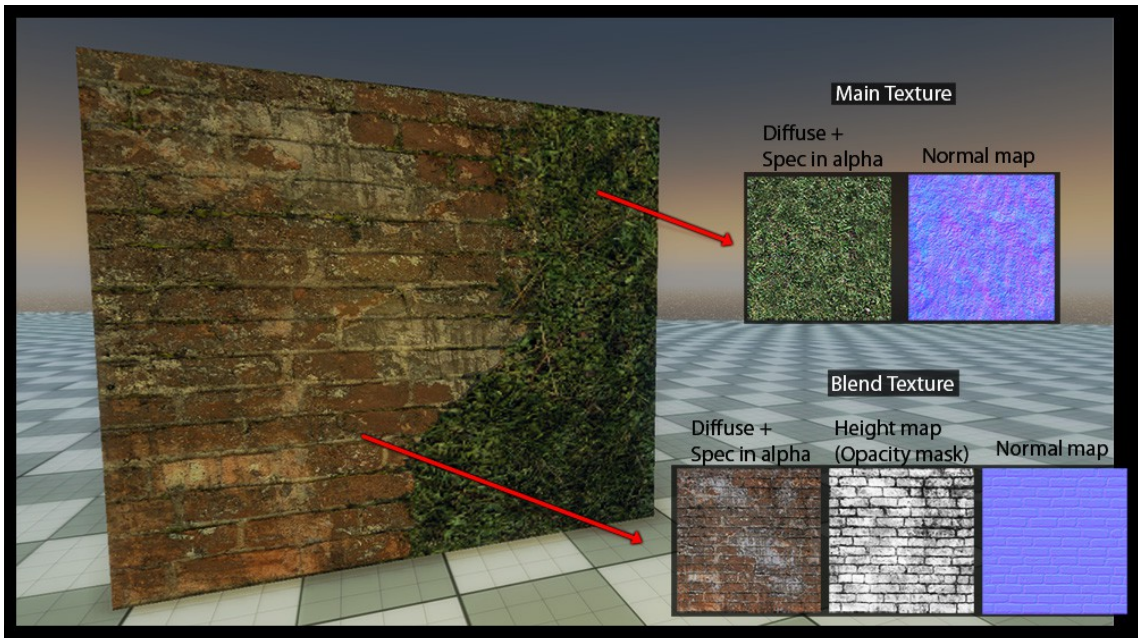


Image 16. An example of a height map controlled blending of textures. Note how the crevices of the bricks are filled with foliage near the transition area, which makes the transition look natural. (Cryengine Manual 2014.)

4.4 Vertex blending

In addition to texture-driven blend mapping, the vertex colors of a mesh can be used as the base for texture blending. This method is called vertex blending. A 3D mesh can contain color data in each of its vertices which can be utilized for different things such as animation, or, in our case, texturing. The color data in the vertices can be used in the same way a blend texture would be set up, but instead of having a bitmap image contain the needed colors, the data would rather be stored in each vertex of the mesh. (Polycount Wiki 2011.)

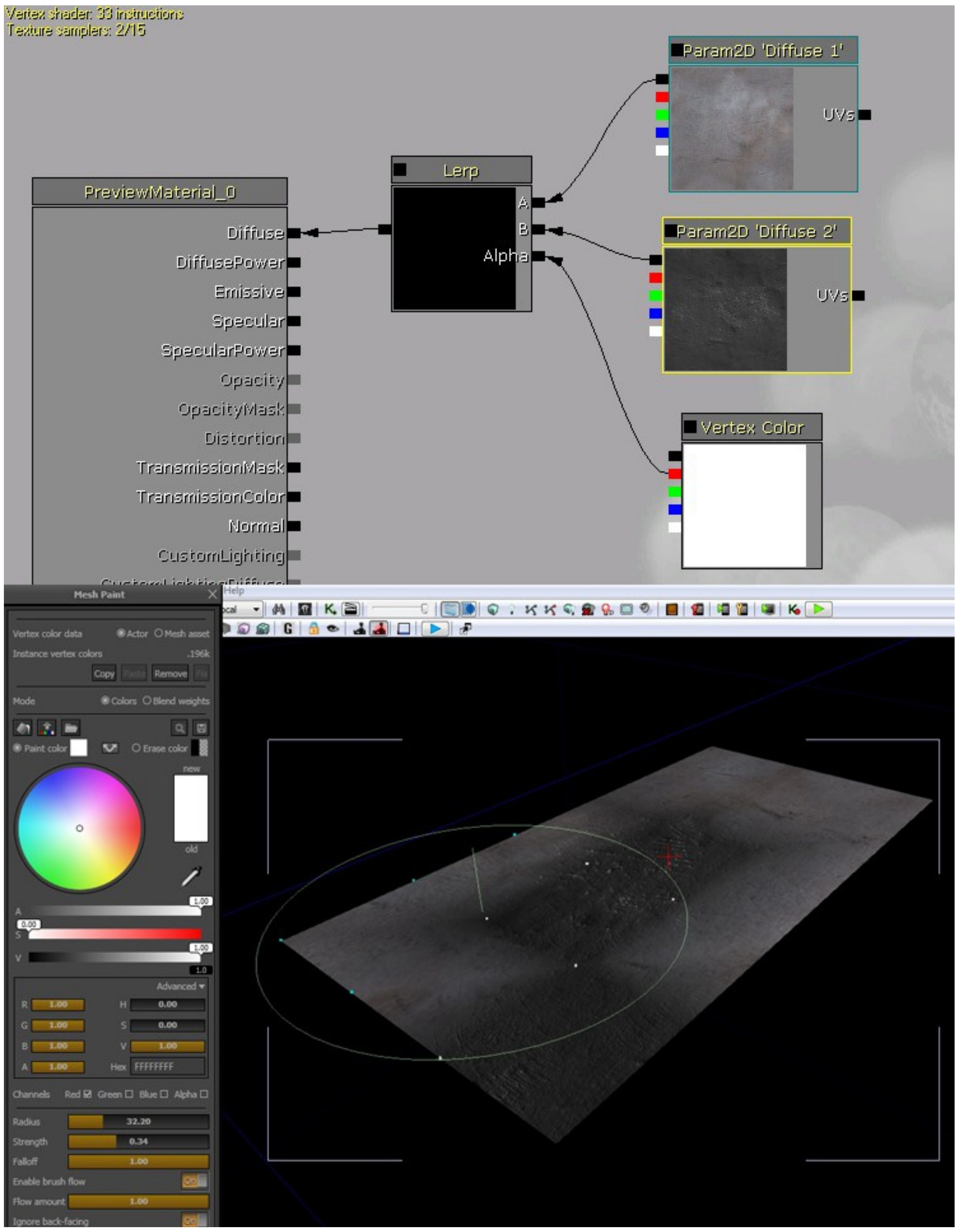


Image 17. An example of a simple vertex color blend made inside UDK (darktype 2012).

A drawback of using vertex colors is that in order to get enough fine control, this method requires quite a lot of vertices. With a low vertex count, the transitions might look quite rough as there is very little data to interpolate between. To overcome this problem, there are different ways of filtering the vertex color data with, for example, texture samples or procedural noise patterns mentioned earlier that break up the transitions between the vertices. In this case, instead of countering the effects of a lack of pixels, it is needed to fight the lack of detail brought by a low number of vertices. (Eat 3D 2012.)

5 Blend map workflow improvement

Blend maps have been used at Bugbear for years, but when I was working there, the tools for the blend mapping workflow had not really been given much attention, so I aimed to look into the issue by myself.

5.1 Blend map creation workflow at Bugbear

Large scale environments are a common task at Bugbear Entertainment when creating race tracks. Sometimes the tracks are quite vast and at the same time they still need a great amount of detail and visual fidelity to look convincing and pleasing to the player up-close as well as from a longer distance. A blend map workflow comes in handy in situations like these.

Image 18. The ground up-close as well as the terrain in the distance need to have enough detail to be pleasing to the viewer (Next Car Game: Wreckfest 2015c).

An in-house implementation of blend maps has been developed to make larger environment texturing tasks easier to accomplish. At the time of writing, Photoshop was the main tool for painting blend maps at Bugbear. Arguably, it is a less than ideal way of creating blend maps, as there is no way of telling where different materials are placed other than by remembering which color corresponds to which material. Even if the material corresponding to a specific color in the blend map is known, it is very hard to visualize and results in a frustrating workflow of constant trial and error. This is why I started experimenting with real-time viewport methods in Blender to make this process easier to visualize by offering instant visual feedback of the end result while creating the map.

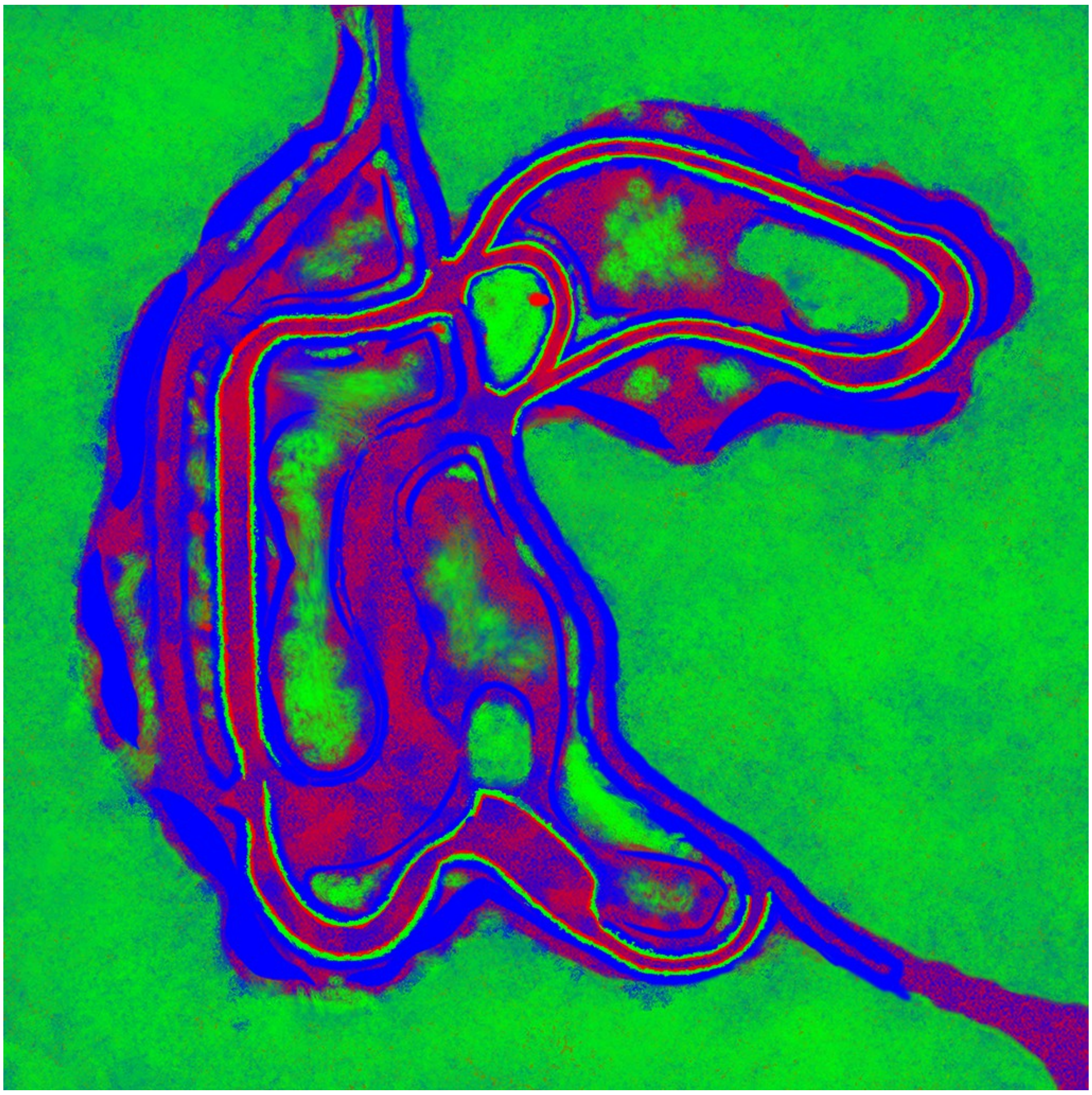


Image 19. A blend map by itself is quite hard to visualize (Bugbear Entertainment 2013a).

5.2 Blend shader development

As Blender started out as a fairly simple 3D graphics software, a lot of features available in other software have been missing from Blender. This has led to a lot of tinkering and innovative thinking in the early days of its use. Even though Blender has taken huge leaps in development since I started using it, this background has proved useful on numerous occasions as it has encouraged me to always try out new things by myself, even when one might easily feel it is too time consuming or challenging to do.

I started out by experimenting in Blender utilizing the node-based shader editor available by default inside Blender.

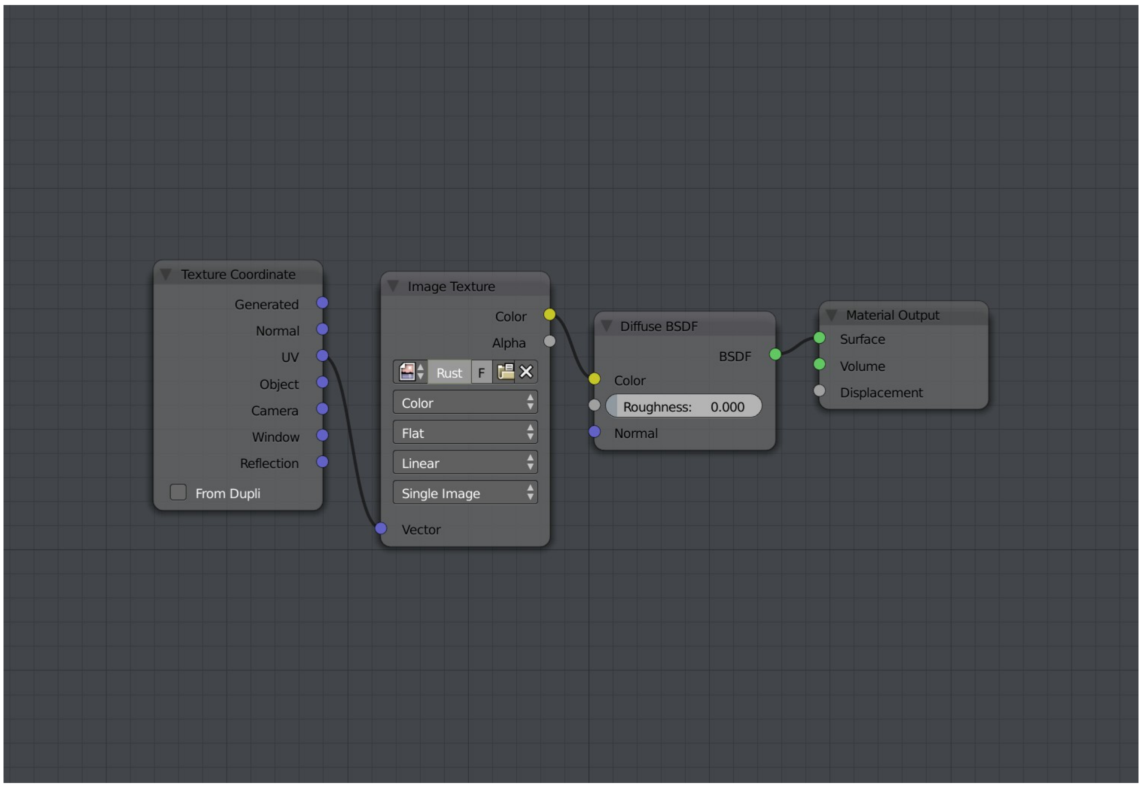


Image 20. An example of a very simple shader made within Blender's node editor. The shader only has a single UV mapped image texture with a diffuse component.

I was already familiar with Blender's capabilities of showing real-time feedback of the assigned textures in different materials, but I was uncertain, if my idea could be achieved the way I wanted. This knowledge of Blender's viewport capabilities did, however, inspire me to start trying out different ways of masking out textures via the node editor. Starting out, I aimed to understand the logic behind the way Bugbear's game engine for Next Car Game: Wreckfest works with blend maps and I tried to mimic this behavior inside Blender. I received a lot of help from my colleagues at Bugbear and the other artists at the company were encouraging and curious about my experiments. With various phases of testing and refining, I was finally able to come up with a shader that was similar enough to the Bugbear shader to be usable.

The way Bugbear's engine works with blend maps allows the different color channels to assign different materials according to the specific color channel within an image texture. However, the shader does not support a blend map alpha channel, so only a three-texture blend is possible with it. As the Bugbear shader works with materials, the blue color channel could, for example, specify a sand surface that has an accompanying diffuse, specular, glossy and normal texture defined in the shader. Even though Bugbear's engine works with materials that have reflections and normal maps blending into each other, I came to the conclusion that just visualizing the diffuse texture blending would be sufficient in order to differentiate surfaces from each other.

The shader I developed also does not treat the blend maps exactly the same way as the one used at Bugbear since I had to discover a lot of issues by myself. I made a simpler version that met the needs I had at the time. I had not implemented normalization of the color channels, which led to some discrepancies, but the main point of the tool was to give some form of visual feedback and as such, I felt that the shader I had developed was a better alternative than half-blindly painting inside Photoshop. The instant visual feedback is not only incredibly useful in regards to the shorter iteration times, but also being able to paint in a 3D-view directly on the mesh makes it so much easier to visualize. As being able to visualize an environment is the main thing an environment artist is supposed to do, the workflow for that visualization is very important.

The setup in Blender is actually very simple, but initially I was struggling quite a lot, since I did not look up online, if anyone else had done something similar in a node editor. I later received some pointers from a colleague who had seen a similar setup done by Ryan James Smith utilizing vertex colors in UDK (Smith). This helped me in my efforts to understand how these kinds of shaders work in practice. Since then, I have put a lot of effort into understanding how different texture blends can be achieved and identifying the available methods. I have also developed different versions of my shader, but the one I am demonstrating here aims to show the basic principle of a blend shader inside Blender and how it can help in environment texturing.

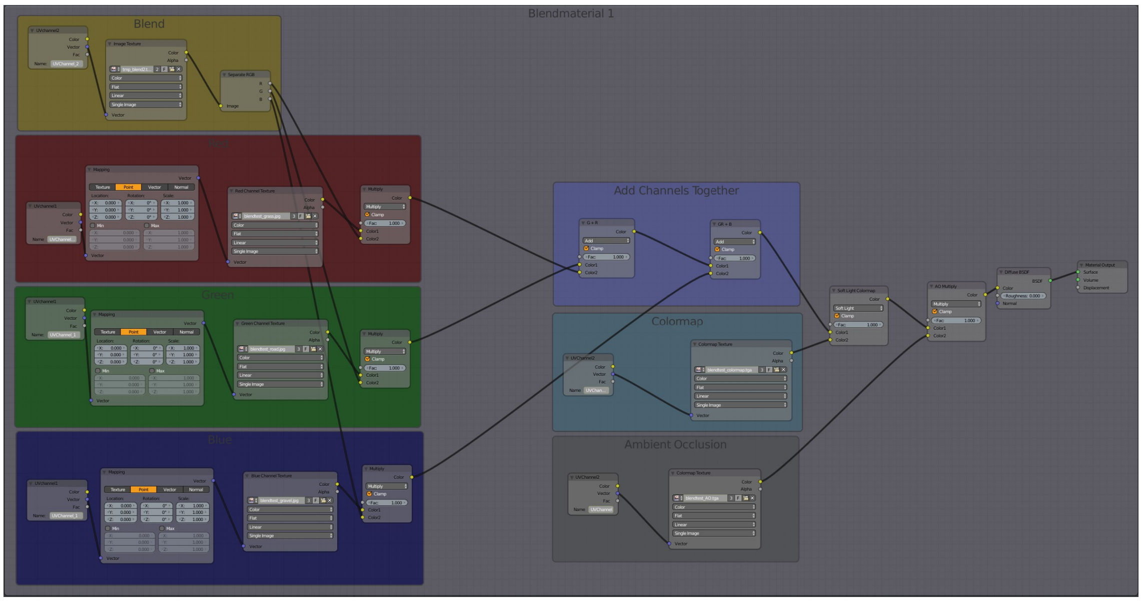


Image 21. An overview of the blend shader inside Blender's node editor.

5.3 Blend shader dissection

In this chapter, I give a detailed account of how to set up the mesh in order to use the blend shader, how the blend shader works, how it differs from the Bugbear shader and give some pointers on how to make the best out of it.

The mesh using this blend mapping method benefits from having at least a couple of UV channels. To work efficiently, even a third channel might be needed in some cases. One of the channels is used for the tileable textures. This channel can be unwrapped quite freely as preferred for the best visual result. Following terrain geometry while unwrapping is a good way to avoid stretching and discontinuity though. A winding road could be straightened and tiled, for example. Straightening the road in the UV map will make a sandy road with tire tracks, for example, follow the curves of the road instead of just tiling in the same world coordinate direction, as in the second example in chapter 4.1.

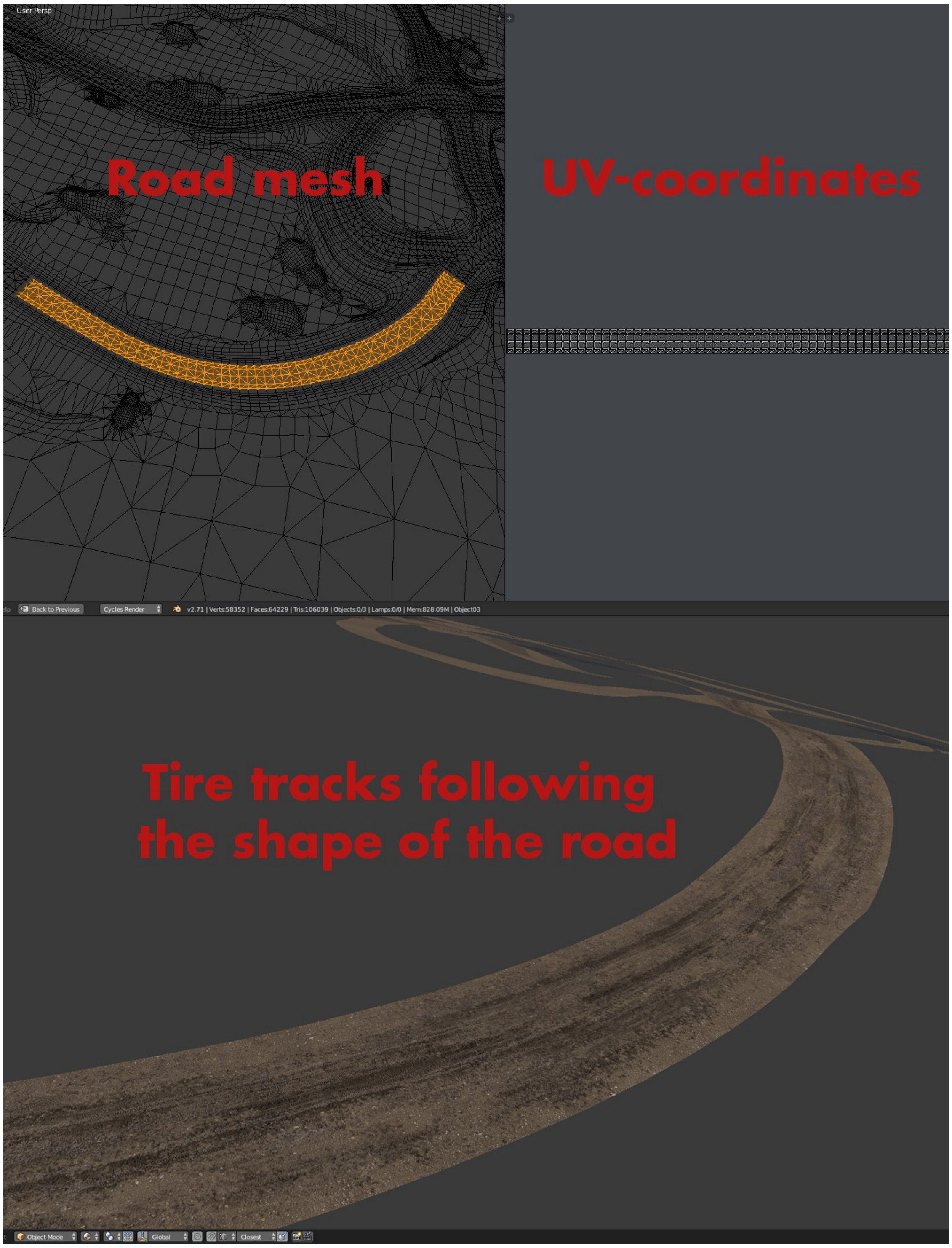

User Persp
Road mesh
UV-coordinates
Back to Previous
Cycles Render
v2.71 | Verts:58352 | Faces:64229 | Tris:106039 | Objects:0/3 | Lamps:0/0 | Mem:828.09M | Object03
Tire tracks following the shape of the road
Object Mode
Global
Closest

Image 22. A demonstration of the unwrapping described on the last page. The curved road has been unwrapped so that it is perfectly straight in order for the tire tracks and other details in the tileable textures to follow the shape of the road. (Bugbear Entertainment 2014b.)

The other UV channel is used for the blend map texture. In the case of creating a terrain, it should be unwrapped from the top and laid out inside the 1x1 square of the UV editor. This is important since the blend map texture generally should not be tiled or overlapped, so that unique detail can be painted over the whole terrain. In the case of having roads in the terrain, they can either be mapped onto the same blend map as the terrain or they can be straightened out and mapped onto a separate blend texture. As the roads usually require a lot of detail to look convincing and separate materials are needed anyway, it is usually preferable to separate them from the rest of the terrain.

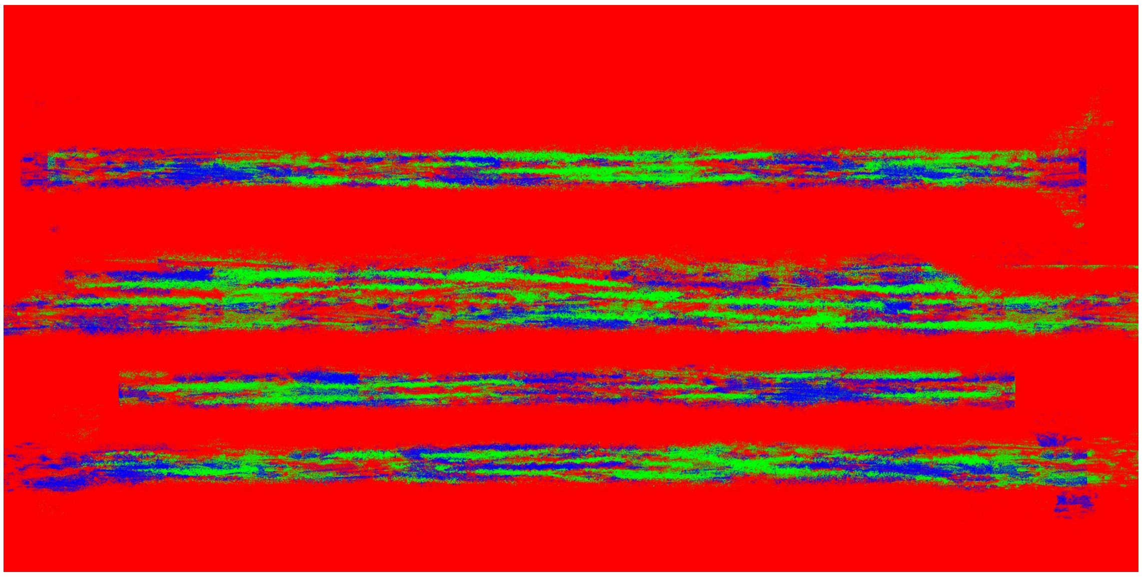


Image 23. A separate blend map for the roads of a terrain. The roads have been straightened out for efficiency. (Bugbear Entertainment 2014c.)

By straightening out the roads for the blend map, a lot of unused space can be saved and a greater texel density can be achieved with a smaller texture. Although a straightened texture is generally easier to paint in Photoshop, the problem with this method for the roads is that dividing the long strips of road into individual smaller pieces stacked beside each other makes it a lot harder to paint the blend map without introducing texture seams. Each island would have to end and continue in the next island with exactly the same color in the same location in order to produce a perfect blend. This is extremely hard and time-consuming to do by hand in Photoshop, if a lot of interesting surface variation is wanted in the blend.

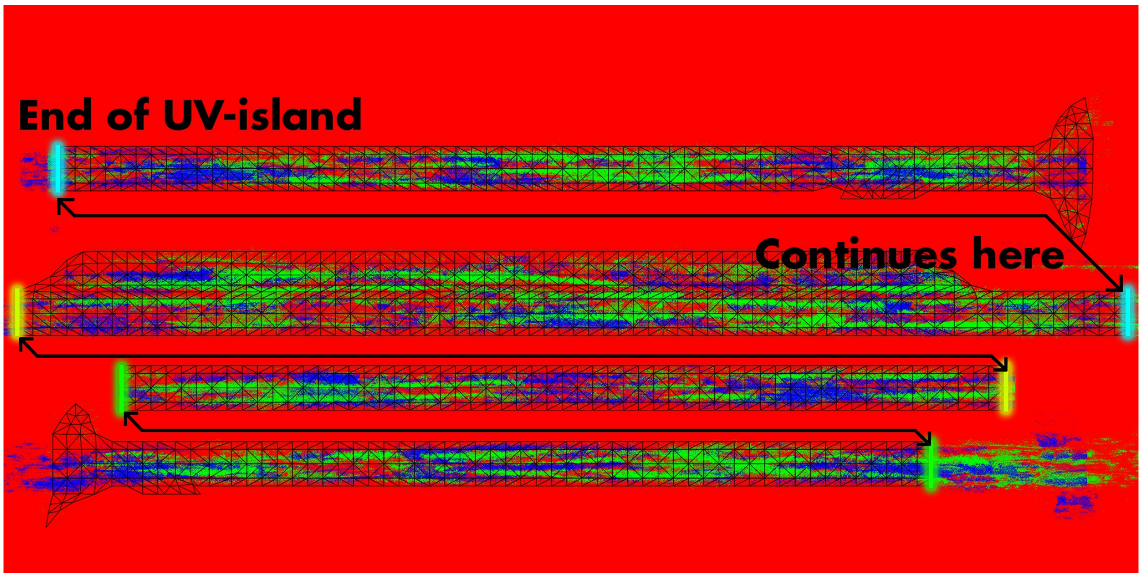


Image 24. The road blend map with the UV map outlined on top and the seams marked with different colors corresponding to how they are connected to each other. Note how each division of the road might introduce a texture seam unless continued perfectly. (Bugbear Entertainment 2014c.)

Painting the same straightened and divided strip of road with the help of the blend shader I developed would make this problem disappear altogether, as the actual road mesh could be painted in the 3D-view and the strokes would automatically be continued from one UV island to another. A single fast stroke could be painted instead of tediously matching the stroke at the right edge of one UV island to the left edge of another in Photoshop.

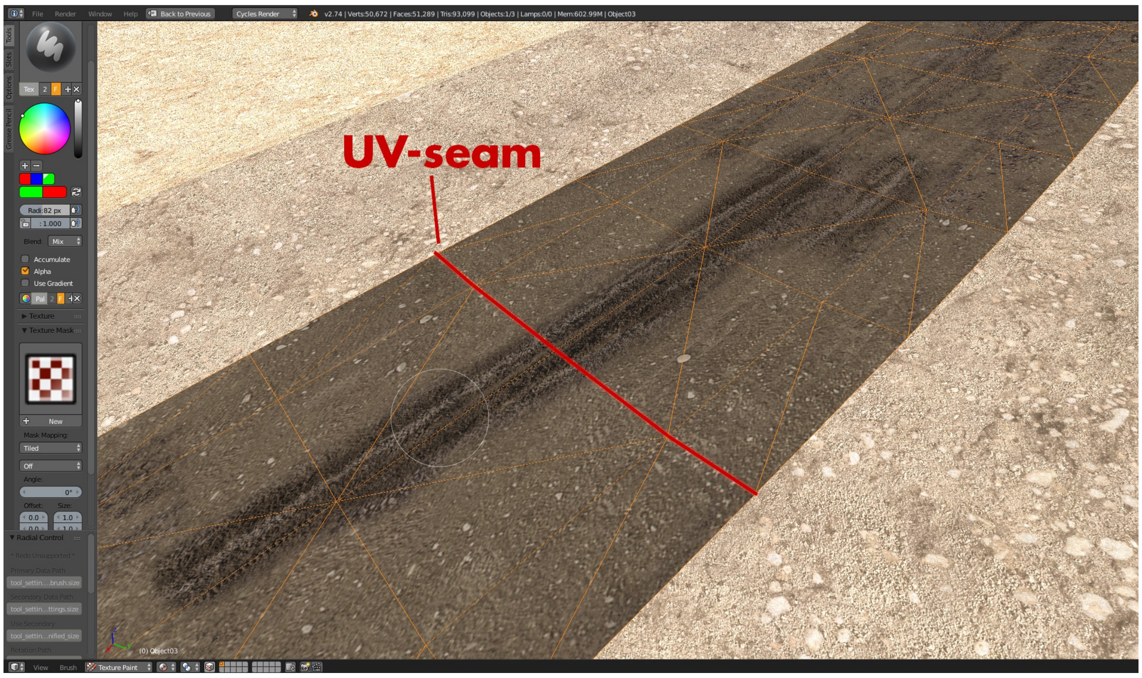


Image 25. This continuous tire track flowing from one UV island to another has effortlessly been painted by using the blend shader in the 3D-view in Blender (Bugbear Entertainment 2014d).

The shader uses the blend map as one of the inputs and separates it into the individual color channels in order to use them as masks for different tileable textures. Every color channel in the blend map corresponds to a tileable texture. The tileable input textures are given individual mapping data for location, scale and rotation to give the artist using the tool more control of the texture mapping without modifying the UV mapping. This was a feature we used at Bugbear which I implemented into my shader. It is quite handy for making, for example, quick changes in the texture density.

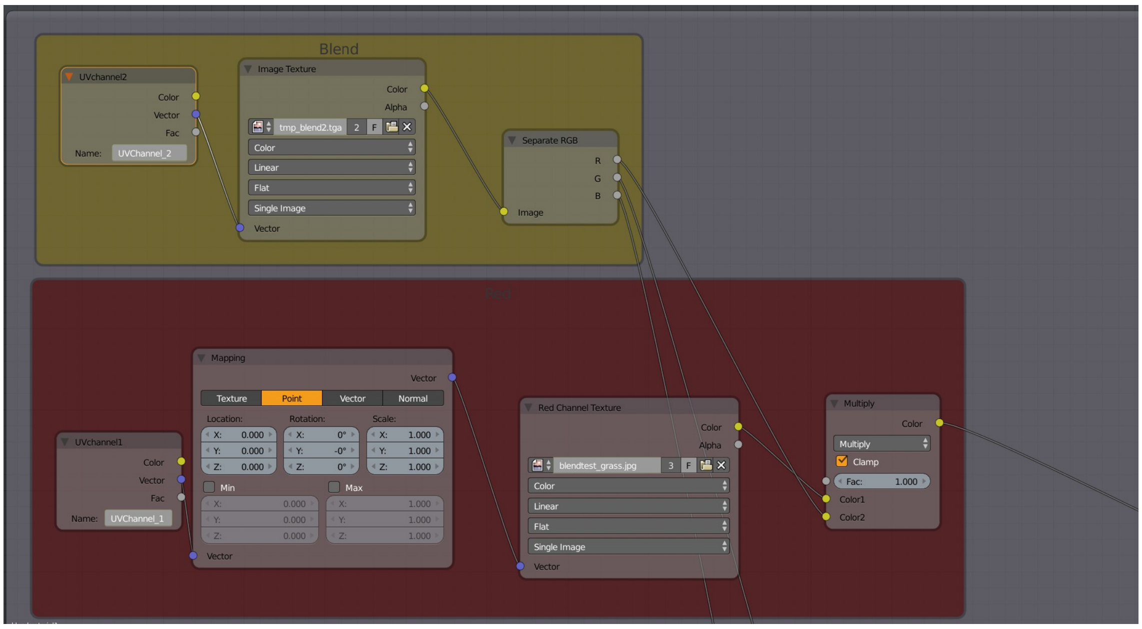


Image 26. As seen in this image, the blend texture is separated into the individual channels which are then multiplied by the texture that is desired to correspond to that channel. The mapping node is for tweaking the tileable texture.

The way the blend shader is set up, a tileable texture is multiplied by the color channel of the blend map that the texture is to be associated with. As an example, the green channel in the blend map could be painted to control where the grass on the surface of the mesh is drawn. In that case, the green channel of the blend map would be multiplied with a user-chosen tileable texture of grass. As previously stated, each color channel is like an alpha map with a scale from black to white. The white in the blend map's green channel means the texture is multiplied by one and therefore draws the texture with a 100% strength. A black value results in the texture being multiplied by zero and means the texture will not show up at all. All the gray values in between work as linear blends, blending in textures from whatever other color channel also has information in that specific region. This process is done for every channel of the blend map and each corresponding texture. These are then finally combined together with a simple add operation. All the information lacking in a specific channel exists in another channel as long as the image does not contain any black color. Adding the channels back together will result in a nice blended terrain of the different textures where they were painted.

An important detail for this shader is that it does not correct situations where the combined value of the color channels add up to a larger value than the maximum value of a single color channel. As a result, a blend map painted in such a manner will lead to incorrect results. In this regard, the shader I developed differs considerably from the one at Bugbear, as no normalizing takes place and no threshold is set to override low values and everything is blended linearly instead. Usually this does not create serious problems as long as the blend map is painted with pure red, green or blue values.

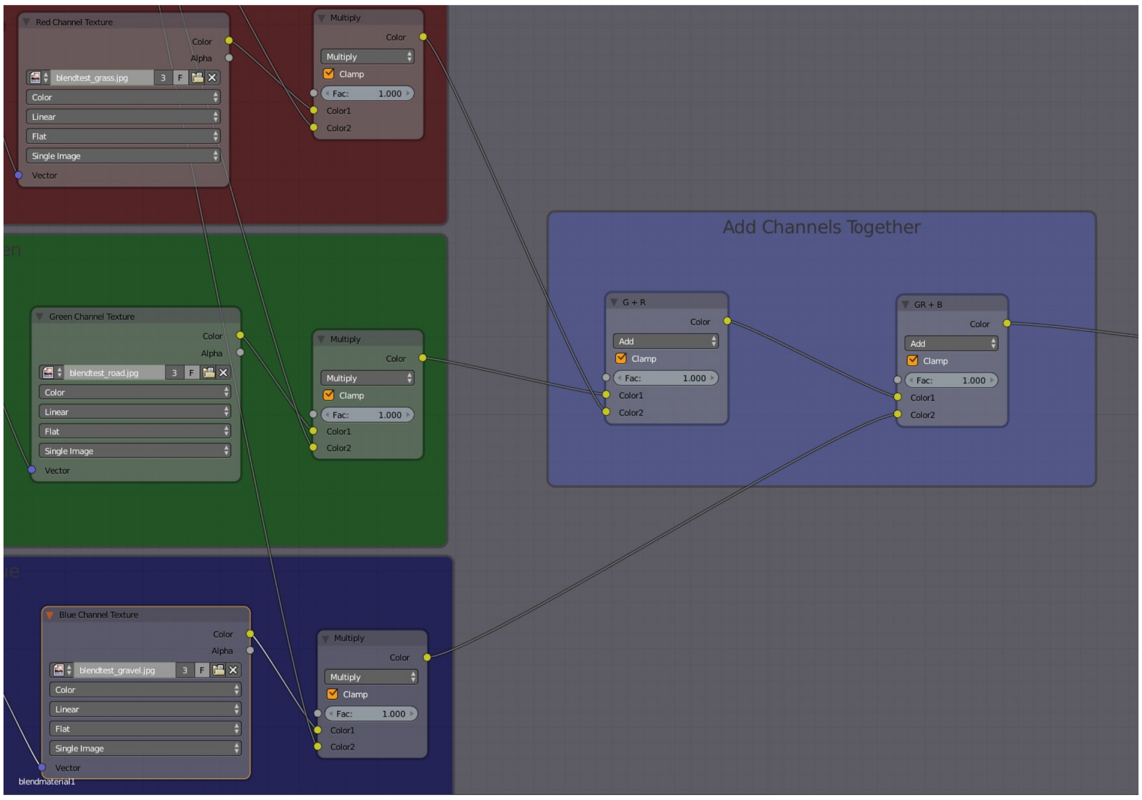


Image 27. After the tileable textures are multiplied by the respective blend map color channels, the outputs are added together resulting in the texture blend.

There are some additional things that can be done to add even more visual fidelity to the terrain. This single blend material only allows for three different surfaces and sometimes even more variation on a single terrain is desired. A situation where a road crosses a large terrain might be difficult to accomplish, if, for example, two slots are already put to use for grass and sand on the terrain and some variation in the road is desired as well. Using a single blend material with a blend texture containing three color channels would not suffice to do this and the road would only get that one leftover texture slot. In order to achieve some variation in the road with this method, the terrain would have to be split into multiple blend materials assigned to different polygon groups. The road could be separated into a different blend material. Separating into different blend materials does not mean a separate blend map has to be painted though. The same blend can be used but the textures assigned in the material can be different from what is set up in the other material.

The initial reaction might be that this would offer a situation where one could have three materials for the terrain as well as the road, but this would unfortunately lead to sharp seams where the materials change from one to another. To avoid this problem, one color channel is sacrificed for sharing a specific surface between the two materials. In a hypothetical situation where the terrain edge facing the road is sand and corresponds to the blue channel in the blend map, the same texture could be assigned to the road's blue channel as well. By painting blue along the polygon seam, the transition between the materials would be smooth. Although one color channel is sacrificed, this technique adds one additional surface that could be used for variation in the road.

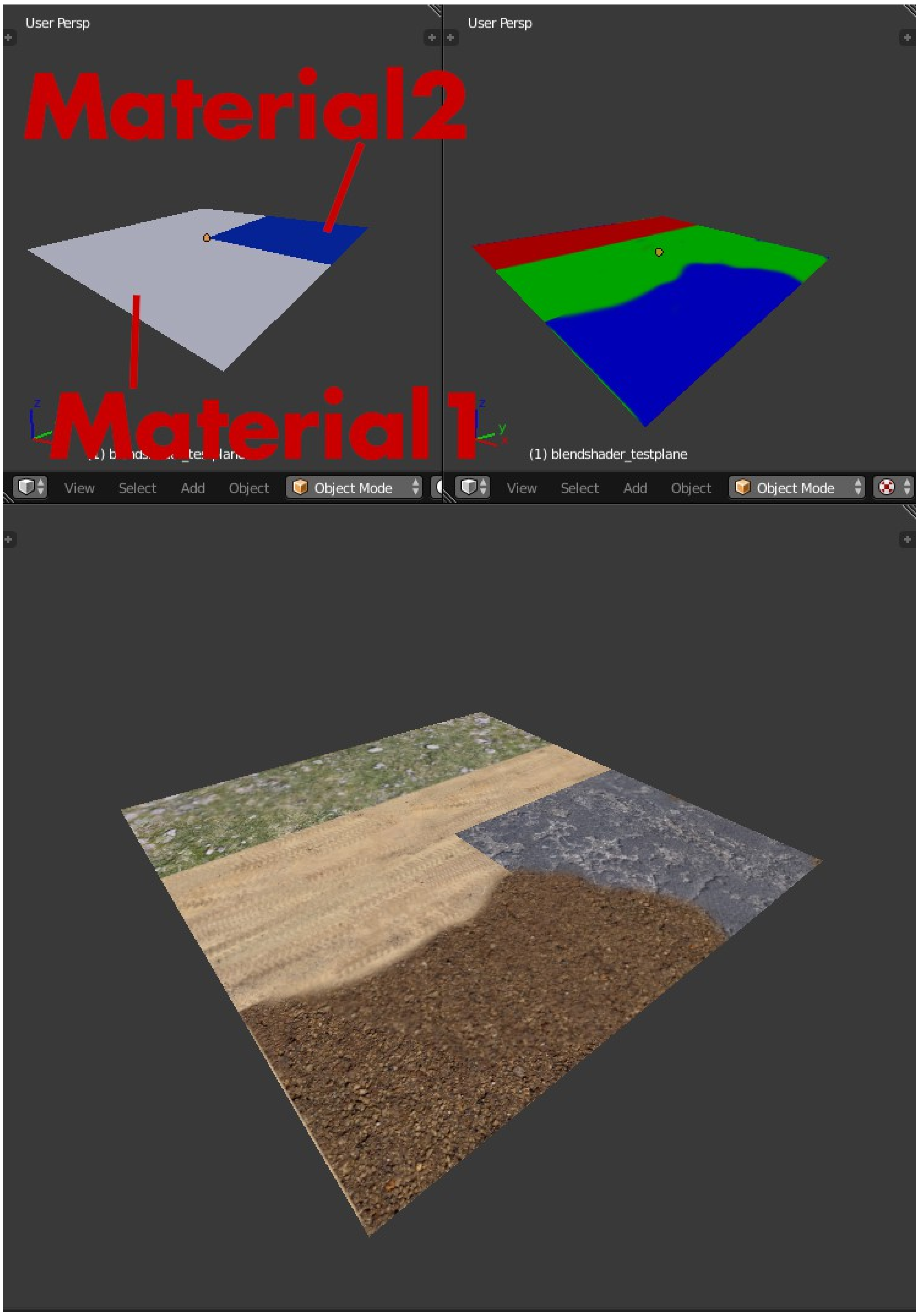


Image 28. An example of how sharing a texture, in this case assigned to the blue channel, between two materials results in a smooth blend. The green channel textures are completely different and thus result in a sharp seam where the material changes.

In case even more surface variation is needed, there are still additional techniques that can be used to get the most out of the situation without adding more blend map channels or materials. A separate uniquely mapped image texture with a color tint could be overlaid on top of the terrain to add subtle variation. This is quite useful on larger areas of grass, for example, where subtle color variations help break up the repetitiveness of a single grass texture. Ambient occlusion data could also be baked and multiplied on top of the terrain to give it more depth and realism.

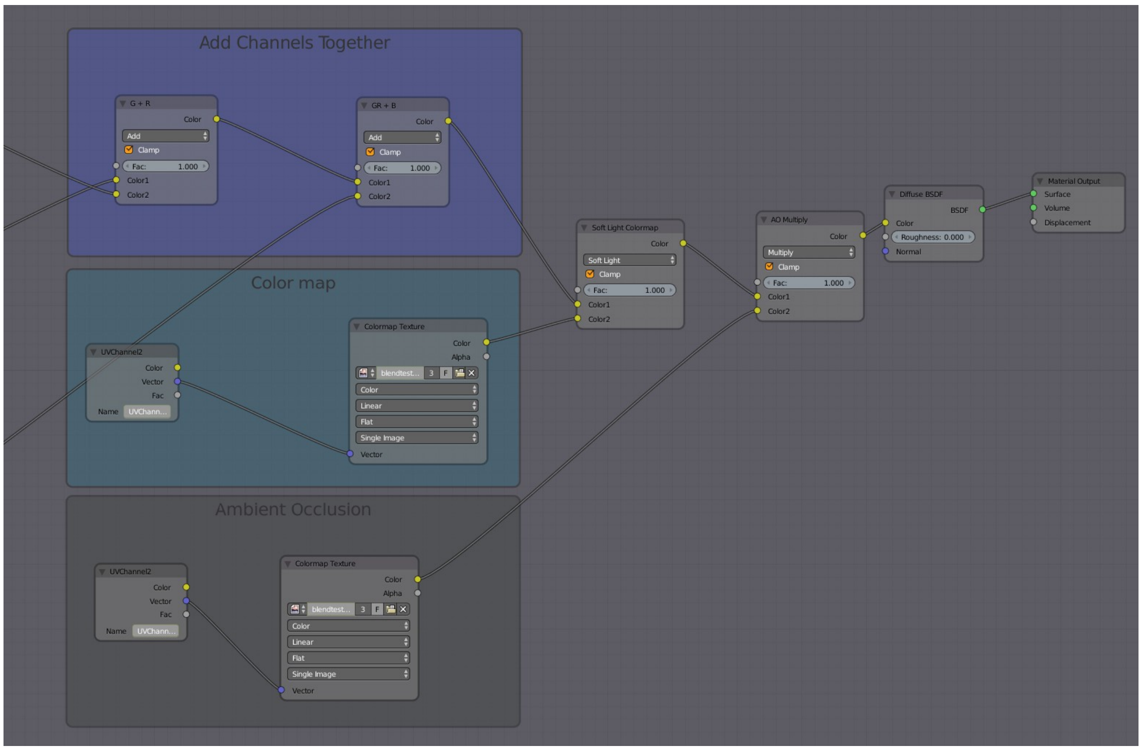


Image 29. An example of how a color map and an ambient occlusion map can be added to the final blend result.

6 Other environment texturing techniques

Despite the usability of blend mapping in environment texturing tasks, there are also a lot of alternative and supplementary texturing methods. This chapter highlights some additional texturing methods that are not necessarily a replacement for blend mapping altogether, but some situations might require different approaches and awareness of alternatives gives an artist a greater tool-set and flexibility to tackle these tasks.

6.1 Modularity

In addition to the previous texturing techniques, one could also create big detailed environments by utilizing modular workflows. These techniques require a lot of thought and planning, but can give incredibly rich results if done imaginatively. By using a single texture atlas tailored for adaptable usage and creating geometry that utilize this texture in the most varied way possible can give practically endless amounts of variation. (Cowley 2012; Burgess & Purkeypile 2013.)

In the example of Tor Frick's one-texture environment, he utilizes the texture efficiently and models the environment according to his texture's limits, constantly finding different ways of making it even more efficient. He also manages his UV space imaginatively and takes advantage of the different shader tricks and material instancing, which allows him to create many different variations with the same texture. All of this is possible thanks to the incredible flexibility of the Unreal Engine shader editor and material system. His method is highly efficient in terms of memory usage, but he admits to it being quite extreme regarding optimization and mainly a good exercise that probably would not be suitable for production. This method taken to a less extreme is, however, no doubt a method that can be utilized even in production and can be a great resource for anyone who wishes to optimize their texture usage. (Cowley 2012.)

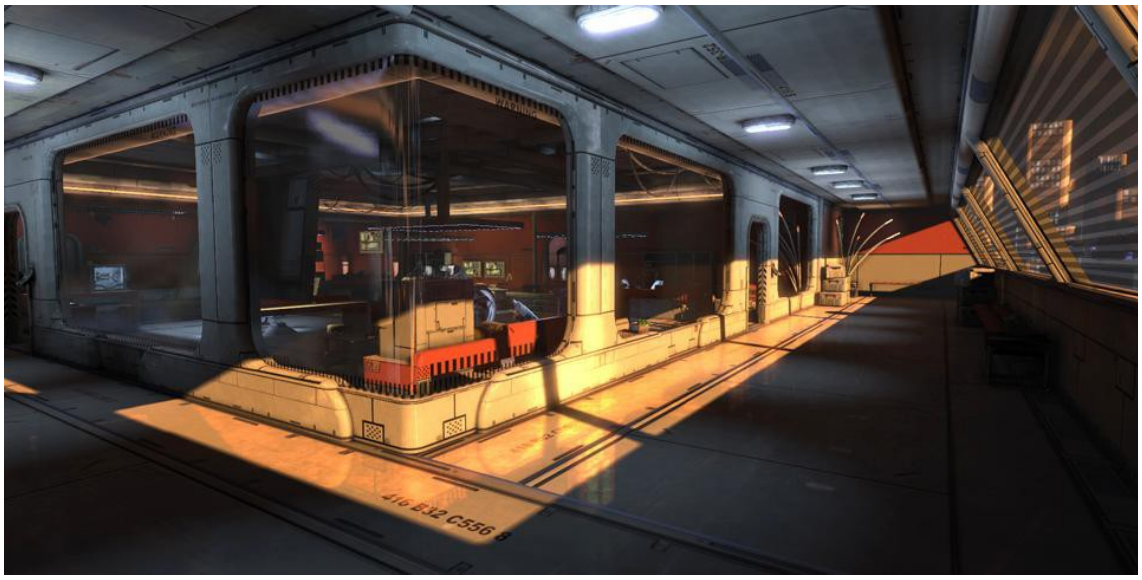


Image 30. Tor Frick's one-texture environment (Frick 2011).

A slightly different approach involves creating many different modular pieces that fit together but use different areas of the texture or have differing geometry to create variation. Placing a brick wall in one area of a texture atlas and a concrete pillar in another area of the texture would, with a bit of imagination, open up the possibility to create literally dozens of different assets just from these two surfaces in the texture. With this technique, a big set of pieces is created that can be used over and over again while still giving a lot of detail and complexity and saving a lot of memory. (Eat 3D 2013.)

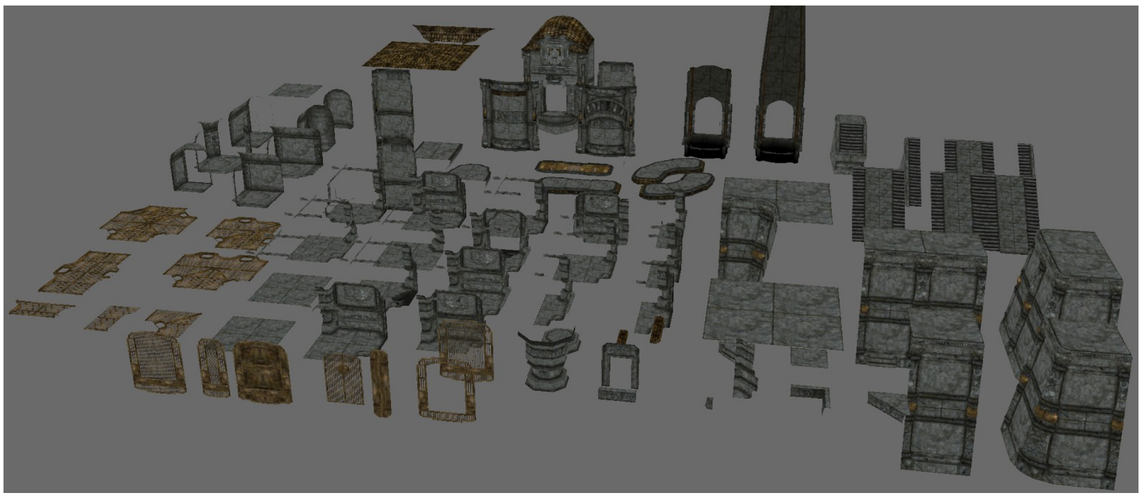


Image 31. A modular set created for Skyrim (Burgess & Purkeypile 2013a).

The team at Bethesda Game Studios has mastered the different techniques of modular environment creation. They plan their modular kits meticulously and have the kit artist work in parallel with the level designer testing out the kit as it is being developed. The pieces the artist creates are being used in numerous ways that allow the level designers to create countless unique environments with great speed. These techniques have been used to a great extent in most of their recent games like Oblivion, Fallout 3 and Skyrim. (Burgess & Purkeypile 2013.)

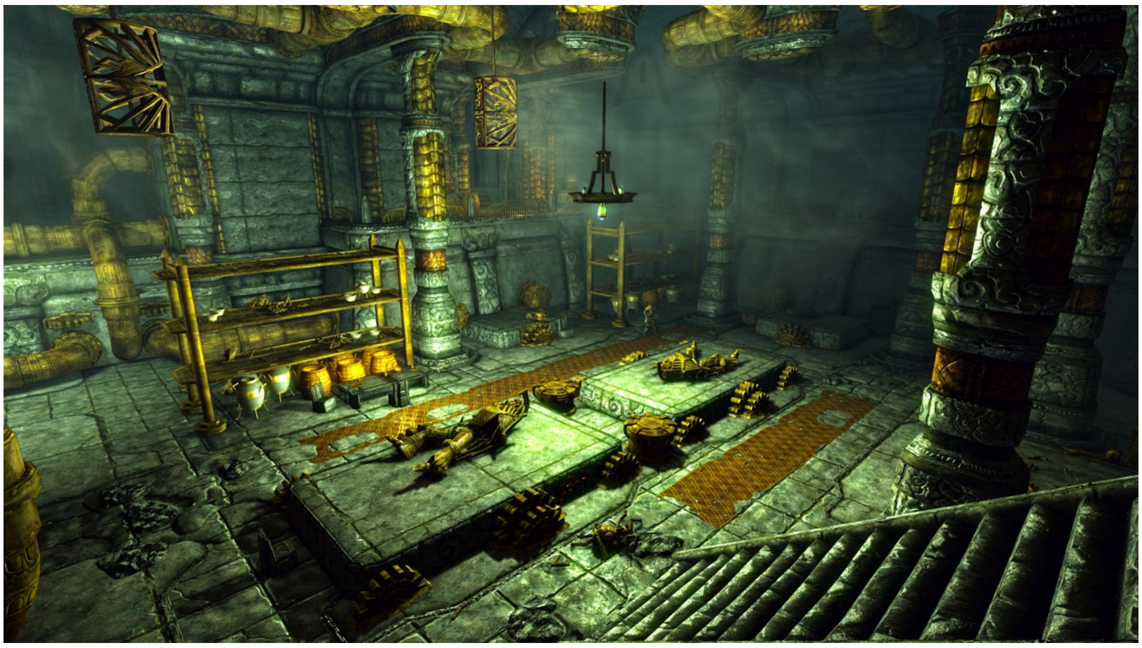


Image 32. A dwarven dungeon in Skyrim created by using a modular set (Burgess & Purkeypile 2013b).

During my work on an unreleased mobile project at Remedy Entertainment, I have worked together with a colleague on making big diverse environments utilizing many of the same techniques described in this chapter. Tor Frick's one-texture environment and his Modular Masterclass have been great inspirations in creating all of the modular assets that we did during the development of the project (Cowley 2012; Eat 3D 2013).

My colleague initially created a fairly small set of assets such as roofs, walls, doors and windows that could be combined in different ways to form various kinds of buildings. I eventually took over the tileset, as we called it, and expanded and improved it further. I created a lot of new tiles that, at the end, allowed us to create quite large environments with relative ease and speed.

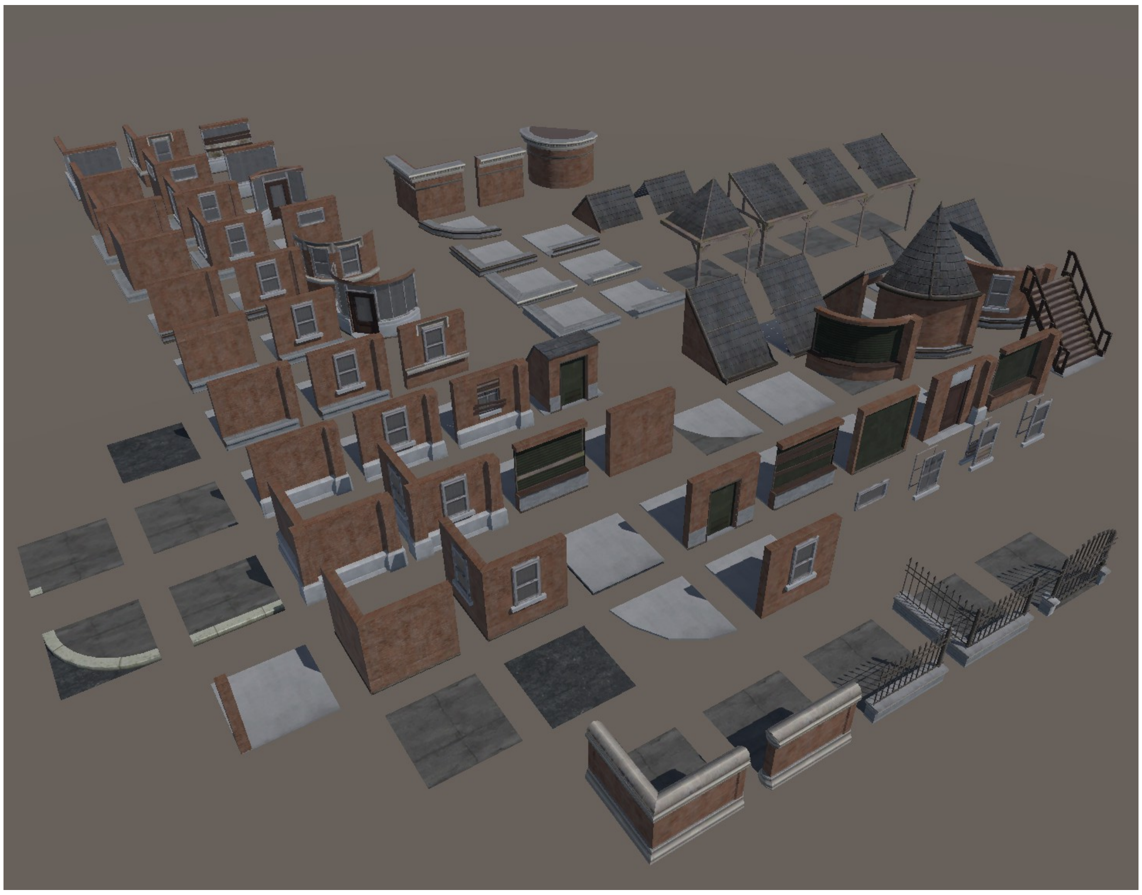


Image 33. A modular building set for an unreleased Remedy project (Remedy Entertainment 2015b).

As with many things in the game industry, this part of the game that we had worked on for several months got cut midway through the project. Ultimately the whole game project got canceled, but even though a large part of the system we developed did end up unused, at least for now, I learned a lot about different techniques related to modular content creation. Making modular textures has allowed me to create high-fidelity assets without making huge textures that hog all of the available memory.

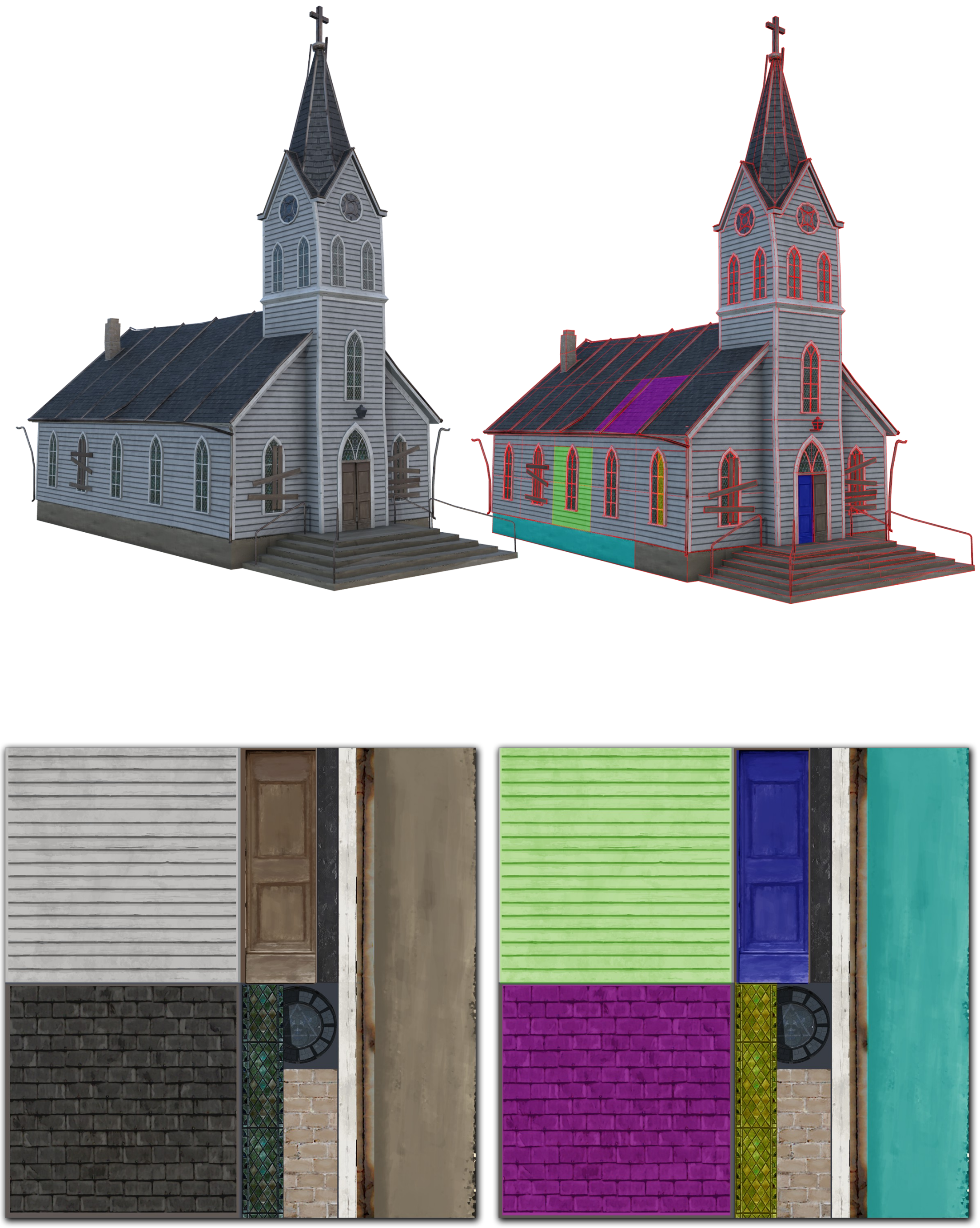


Image 34. A church I made for an unreleased Remedy project. Instead of solely creating textures that tile from the image edges, geometry has been used to create tiles that can be repeated to build larger surfaces. (Remedy Entertainment 2015c.)

The challenge of modular environment creation is to hide the repetitiveness so the player does not encounter visual fatigue. One of these methods is addressed in the next chapter.

6.2 Decals

In case some smaller details like leaks or imperfections are needed for a scene to give it more life or technical restrictions prohibit any kind of texture blending, the situation might be solved with the help of decals. Generally a decal refers to a simple piece of geometry that has some nature of transparency set in its texture, which allows for some of the underlying surface to show through. Adding decals to a scene is rarely the only method used for texturing an environment, but it can be a considerable help in breaking up repetitiveness or adding variation and detail. The beauty of decals is that they can be placed anywhere while freely being scaled and rotated in whichever way the artist wishes. Decals can also easily be removed or added depending on the situation without the need for redoing any textures. This makes them ideal in conjunction with the use of modular environment workflows. (Cryengine Manual 2014c.)

The downside of decals is that they have a lot of alpha texture space that can have a significant effect on performance, if too many of them are added. There are some options for balancing the quality level and the effects on performance. Decals can be configured to blend smoothly into the background surface via alpha blending. The alpha blending uses the whole range of values stored in the alpha channel which allows for smooth transitions and semitransparent surfaces. Alpha blending is quite demanding on the hardware and issues with sorting can occur due to limitations of the z-buffer. A simpler and less demanding way to render decals is by using the alpha channel as a basis for a threshold that determines, if a pixel is rendered or not. This method, called alpha testing, only renders a pixel either fully opaque or fully transparent according to a user defined threshold and no smooth transitions or semitransparent surfaces can be achieved. Despite the somewhat rugged look of this method, there are no issues with sorting and it is therefore useful in many situations. (McAnlis 2014; Second Life Wiki 2013; Marmoset LLC; Polycount Wiki 2015b.)

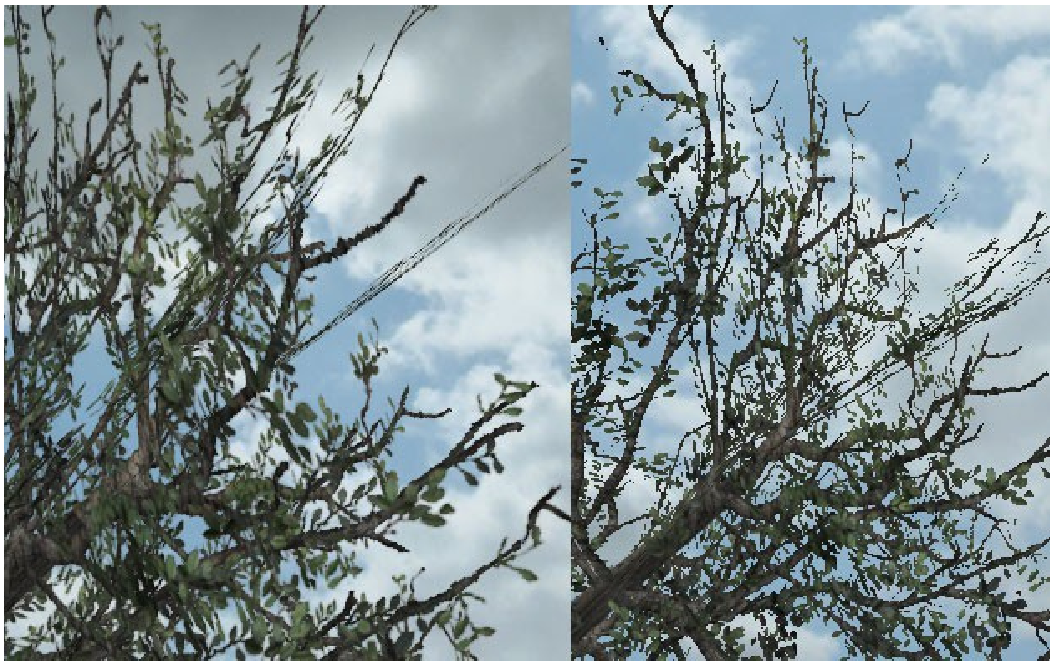


Image 35. Comparison between alpha blend and alpha test (Justafin 2013).

Both of the demonstrated methods of transparency have some benefits and drawbacks. There are some additional methods for tackling the problem of transparent surfaces that try to solve some of the previously stated issues. Methods include, for example, alpha to coverage and hybrid solutions that combine alpha blending and testing. (Marmoset LLC; Polycount Wiki 2015b.)

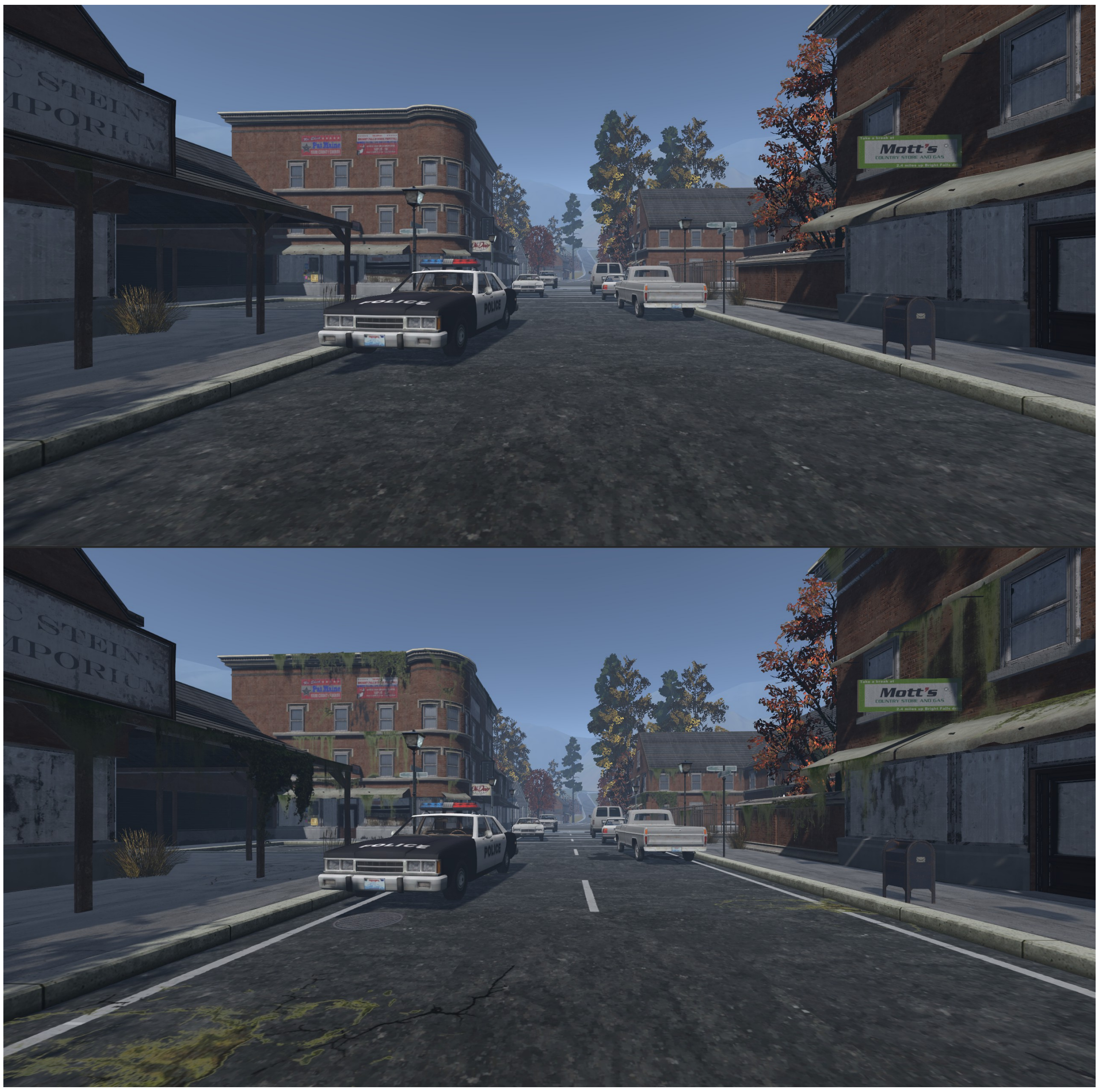


Image 36. Comparison between a modular scene without and with decals (Remedy Entertainment 2015d).

Generally decals are placed very close to the surface they are meant to add detail to as to avoid breaking the illusion of one continuous surface. The viewer should not be able to tell that the decals are levitating above the surface. This can, however, lead to another issue with the way the game engine determines which surfaces are rendered on top of another in the camera frustum. The method for determining this in computer graphics is limited in how accurate it is. Inaccuracies in these calculations might lead to z-fighting. This error gets worse the further the in-game camera is from the problem area in question, as the accuracy is dependent on the distance from the camera, the near and far clipping planes as well as the bit-depth of the z-buffer. (Baker.)

6.3 Virtual Texturing

An interesting method of solving large texturing tasks is called megatexturing or more recently, virtual texturing. Megatexturing refers to a method developed by id Software's former technical director John Carmack where a single enormous texture is used for texturing the terrain in a game level. Initially only static terrain used this technique, but id Software has developed this technology further, which has allowed for an even larger texture also accounting for model textures and sprites. This improved megatexturing technology called virtual texturing has been used, for example, in the first person shooter Rage. (id Software 2009; Youtube 2010a; Youtube 2010b; Rendering Pipeline 2012; Wikipedia 2014.)

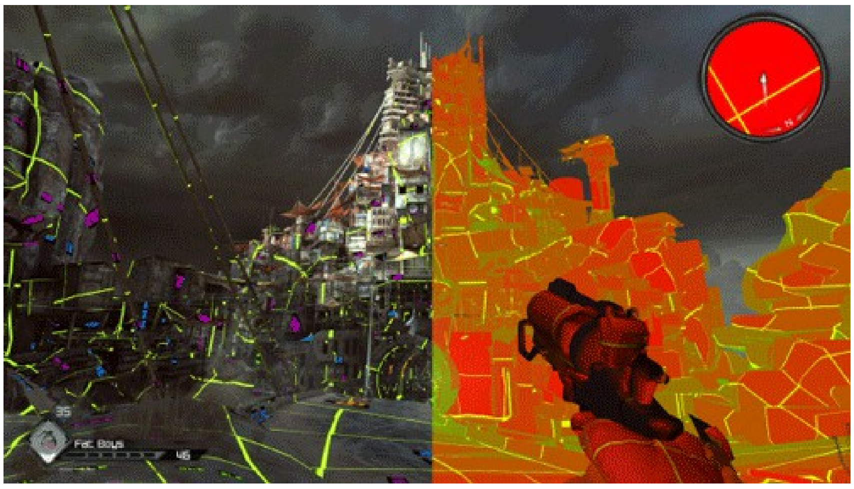


Image 37. Visualization of the virtual texture loading in Rage (Rage 2011a).

Since everything is mapped to an extremely large texture, every single object or area can have uniquely textured surfaces, the only restriction being the maximum size of the virtual texture. This limitation should not pose a problem, however, as the maximum size of the virtual texture can be as large as 128 000 by 128 000 pixels. Naturally, all of this texture can not be loaded into memory at the same time, so the engine only loads the parts needed at the time into the GPU memory. The virtual texture is divided into tiles that are loaded progressively from low resolution mip map levels to higher resolution levels. The process of loading in higher mip levels as needed is called texture streaming. The game editor used at id Software allows the artists to paint tiles or stamp decals onto the terrain or objects freely without having to worry about texture budgets or ugly tiling patterns. The methods used for creating game worlds at id

Software give an immense amount of flexibility and control to the artists as they are able to paint intricate details wherever they wish almost without limits. (Hastings 2007; id Software 2009; Youtube 2010a; Youtube 2010b; Rendering Pipeline 2012; Wikipedia 2014.)

Image 38. The rich world of Rage was created with the help of virtual texturing (Rage 2011b).

6.4 Procedural texturing

Procedural texturing is a very old method of using different kinds of mathematical algorithms to generate patterns and noise that mimic various real life phenomenon. These do not require any initial image sources, if preferred, but generates everything mathematically with different algorithms. This means that the memory requirements for this method are very low. However, using this method exclusively can lead to a lot of difficulty in achieving specific details, as everything needs to be controlled by algorithms instead of an artists eye and hand. An incredible amount of variation can be achieved procedurally, but in some cases it can require highly complex algorithms that could be easier to achieve with an image. (Animation supplement 2012.)

A more useful method of procedural texturing combines initial image sources with mathematical algorithms to create complex textures with a wide range of flexibility and modifiability. An example of a program that utilizes this method is called Substance Designer, which has a node-based texturing system that allows the user to create various textures with the help of built-in algorithms that treat the source textures in different ways. Using these algorithms helps the artist create new interesting materials without the need for a specific effect in an image. The artist is able to use an image source that can be enhanced and modified in different ways by plugging in nodes that affect the final output texture. The beauty of this system is that it is completely non-destructive. Textures can be created entirely by procedural methods or an image can be used as a base for everything. Having an image as a base allows the artist to create all the additional modifications and additions procedurally which means that anything can be changed whenever it is required without any of the definition or quality being lost from the initial texture. Any changes can be made on the fly even after making the first iteration of a final texture and changes to something down the line of production are needed. (Allegorithmic 2015.)

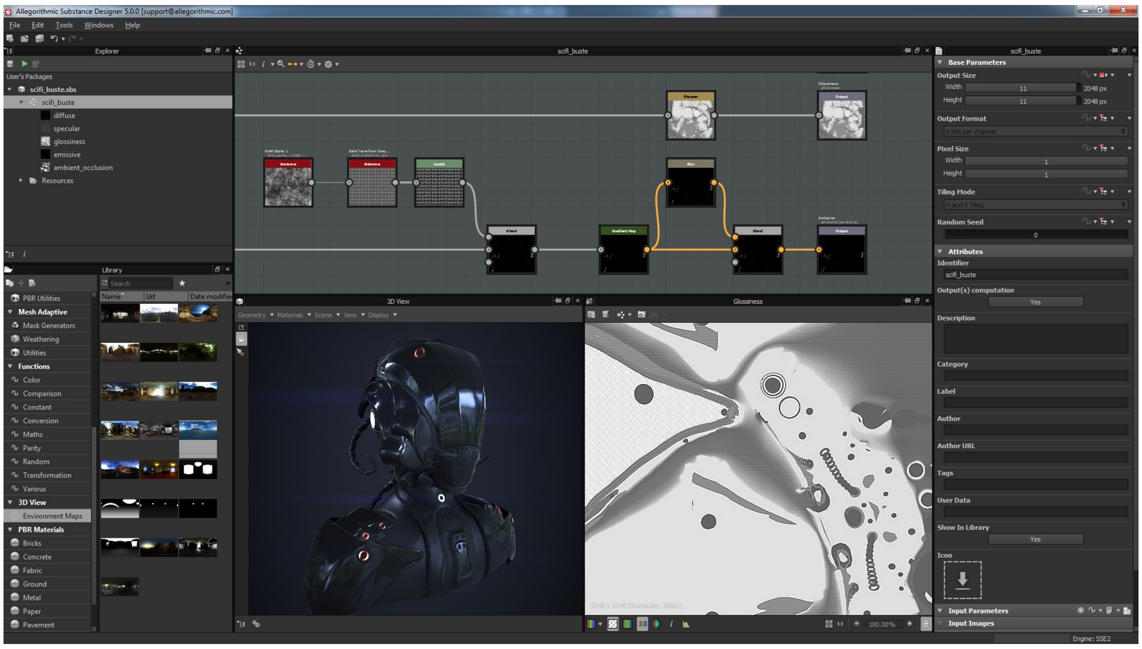
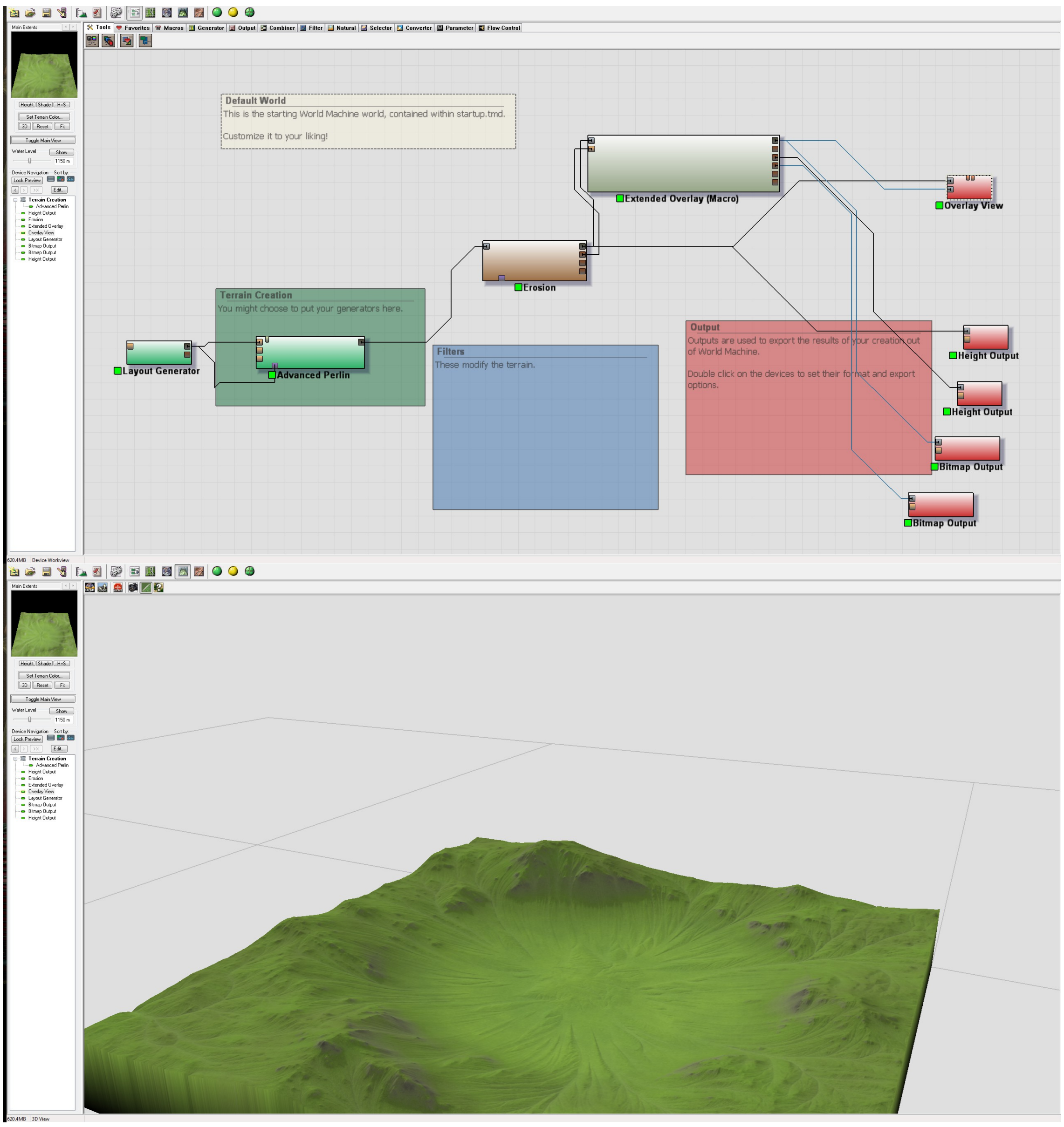


Image 39. The node-based texturing of Substance Designer allows for a very flexible workflow (Allegorithmic 2015).

A program mentioned earlier, World Machine, is also relevant in regards to procedural texturing as the different kinds of terrain formations are generated procedurally. The user can select and modify nodes that contain algorithms that produce the final terrain. There are countless ways to create terrains and the user can even draw shapes to affect the final outcome. The output from World Machine can be, for example, height data, blend maps or color information. (World Machine Software LLC 2013.)



Default World
This is the starting World Machine world, contained within startup.tmd.

Customize it to your liking!

Terrain Creation
You might choose to put your generators here.

Filters
These modify the terrain.

Output
Outputs are used to export the results of your creation out of World Machine.

Double click on the devices to set their format and export options.

Image 40. The textures are created procedurally in World Machine.

6.5 Layered materials

Workflows have improved in many ways to allow a lot more flexibility in the process of content creation. Even with traditional blend mapping methods a lot of textures would require modifications in order to make changes to a specific material on a surface. Blend mapping is also generally only used for terrains and not on smaller objects or environments.

Epic Games has worked on a flexible method of producing content with multiple materials that is a lot more flexible than traditional material blending thanks to their in-engine material editor. The layered materials are basically materials within a material that are blended together similarly to traditional blend mapping, but with a lot more fine control. Different material functions can be created separately in any way needed and they can then be combined in a layered material defined to blend between the materials. The blending can be defined by masks, as in traditional blend mapping, or different kinds of parameters such as world direction or position can be used. The real benefit of the layered material system is that materials can be created and perfected on their own. When the desired result has been achieved in an individual material, it can be combined with other materials through a variety of available blending methods. This system leads to a less cluttered material network as all the material functions for a specific material are contained in its own material layer. (Youtube 2013; Unreal Engine Docs 2014a.)

UNREAL
ENGINE

Image 41. Rockets demonstrating the layered materials available in Unreal Engine 4 (Unreal Engine Docs 2014).

The example on the Unreal Engine Wiki for creating layered materials demonstrates the creation of a metal material with snow on top of it. The blend between the materials is defined according to the world direction so that the snow always stays on the top of the mesh. The power of Epic Games's in-engine material editor allows for a lot of changes inside the editor without the need for opening Photoshop to make changes. This helps to avoid long iteration times and frustration during the process. (Unreal Engine Wiki 2014.)

6.6 Photogrammetry

Photogrammetry is a technique that has been used extensively to model and measure in fields like engineering, the film industry, accident scene forensics and more recently, even in the game industry. The technique requires multiple photographs of a real world object or environment from different angles that get stitched together by software that try to match up the details in the photographs and calculate points in space to form a 3D representation of said object or environment. (Walford 2007; Eos Systems Inc. 2013; Youtube 2015.)

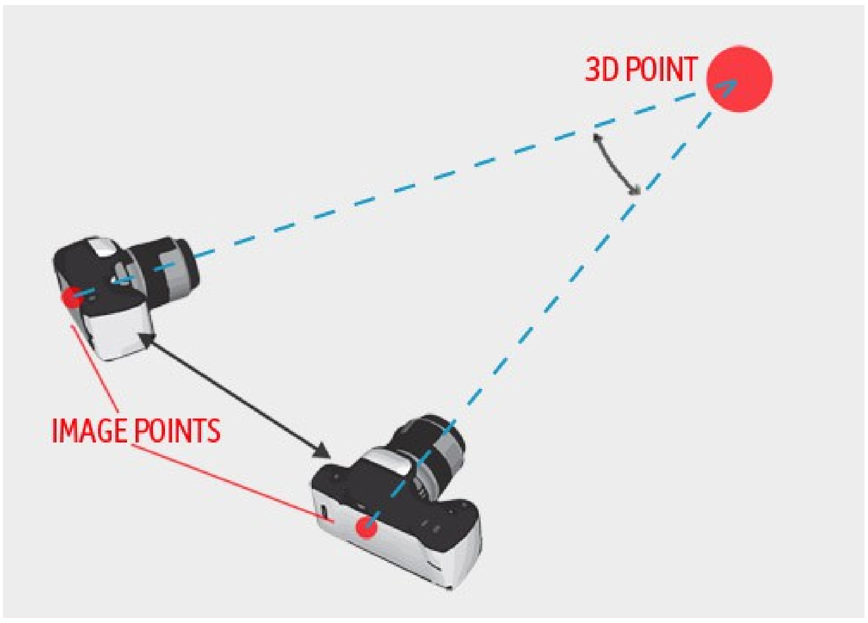


Image 42. By knowing the parameters of the cameras the location of a point in space can be calculated with simple geometry (Eos Systems Inc. 2013).

The point cloud generated by photogrammetry software, like Agisoft's Photoscan, are connected to form a polygonal mesh. The images used in the generation of the 3D mesh then get projected onto the model to form the basis of the textures. (Poznanski 2014.)

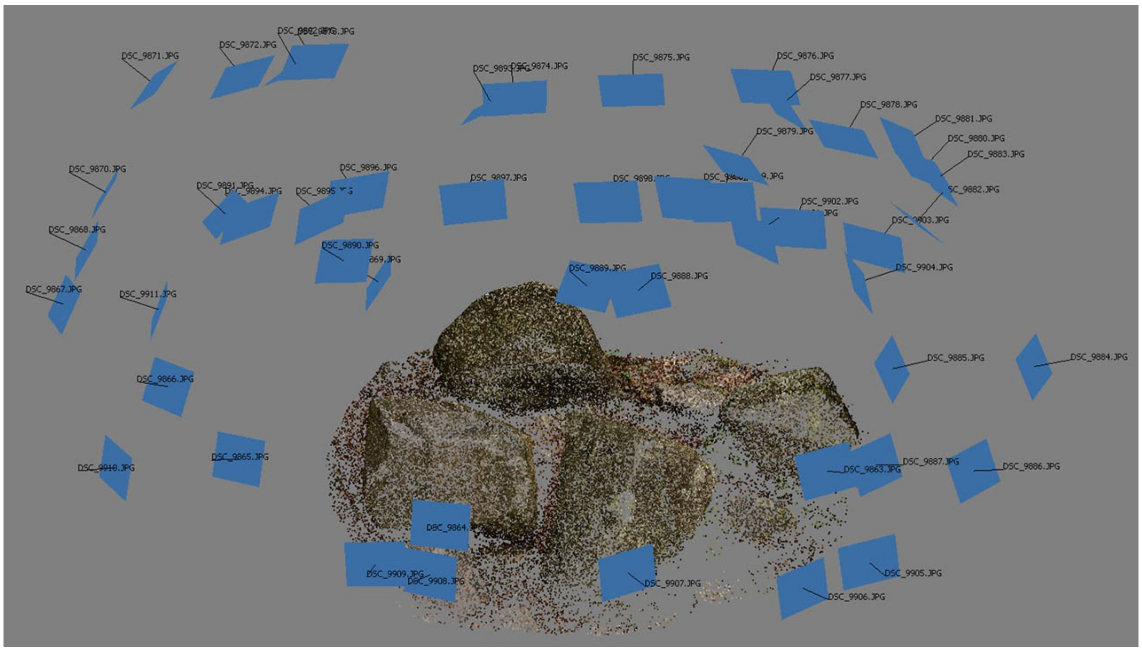


Image 43. Photogrammetry generated boulders for The Vanishing of Ethan Carter. The process requires several photographs shot from different angles of the subject to create the final mesh. (The Astronauts 2014a.)

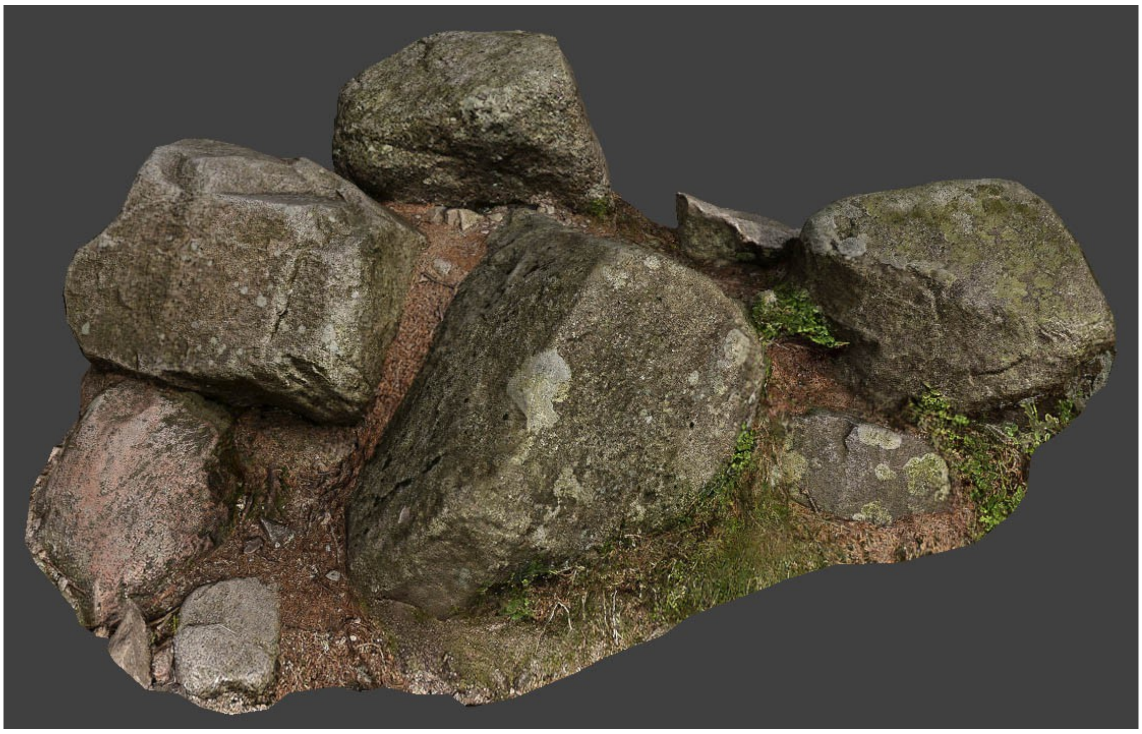


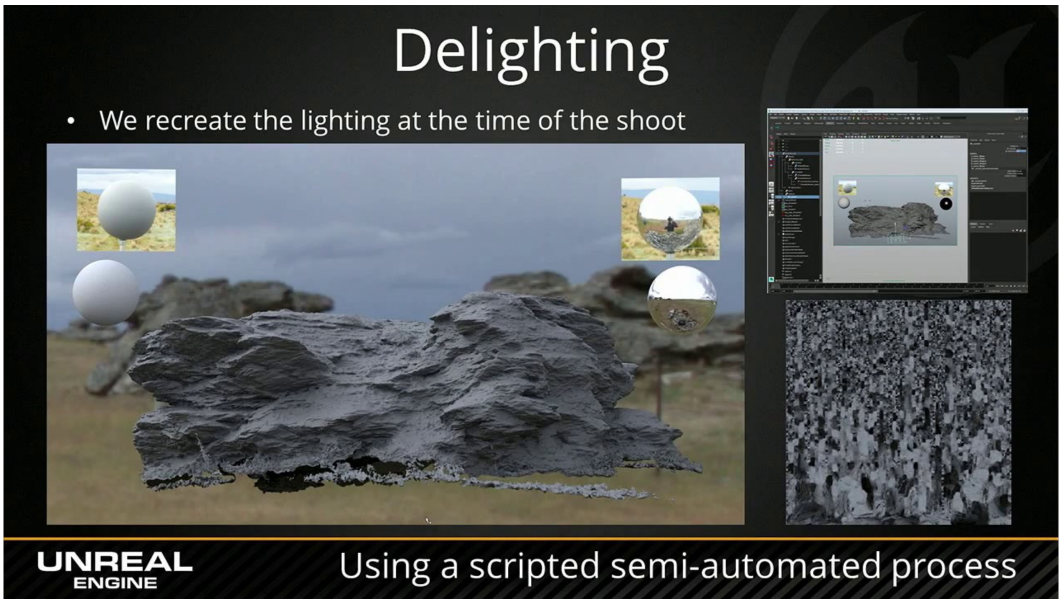
Image 44. The final in-game asset (The Astronauts 2014b).

The developers of The Vanishing of Ethan Carter, The Astronauts, used a lot of photogrammetry-generated assets to create the breathtaking world for their game. Even though a lot of the work is done by the software generating the 3D assets, there are a lot of factors that can go wrong if the user is inexperienced. Techniques that are valued in traditional photography can have negative consequences when working with photogrammetry. Shallow depth of field, highlights and noise can absolutely ruin a photogrammetry asset. Also, all of the lighting and shadows need to match, which means that changing weather conditions can prove to be highly problematic. After all pictures are shot, any errors that the software can not resolve need to be fixed. The last step involves a lot of optimization until the final in-game asset is complete. (Poznanski 2014.)

The Vanishing of Ethan Carter

Image 45. A scene from The Vanishing of Ethan Carter (The Astronauts 2014c).

Although photogrammetry can produce excellent content, it can often look static, as all the lighting and shadow information is baked into the textures. Epic Games have tried to counter this effect by doing delighting algorithms with reference to the real world lighting conditions, at the time of the shot, as a base. These lighting conditions are matched digitally and then reversed in order to create a neutral delighted color surface that can be used as a base for the textures. (Youtube 2015.)



Delighting

- We recreate the lighting at the time of the shoot

Using a scripted semi-automated process

Image 46. Epic Games's process of delighting a captured asset relies on recreating the lighting as it was at the time of the shot (Moritz Weller CGI & Photography 2015).

7 Conclusion

Although the blend shader I developed ended up being quite simple, I am still of the opinion that it could be of great help to anyone interested in gaining a deeper understanding of the topic and that it could be used by anyone willing to learn the somewhat unintuitive user interface (UI). The shader could be developed further in many ways. One such improvement would be to account for a fourth channel in the blend map, the alpha channel, available in certain image formats such as Targa. This would allow for four different textures to be blended together instead of the previous three, which naturally would lead to a more flexible way to add variation without having to split the mesh into multiple materials by polygon groups. At the time of development, I encountered a bug in the way Blender handles the alpha channel in the viewport, so the result was less than useful for the real-time texturing workflow I was hoping for.

In addition, I would like to add support for actual visual feedback of materials instead of simple diffuse textures. Having support for blending between materials would add even more visual feedback and look more like the final product seen on-screen in-game. As the point of the real-time preview was to improve the visual feedback, I think this is one of the essential parts that could be improved to make the shader more useful. The normalization and threshold values are improvements that would affect the final outcome in a more intuitive manner as well.

I also find that the user-friendliness could be improved by hiding the unnecessary components from the user to be less of a distraction while working with the shader. The lack of a UI has proven problematic and it is something I believe would be necessary to fix. I am of the opinion that currently the shader has a lot of potential, but is unfortunately not pleasant to work with. An integrated color palette has been added in a recent Blender version which was a change that made working with the shader a lot less tedious.

Despite there being a lot of interest in the shader at Bugbear and some artists even using it occasionally, discussions with an old colleague at Bugbear revealed that it is currently not being used. I think the reason for this is partly due to the shader only working inside Blender and the artists at Bugbear not being proficient in the software. Another reason, which I think is the biggest hurdle, is the complexity of the process involved with working with the shader due to the lack of a proper UI. Although the blend shader could be improved further, I am more than happy with the result I achieved. Blend maps and blend materials are techniques still used at Bugbear and an in-house tool has recently been put into development, so I hope that they will eventually have a user-friendly tool for the blend mapping process.

I consider blend mapping a highly efficient texturing technique, but it is getting outdated as other, more refined and flexible technologies are available. There are a lot of different ways to create textures nowadays that minimize the time wasted on recreating things due to iteration and general guesswork. An immense amount of different texturing techniques and alternatives to blend mapping are available and only some of them were briefly addressed in this thesis, but I think it is safe to say that there are a great deal of options to choose from for most texturing tasks. Procedural generation combined with artistic skill is something that is incredibly powerful and which I believe will take a stronger footing in the coming years, as more effort can be concentrated on the actual artistic parts of the content creation instead of spending countless hours on miniscule details such as painting scratches in the right places. I reckon that blend mapping still has its uses, but the limitations of the shader at Bugbear were quite severe compared to similar methods in Unreal Engine, for example.

Despite relying a lot on the experience I have gained working in the game industry, the research I conducted for this thesis during my free time has taught me a great deal regarding many topics, not just limited to texturing, and I believe a lot of people could learn something from the techniques I have studied.

8 Sources

3D Buzz 2008. Blend Textures Via Texture Alphas. [video] <https://www.3dbuzz.com/training/view/how-do-i-unreal/how-do-i/blend-textures-via-texture-alphas> (watched 27.2.2015).

Allegorithmic 2015. Substance Designer 5. [web page] <https://www.allegorithmic.com/products/substance-designer> (read 16.4.2015).

Animation Supplement 2012. History of texturing and shading. [web page] <http://animationsupplement.com/content/content/Texture%20History.htm> (read 24.3.2015).

Baker, Steve n.d. Learning to Love your Z-buffer. [web page] <http://www.sjbaker.org/steve/omniv/love_your_z_buffer.html> (read 14.2.2015).

Blender Wiki 2011. Stencil. [web page] <http://wiki.blender.org/index.php/Doc:2.4/Manual/Textures/Influence/Material#Stencil> (read 23.4.2014).

Blender Wiki 2014. Nodes. [web page] <http://wiki.blender.org/index.php/Doc:2.6/Manual/Materials/Nodes> (read 27.4.2015).

Burgess, Joel & Purkeypile, Nathan 2013. Skyrim's modular level design. [web page] <http://blog.joelburgess.com/2013/04/skyrims-modular-level-design-gdc-2013.html> (read 15.2.2015).

Computer Graphics Laboratory n.d. Mip Map Texturing. [web document] <https://graphics.ethz.ch/teaching/former/vc_master_06/Downloads/Mipmaps_1.pdf> (read 27.4.2015).

Cowley, Dana 2012. Amazing one texture environment. Unreal Engine Blog. [web page] <https://www.unrealengine.com/showcase/amazing-one-texture-enviroment> (read 10.11.2014).

Cryengine Manual 2014a. Blend Layer. [web page] <http://docs.cryengine.com/display/SDKDOC2/Blend+Layer> (read 19.3.2015).

Cryengine Manual 2014b. Blend mapping in Cryengine. [web page] <http://docs.cryengine.com/display/SDKDOC2/Blend+mapping+in+Cryengine> (read 19.3.2015).

Cryengine Manual 2014c. Creating Decal Textures and Materials. [web page] <http://docs.cryengine.com/display/SDKDOC2/Creating+Decal+Textures+and+Materials>
(read 4.4.2015).

DirectX.com n.d. Shader Overview – why shaders? [web page] <http://web.archive.org/web/20050505141636/http://www.directx.com/shader/> (read 27.4.2015).

Eat 3D 2012. Vertex Painting in UDK. [video]
<http://eat3d.com/free/vertex_painting> (watched 18.3.2015).

Eat 3D 2013. UDK Modular Masterclass - Efficiently Creating and Entire Scene with Tor Frick. [video]
<http://eat3d.com/udk_modular> (watched 29.4.2015).
Eos Systems Inc. 2013. How it works. PhotoModeler.com [web page]
<http://www.photomodeler.com/products/how-it-works.html> (read 20.3.2015).

Erdőkövy, Zoltán 2013. One-minute Dungeon: Behind The Scenes. [web page]
<http://www.zspline.net/blog/2013/12/16/one-minute-dungeon-behind-the-scenes/> (read 3.4.2015).

GitHub 2013. Screen & viewport. [web page]
<https://github.com/libgdx/libgdx/wiki/Screen-%26-viewport> (read 30.4.2015).

Glasser, Nate 2005. Texture Splatting in Direct3D. Gamedev.net. [web page]
<http://www.gamedev.net/page/resources/_/technical/game-programming/texture-splatting-in-direct3d-r2238> (read 8.4.2015).

Hastings, Al 2007. Texture Streaming. [web document]
<http://www.insomniacgames.com/tech/articles/1107/files/texture_streaming.pdf> (read 16.7.2014).

id Software 2009. From Texture Virtualization to Massive Parallelization. [web document]
<http://s09.idav.ucdavis.edu/talks/05-JP_id_Tech_5_Challenges.pdf> (read 16.3.2015).

L3DT documentation 2013. Texture Splatting. [web page]
<http://www.bundysoft.com/docs/doku.php?id=l3dt:algorithms:splatting> (read 24.4.2014).

Marmoset LLC n.d. Tutorial: The Miracle of Blending. [web page]
<https://www.marmoset.co/toolbag/learn/blending> (read 8.4.2015).

McAnlis, Colt 2014. Don't Alpha That Pixel!! [web page]
<http://mainroach.blogspot.fi/2014/04/dont-alpha-that-pixel.html> (read 4.4.2015).

McGuire, Max 2010. Blending Terrain Textures. [web page]
<http://web.archive.org/web/20140809193644/http://www.m4x0r.com/blog/2010/05/blending-terrain-textures/> (read 19.3.2015).

Microsoft Developer Network 2015a. What Is a View Frustum? [web page]
<https://msdn.microsoft.com/en-us/library/ff634570.aspx> (read 27.4.2015).

Microsoft Developer Network 2015b. What Is a Depth Buffer? [web page]
<https://msdn.microsoft.com/en-us/library/bb976071.aspx> (read 27.4.2015).

Panda3D Manual n.d. Texture Management. [web page]
<https://www.panda3d.org/manual/index.php/Texture_Management#Monitoring_memory_usage> (read 3.4.2015).

Polycount Wiki 2011. Vertex Color. [web page]
<http://wiki.polycount.com/wiki/Vertex_color> (read 19.5.2014).

Polycount Wiki 2014. Tiling. [web page]
<http://wiki.polycount.com/wiki/Tiling> (read 28.4.2015).

Polycount Wiki 2015a. Texture Coordinates. [web page]
<http://wiki.polycount.com/wiki/Uv> (read 28.4.2015).

Polycount Wiki 2015b. Transparency map. [web page]
<http://wiki.polycount.com/wiki/Transparency_map> (read 8.4.2015).

Poznanski, Andrzej 2014. Visual Revolution of The Vanishing of Ethan Carter. The Astronauts.com [web page]
<http://www.theastronauts.com/2014/03/visual-revolution-vanishing-ethan-carter/> (read 20.3.2015).

Rendering Pipeline 2012. Megatextures in Rage. [web page]
<http://renderingpipeline.com/2012/03/megatextures-in-rage/> (read 16.3.2015).

Second Life Wiki 2013. Alpha Modes Do's and Don'ts. [web page]
<http://wiki.secondlife.com/wiki/Alpha_Modes_Do%27s_and_Don%27ts> (read 4.4.2015).

Shader Forge Wiki 2014. Texture Splatting. [web page]
<http://acegikmo.com/shaderforge/wiki/index.php?title=Texture_Splatting> (read 19.3.2015).

Smith, Ryan James n.d. Advanced mesh paint in UDK. 3dmotive.com. [video]
<http://3dmotive.com/series/advanced-mesh-paint-in-udk/129/941> (watched 14.3.2015).

Tamakuwala, Dhawal 2013. Pixel VS. Texel. [web page]
<http://dmtamakuwala.blogspot.fi/2013/07/pixel-vs-texel.html> (read 28.4.2015).

Unreal Engine Docs 2014a. Layered materials. [web page]
<https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/LayeredMaterials/index.html> (read 1.1.2015).

Unreal Engine Docs 2014b. Landscape Materials. [web page]
<https://docs.unrealengine.com/latest/INT/Engine/Landscape/Materials/index.html> (read 29.4.2015).

Unreal Engine Wiki 2014. Creating layered materials. [web page]
<https://wiki.unrealengine.com/Creating_Layered_Materials_%28Tutorial%29> (read 1.1.2015).

Walford, Alan 2007. What is photogrammetry? Photogrammetry.com. [web page]
<http://www.photogrammetry.com/index.htm> (read 20.3.2015).

Wikipedia 2013. Texture Splatting. [web page]
<http://en.wikipedia.org/wiki/Texture_splatting> (read 24.4.2014).

Wikipedia 2014. Megatexture. [web page]
http://en.wikipedia.org/wiki/MegaTexture
(read 16.7.2014).

World Machine Software LLC 2013. Latest update. [web page] <http://www.world-machine.com/about.php?page=latest> (read 17.3.2015).

Yang, Robert 2013. Hacking blend transition masks into the Unity terrain shader. Radiator Design Blog. [web page] <http://www.blog.radiator.debacle.us/2013/09/hacking-blend-transition-masks-into.html> (read 14.4.2015).

Youtube 2010a. Id Tech 5 Stage Demo footage Part 1. [video] <https://www.youtube.com/watch?v=dbgZJPVbFAg> (watched 16.3.2015).

Youtube 2010b. Id Tech 5 Stage Demo footage Part 2. [video] <https://www.youtube.com/watch?v=unLG8qY17DQ> (watched 16.3.2015).

Youtube 2013. Inside Unreal: Layered Materials. [video] <https://www.youtube.com/watch?v=PjSFbPv3SLc> (watched 22.11.2014).

Youtube 2015. GDC 2015: Creating the Open World Kite Real-Time Demo in Unreal Engine 4. [video] <https://www.youtube.com/watch?v=clakekAHQx0> (watched 18.3.2015).

Image Sources

Image 1. Bugbear Entertainment 2015a. <http://bugbear.fi/> (available 30.4.2015).

Image 2. Next Car Game: Wreckfest 2015a. Finland. Bugbear Entertainment.

Image 3. Remedy Entertainment 2015a. <http://remedygames.com/> (available 30.4.2015).

Image 4. GitHub 2013. <https://raw.githubusercontent.com/wiki/libgdx/libgdx/images/screen-and-viewport1.png> (available 30.4.2015).

Image 5. rouncer 2013. gamedev.net. <http://img685.imageshack.us/img685/392/justuseblender.png> (available 30.4.2015).

Image 7. Thief II: The Metal Age 2000. United States. Looking Glass Studios. <http://static1.gamespot.com/uploads/original/mig/7/3/1/9/267319-thief2_003.jpg> (available 30.4.2015).

Image 11. Bugbear Entertainment 2013a.

Image 12. Next Car Game: Wreckfest 2015b. Finland. Bugbear Entertainment.

Image 13. Shader Forge Wiki 2014. Texture Splatting. <http://acegikmo.com/shaderforge/wiki/index.php?title=File:Splatmap-blending-example.png> (available 30.4.2015).

Image 14. Bugbear Entertainment 2013b.

Image 15. Bugbear Entertainment 2014a.

Image 16. Cryengine Manual 2014. Blend Layer. <http://docs.cryengine.com/download/attachments/1048616/blend_layer_01.jpg?version=1&modificationDate=1269272488000&api=v2> (available 30.4.2015).

Image 17. darktype 2012. Epic Games Community Forum. <https://forums.epicgames.com/threads/893085-Vertex-Painting-Need-Help!?p=30140394&viewfull=1#post30140394> (available 30.4.2015).

Image 18. Next Car Game: Wreckfest 2015c. Finland. Bugbear Entertainment.

Image 19. Bugbear Entertainment 2013a.

Image 22. Bugbear Entertainment 2014b.

Image 23. Bugbear Entertainment 2014c.

Image 24. Bugbear Entertainment 2014c.

Image 25. Bugbear Entertainment 2014d.

Image 30. Frick, Tor 2011. Amazing one-texture environment. Unreal Engine Blog. <https://de45xmedrsdbp.cloudfront.net/Showcase/tor-frick1-960x482-1655951775.jpg> (available 30.4.2015).

Image 31. Burgess, Joel & Purkeypile, Nathan 2013a. Skyrim's Modular Approach to Level Design. <http://2.bp.blogspot.com/-q7M9ZyKUWpI/UXAcJT3pSUI/AAAAAAAAAak/FvXRSHtOsik/s1600/DwarvenKitShot.png> (available 30.4.2015).

Image 32. Burgess, Joel & Purkeypile, Nathan 2013b. Skyrim's Modular Approach to Level Design. <http://4.bp.blogspot.com/-9hZKU_VTTZg/UWuPbOKCo9I/AAAAAAAAAXc/6_J0zSHdBAs/s1600/Dwem01.png> (available 30.4.2015).

Image 33. Remedy Entertainment 2015b.

Image 34. Remedy Entertainment 2015c.

Image 35. Justafin 2013. Alpha Test with Alpha Blend? Unity Forums. <http://forum.unity3d.com/threads/alpha-test-with-alpha-blend.175877/#post-1203792> (available 30.4.2015).

Image 36. Remedy Entertainment 2015d.

Image 37. Rage 2011. United States. id Software. <http://renderingpipeline.com/wp-content/uploads/2012/03/virtualtextureloading.gif> (available 30.4.2015).

Image 38. Rage 2011b. United States. id Software. <http://images4.alphacoders.com/184/184744.jpg> (available 30.4.2015).

Image 39. Allegorithmic 2015. Substance Designer 5. <https://www.allegorithmic.com/sites/default/files/sd_5_graph.jpg> (available 30.4.2015).

Image 41. Unreal Engine Docs 2014. Layered materials. <https://docs.unrealengine.com/latest/images/Engine/Rendering/Materials/LayeredMaterials/LayeredMaterials.jpg> (available 30.4.2015).

Image 42. Eos Systems Inc. 2013. How it works. <http://www.photomodeler.com/products/images/howto/Img2TwoCams.jpg> (available 30.4.2015).

Image 43. The Astronauts 2014a. Visual Revolution of The Vanishing of Ethan Carter. <http://i.imgur.com/HTxtvZI.jpg> (available 30.4.2015).

Image 44. The Astronauts 2014b. Visual Revolution of The Vanishing of Ethan Carter. <http://i.imgur.com/5ttYL4J.jpg> (available 30.4.2015).

Image 45. The Astronauts 2014c. Visual Revolution of The Vanishing of Ethan Carter. <http://www.theastronauts.com/wordpress/wp-content/uploads/2014/03/The_Vanishing_of_Ethan_Carter_Church.jpg> (available 30.4.2015).

Image 46. Moritz Weller CGI & Photography. The Tech & Beauty behind Epic’s UE4 Open World Demo. <https://moritzweller.files.wordpress.com/2015/03/epic_delighting_03.jpg> (available 30.4.2015).

Blend shader

The blend shader can be downloaded from the link below.

https://dl.dropboxusercontent.com/u/2344342/Blendshader.7z