Creation of Procedural Materials in Substance Designer and their Application in Unity

Anna Sanchis Albert

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

July 1, 2019

Supervised by: Juan Miguel Vilar Torres

Acknowledgments

I would like to thank my supervisor, Juan Miguel Vilar, who allowed me to work in this project and helped me to organize the ideas for it.

I would also like to thank my Procedural Art teacher in Saxion University, Mark Schipper, who showed me Substance Designer for the first time and made me aware of its existence.

And, of course, thanks to Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Abstract

This project was born to show the potential of the software Substance Designer and its uses in Unity. For this purpose, I created a collection of different materials using the software and implemented them in Unity.

In this document I will explain the characteristics of Substance Designer, the workflow I followed using the software, how a material can be converted into a procedural one, how to set up a Unity project to adapt well to the procedural materials and the demo I've done to show the adaptability of the materials, first with an static environment and later, in another scene, how the procedural materials work and change in runtime.

I hope that, with this project, I can show the importance of materials and all the features they have to offer, as they can replace a lot of geometry and lessen the load of polygons in a game environment.

Contents

CHAPTER 1

Introduction

Contents

1.1 Work Motivation

I discovered Substance Designer some months ago in my Erasmus stay and I became amazed by its versatility when I found materials done by professionals. They replicated complex geometry like tentacles, grass or the Iron throne (from the Game of throne series) using only this software and its features, not modeling involved.

I was impressed by these materials, and more so after I learnt that every parameter of every part of a material done in S. Designer could be modified in the editor and in runtime, allowing the creation of slightly (or hugely) different materials using one as the base, something really useful to reutilize materials one after another, twerking the parameters a bit to trick the eye, and making a person not realize that it is the same material repeated all over again. In short, something essential to bring realism to a 3D scene without wasting a lot of resources.

Later I found out that this software is being used by AAA such as Horizon 0 Dawn or uncharted 4 and how it has become the standard for PBR (Physical Based Rendering) content.

I read opinions of professionals about Substance Designer and how it had changed his work for the better. They talked about how it was so time saving, as it allows iterations

in the same materials with minimum effort and how fast it is. Work that used to take days with other softwares is now done in hours. It has been a revolution that is still happening as the Allegorithmic team keeps updating and improving the software.

I looked for jobs related to the use of this software and I found that there is a demand of professionals capable of using the Substance's family software (specially S. Painter and S. Designer) and I found out what I wanted to try and become in the future.

When the opportunity to dedicate hours of my degree into something I wanted to improve and work in, a collection of materials for my portfolio was the first thing that arose to my mind.

With the main objective established and with the will to show others the potential of Substance Designer and how much it has to offer, I decided to take up the opportunity to do this as my project.

1.2 Objectives

There were two main objectives in this project, the creation of procedural materials in Substance Designer and a demo to show them in Unity, that can be divided in subtasks or objectives:

Substance Designer:

- Learn the basics of the software, making use of the Allegorithmics' (developer of Substance) webpage, documentation and its tutorials and watching and following the ones done by professionals in YouTube.
- Create a diverse array of materials, following the distinct thematics of Modern, Medieval, Victorian, Rustic and Outdoor rooms.
- Learn how to parametrize material variables to turn them into procedural ones.
- Learn how to make them compatible with Unity.

Unity:

- Learn how to set up PBR in Unity, using cube maps and the Unity asset "Post Processing Stack"
- Install the "Substance in Unity" plug-in [3] and learn its characteristics.
- Set up the materials in a compelling way and make a house structure.
- Create a character and a simple movement for it to be able to tour the scene without problems.
- Code and attach a first-person camera to the character and add and create a Post Processing Behavior for it.

- Code an interface that appears when you click in a material (from a new scene) that will show the parameters and a slide to change the values and the material in runtime.

When everything is done, the objective of having a Unity project that shows the potential of Substance designer should be done. However, it can be extended at any time with new textures and rooms to show more variety.

1.3 Environment and Initial State

This project started with the idea of creating a collection of materials done in Substance Designer with no implementation in Unity but, as I investigated further and got to know the possibilities of the software, I realized the potential it has in Unity.

After consulting with my tutor the best way of adding the materials to a Unity project, I decided to make a demo where the player could walk around the environment and change the procedural materials in runtime as I thought was a pity to just show individual materials, with no connection with each other.

Fortunately, there was plenty of documentation to implement this thanks to Allegorithmic's team as well as a plug-in to use Substance materials in Unity and thus, the second part of this project was born.

CHAPTER 2

Planning and Resource Evaluation

Contents

2.1 Planning

As stated before, this project was divided into two parts: The creation of materials in Substance Designer and their implementation in Unity that can be divided in smaller tasks:

Substance Designer: Estimated 200 hours.

- Research: Find out how the software works, using the documentation from the developer's page and watching tutorials of various people in youtube while taking notes. Here are also included the time spending looking for solutions to complications. Estimated of 50 hours.
- Creation of materials, these also include the time it took to expose their variables to turn them into procedural ones and the materials that may not make it into the final demo. Estimated 150 hours.

Unity implementation: Estimated 50 hours

- Research: How to set the properly PBR implementation in Unity, what plugins were necessary for a correct visualization of Substance Designer's materials, how to code a script that could enable runtime modification and other miscellaneous things. Estimated 5 hours.
- Setting up the first demo, with all the materials, creating rooms of different themes and with the properly PBR implementation. Estimated 20 hours.
- Scripting the camera and movement of the player inside the first demo. Estimated 5 hours.
- Setting up a second demo, with only one sample of every material. Estimated 5 hours.
- Creating a script and menu for the materials in the second scene that will enable them to change in runtime. Estimated 15 hours.

Writing of the project memory: Estimated 50 hours.

2.2 Resource Evaluation

The cost of this project depends on a lot of different factors such as the condition of the workers, the business' revenue, the quality of the equipment that is used for it, how many materials have to be done etc. I show here some differences in cost according to the softwares' official page and standard salaries, to give an idea of how much the final price depends on those factors (the money has been rounded after converting it from dollars to euros):

Manpower The average salary for this kind of work oscillate between the 34K and 77K euros annually.

Unity Free personal license (yearly revenue under 90K euros). Plus, 22 euros/month. Pro 111 euros/month [10].

Substance Designer Substance Designer: One year student license for free. 132 euros/year for the inde license (yearly revenue under 90K euros). 280 euros/year for the pro license (revenue between 99K-99M euros) and tailored solutions for those who have a revenue higher than 90M euros [2].

Hardware The computers recommended for the use of Substance Designer, according to various forums, have a price of 1200 euros approximately, as the hardware needed to have a good performance is quite pricey.

To calculate an approximation of the real cost for this project, I have considered the 250 hours it has took me to create both materials and their implementation.

As a Substance Designer beginner, the salary would probably be the minimum one so 20K euros annually. With a timetable of 8 hours a day, 5 days a week, there is a total of 200 hours a month worked approximately. Dividing the annual revenue by 12, the result turns to be 1666 euros per 200 hours. In conclusion, 2082 euros for 250 hours of work.

Now, concerning the software and hardware. Acquiring the indie license for both programs would be the appropriate thing to do considering the income in this situation.

To calculate the expense that Unity generates for the hours that this project takes, like in the salary case. For 200 hours a month the price is 22 euros, so for 250 the expenses sum to 27.5 euros. Likely, if Substance Designer costs 132 euros yearly, divided by 12 it's 11 euros for 200 hours and 13.75 euros for 250. Adding the two costs together, the software expenses for this particular project turn out to be 41.25 euros.

The computer can be used for two to three years, in this case let's use the average of two and a half years, in other words, 30 months. 1200 (cost of the computer) divided by 30 is 40 euros a month, so, 50 euros for 250 hours.

Taking into consideration all the factors, the total cost of this project is 941.25 euros.

Chapter 3

System Analysis and Design

Contents

3.1 GDD

Even thought this project is not a game at is own, it does have some common ground with them. There is interaction with the environment, an exploration objective and the user does control an avatar in a virtual environment.

Taking this into account, I think it's important to make a brief summary in a form of a simple GDD to introduce the project to the reader.

Game Concept Two demos linked together that display and let the user interact with the materials that were created for the environment.

Target Audience Students of all ages that are interested in seeing how Substance Designer works when implemented in Unity.

Gameflow Summary The user can jump back and forth between the two demos, by pressing the key 2 to go from the first one to the second one and by pressing the key 1 to do the inverse thing.

Gameplay When the user starts the demo, the first thing they are going to notice is that there is no interface and that the environment takes all the screen space. They can explore the surrounding by pressing the WASD keys and rotating the camera with the mouse. The space to explore is not that large, as it only consist of 4 connected rooms that form a closed space.

When the user is done exploring the first demo they can press the key 2 to switch to the second one. In this scene, the user can see the individual materials displayed in cubes and can interact with them by clicking. Whenever they press one, a menu with slider appears that allows the user to modify the materials at runtime and see how they change immediately.

3.2 Requirements Analysis

3.2.1 Functional Requirements

The few functional requirements of this project are found in the two demos and all of them relate to the navigation of the corresponding scene:

The tables from 3.1 to 3.5 correspond to the first demo, and the one from 3.6 to 3.8 to the second :

Input:	Key W
Output:	Movement
The user moves the avatar forward in the Z axis by a set speed.	

Table 3.1: Functional requirement "MOV1."

Input:	Key S
Output:	Movement
The user moves the avatar backwards in the Z axis by a set speed.	

Table 3.2: Functional requirement "MOV2."

Input:	Key D
Output:	Movement
The user moves the avatar forward in the X axis by a set speed.	

Table 3.3: Functional requirement "MOV3."

Input:	Key A
Output:	Movement
The user moves the avatar backwards in the X axis by a set speed.	

Table 3.4: Functional requirement "MOV4."

Input:	Mouse movement
Output:	Camera rotation
The user moves the mouse, the camera turns accordingly to enable a 360 view of the scene.	

Table 3.5: Functional requirement "CAM"

Input:	Mouse movement
Output:	Cursor movement
The user navigates the scene moving the mouse, and the cursor will be translated accordingly	

Table 3.6: Functional requirement "NAV."

Input:	Mouse left click
Output:	Material selection
The user selects the material they want to modify by clicking on it in the scene. Later, the menu used for this changes will be displayed.	

Table 3.7: Functional requirement "CLICK ON MATERIAL"

Input:	Mouse left click
Output:	Material change
The user selects one of the slider that appears after clicking on a material. Once he varies the position of the handle, the material will show the changes according to this variation.	

Table 3.8: Functional requirement "CLICK ON SLIDER"

3.2.2 Non-Functional Requirements

There are none in this project.

3.3 System Design

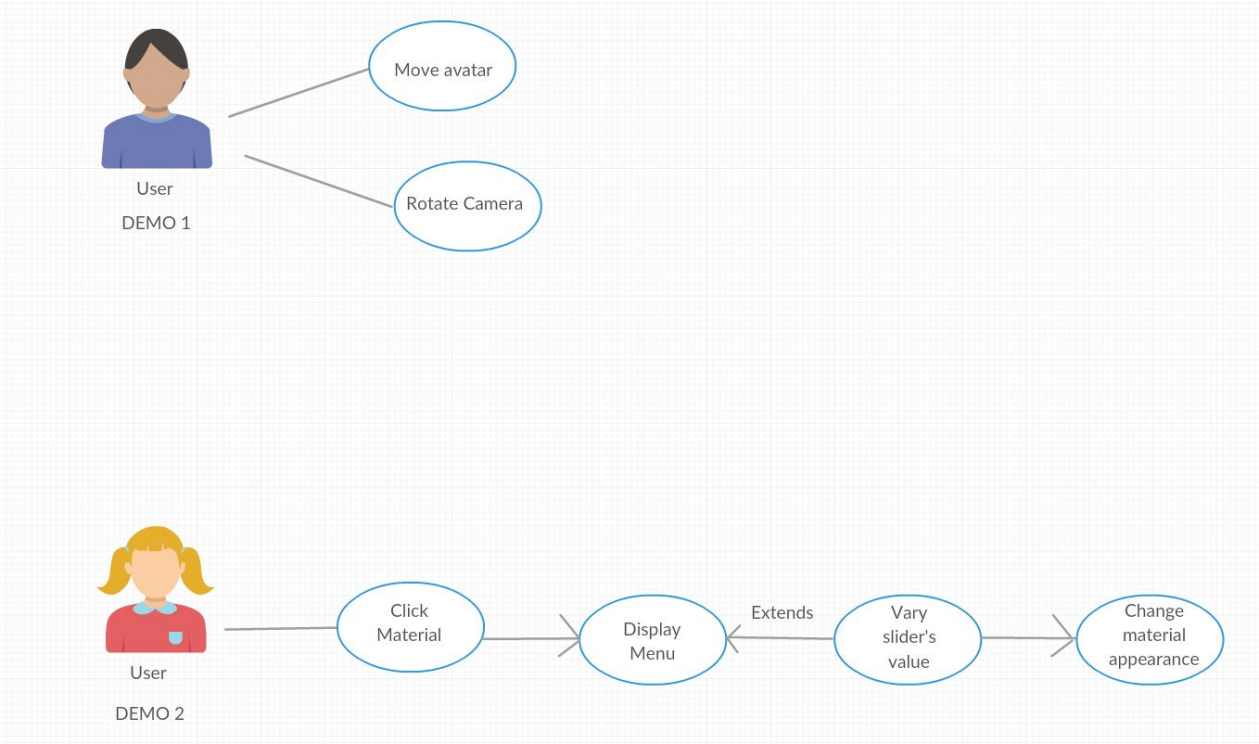

Figure 3.1: Case use diagram

The case use diagram is divided by demos, representing the actions that can be done in each one.

DEMO 1 The user can interact with the scene using the keyboard and mouse. With the former they can rotate the camera to look around and, with the latter, they control the movement of the avatar to explore the demo.

DEMO 2 At the start of the demo, the user can only move around the mouse to click on a material. When they do, a menu is shown with sliders that the user can interact with to change the material's appearance and/or click in another material to modify it instead.

3.4 System Architecture

As this project was done in Unity, I made use of its characteristics to develop it, this is the reason why all of my scripts use the System Collections and the UnityEngine extensions in them. Moreover, the use of its function `FindObjectsOfType Type` to be able to create references to other scripts to access and use their public variables from others has been key to connect scripts to each other to achieve the desired results. Another really useful characteristic of Unity that I used were the functions that detected the event of the mouse or keys being clicked, necessary for the overall navigation and use of the demo.

However, the scripts structure I used for each demo varies enormously as the second one has much more things to take care of than the first one to manage how the menus are displayed and how to modify the materials in runtime:

First demo: This demo only needs to control the avatar and respond to the WASD and mouse input to control what the avatar can see. To this end, two scripts are needed:

- Camera script attached to the avatar that reads the mouse input to rotate the view accordingly.
- Movement script attached to the avatar that reads the WASD input and makes a translation to the rigidbody accordingly.

Second demo: This demo is a bit more complicated. All the scripts are connected to each other to reflect the actions of the user in the view. Clicking in a material means that the corresponding menu has to be shown, and if the user makes changes in any slider in that menu the same material has to change accordingly. To achieve this various scripts and UI elements were implemented:

- UI Menu divided in fourteen subgroups, one for each material. Inside them are sliders for each property of that material that can be altered. These sliders have a parametre shown in the inspector called On Value Changed that passes the value of that slider when it's been modified to the function it has been assigned to. This way, the script has access to the slider's value and can actualize it.
- Material catalog script attached to an empty object that makes use of the `Substance.Game` namespace to be able to access the materials' properties from the script. It has a public reference for each material in the scene, and a function for every property of them that can be modified. These functions are called when the value of the slider that references them is changed.
- Menu manager scripts, fourteen inheritances of the same script that are attached to the parents of the subgroups in the UI. This script has two public functions that disable or enable all of the children of the object where the script is attached to.

- Raycast script that reference all of the Menu manager scripts in it to be able to access to their public functions. With a raycast cast from the camera to the mouse position triggered when the user has clicked on the demo, the script obtains the tag of the object that it has hit. Depending of the tag that is returned, the script calls the hide or show functions of the Menu manager scripts to make them visible or invisible from the view.

3.5 Interface Design

I developed a simple interface for the second part of the Unity demo.

When the user clicks on one of the materials displayed in the scene, a menu appears in the left part of the screen. This menu consists of a list of parameters, with every one of them having a slider below, that allows the user to change the corresponding value and see the modification in the object at runtime.

Once the user clicks in another element of the screen, the menu changes accordingly to display the new variables of the corresponding material. Here a visual example of what I mean:

Figure 3.2: Interface initial state

Figure 3.3: Interface after playing with the parameters

Chapter 4

Work development and Results

Contents

The work development, as said before, can be divided, mostly, in two parts: The procedural material creation in Substance Designer and the creation of two demos in Unity to showcast them.

This chapter is divided in four parts: a brief introduction to Substance Designer and the investigation done before starting the materials; an analysis of one them to show the workflow used; a part about the learning curve in Substance Designer; and finally a brief explanation on how the demos were set up and developed in Unity.

4.1 Introduction to Substance Designer

As stated before, Substance Designer is a node based, non-linear and non-destructive software that allows the user to generate procedural textures and materials developed by the Allegorithmic's team. But, what does it actually mean? How does it work?

Node Based To start, we have to know what a node contains, what it does and how it interacts with the rest of the project.

A node contains a 2D image data that stores all the information, this information can be generated by the node or it can be passed from another one.

Each node has its own function with its own parameters that the user can alter to modify the result. For example, a plasma generator has scale and disorder parameters while a transformation 2D has rotation, stretch, offset (X and Y), mipmap mode, mipmap level, matte color and filtering. These variables are later used in Unity to modify the material in real time after exposing them. It is important to experiment with the nodes to learn and understand which one does what and where to use it as they may have unexpected uses that the user has never thought of before... and watch a lot of graph breakdown videos made by people that has realized it before you have, to save time reinventing the wheel.

The nodes can be linked to interact and exchange information with each other. The places where they can be connected are represented by little circles, if the circle is at the left of the node it means that the node can receive information, and if it is placed at the right that the node can pass information to another one.

Nodes that only have right circles are called generators because they create new information (for example, basic shapes or noises), not modify it, and the ones that only have left circles are the outputs, special nodes that end the graphs. The nodes that have both connections are called filters because they modify existing information. They have left and right connections to receive information, the user modifies it making use of the node's specific parameters and later passes the altered image to an output to end the graph or to another filter to continue with the variations.

In Figure 4.1 we can find one example for each of the nodes explained before.

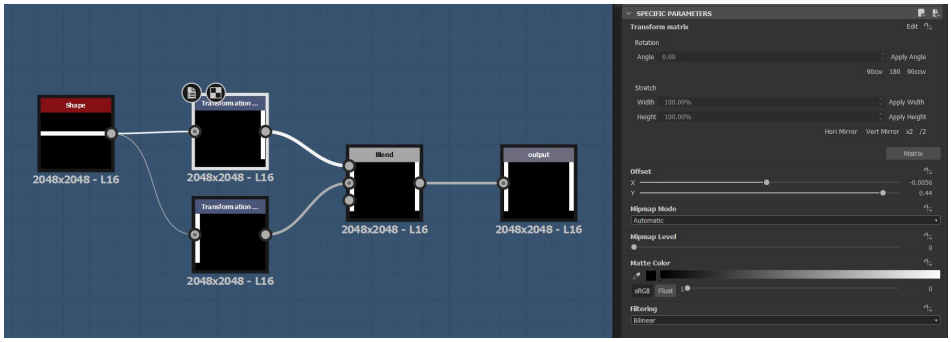

Figure 4.1: Visual example of links, inputs and the specific parameters of the transformation 2D

Non-Linear That the software is non-linear means that, when the user modifies a node's parameter, all the nodes that are linked to it and receive information from that one change accordingly even to the outputs. This is one amazing feature, as

one little change can greatly vary the final result, and even create different looking materials by just touching one simple variable. It also works really quickly and allows the user to see the results in the 2D and 3D portviews almost immediately.

Non-Destructive All actions can be reversed without losing any part of the work. Even though all nodes are connected to each other, none of them depends on another to function, so the user can delete whatever they want whenever they want without crashes or incompatibilities. The only information that the user may loose are connections from that node, maybe causing an interruption in the graph, but it is as easy as to reconnect with the remaining nodes.

All in all and in my opinion, Substance Designer is a software created to allow as much experimentation as the user wants, rewarding them with amazing visuals and eye catching results that can be modified at any time.

4.2 Procedural Materials

To understand the workflow for the creation of all the materials I will explain a graphic example of how a material builds up. For this purpose, I will analyze my heaviest material, the castle wall with an arch, and explain step by step how it was created, the problems that arose and how to solve them.

Why have I picked this material? Because apart from being the most complex that I have created with around 120 nodes, it contains all the problems that I had encountered while doing the materials before this one: Opacity, height blend and color blend as well as having some complex geometry conditions. The whole material's graph can be seen in Figure 4.2.

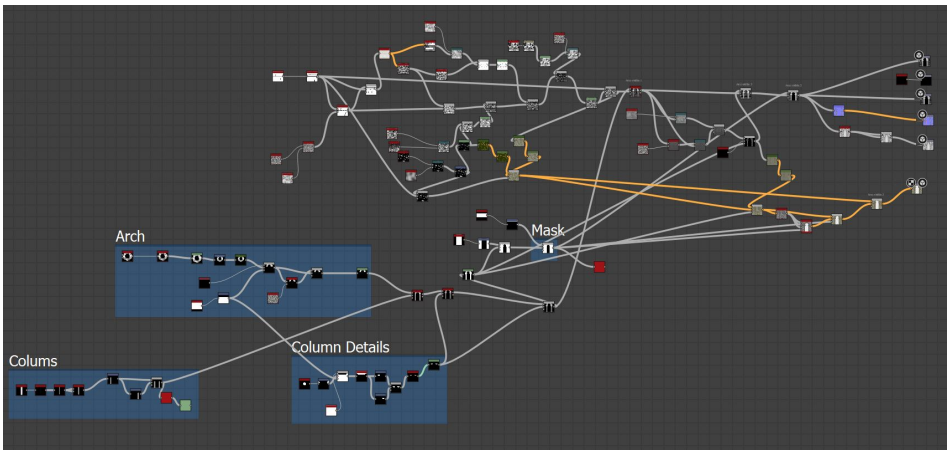

Figure 4.2: Whole material's graph

The question is, where to start? The first step was to think about what kind of material I wanted to simulate and, once I knew, to look for pictures to use as references. For this project the themes were: Victorian, Modern, Castle, Garden and Rustic rooms that would develop into floors, ceilings, walls, doors and windows.

The next step after finding a good reference picture or just looking through the materials surrounding me, was to divide the material into its core shapes. One example of breaking down a material can be found in the stone arch: It is an agrupation of irregular squares arranged in a semi-circular way, with two other larger squares supporting them that together form a more complicated shape.

As seen in this Figure 4.2, the creation of the material is carried out by an array of nodes that are interconnected until they reach the output nodes.

The output nodes are the ones that generate the maps and store the material's information, so we can later export and use it in other programs such as Unity. They are the ones shown in Figure 4.3.

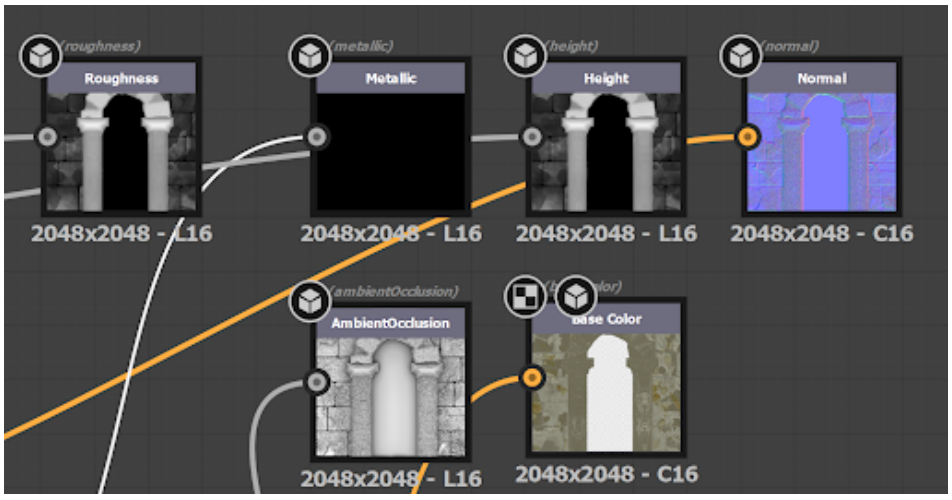

Figure 4.3: Outputs and the information each one of them stores.

As a brief summary from the Allegorithmic's documentation [5] [6], the roughness map (upper left node in Figure 4.3) describes the surface irregularities that cause light diffusion, the lighter it is the smoother that part is, the more light is reflected. This one, combined with the metallic map (second upper node in Figure 4.3), dictate how the light is reflected and so the metallic appearance of the material.

The height map (third upper node in Figure 4.3) is often used for displacement in rendering. It can be used for parallax mapping, helping to add more apparent depth and thus greater realism (basically, with this information Unity can expand the material giving it depth according to its values, giving realism and enabling 3D modeling within the software).

The ambient occlusion (first lower node in Figure 4.3) defines how much of the ambient environment lighting is accessible to a surface point, it creates shadows.

The normal (last upper node in Figure 4.3) map is used to simulate surface details. It is an RGB map where each component corresponds to the X, Y and Z coordinates of the surface normal. It can be used to store the projected details of a high-resolution model to a low-resolution model.

The base color (second lower node in Figure 4.3) gives color to the vertices of the material.

You can create more outputs like opacity, glossiness, specular etc. but they are used in other contexts and, in unity, the opacity is stored in the base color map, and doesn't read the one generated by the opacity output (will explain later in detail as it was a problem I encountered).

Now it is time to explain step by step a breakdown of the graph of Figure 4.2. shown before to clarify the workflow I followed with all the materials and to demonstrate the uses of nodes mentioned in the 'Introduction To Substance Designer' section.

4.2.1 Arch and columns

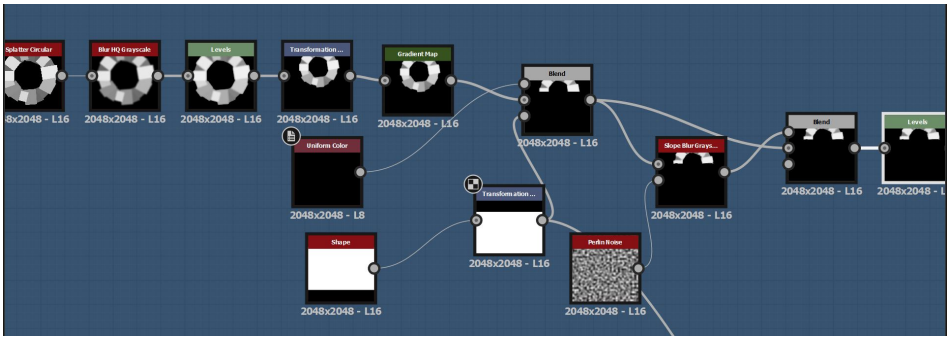

Figure 4.4: Part of the graph where the superior part of the arch is created

In Figure 4.4 we can see the part of the graph where the superior part of the arch is created: The splatter circular is a node that creates copies of a chosen shape (implemented or plugged in at the left of the node) and arranges them around a pivot point, creating a circular form. The user can select if they want randomness in the width, height, luminance etc. and choose how many rings to display, the distance from the pivot, the number shapes... It has a lot of parameters that are worth experimenting with.

Later, the blur is used to make the edges less sharp and to erase the sawtooth effect geometrical shapes have in Substance Designer.

The levels adjust the white balance in the part of the geometry that has been plugged to its left. This node is frequently used because it helps balance the heights between element as is easy to make mistakes in that regard when the material is not yet done.

The transformation 2D is the node you want to use to move the geometry, expand it, make symmetries and so on, as its names indicates.

The blend node is one of the most importants nodes in all the software as it is used to merge different nodes. In this case it mixes the circle of squares with a half white shape, to erase half of it. This node works almost like the blending layer mode in Photoshop, allowing mergings like add, subtract, multiply, lighten... the layers being the two input nodes.

Another widely used node is the Slope Blur Grayscale, which using a texture (normally a noise) erodes the edges of the target node and makes them more imperfect. It is really good to attain an effect of erosion, like in this case.

The creation of the columns supporting the arch was done as shown in Figure 4.5.

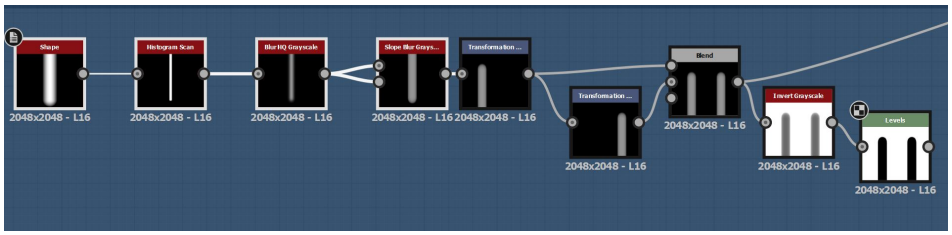

Figure 4.5: Part of the graph where the columns of the arch are created.

The columns are really simple things to do: First, using a shape node, select the Capsule pattern, plug it to a Histogram scan to be able to change its width whenever you need, slope blur it with itself to get a more smooth transition between black and gray and do some transformation to make the columns stand under the arch. However, just leaving two straight columns seems kind of boring, so a new shape could be created to stand between the columns and the arch as seen in Figure 4.6.

The process is quite similar to the ones done before: go from simple shapes, blend them with others to obtain more complex ones and blur them to make a smooth transition between the black of the background and the white (tallest vertex). In this case the blurring is done by using a bevel, a node that identifies the edges of the geometry and creates a gradient following the parametres you give to it.

For the Arch microsurface, I did what is shown in Figure 4.7.

I used two textures and passed them to a directional warp. This node is really useful because the directional warp transform a given texture using the grayscale parameters, if we do it in a really big scale, the texture will adapt to the contour of the material breaking the continuity of the texture and making it more natural looking.

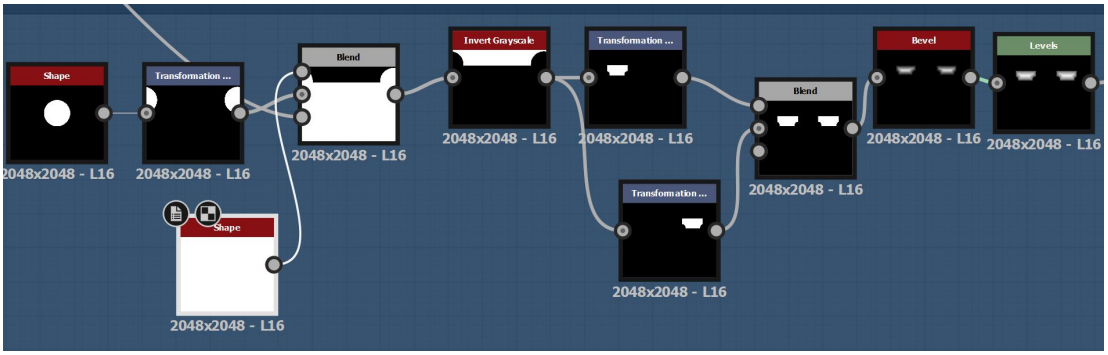

Figure 4.6: Part of the graph where the superior part of the columns are created.

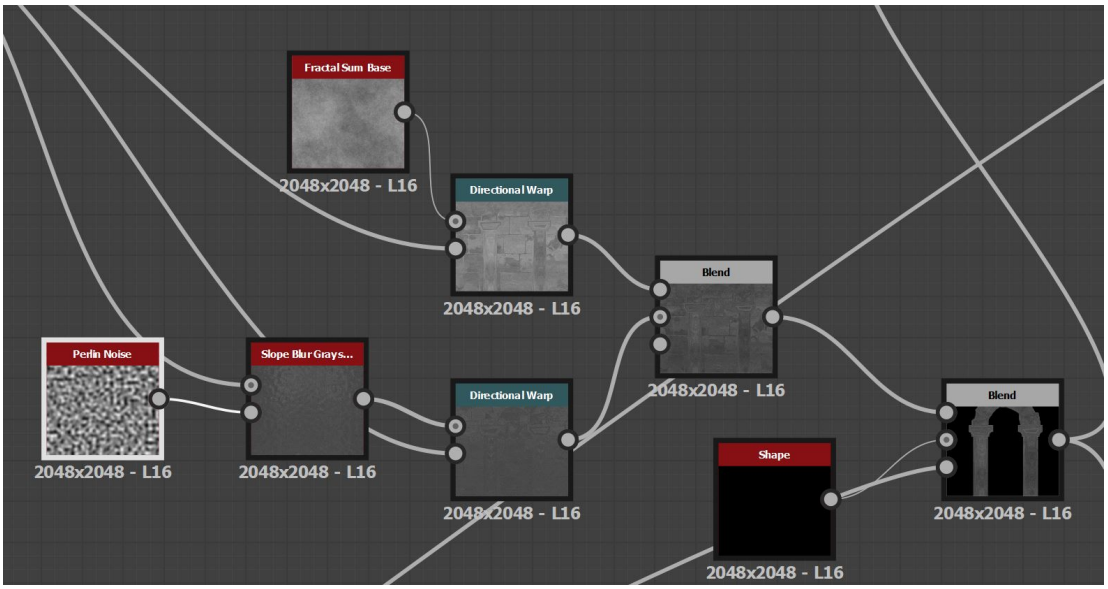

Figure 4.7: Part of the graph where the arch microsurface is created.

4.2.2 Stone Wall

Having a lone arch could be perfectly fine if we so desired, it could serve as decoration or entry to a area in ruins but, as the objective is to create delimited rooms, the arch alone is not enough. In this part I explain how the stone wall made of bricks was created, starting from the basic shapes to all the details of its surface.

The geometry's base was done as seen in Figure 4.8.

The geometry is really easy to understand in this case: The brick generator can do a lot of work for you. The edge detect, as the name says, detects the edges of the geometry and allows you to modify their thickness, roundness in the intersections... and, of course, a bit more of slope blurr to give them an eroded look.

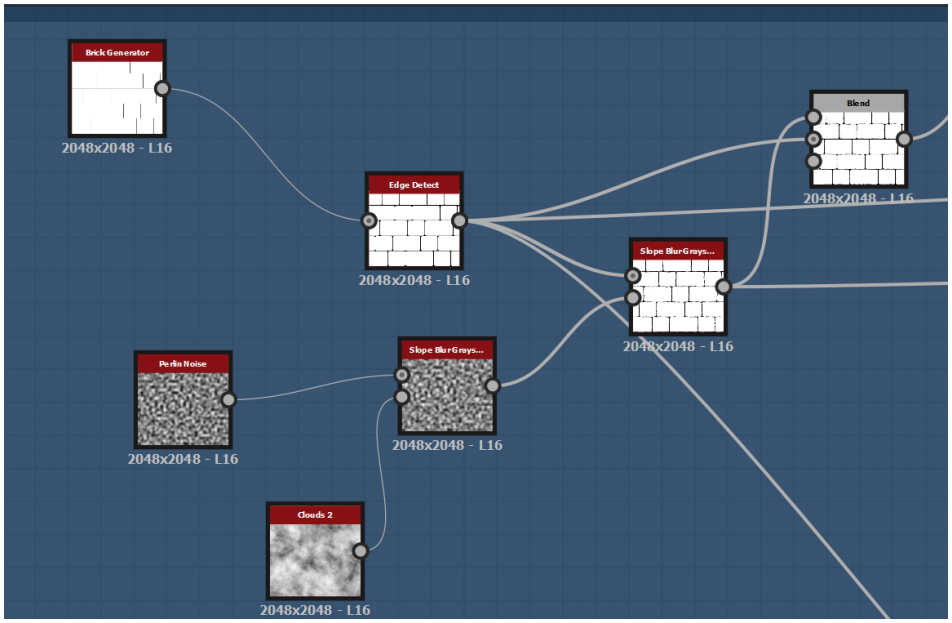

Figure 4.8: Part of the graph where the brick wall basic shape is created

And now to the fun part of this project: The micro surface. There are a lot of nodes involved, as it came mostly from trial and error, so it is a bit hard to explain with words. This said, let's start by the beginning that can be seen in Figure 4.9. The flood fill is used to give every brick a random gradient. This gradient becomes solid colors in one part of the graph so as to use it in a Direction warp with a cloud texture.

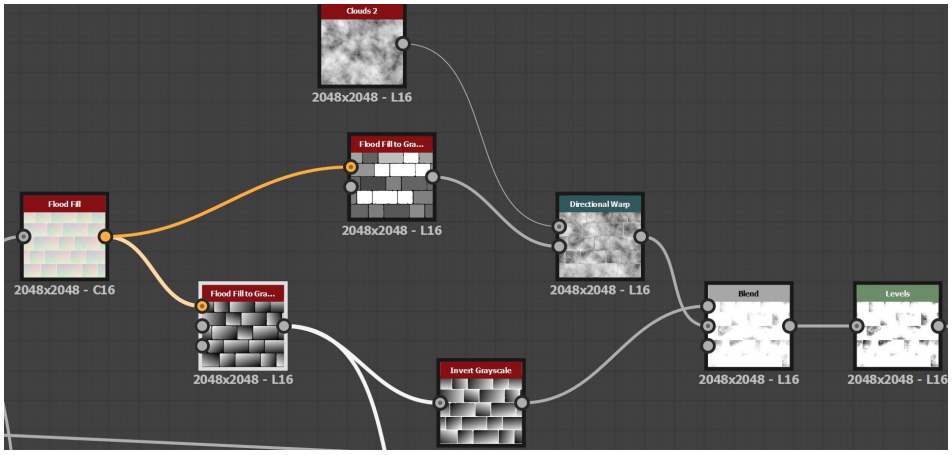

Figure 4.9: Part of the graph where the bricks' microsurface is created.

The parts showed in Figure 4.10 and Figure 4.11, blended with other textures (mostly

grunge maps, clouds and noises) are in charge of giving every brick an interesting a unique pattern to make it look more realistic. In them I wanted to mimic the feel of eroded stone and give an extra layer of moss, to simulate the passing of time and what it does to the material.

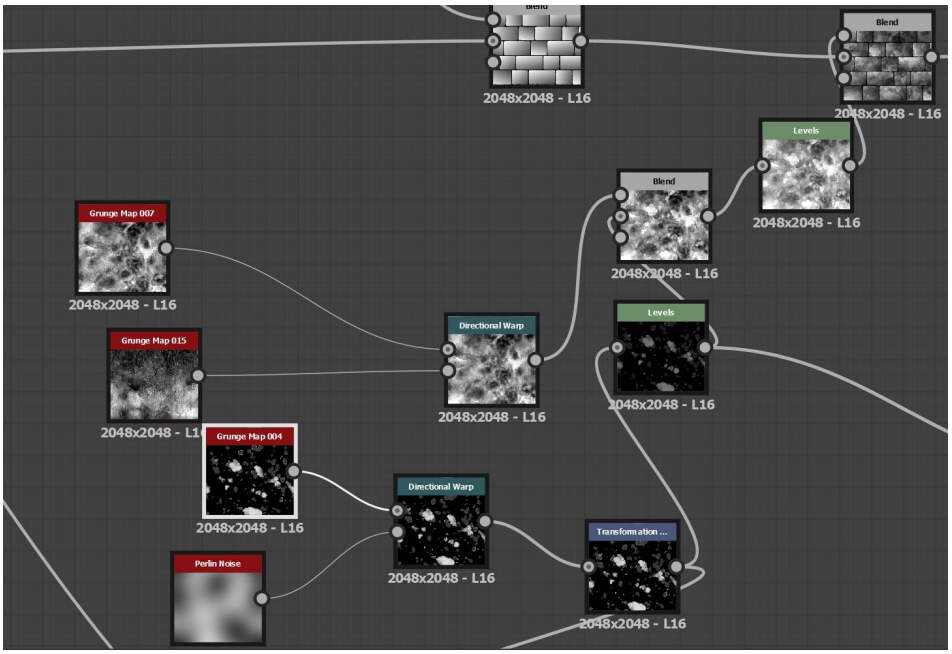

Figure 4.10: Part used to add moss to the bricks.

So now that the two materials are done, the bricks and the arch, what is the best way to join them? The best solution is to use a height blend. This node accepts two inputs: a node that is supposed to be at the top and another that is supposed to be under the other. In this case the bricks are under the arch so we will appoint the bricks to be at the bottom. The blending then is really simple: When the blending value is 0, the brick texture appears without any changes and the arch one in not visible but, as we increase the value, the arch starts to appear and the overall white balance of the bricks starts to gets darker until it hits 1 and it totally disappears and only the arch remains. This is a really neat way of mixing materials as you can control how separated they become and it can lead to one material surging from another giving it a natural look.

But the main reason I used this method was because, this way, I could have two materials at a price of one: a plain brick material and another that has a wall and an arch that serves as an entrance.

Now that I have mentioned the entrance I will take the opportunity to talk about how opacity works in Substance Designer. I mentioned this before but it was kind of a

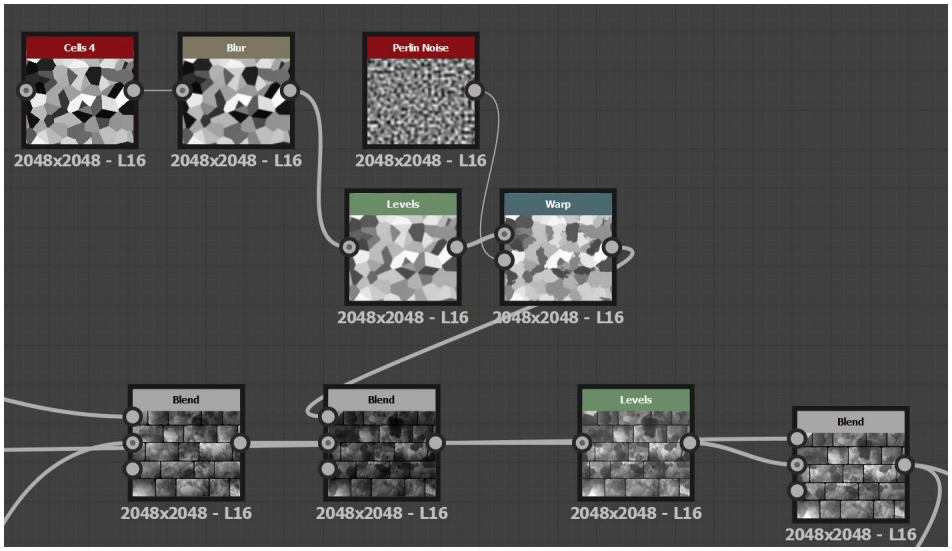

Figure 4.11: Part used to give more texture to the bricks.

headache for me. At first, I created a mask for the entrance, where the hollow part was in black and the rest of the material was white and attached it to an opacity output I created. Sadly, it didn't work when I exported the material to Unity. Looking for answers I found out that Unity read the opacity from the base color, and the only way to export it was to separate the material's channels using a RBGA merge, where the RBG can be plugged in by the normal material and the A by the opacity mask. Then, after adding color using a Gradient map and being careful to still have the opacity, you can plug it into the base color output. Adding color is as simple as adding gradient maps and to have the materials plugged in. You can merge the distinct parts using a blending nodes and opacity masks (really important to make masks in white and black of the different elements of the materials for using them for things like this).

The normal map can be generated by using a normal node that automatically generates the map for you, the same case as the Ambient Occlusion with the AO node.

4.2.3 Variable Parametrization

Now that all the outputs have been properly generated and the material is finished it's time to decide what variables to parametrize. This process is what makes a material procedural, as it allows the user to be able to change it whenever they decide.

How is it done? First, you have to think what aspects of the material do you want to be able to modify later. I recommend to go node by node and experiment with all the parameters every node offers. For example, the node HSL has three parameters: Hue, saturation and luminosity that affect the overall coloration of the gradient map plugged

into it, so it could be quite useful to have them in Unity. Now that we have selected a parameter we only need to click in the function symbol above it, select 'expose' and save it with a name. We can now double click on any part of the blank space of the graph to see the material's properties and, if we scroll down, we will find the variable we exposed with the name we saved it. From here we can put on a label, to group variables, write a different name for Unity to display, tweak the maximum and minimum values the can take... as displayed in Figure 4.12.

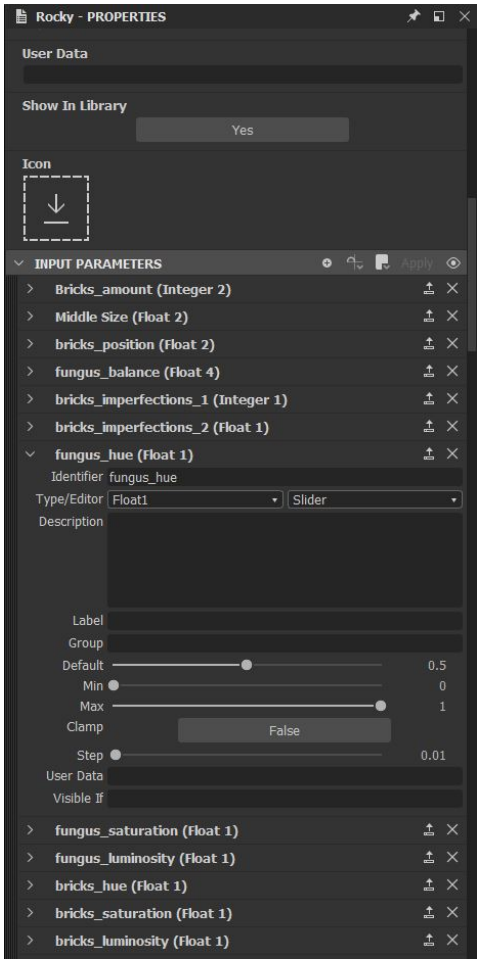

Figure 4.12: Input parameters

Finally, the material is completed and can be exported as an .sbsar file to use in Unity. (I left a lot of other interesting nodes, textures and techniques in here that I used in other materials, but I think this is enough to comprehend the workflow I used in the project).

4.3 Learning Curve in Substance Designer

Now that the process of creating a material has been explained I want to mention the steps and the effort put behind it. If the reader goes back to Figure 4.2 they can notice how the upper part (the one where the wall was created) is quite more chaotic than the one below, where every node seems to be shorted in its corresponding blue box. The reason why it is this way is because the wall was one of the first materials I did while the addition of the arch was done almost at the end of this phase of the project.

It's natural that, if you are inexperienced with a software, the first attempts are slower and not so well executed in comparison to the ones done later, and I think is important to address that.

In Figure 4.13 and in Figure 4.14 the first materials I ever did can be seen. If I compare them now to the ones that actually made it into the demo I realize how much I improved, that is the reason why the former were not included in the final demo, the quality difference was too large and some of them were actually unusable because of the way they were made. For example, the one in Figure 4.14 can't be tiled and that represents a big problem that I wasn't aware of at first.

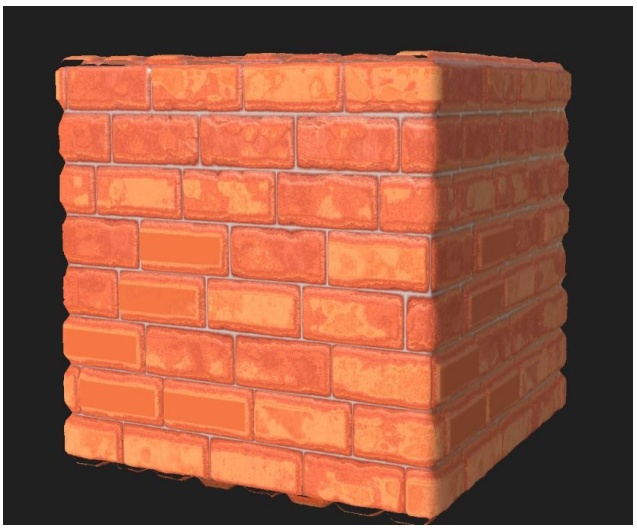

Figure 4.13: My first material.

As I continued to learn how to use the software and the possibilities it has I became aware of how terrible my first attempts actually were. After I realized this, I decided to stop doing materials for a while and to just focus on watching tutorials and reading about the software to take notes of everything that I found interesting or useful. This actually affected my planification a lot, as some materials were reworked while others were totally replaced by different materials as I spent more time into research and taking

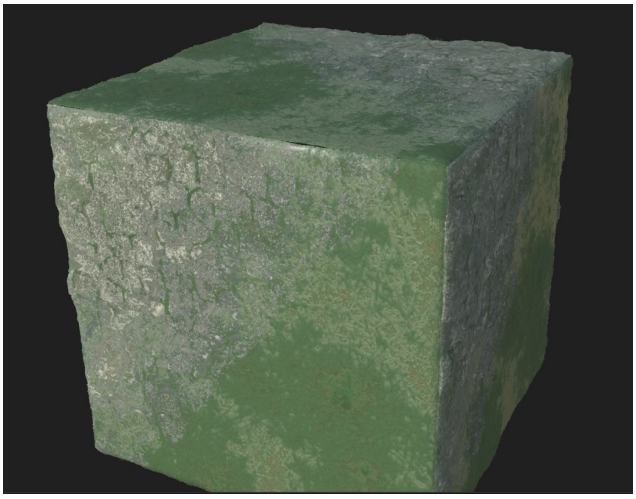

Figure 4.14: My second material.

notes. However, I do not regret this decision, as I think that it made everything look better and that, in the long run, will benefit me.

I took notes about node uses, basic geometry tips, annotations on certain processes that can appear in different materials and so on.

Here some examples of things I wrote down:

- Edge detect: Node that identifies where the limits of the geometry are, if before it you pass the image through a levels node to control the contrats, you can generate a black and white separation of geometry that is perfect to act as masks.
- Slope blur grayscale: The Max mode generates a blur that goes outside the geometry, the Min goes inside it and the Blur one does both. The Min can be used for a eroded material, for example, and the Max to make something bumpy.
- Solution to the Opacity problem explained in the past section.

I am still watching breakdown videos and taking notes to fill my notebook to the brim at this point in the project and I don't plan to stop now. I just hope to improve in the future to be able to do more complicated materials in the future.

4.4 Implementation of the Demos in Unity

Once the materials were done it was time to showcase them in Unity and to show their potential. The first step was to download the Substance in Unity plugin made by Allegorithmic [3] and the Post processing stack made by Unity [9]. After installing them, we can now use Substance Designer materials without a problem.

The next step was to set up the PBR in Unity for good visual results. According to Allegorithmic's tutorial these are the steps to take [4]:

- Select the camera and check HDR (HDR mean high dynamic range and makes sure to reflect the surroundings of the skybox) HDR is a 32 bit .EXR format which contains more data per pixel than the average .jpg.
- Go to edit → project settings → quality and disable anti aliasing:
- Go to edit → project settings → graphics and set all TIERS to to deferred rendering.
- Go to edit → project settings → player and set the color space to Linear.
- Add a cubemap. In my case I used one from the asset store by proassets called Free HDR Sky [7].
- Create a new Post processing behaviour and attach it to the camera, tweak the parameters until you have a satisfactory result.

4.4.1 First Demo

The first demo was intended to show the assests in four rooms and I based them on different themes: the Victorian, Castle, Modern and Rustic ones with the use of cube and planes and assigned a material to each figure.

This was where the exposed variables paid off. Let me explain, let's say a room has three cubes (Unity basic primitive), next to each other in the same axis, to form a single, large wall. If a room needs four walls to be complete, to have twelve cubes with the same texture would be obvious. If every cube had the same exact material it would cause a repetition that would be really apparent to the naked eye. To avoid this, Substance for Unity allows to make copies of the materials and to tweak the variables, even having a 'randomize' button that changes the variables for you with just one click. Thanks to this, the scene gains a lot of realism as each cube has slightly different changes that differentiates it from the others, making the scene a lot more natural.

After this and for the first demo, I created a 3D capsule and coded a simple script that allows the player to move around using the WASD keys and a first person camera that follows it.

The camera is just used to allow the player to look around him, so it rotates accordingly to the vertical and horizontal speed of the mouse.

The capsule movement reads when the WASD keys have been pressed and performs a translation of a speed value (set to 0.1) in the corresponding axis.

Figure 4.15: Example of a floor made of 9 cubes with the same source material but cloned 9 times to give it different parameters.

And the last code for this demo was the creation of a script that, when it detects the player approximating, rotates the door to open it so the player can enter a new room.

This demo was done to show the potential of the material done in Substance Designer and how good they can look in Unity. However, this demo doesn't allow the tweaking of parameters nor the transformation of materials in running time, to say, they are not procedural as the strictest definition of the word.

4.4.2 Second Demo

So, to show the changes of the materials in running time, a second demo was created. In this one there are less effects and the textures have worse resolution to allow the real time rendering without my computer dying. This is also the demo where the interface showed before in the document is from (page 15).

This part of the project only consist of 14 cubes, each one with a different material I created, an interface, and a fixed camera.

The first step was to create an script that references to the materials in the scene with variables of type 'Substance.Game.SubstanceGraph'. With this references, the script can access every parameter that that material has, so here is where the functions to change

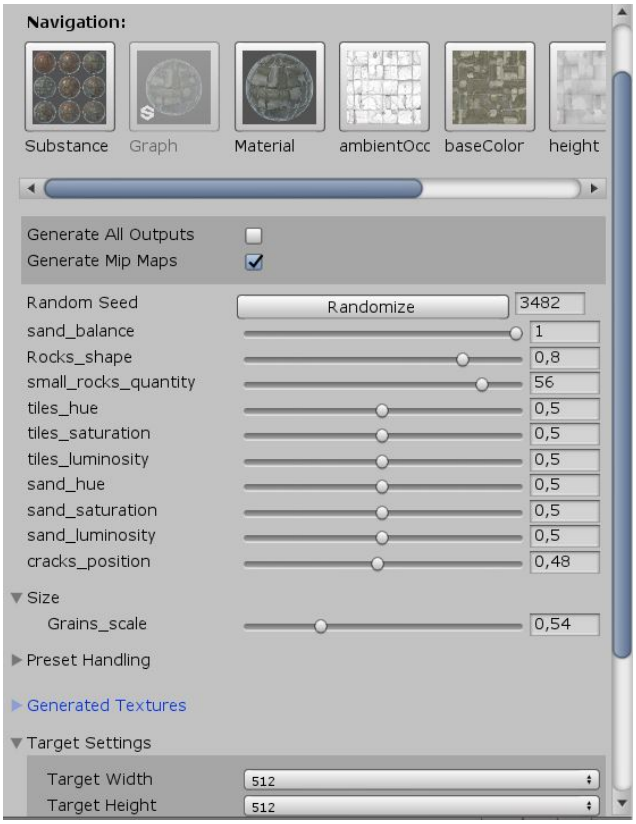

Figure 4.16: Material menu where the variables and material resolution can be changed.

them are, one for every exposed variable the material has. These are called when a slider that references that function has its value changed. Once this is done, there's a call to render the modified material again to show the changes in the scene.

After that, I created the sliders needed for the last script to work. I created a menu for each material that showed a list of the individual parameters that can be changed, with a slider under it. Every slider have a min, a max, a current value and accept only integer or decimal numbers according to the ones in Substance Designer to avoid errors. The sliders were then assigned to the corresponding function in the latter script using the 'On value changed (Single)' that the Unity editor offers.

The menus were done but they only need to be visible when the corresponding material is pressed, not all of them at the same time.

To achieve this, I tagged every cube differently and created a script that performs a raycast from the camera to where the mouse is clicking and returns the tag of the object it crossed, if there is one.

I created fourteen other scripts, one for each material, that are in charge of storing all the children of the menu (slider and text) into a list. Then, I created two public methods, disableMenu that disables all the childrens in the list, and enableMenu that enables all the children in the list.

With this, I created a reference for each of the last fourteen scripts in the raycast one so, every time the user clicks and the tag changes, the script displays the menu corresponding to the selected material and hides all the other.

Finally, both demos for Unity were completed. They and can be selected by pressing the 1 and 2 bottoms of the keyboard.

4.5 Results

I am pretty happy about the knowledge I gained from this project and with the results I've obtained. To specify a bit more:

Substance Designer

- I learnt a lot about how the software works and about how to use its tools. I hope to learn even more in the future and reach a higher level of proficiency in it from this point, but this project surely helped me at gaining experience. If you are interested, you can find the renders of all the materials in the Appendix, as they are displayed in Substance Designer (better quality than Unity) and with the corresponding credits, if they were made following a tutorial.
- I created various materials that are unique between them but I actually made less than I expected at first, as I underestimate the effort it takes to learn how to mix the nodes and how to exploit all their potential.
- My materials were all procedural with exposed parameters, so this objective was accomplished moreover, everyone of them could be exported to Unity so it all turned well.

Unity

- Thanks to Allegorithmic's video tutorials and documentation, I learnt the correct PBR implementation and the use of their plugin without difficulties.
- I created a demo that set up the materials in a compelling way, creating rooms for a house and that the user can navigate making use of the 'WASD' keys and a first person camera.

- I created a second demo that allows the user to interact with the materials at run time, making use of an interface that changes depending on the material that they want to modificate.

The materials done here that are not based on a tutorial will be later displayed in my artstation, to add to my portfolio. However, I don't plan to release the Unity project anywhere, as it only serves to show the materials but at a lower quality due to the limitations of Unity.

I also hope that this document can be used for the alumni or people that want to start with Substance Designer and use their work in their own projects, as I hope I have explained simply and clearly how it works and how the materials can be implemented in their own games.

CHAPTER 5

Conclusions and future work

Contents

5.1 Conclusions

I really enjoyed working in this project, as I got to learnt a new skill and discover the amazing community that is behind the software I used. I would have liked to spend more time doing materials instead of coding the different demos but I understand that coding is a requirement of this degree.

I would have liked to extend the explanations in the materials, talking about more tricks and uses I discovered and wrote down in my notebook, but I was afraid to bore the reader, as it is quite the technical stuff and it's hard to explain notes made for me to someone that hasn't used Substance Designer.

About LaTeX, the software used to write this document. At first I found it really difficult to work with it, as we have never used it in the degree and there was no documentation in the aula virtual of the subject. I would have appreciated a bit of insight on how to work with the software as it really is great to use, maybe a seminary or master class with the basics. I know of a lot of people that passed the opportunity to work with it and did a Word document instead for such reasons so I think that all of us would be grateful.

Overall I am satisfied with this project and hope that it will give a new insight into a subject that we don't get to touch in our degree and, in my opinion, it's a pity.

5.2 Future work

I do plan to continue this project, as there are still a lot of tools and nodes that I still want to experiment with. I will create more materials but I don't think I will continue adding them to the demo, as the latter was done as a mean to show the potential of Unity and S. Designer together, and thus needs no more content.

Bibliography

[1] 3dEx. 3DEX TUTORIALS. https://www.youtube.com/user/irvin390. Accessed: 2019-05-28.

[2] Allegorithmic. SUBSTANCE DESIGNER PRICE. https://www.substance3d.com/buy/download. Accessed: 2019-05-28.

[3] Allegorithmic. SUBSTANCE IN UNITY PLUGIN. https://assetstore.unity.com/packages/tools/utilities/substance-in-unity-110555. Accessed: 2019-05-28.

[4] Allegorithmic. SUBSTANCE IN UNITY TUTORIAL BY ALLEGORITHMIC. https://academy.substance3d.com/courses/substance-in-unity/youtube-Dy177PrxRfs. Accessed: 2019-05-28.

[5] Allegorithmic. THE PBR GUIDE BY ALLEGORITHMIC - PART 1. https://academy.substance3d.com/courses/the-pbr-guide-part-1. Accessed: 2019-05-28.

[6] Allegorithmic. THE PBR GUIDE BY ALLEGORITHMIC - PART 2. https://academy.substance3d.com/courses/the-pbr-guide-part-2. Accessed: 2019-05-28.

[7] ProAssets. HDR SKY. https://assetstore.unity.com/packages/2d/textures-materials/sky/free-hdr-sky-61217. Accessed: 2019-05-28.

[8] Sharpstance. SHARPSTANCE TUTORIALS. https://www.youtube.com/channel/UCvrzoT49CiRy1hvgb
Accessed: 2019-05-28.

[9] Unity. POST PROCESSING STACK PLUGIN. https://assetstore.unity.com/packages/essentials/post-processing-stack-83912. Accessed: 2019-05-28.

[10] Unity. UNITY PRICE. https://store.unity.com/es. Accessed: 2019-05-28.

Appendix A

Renders

A.1 Appendices

To finish the document, I will show the renders of the materials done in Substance Designer. They are shown as displayed in that software, as the quality is far superior to the one in Unity so the readers can see how they really look. To finish the document, I will show the renders of the materials done in Substance Designer. They are shown as displayed in that software, as the quality is far superior to the one in Unity so the readers can see how they really look. Credits when pertaining for materials done while following a tutorial.

A.2 Link to Demo and Video

Link to the google drive folder where the project .exe and a video to show it is:

https://drive.google.com/drive/folders/1rya40uYruBvaQkPA4I2Kxeji6yRpxu-p?usp=sharing

Figure A.1: Grass, made following the 3dEx youtube's channel tutorial [1]

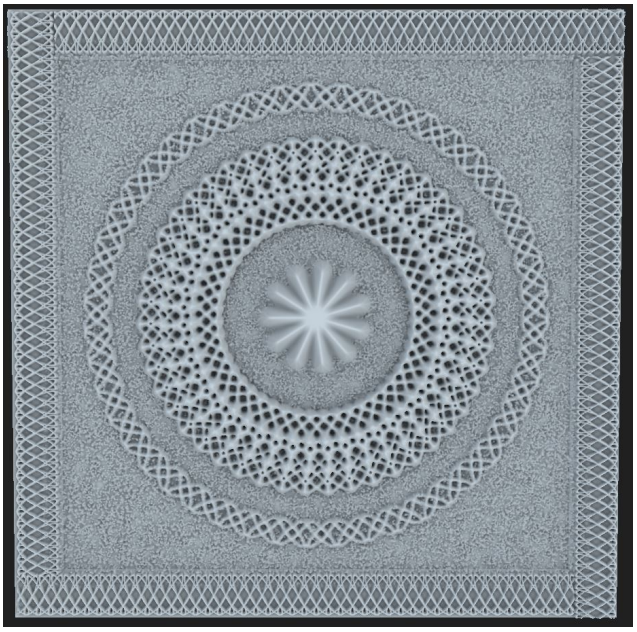

Figure A.2: Modern ceiling, original

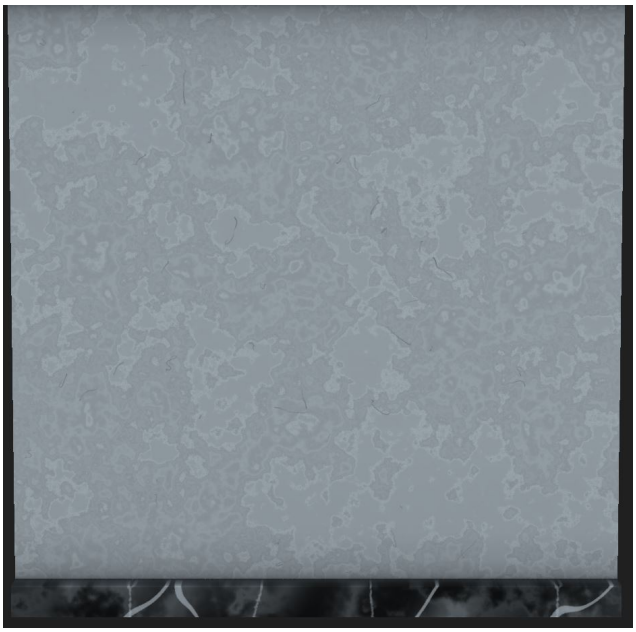

Figure A.3: Modern wall, original

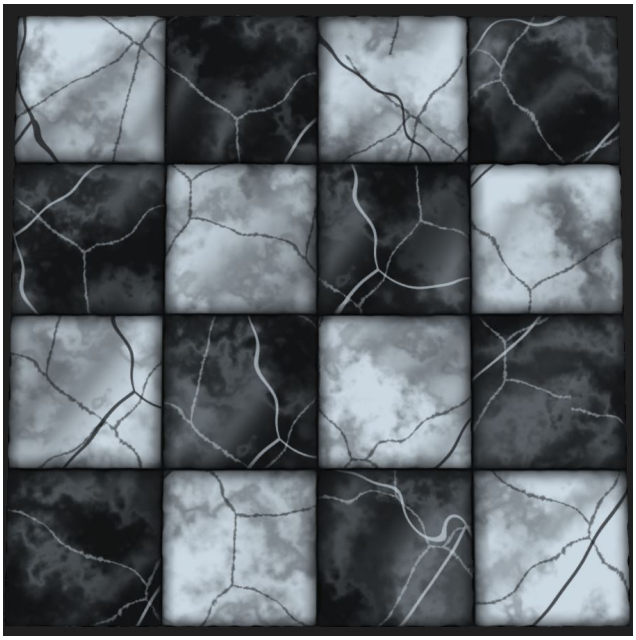

Figure A.4: Modern floor, original

Figure A.5: Rustic wall, original

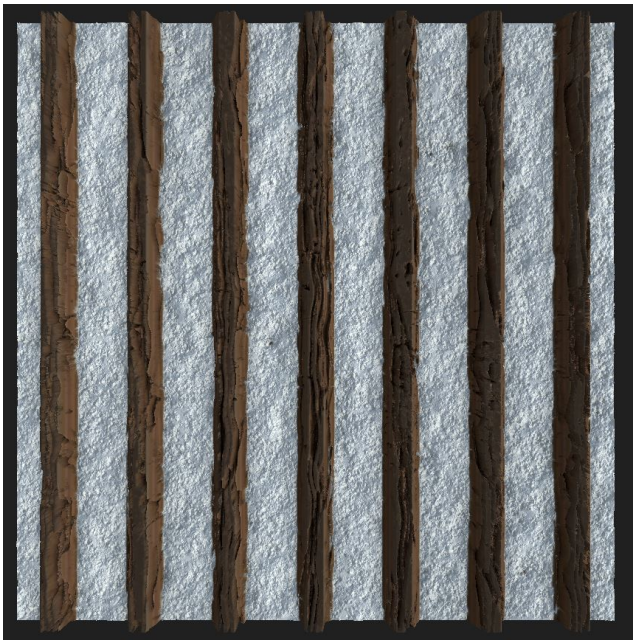

Figure A.6: Rustic ceiling, original

Figure A.7: Rustic floor, based on Mark Schipper's teachings

Figure A.8: Victorian wall, original

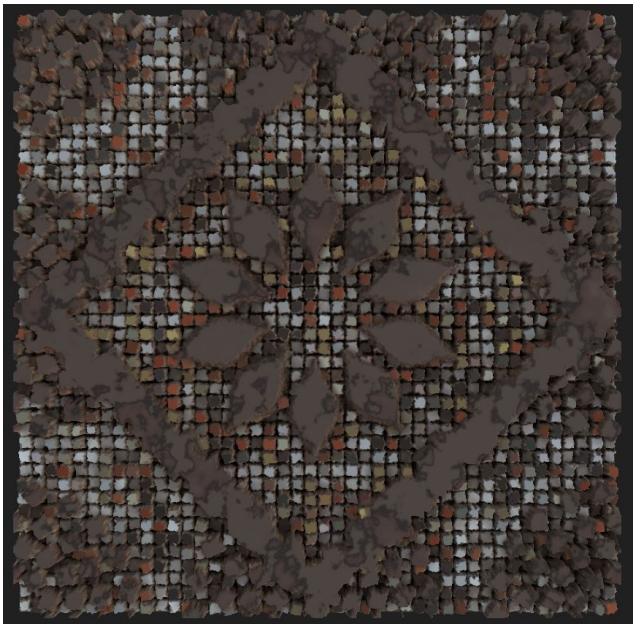

Figure A.9: Victorian floor, made following the sharpstance youtube's channel tutorial [8]

Figure A.10: Castle floor, made following the sharpstance youtube's channel tutorial [8]

Figure A.11: Castle wall + arch, original

Figure A.12: Door pane, original

Figure A.13: Door frame, original

Figure A.14: Window, original