

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Computer Graphics and Interaction**

Materials Library in Unity

Timushev Fedor

**Supervisor: Ing. Jiří Bittner, Ph.D.
May 2023**

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Timushev** Jméno: **Fedor** Osobní číslo: **496153**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávací katedra/ústav: **Katedra počítačové grafiky a interakce**
Studijní program: **Otevřená informatika**
Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Knihovna materiálů v Unity

Název bakalářské práce anglicky:

Materials Library in Unity

Pokyny pro vypracování:

Zmapujte problematiku simulace vzhledu materiálů se zaměřením na fyzikálně založené modely. Popište podporu vytváření materiálů pomocí shader grafů. Zmapujte možnosti vytvoření specializovaných dynamicky se měnících materiálů. Pomocí shader grafu vytvořte vzorkovník nejméně deseti často používaných materiálů (barevné plasty, různé typy dřeva, měď, zlato, stříbro, hliník, sklo, diamant, apod.) a nejméně deseti dynamicky se měnících komplexních materiálů jako je vodní plocha, tráva ve větru, zářící tekutina, apod. Zaměřte se na kvalitní simulaci detailů s využitím textur detailů, šumových funkcí, normálových map a výškových map. Na několika příkladech popište proces vytváření procedurálních materiálů pomocí shader grafu a pokuste se shrnout důležitá obecná doporučení. Vymodelujte testovací scénu, kde bude možné materiály z vytvořeného vzorkovníku jednoduše přepínat.

Seznam doporučené literatury:

- [1] Ebert, David S., et al. Texturing & modeling: a procedural approach. Morgan Kaufmann, 2003.
- [2] Burley, Brent, and Walt Disney Animation Studios. 'Physically-based shading at Disney.' ACM SIGGRAPH. Vol. 2012. vol. 2012, 2012.
- [3] Schmidt, T. W., Pellacini, F., Nowrouzezahrai, D., Jarosz, W., & Dachsbacher, C. (2016, February). State of the art in artistic editing of appearance, lighting and material. In Computer Graphics Forum (Vol. 35, No. 1, pp. 216-233).
- [4] Materials in Unity. Online: <https://docs.unity3d.com/Manual/Materials.html>
- [5] Tomáš Cicvárek. Materials in Computer Graphics. Bakalářská práce, ČVUT FEL, 2021.
- [6] Jakub Kyselka. Vzhled materiálů v Unity. Bakalářská práce, ČVUT FEL, 2022.
- [7] Boulanger, K., Pattanaik, S. N., & Bouatouch, K. (2008). Rendering grass in real time with dynamic lighting. IEEE Computer Graphics and Applications, 29(1), 32-41.
- [8] Fan, Z., Li, H., Hillesland, K., & Sheng, B. (2015). Simulation and rendering for millions of grass blades. In Symposium on Interactive 3D Graphics and Games (pp. 55-60).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jiří Bittner, Ph.D. Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.02.2023**

Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Jiří Bittner, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to extend my gratitude to my supervisor, Ing. Jiří Bittner, for his unwavering guidance and advice throughout the creation of this bachelor's thesis. I am also deeply grateful to my family and friends: Veronika Shirochenkova, Vasily Levitskiy, Daniil Lebedev, Ing. Daria Antonova, Daniil Nikolaev, Irina Kobzhitskaya, Alexandra Sannikova, and Štěpán Meister for their unwavering support during my academic journey. Their encouragement and belief in me have been instrumental in my result.

Declaration

I hereby declare that I have independently conducted and completed all the work presented in this bachelor thesis. I have diligently followed the established rules and guidelines for formatting and citation, ensuring that all sources and references are accurately cited and acknowledged.

Prague, May , 2023

Abstract

This project has covered and described the problems of modeling the appearance of materials, with an emphasis on physically based models. The topics of material creation with shader graphs were also touched, with which specialized dynamically changing materials were created using the capabilities of the shader graph in a broad aspect.

The shader graph tool was used to create a sampling of frequently used materials as well as unique styles, inspired by different games. The work focused on high-quality detail modeling using detail textures, noise features, normal maps, elevations, etc.

The final demo is a test scene where materials from the sample album can easily be swapped, and each material has its own parameters that affect the display of the material in real time and can be accessed by the user at any moment.

Keywords: Rendering, Unity, Shader Graph, Visual Programming, Texturing, Shading

Supervisor: Ing. Jiří Bittner, Ph.D.
Praha 2, Karlovo náměstí 13, E-420

Abstrakt

Tento projekt se zabýval a popsal problematiku modelování vzhledu materiálů s důrazem na fyzikálně založené modely. Dotýká se také témat tvorby materiálů pomocí shader grafů, pomocí nichž byly vytvořeny specializované dynamicky se měnící materiály využívající široké možnosti shader grafu.

Pomocí nástroje shader graph byla vytvořena ukázka často používaných materiálů i unikátních stylů, inspirovaných různými videohrami. Práce se zaměřila na kvalitní modelování detailů pomocí detailních textur, šumových prvků, normálových map, výšek atd.

Závěrečnou ukázkou je testovací scéna, kde lze materiály z ukázkového alba snadno vyměňovat a každý materiál má své vlastní parametry, které ovlivňují zobrazení materiálu v reálném čase a uživatel k nim má kdykoli přístup.

Klíčová slova: Vykreslování, Unity, Shader Graph, Vizuelní programování, Textury, Stínování

Překlad názvu: Knihovna materiálů v Unity

Contents

1 Introduction	1	5.23 Pearl	82
2 Materials in games	3	5.24 Glass	84
2.1 Similarity of approaches	3	6 Conclusion	87
2.2 Basic terms	5	Bibliography	89
2.3 The BRDF model	6		
2.3.1 Applicability	6		
2.3.2 BRDF in Unity	7		
3 Material model and pipeline	9		
3.1 PBR material model in Unity . . .	9		
3.2 Choosing pipeline	10		
4 Material creation	11		
4.1 Approaches in creating materials	11		
4.2 Nodes library in Unity	12		
4.2.1 Artistic	12		
4.2.2 Channel	13		
4.2.3 Input	14		
4.2.4 Math	15		
4.2.5 Procedural	16		
4.2.6 Utility	16		
4.2.7 UV	18		
4.2.8 How to create Shader Graph and material based on it	18		
5 Materials library	19		
5.1 Acid	22		
5.2 Candy	24		
5.3 Clay	26		
5.4 Core	28		
5.5 Dissolve	32		
5.6 Filler	34		
5.7 Flame	36		
5.8 Triplanar projection	40		
5.9 Frost	43		
5.10 Glitter	46		
5.11 Grid	48		
5.12 Halftone	51		
5.13 Hologram	56		
5.14 Liquid	58		
5.15 Minecraft	60		
5.16 Mirage	63		
5.17 PewDiePie	65		
5.18 Soap	67		
5.19 Toon	69		
5.20 Water	71		
5.21 Wood	75		
5.22 Metal	78		

Figures

2.1 Metallic material simulating the structure of carbon fiber reinforced plastic in the Unity Engine [1].	3
2.2 An example of using nodes in the Blender3D graph to apply textures to the different channels of the model (in this case albedo and normal).	4
2.3 Example of using Shader Graph nodes to deform the model mesh (Vertex shader).	5
2.4 Phong Equation (visual illustration) by Brad Smith [12].	7
2.5 A Standard Shader material with default parameters.	8
3.1 As the smoothness of a material's surface increases, the fresnel effect becomes more noticeable at grazing angles in relation to the viewer.	10
4.1 This material is a combination of the following 4 maps (from left to right): color, normal, roughness, height [6].	11
4.2 Example of procedural texture generation [7].	12
4.3 Creating texture with Voronoi and Blend nodes.	13
4.4 An example of using the Normal From Height node.	13
4.5 Examples of using the Split node.	14
4.6 An example of using the Branch node.	16
4.7 The principle of creating a sub graph.	17
4.8 Offset Over Time Sub Graph with its input parameters.	17
5.1 All materials produced as part of this project in Unity application.	19
5.2 "Acid" Material on a variety of objects.	22
5.3 Hazardous waste tanks inside the plant.	22
5.4 "Acid" Shader Graph.	23
5.5 "Candy" Material on a variety of objects.	24
5.6 Candy Cane decorations in Stardew Valley.	24
5.7 "Candy" Shader Graph.	25
5.8 "Clay" Material on a variety of objects.	26
5.9 Claymation animation in Blender 2.81 by J Middleton [19].	26
5.10 "Clay" Shader Graph.	27
5.11 "Core" Material on a variety of objects.	28
5.12 Avatar Korra opens the northern portal to the Spirit World, S02E10.	28
5.13 "Core" Shader Graph.	29
5.14 Section 1 of the "Core" Shader Graph.	30
5.15 Section 2 of the "Core" Shader Graph.	30
5.16 Section 3 & 4 of the "Core" Shader Graph.	31
5.17 "Dissolve" Material on a variety of objects.	32
5.18 "Dissolve" effect on corpses in BioShock Infinite.	32
5.19 "Dissolve" Shader Graph.	33
5.20 "Filler" Material on a variety of objects.	34
5.21 UE4 Transition Effect Shader by TGA Digital [20].	34
5.22 "Filler" Shader Graph.	35
5.23 "Flame" Material on a variety of objects.	36
5.24 Example of using the "Flame" material with a model of a fireplace. Models by Rocco Giandomenico (logs) [21], Michalina "Miszla" Gąsienica-Laskowy (fireplace) [22].	36
5.25 "Flame" Shader Graph.	37
5.26 Section 1 of the "Flame" Shader Graph.	38
5.27 Section 2 of the "Flame" Shader Graph.	38
5.28 Section 3 of the "Flame" Shader Graph.	39
5.29 Triplanar Projection feature in Adobe Substance Painter. [23]	40

5.30 Triplanar maps over the surface from three different axes (X, Y, Z) by Martin Palko.	40	5.53 “ColorValueReduce“ Shader Graph.	53
5.31 Triplanar mapped on a surface (left) and maps blending view (right) by Martin Palko.	41	5.54 “Halftone“ Shader Graph.	54
5.32 Example code represents the possible outcome of Triplanar Texture node (Default mode).	41	5.55 Section 1 of the “Halftone“ Shader Graph.	55
5.33 Triplanar Simple Noise sub graph.	42	5.56 Section 2 of the “Halftone“ Shader Graph.	55
5.34 “Frost“ Material on a variety of objects.	43	5.57 “Hologram“ Material on a variety of objects.	56
5.35 Frostpunk artwork by 11-bit Studios [26].	43	5.58 Cortana as a hologram in Halo 5.	56
5.36 “Frost“ Shader Graph.	44	5.59 “Hologram“ Shader Graph.	57
5.37 Section 1 of the “Frost“ Shader Graph.	44	5.60 “Liquid“ Material on a variety of objects.	58
5.38 Section 2 of the “Frost“ Shader Graph.	45	5.61 Liquid in the bottle in Half-Life: Alyx (waves and the movement of the liquid depends on the movement of the bottle).	58
5.39 Section 3 of the “Frost“ Shader Graph.	45	5.62 “Liquid“ Shader Graph.	59
5.40 “Glitter“ Material on a variety of objects.	46	5.63 “Minecraft“ Material on a variety of objects.	60
5.41 Galaxy by rawpixel.com [27]. ...	46	5.64 Grass block in Minecraft.	60
5.42 “Glitter“ Shader Graph.	47	5.65 “Minecraft“ Shader Graph.	61
5.43 “Grid“ Material on a variety of objects.	48	5.66 Sections 1 & 2 of the “Minecraft“ Shader Graph.	61
5.44 Retrowave Neon 80’s Background by Rafael-De-Jongh [28].	48	5.67 Section 3 of the “Minecraft“ Shader Graph.	62
5.45 “Grid“ Shader Graph.	49	5.68 Section 4 of the “Minecraft“ Shader Graph.	62
5.46 Sections 1 & 2 of the “Grid“ Shader Graph.	50	5.69 “Mirage“ Material applied on a plane near the camera, so the whole view is distorted.	63
5.47 Sections 3 & 4 of the “Grid“ Shader Graph.	50	5.70 Heat Haze near plane engines by Scott Barbour [31].	63
5.48 “Halftone“ Material on a variety of objects.	51	5.71 “Mirage“ Shader Graph.	64
5.49 Halftone effect example (source image on the left, final transition on the right) [29].	51	5.72 “PewDiePie“ Material on a variety of objects.	65
5.50 Outline effect in Left4Dead game is used to distinct player’s allies. ...	52	5.73 PewDiePie’s YouTube channel pattern. Used as a YouTube banner image, as a screen saver while waiting for the start of the stream, and as a recognizable texture for branded merchandise [33].	65
5.51 “Outline“ Shader Graph.	52	5.74 “PewDiePie“ Shader Graph.	66
5.52 GetMainLight custom function node source.	53	5.75 “Soap“ Material on a variety of objects.	67

5.76 Group of soap bubbles with good visible film [35].....	67	5.101 “Glass“ Material on a variety of objects.....	84
5.77 “Soap“ Shader Graph.....	68	5.102 Photo of the distorted glass by Image*After [44].	84
5.78 “Toon“ Material on a variety of objects.....	69	5.103 “Glass“ Shader Graph.	85
5.79 Cel shading in The Legend of Zelda: Breath of the Wild.....	69		
5.80 “Toon“ Shader Graph.	70		
5.81 “Water“ Material on a variety of objects.....	71		
5.82 Water surface in Zelda: Wind Waker.	71		
5.83 “Water“ Shader Graph.....	72		
5.84 Section 1 of the “Water“ Shader Graph.	73		
5.85 Section 2 of the “Water“ Shader Graph.	73		
5.86 Section 3 of the “Water“ Shader Graph.	74		
5.87 “Wood“ Material on a variety of objects.....	75		
5.88 Photo of the wooden surface from close up by FWStudio [36].	75		
5.89 “Wood“ Shader Graph.....	76		
5.90 Section 1 of the “Wood“ Shader Graph.	76		
5.91 Section 2 of the “Wood“ Shader Graph.	77		
5.92 “Metal“ Material on a variety of objects (steel, copper, titanium)...	78		
5.93 Examples of heat coloring on metals: steel by Jeffrey H Dean [39], copper by Mari [40], titanium by Robert Lopez [41].	78		
5.94 “Metal“ Shader Graph.....	79		
5.95 Section 1 of the “Metal“ Shader Graph.	80		
5.96 Section 2 of the “Metal“ Shader Graph.	80		
5.97 Section 3 of the “Metal“ Shader Graph.	81		
5.98 “Pearl“ Material on a variety of objects.....	82		
5.99 Different types of pearls by GIA.	82		
5.100 “Pearl“ Shader Graph.	83		

Tables

5.1 Table of materials divided into categories.	21	5.24 “Wood“ Shader Graph input parameters.	76
5.2 “Acid“ Shader Graph input parameters.	23	5.25 “Metal“ Shader Graph input parameters.	79
5.3 “Candy“ Shader Graph input parameters.	25	5.26 “Pearl“ Shader Graph input parameters.	83
5.4 “Clay“ Shader Graph input parameters.	27	5.27 “Glass“ Shader Graph input parameters.	85
5.5 “Core“ Shader Graph input parameters.	29		
5.6 “Dissolve“ Shader Graph input parameters.	33		
5.7 “Filler“ Shader Graph input parameters.	35		
5.8 “Flame“ Shader Graph input parameters.	37		
5.9 Triplanar sub graph input parameters.	42		
5.10 “Frost“ Shader Graph input parameters.	44		
5.11 “Glitter“ Shader Graph input parameters.	47		
5.12 “Grid“ Shader Graph input parameters.	49		
5.13 “Outline“ Shader Graph input parameters.	52		
5.14 “ColorValueReduce“ Shader Graph input parameters.	54		
5.15 “Halftone“ Shader Graph input parameters.	54		
5.16 “Hologram“ Shader Graph input parameters.	57		
5.17 “Liquid“ Shader Graph input parameters.	59		
5.18 “Minecraft“ Shader Graph input parameters.	61		
5.19 “Mirage“ Shader Graph input parameters.	64		
5.20 “PewDiePie“ Shader Graph input parameters.	66		
5.21 “Soap“ Shader Graph input parameters.	68		
5.22 “Toon“ Shader Graph input parameters.	70		
5.23 “Water“ Shader Graph input parameters.	72		



Chapter 1

Introduction

Throughout the history of the game industry, visual appearance has been an integral part of game creation. Human perception is based not only on gameplay but also on memorable images. Behind the creation of each game object, there is an idea, a design, and its final appearance within the created world. This work will address the problem of passing visual information about the object by giving it certain graphical parameters, or in other words - the use of materials.

Creating materials and textures for games can be a challenging task. The demands on the quality and performance of materials are high, as they play a crucial role in the overall visual appearance of a game. Materials need to not only look realistic and visually appealing but also be optimized for performance to ensure a smooth gaming experience. In addition, materials often need to be easily adjustable and flexible, allowing game developers to quickly make changes and iterate on the design.

Another problem that game developers face when creating materials is the need to adhere to strict memory and file size limits. Materials and textures can quickly eat up a significant portion of a game's memory and storage, making it important to find a balance between high-quality visuals and efficient use of resources.

This work explores various techniques for creating materials in Unity, including both traditional techniques and the use of procedural methods. It will cover the tools and features available in the Unity engine for material creation, and discuss practices for designing materials for use in games.

Chapter 2

Materials in games

What is the material itself? Material is a set of properties responsible for the appearance of a geometric object: it's color spectrum, illumination, surface grain, shading, reflection, refractive power, and other visualization properties.

For example, in the real world, every object reacts differently to light. Steel objects often shine better than, for example, rough objects (as an example, compare the reflection from metal and wood). Therefore, the material properties must be determined specific to each surface. A quality and correctly created material (such as the one in the **Figure 2.1**) can greatly improve the visual aspect and convey the parameters of the surface.

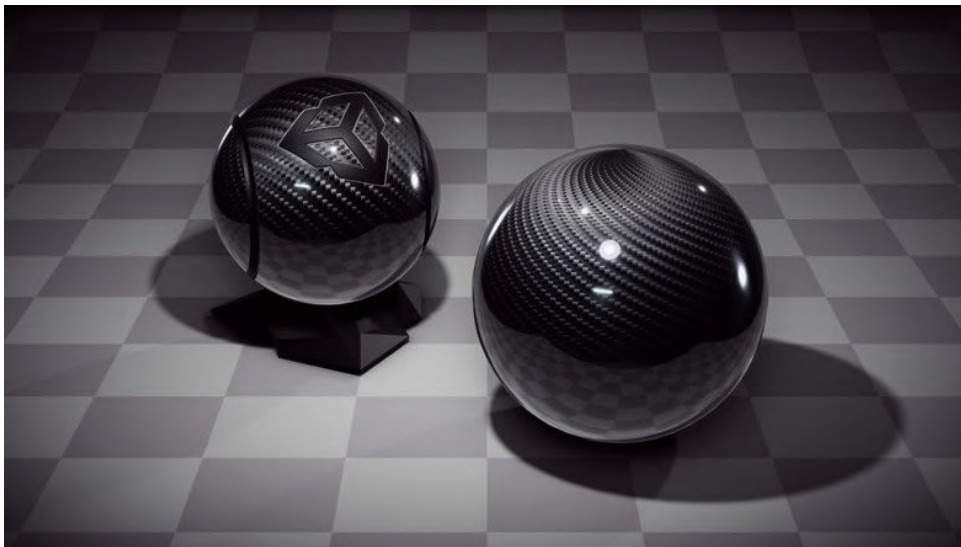


Figure 2.1: Metallic material simulating the structure of carbon fiber reinforced plastic in the Unity Engine [1].

2.1 Similarity of approaches

On the programming side, materials in graphics and game engines are collections of numerical characteristics, textures, and program code (often referred to as a “shader”) that processes this data. In OpenGL, one of the oldest

graphics languages, materials are specified as data structures that store surface properties such as color, reflectivity, and texture. This method of storing materials has been practiced in OpenGL for a long time. While OpenGL has the ability to store data as uniform values, storing it as a structure makes it more organized and easier to work with.

```
#version 330 core
struct Material {
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};

uniform Material material;
```

Creating a compact storage of materials as a set of data that is always available to the user is a very practical feature of graphics and game engines. This is why many engines use a “binding” system for working with materials, which includes an external material editor and an internal manager that implements a subsystem for managing properties and their direct processing, with the resulting material applied to a graphics object.

For example (**Figure 2.2**), Blender 3D uses a shader editor with components called “nodes” that allow users to specify the properties and behavior of a material. Similarly, Unity’s material editor is called Shader Graph, which also uses nodes to represent the different aspects of a material (**Figure 2.3**).

By using a node-based system, users can easily see the relationships between different material properties and adjust them as needed. You can see the similarity in the input parameters of these systems, so if you are familiar with one, it should be easy to understand the other.

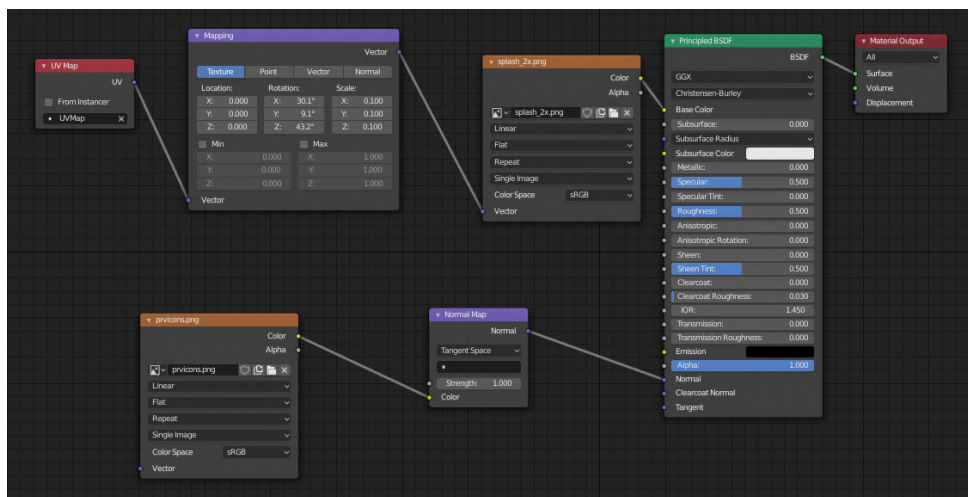


Figure 2.2: An example of using nodes in the Blender3D graph to apply textures to the different channels of the model (in this case albedo and normal).

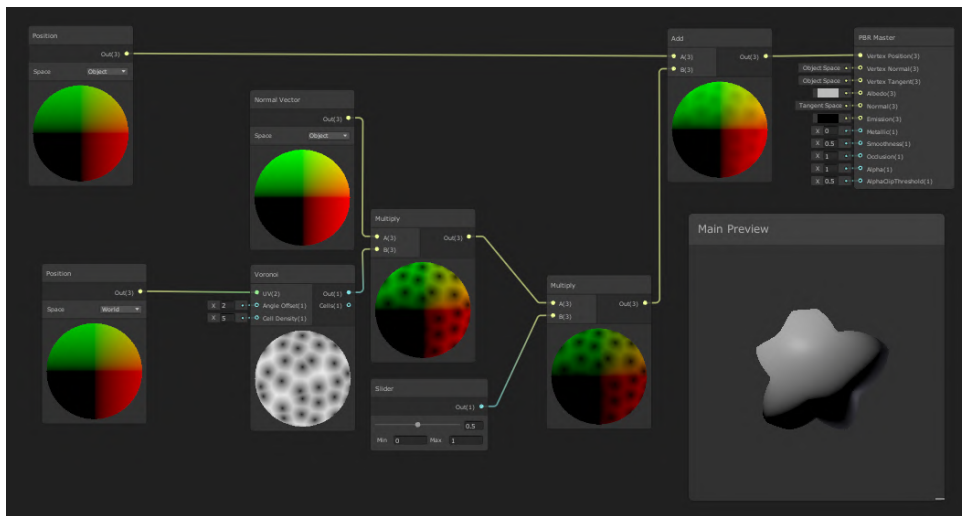


Figure 2.3: Example of using Shader Graph nodes to deform the model mesh (Vertex shader).

2.2 Basic terms

Before delving into the Unity material creation system (the main goal of this project), it's worth getting familiar with some basic terms and concepts related to materials in 3D graphics.

- **UV mapping** is the process of projecting a 3D model's surface onto a 2D plane, with the U and V coordinate corresponding to the axes of the 2D texture. This flat representation of the 3D model is used for texture mapping, and the process of creating a UV map is called UV unwrapping. UV unwrapping is typically done using 3D modeling software. The letters U and V denote the coordinates in the 2D texture axes, as X, Y, and Z are used in the 3D model space.
- A **texture**, or texture map, is an image that is applied to the surface of a 3D model to add detail and realism. Textures can be used to simulate the appearance of various surface characteristics, such as shape, density, and other visual details.
- The **albedo** of material represents its pure color information. It is used to define the base color of the material and can be adjusted to achieve a desired visual effect.
- The **normal map** of material is used to simulate more detailed surface features, such as bumps and dents, by spoofing the height and depth information. This allows the surface to appear more detailed and realistic, even when using a low-polygon 3D model.
- The **metallic** value of a material represents how it reacts to light, with higher values producing more metallic-looking materials and lower values

producing more diffuse materials. This property is often used to create metallic or reflective surfaces, such as metal or glass.

- The **specular** value of a material represents the strength, color, and reflections of light sources that should appear on the material. Higher specular values will produce shinier materials, while lower values will produce duller materials.
- The **roughness (or smoothness)** of a material determines how light is distributed over its surface, with rougher materials scattering light in more directions and smoother materials reflecting light in fewer directions. This property is often used to control the appearance of specular reflections on a material.
- **Glossiness** is a similar concept to roughness, and it can be used to control the appearance of specular reflections on a material. Higher glossiness values will produce shinier materials with more concentrated specular reflections, while lower values will produce duller materials with more diffuse reflections.
- The **height map** of a material is used to simulate depth information based on the camera's position, and materials with a height map assigned will appear to have occluding surface details. This property is often used to create more realistic and detailed surfaces, such as rough terrain or bumpy surfaces.
- **Ambient occlusion** is a value that represents surfaces on the material that are naturally self-shading or appear darker than the rest of the surface. This property is often used to create more realistic and realistic-looking materials, as it simulates the way that light is occluded in the real world.
- **Emission** determines the areas of the material that should glow and what color the glow should be. This property is often used to create materials that emit light

2.3 The BRDF model

The *Bidirectional Reflectance Distribution Function* (BRDF) is a fundamental concept in computer graphics and game development that governs the behavior of light reflected from surfaces.

2.3.1 Applicability

The BRDF model is crucial in creating realistic lighting effects and producing accurate visual simulations of real-world materials. For instance, in the book “*Real-Time Rendering*” [9], the authors explain the importance of the BRDF

model in producing realistic material rendering in real-time applications such as games.

The BRDF model describes the relationship between the incoming light direction, the surface normal, the observer's position, and the reflected light direction. The model takes into account the surface's material properties, including its diffuse and specular reflection properties, as well as any other surface features such as roughness or texture. So In game development, the BRDF model allows game designers to create realistic lighting effects that mimic the way light behaves in the real world. For example metallic or wet surfaces.

The BRDF model calculates the ratio of outgoing light to incoming light at a given point on a surface, taking into account the direction of the incoming light, the surface normal, and the angle of reflection. Also, it is used to calculate the color and brightness of each pixel in an image based on the properties of the surface and the direction of the light sources.

The BRDF model is typically divided into two components: the *diffuse* component and the *specular* component.

1. The diffuse component is modeled using Lambert's law, which assumes that light is reflected uniformly in all directions from a rough surface.
2. The specular component is typically modeled using either the *Phong* (or *Blinn-Phong*) model [11], which assumes that light is reflected only in the direction of the mirror reflection (visual illustration can be seen below).

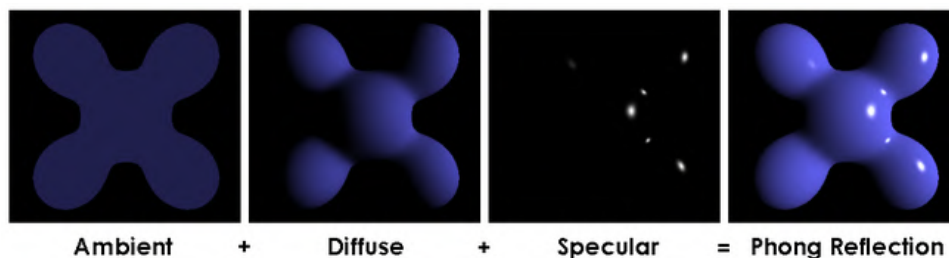


Figure 2.4: Phong Equation (visual illustration) by Brad Smith [12].

2.3.2 BRDF in Unity

The BRDF model is an essential component in creating material in the Unity engine. Unity provides several built-in shader systems that use BRDF models to simulate the behavior of light reflected from surfaces, including Standard Shader, *HDRP* (High Definition Render Pipeline), and *URP* (Universal Render Pipeline), which will be further explained later in **3.2**.

According to the Unity documentation [13], the Standard Shader in Unity uses a BRDF model to simulate metallic and non-metallic surfaces. The HDRP, which is designed for high-end graphics and visual quality, uses a physically-based BRDF model to create photorealistic materials. The URP,

on the other hand, is a lightweight rendering pipeline that uses a simplified version of the BRDF model to render materials efficiently.

The Unity engine also provides various tools for adjusting the material properties of objects. These properties affect the behavior of the BRDF model and enable developers to create a wide range of materials with different properties and textures (the properties for the standard Standard Shader can be seen below in **Figure 2.5**). That's what this project is focused on.

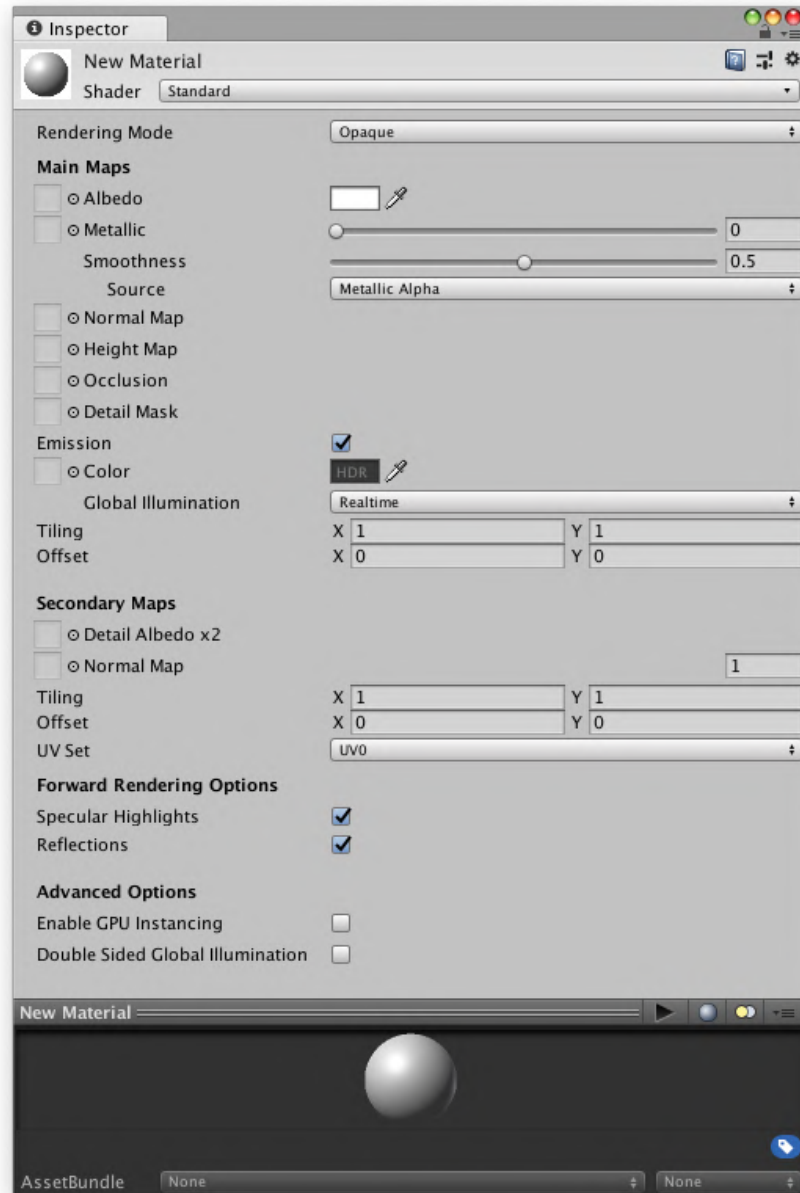


Figure 2.5: A Standard Shader material with default parameters.

Chapter 3

Material model and pipeline

3.1 PBR material model in Unity

Physically-based rendering (PBR) is a technique used to create more realistic 3D models by simulating how light interacts with surfaces. There are two main approaches to PBR, which differ mainly in the way they describe the reflective properties of materials: *PBR Metalness* and *PBR Specular*. The choice of approach typically depends on the specific needs and preferences of the artist or designer. However, both techniques aim to achieve the same goal of producing more physically accurate and visually appealing rendering.

- **PBR Metalness:** albedo, normal, metallic (scalar), roughness
- **PBR Specular:** albedo, normal, specular (vector), glossiness

In PBR Metalness and PBR Specular, there are differences in the way that material's reflective properties are defined. For instance, in PBR Metalness, a metallic map is used to describe how much of the material's color comes from metallic reflection, while a roughness map is used to control the material's smoothness.

In contrast, PBR Specular uses a specular map to define the material's reflectivity, which affects both the color and the intensity of the reflection. Additionally, the way that light interacts with the surface of a material is also different between the two approaches.

Regarding Unity's use of PBR, the engine does indeed offer two different material setups: *Standard* and *Standard (Specular setup)*. These provide different options for how a material's reflective properties are defined and can be chosen. Both of the setups can be seen below in **Figure 3.1**. Note the Metallic parameter on the left (which is controlled either by the metallic map or by a float number from 0 to 1) and Specular on the right (which is controlled again by the map or by the color selected, which is a vector).

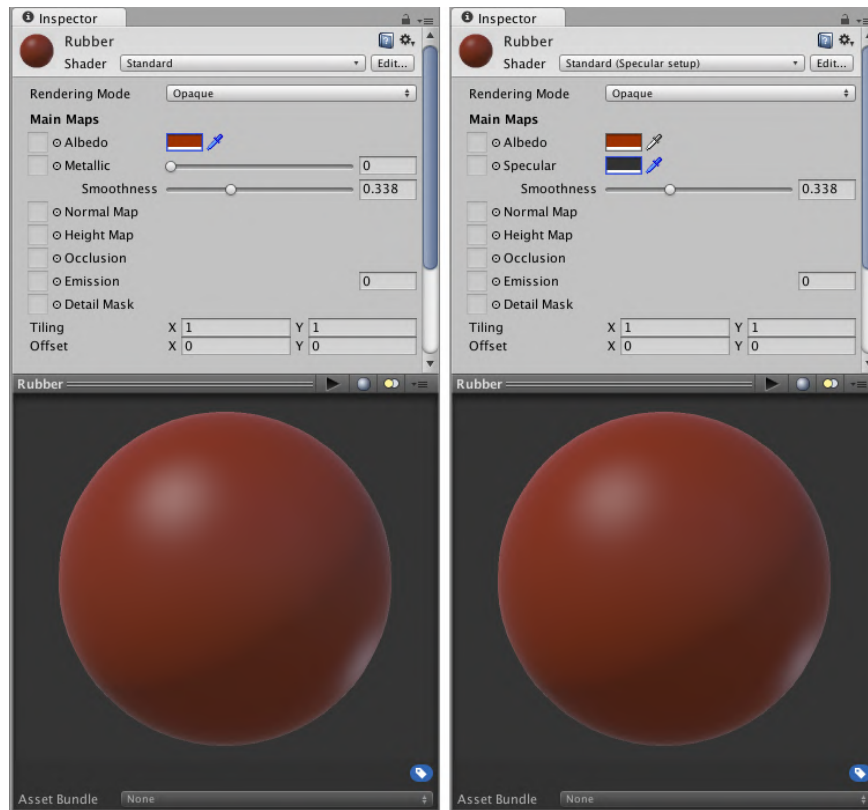


Figure 3.1: As the smoothness of a material's surface increases, the fresnel effect becomes more noticeable at grazing angles in relation to the viewer.

3.2 Choosing pipeline

After choosing a suitable technique (PBR Metallic), it was necessary to choose a suitable rendering pipeline. The choice was made in favor of the Universal Render Pipeline (URP) due to not only experience but also because the URP is designed to optimize the rendering of graphics on all platforms. In contrast, the High Definition Render Pipeline (HDRP) is designed to take advantage of the hardware capabilities for rendering high-definition graphics on powerful platforms.

The URP is an off-the-shelf scripted rendering pipeline created by Unity that provides artist-friendly workflows to quickly and easily create optimized graphics on a variety of platforms (from mobile to PC). The following advantages were considered when choosing the URP for this project:

- It replaces the built-in renderer.
- It supports accelerated rendering.
- It has a visual interface.
- It has a native shader graph.

Chapter 4

Material creation

It is important for game developers to familiarize themselves with the different methods and techniques for creating materials before diving into the process. This can help ensure that materials are not only visually appealing and realistic but also optimized for performance and flexible enough to allow for easy iteration and adjustment. Let's take a look at a few of them.

4.1 Approaches in creating materials

There are two main methods for creating materials in Unity: assigning maps (textures) to input material parameters and using procedural generation.

The first method involves assigning appropriate maps (textures) to the input material parameters, which are then processed by the shader. This allows developers to create materials that are visually appealing and realistic, while also taking advantage of the various tools and features available in Unity for material creation.

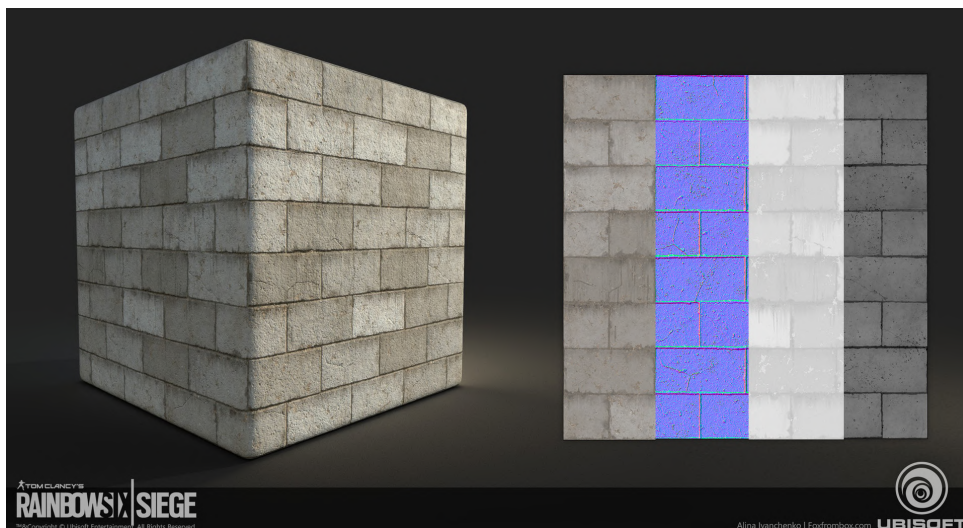


Figure 4.1: This material is a combination of the following 4 maps (from left to right): color, normal, roughness, height [6].

The second method, procedural generation, involves using computer algorithms to generate textures based on a fixed set of parameters. This allows for the creation of unique variations of textures without the need for manual input, making it a useful technique for creating materials that need to be flexible and adjustable. However, it is important to note that procedural textures may not always be as high quality as those created using traditional techniques, so it is important to weigh the benefits and drawbacks of each method when deciding which to use in a given project.

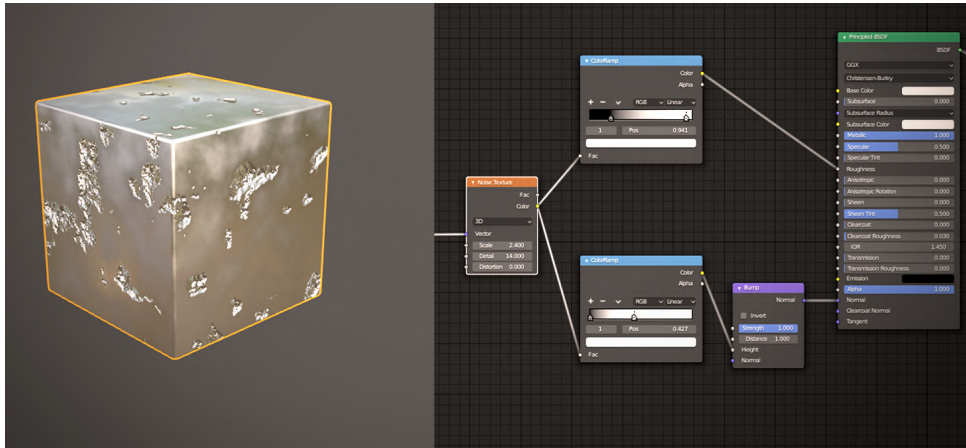


Figure 4.2: Example of procedural texture generation [7].

This project will cover both ways of creating materials.

4.2 Nodes library in Unity

As was mentioned before, the Unity game engine uses the Shader Graph tool to create materials. To create a graph, the nodes are used. So, before going directly to the creation of materials, it is necessary to get familiar with the most commonly used Shader Graph nodes in this project.

In Unity Shader Graph, there are various categories of nodes that can be used to create custom shaders. These nodes are responsible for different tasks and can be combined to achieve the desired effect. Here are the categories of nodes and examples of the most commonly used nodes in this project in each category.

4.2.1 Artistic

Artistic nodes are responsible for adding visual effects that improve the aesthetics of the shader. For example, in this project are used:

- **BLEND** - Blends two input values together based on a specified blend factor.

The Blend is quite a lot of functionality. It is useful not only for working with colors and masks but also for creating complex textures. For example

(**Figure 4.3**), as parameters of the blend were chosen the outermost colors of gray-scale, and Voronoi - as a mask. By changing the mask, it is possible to get fanciful results, such as the one on the right.

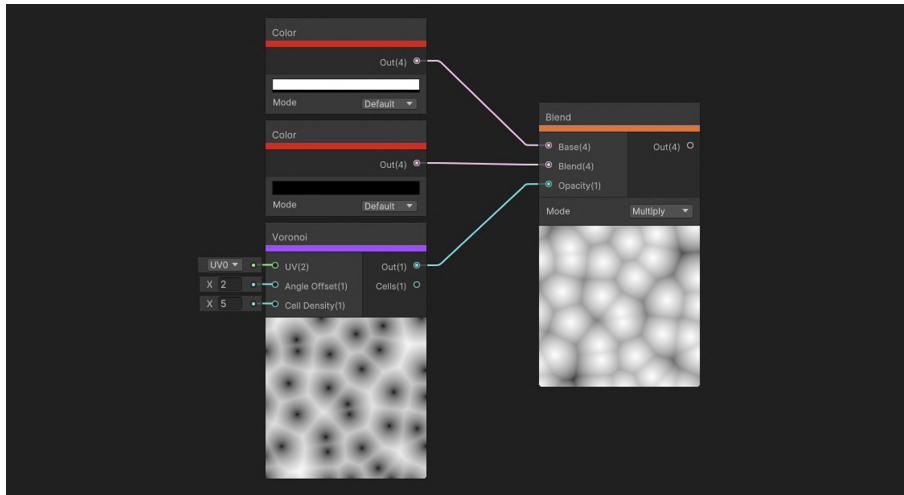


Figure 4.3: Creating texture with Voronoi and Blend nodes.

- **NORMAL FROM HEIGHT** - Converts a height map texture into a normal map texture.

Since the height maps are in gray-scale, any such texture is valid as input for this node. For example, a simple black and white quadrat, as below in **Figure 4.4**.

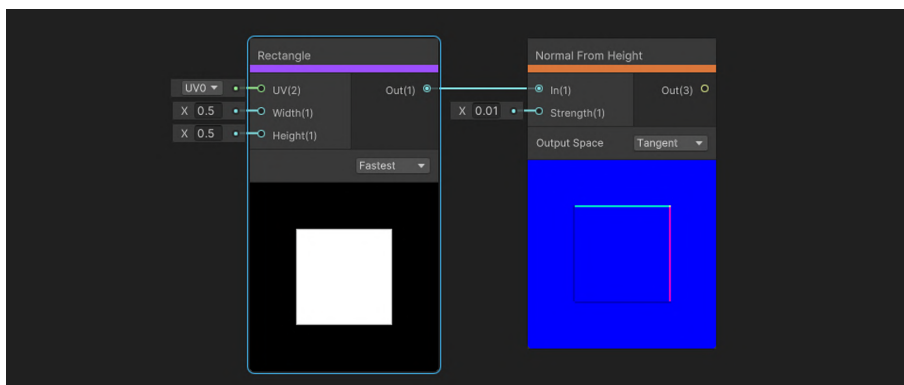


Figure 4.4: An example of using the Normal From Height node.

■ 4.2.2 Channel

Channel nodes work with the RGBA channels of textures or other inputs. For example, in this project is used:

- **SPLIT** - Splits the input vector into RGBA channels (outputs).

For example in the picture below (**Figure 4.5**) the upper part shows an example of the use of different texture channels. It's common practice

to pack different maps/masks (such as height or roughness/metallic) into the texture on different channels for convenience. The bottom part demonstrates creating a simple horizontal/vertical gradient by calling the Split node.

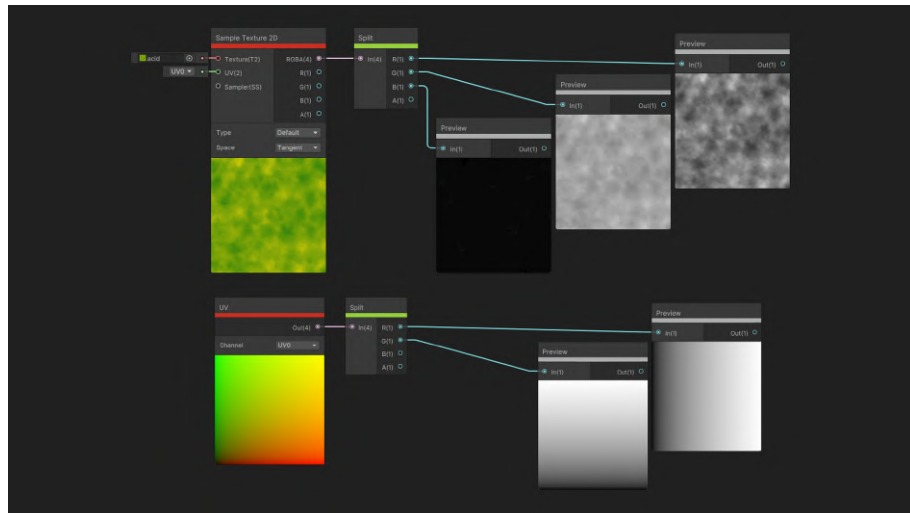


Figure 4.5: Examples of using the Split node.

4.2.3 Input

Input nodes are responsible for getting data into the shader. For example, in this project are used:

- UV - Outputs the UV coordinates of the pixel.
- SAMPLE TEXTURE 2D - Samples a 2D texture at a specific UV coordinate.
- POSITION - Outputs the world space position of a pixel.
- NORMAL VECTOR - Calculates the surface normal at a given point on a mesh or texture.

An example of the use both of vertex vectors Position and Normal to dynamically deform the model mesh using procedural noise can be seen again in **Figure 2.3**.

- TIME - Outputs the current time in seconds.
- CONSTANT - Outputs a constant value (PI, TAU, PHI, E, SQRT2).
- GRADIENT - Creates a gradient between two or more colors.
- SAMPLE GRADIENT - Samples a gradient at a specific point.
- VECTOR2 - Outputs a 2D vector.

■ 4.2.4 Math

Math nodes perform mathematical operations on values. For example, in this project are used:

- ABSOLUTE - Outputs the absolute value of an input.
- ADD - Adds two values together.
- DOT PRODUCT - Calculates the dot product of two vectors.
- MULTIPLY - Multiplies two values together.
- ONE MINUS - Subtracts an input value from one.
- POWER - Raises a value to a specified power.
- SUBTRACT - Subtracts one value from another.
- POSTERIZE - Reduces the number of colors in the image to create a poster-like effect.
- FRESNEL EFFECT - Creates a reflection effect, based on the viewing angle and surface normal of an object.
- LERP - Performs a linear interpolation between two values.
- REMAP - Remaps the range of an input value to a new range.
- STEP - Outputs 1 if an input value is greater than a threshold, and 0 otherwise.
- SMOOTHSTEP - Performs a smooth interpolation between two values.
- TRUNCATE - Truncates the decimal part of an input value.
- SINE - Returns the sine of the giving value.
- SINE WAVE - Returns the sine of the giving value with pseudo-random noise added to the amplitude.
- SATURATE - Clamps the giving value between 0 and 1.
- ROUND - Rounds a given input value to the nearest integer.
- FRACTION - Returns the fractional part of the input value (e.g. the fraction of 1.3 is 0.3).
- NEGATE - Inverts the sign of the input value.
- DISTANCE - Calculates the Euclidean distance between two points in 3D space.
- NORMALIZE - Scales a vector so that it has a length of 1.

1. **“Initial nodes“** - the created pattern of nodes that should be combined into one.
2. **“sub graph“** - the view after combining (select the nodes that needed to be inside, right-click, and then - Create Sub Graph) as a separated graph.
3. **“In another graph“** - the sub graph looks like a separate node now in any other graph in the same project.

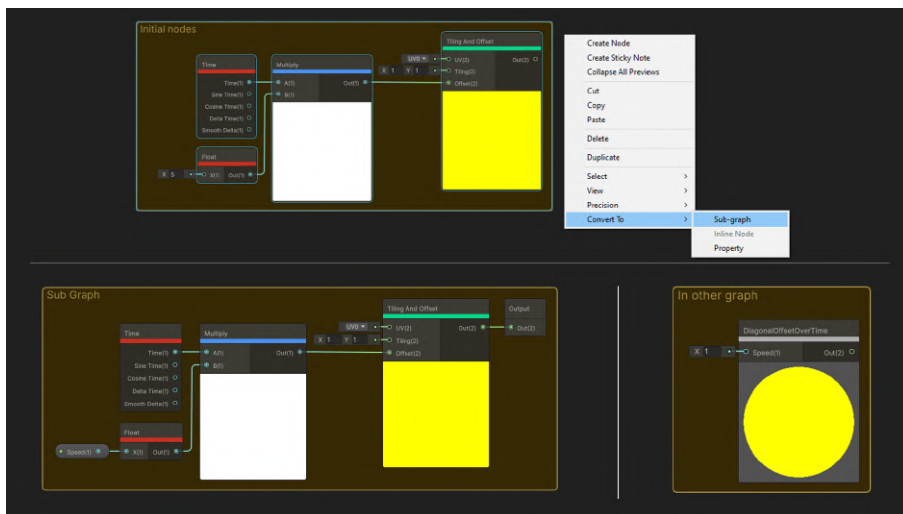


Figure 4.7: The principle of creating a sub graph.

Further on, some materials will use the following sub graph - “Offset Over Time“ (**Figure 4.8**). It is created to speed up the work but will be applied to materials where its use is evident and will reduce the total size of the graph. Its job is to shift the UV coordinates depending on the given speed and direction in the input parameters. Thus, it is possible to quickly interact with the shift which is the main advantage of sub graphs.

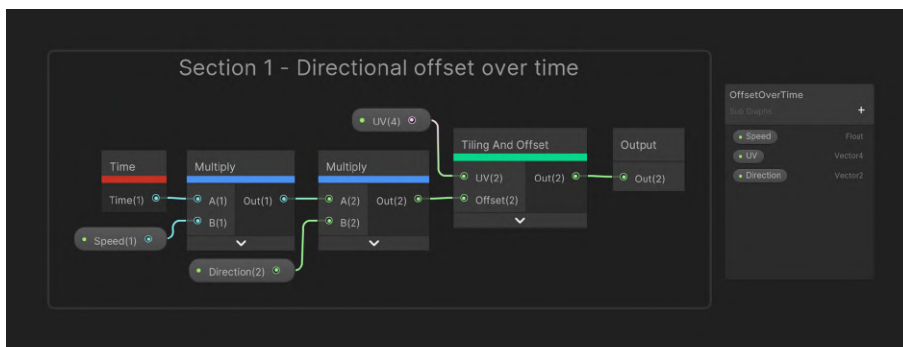


Figure 4.8: Offset Over Time Sub Graph with its input parameters.

■ 4.2.7 UV

UV nodes manipulate the UV coordinates of textures. For example, in this project are used:

- TILING AND OFFSET - Tiles and offsets a texture based on its UV coordinates.
- TWIRL - Twists the pixels of a texture around a central point, creating a spiral effect.
- ROTATE - Rotates UV coordinates around a center point.

■ 4.2.8 How to create Shader Graph and material based on it

1. Creating a new Shader Graph by going to the Assets menu and selecting Create → Shader → PBR Graph.
2. Next, select, place, and connect the necessary nodes to create the desired material. This may include nodes for color, texture, and other material properties.
3. Specifying the input parameters for the material, such as its color, texture, and other properties.
4. Once the Shader Graph has been set up, a new material can be created based on the graph using the Assets menu and selecting Create → Material.
5. Finally, the newly created material can be added to test objects in the scene to see how it looks and functions in the game. This allows us to iterate on the design and make any necessary adjustments before implementing the material in the final game.

Chapter 5

Materials library

The outcome of this project is a library containing over 20 materials (5 stylized static, 12 stylized dynamic, and 6 realistic) with various orientations and designs. Additionally, an application was developed [15] to simulate a test scene, enabling users to easily modify the materials and methods of presentation. Within the application, it is possible to access the complete library observation that can be discerned by referring to **Figure 5.1**.

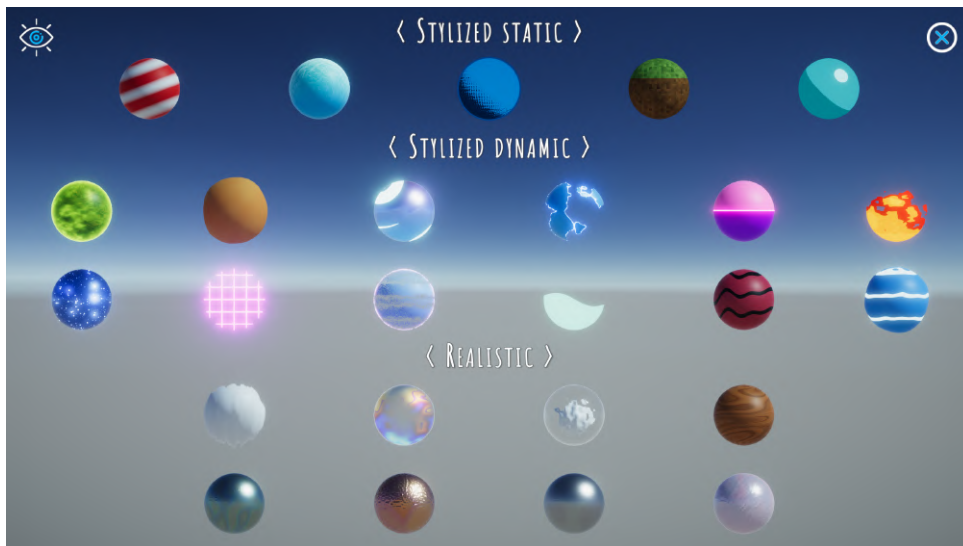


Figure 5.1: All materials produced as part of this project in Unity application.

This section contains the following parts specific to each material created in the project:

- description and demonstration
- inspiration
- general view of the Shader Graph with the table of input parameters

For a better idea and a brief overview of the contents of the library, it is worth referring to the **Table 5.1** of materials presented below. Materials marked with an "*" will be discussed in detail in their own sections.

Demo	Graph settings	Demo	Graph settings
 CANDY	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True	 FROST*	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True
 HALFTONE*	Material: Unlit Surface: Opaque Render Face: Front Cast Shadows: True	 OUTLINE*	Material: Unlit Surface: Opaque Render Face: Back Cast Shadows: False
 MINECRAFT*	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True	 TOON	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True
 ACID	Material: Lit Workflow: Metallic Surface: Transparent Cast Shadows: False Receive Shadows: True	 CLAY	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True
 CORE*	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: False Receive Shadows: False	 DISSOLVE	Material: Lit Workflow: Metallic Surface: Opaque Alpha Clipping: True Cast Shadows: True Receive Shadows: True
 FILLER	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True	 FLAME*	Material: Lit Workflow: Metallic Surface: Transparent Cast Shadows: False Receive Shadows: False

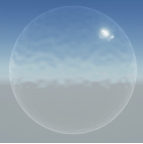
Demo	Graph settings	Demo	Graph settings
 GLITTER	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: False	 GRID*	Material: Lit Workflow: Metallic Surface: Transparent Cast Shadows: True Receive Shadows: False
 HOLO	Material: Lit Workflow: Metallic Surface: Transparent Cast Shadows: False Receive Shadows: False	 LIQUID	Material: Unlit Surface: Transparent Blending: Additive Cast Shadows: True
 PEWDIEPIE	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: False	 WATER*	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True
 MIRAGE	Material: Unlit Surface: Transparent Cast Shadows: False	 SOAP	Material: Lit Workflow: Specular Surface: Transparent Cast Shadows: False Receive Shadows: False
 GLASS	Material: Lit Workflow: Specular Surface: Transparent Cast Shadows: True Receive Shadows: False	 WOOD*	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True
 METAL*	Material: Lit Workflow: Metallic Surface: Opaque Cast Shadows: True Receive Shadows: True	 PEARL	Material: Lit Workflow: Specular Surface: Opaque Cast Shadows: True Receive Shadows: True

Table 5.1: Table of materials divided into categories.

5.1 Acid

An “Acid” material can be used to create the appearance of a corrosive, toxic substance. It can be used to enhance the realism of certain environments, such as a chemical laboratory or an alien planet. By using it, a sense of danger and intrigue can be added to a game.

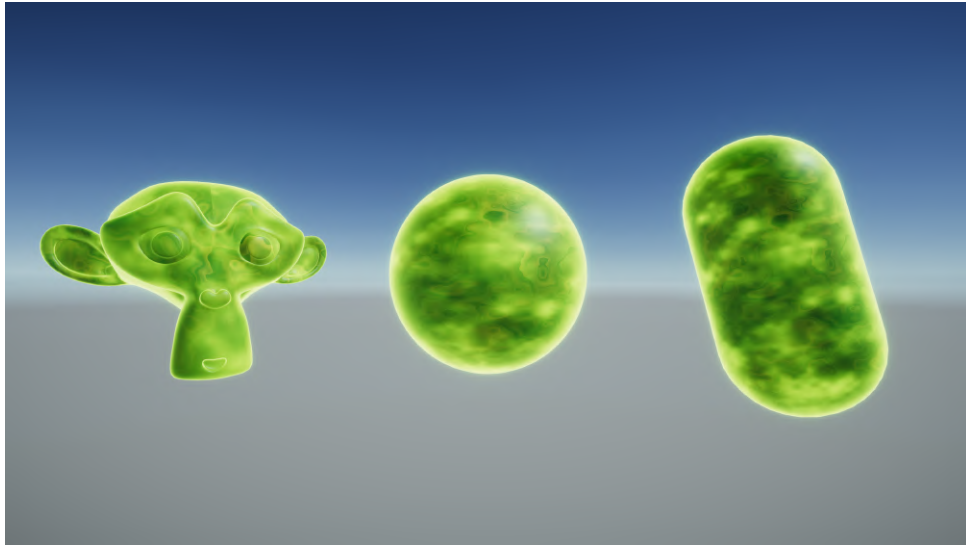


Figure 5.2: “Acid” Material on a variety of objects.

Inspiration was taken from the classic - Half-Life, namely the dangerous materials that can be found as obstacles throughout the entire game, for example, they can be met in the location of the Biological Waste Processing Plant (**Figure 5.2**) [16].

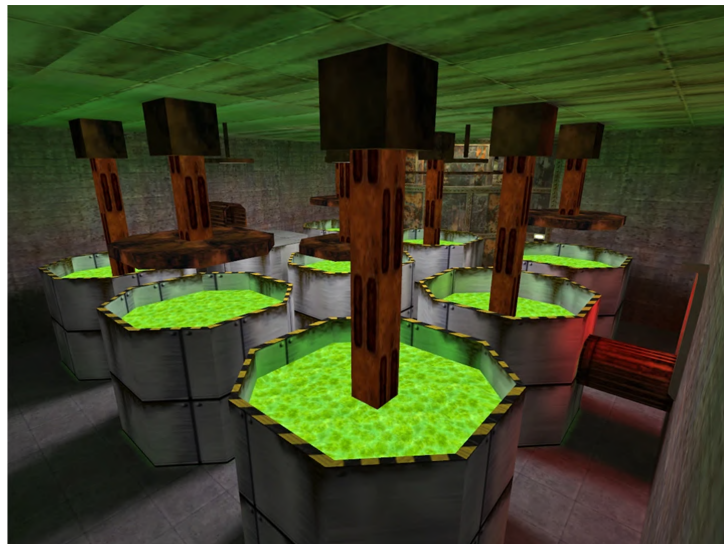


Figure 5.3: Hazardous waste tanks inside the plant.

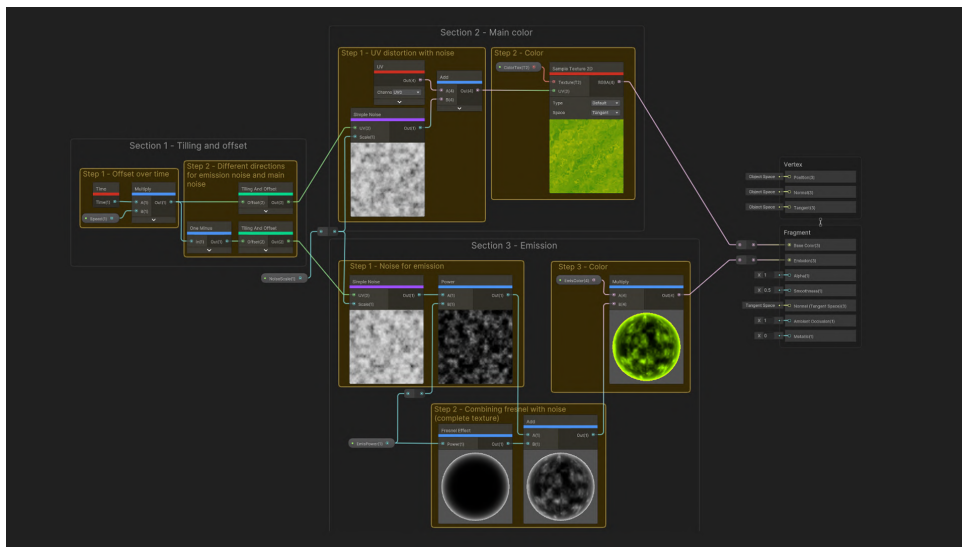


Figure 5.4: “Acid“ Shader Graph.

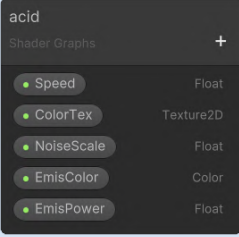
In Unity	Name	Type	Description
	SPEED	Float [-1, 1]	How quickly the noise texture will change over time.
	COLORTEX	Texture2D	The texture that will be used as the main color.
	NOISESCALE	Float	The size of the acid splits of the material on the surface.
	EMISCOLOR	Color HDR	The glow color.
	EMISPOWER	Float [-10, 10]	Glow intensity.

Table 5.2: “Acid“ Shader Graph input parameters.

5.2 Candy

The Christmas “Candy” material is a vibrant and festive material that is perfect for adding some holiday cheer to a game. It can be used on a variety of objects, such as gift boxes, ornaments, and other holiday decorations.

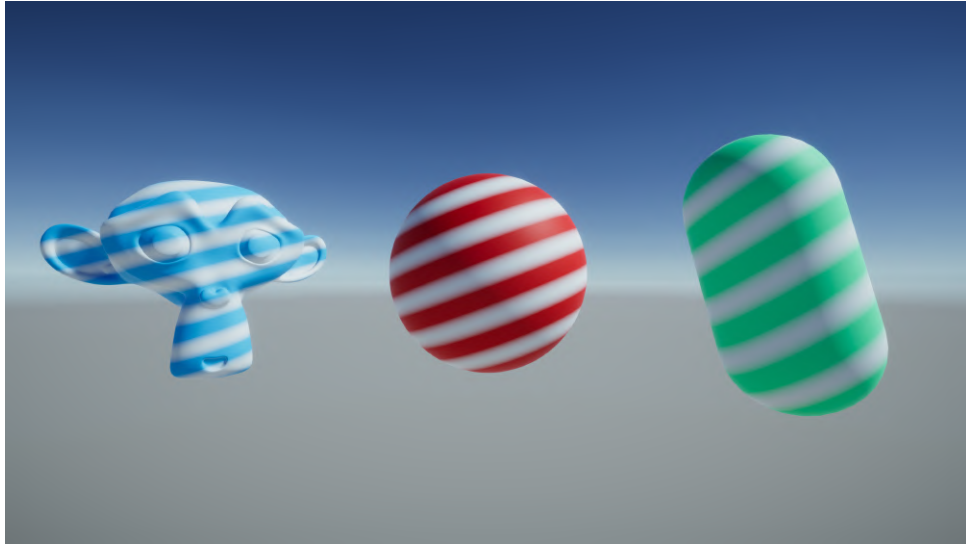


Figure 5.5: “Candy” Material on a variety of objects.

The inspiration came from a traditional Christmas candy - Candy Cane. This part of the holiday can be found everywhere in games (often in pre-Christmas events). For example in high places in Fortnite, in the Super Mario Odyssey’s Secret Candy Kingdom, or even as a simple decoration in Stardew Valley (**Figure 5.15**) [17].



Figure 5.6: Candy Cane decorations in Stardew Valley.



Figure 5.7: “Candy“ Shader Graph.

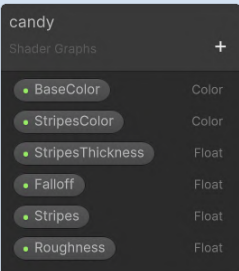
In Unity	Name	Type	Description
	BASECOLOR	Color	Main color of the surface.
	STRIPESCOLOR	Color	Color of stripes.
	STRIPESTHICKNESS	Float [0, 0.5]	How wide the stripes are.
	FALLOFF	Float [1, 10]	The smoothness of the transition from the stripe color to the main color.
	STRIPES	Float	Stripes count control.
	ROUGHNESS	Float [0, 1]	Surface roughness.

Table 5.3: “Candy“ Shader Graph input parameters.

5.3 Clay

The stop-motion “Clay” material is designed to create the appearance of a stop-motion animation that uses clay or plasticine. It features a tactile, textured surface that is reminiscent of handmade animation.

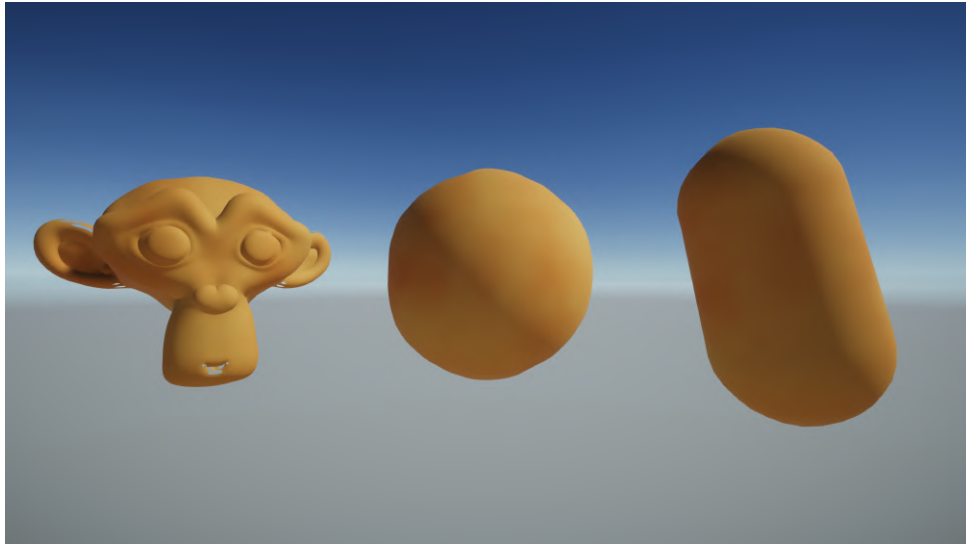


Figure 5.8: “Clay” Material on a variety of objects.

This material represents the style of Claymation - a form of animation where characters and backgrounds are made from plasticine or other malleable materials. Examples of popular movies in this style include *Chicken Run*, *Shaun the Sheep*, and *Wallace & Gromit*, while games such as *Skullmonkeys* and *The Neverhood* have also used this style.

Unlike traditional Claymation (e.g. in **Figure 5.9**) where objects typically retain their basic shape, this material features clay objects that change shape over time, creating a unique and sloppy transition effect between frames [18].



Figure 5.9: Claymation animation in Blender 2.81 by J Middleton [19].

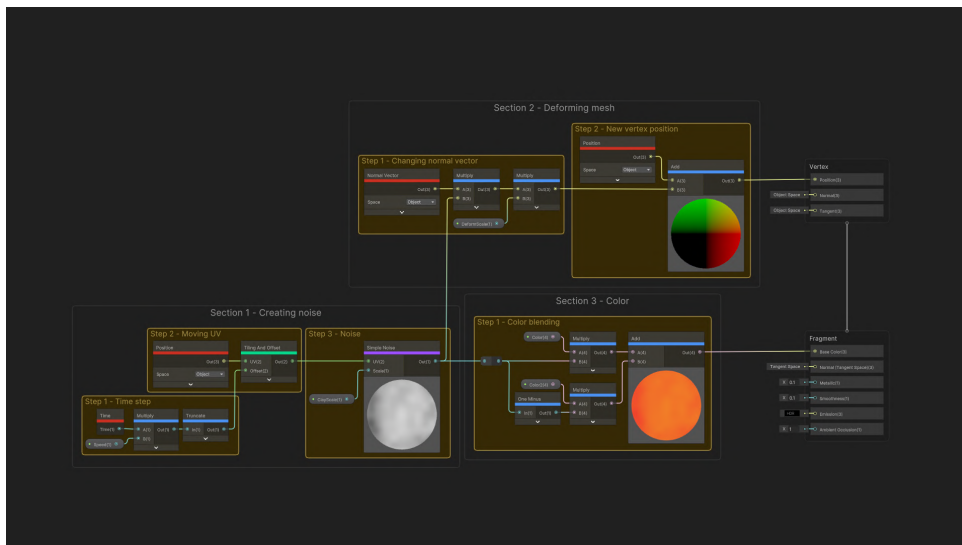


Figure 5.10: “Clay“ Shader Graph.

In Unity	Name	Type	Description
<div style="background-color: #333; color: #fff; padding: 5px;"> clay Shader Graphs + <ul style="list-style-type: none"> • Color Color • Color2 Color • ClayScale Float • DeformScale Float • Speed Float </div>	COLOR	Color	Main color of the surface.
	COLOR2	Color	Additional Color of the surface for blending.
	CLAYSCALE	Float [5, 10]	Noise texture scale.
	DEFORMSCALE	Float [0, 0.2]	How strongly the clay changes shape.
	SPEED	Float [0, 5]	Speed of stop-motion.

Table 5.4: “Clay“ Shader Graph input parameters.

5.4 Core

The “Core“ material can be used to add a mesmerizing and otherworldly touch to an object. It is particularly effective when applied to small, round objects such as gemstones or magical artifacts. The material’s combination of bright, swirling colors and refractive surface properties creates a sense of depth and movement.

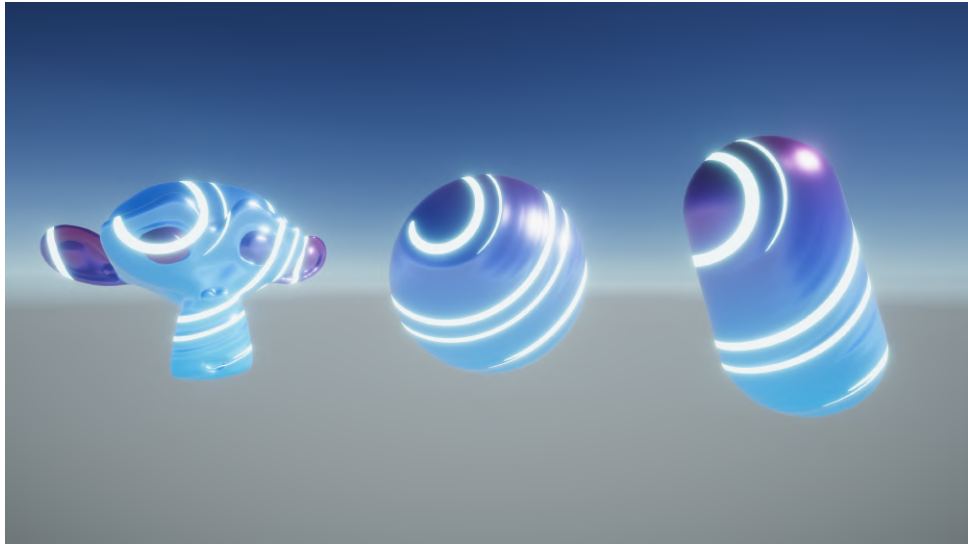


Figure 5.11: “Core“ Material on a variety of objects.

If you are familiar with the Avatar universe, you may know that in the animated series “Avatar Korra“ in the second season of episode ten, the protagonist opens a portal, which is shielded by a wavy-overflowing sphere (Figure 5.12). This effect served as the inspiration for the material.



Figure 5.12: Avatar Korra opens the northern portal to the Spirit World, S02E10.

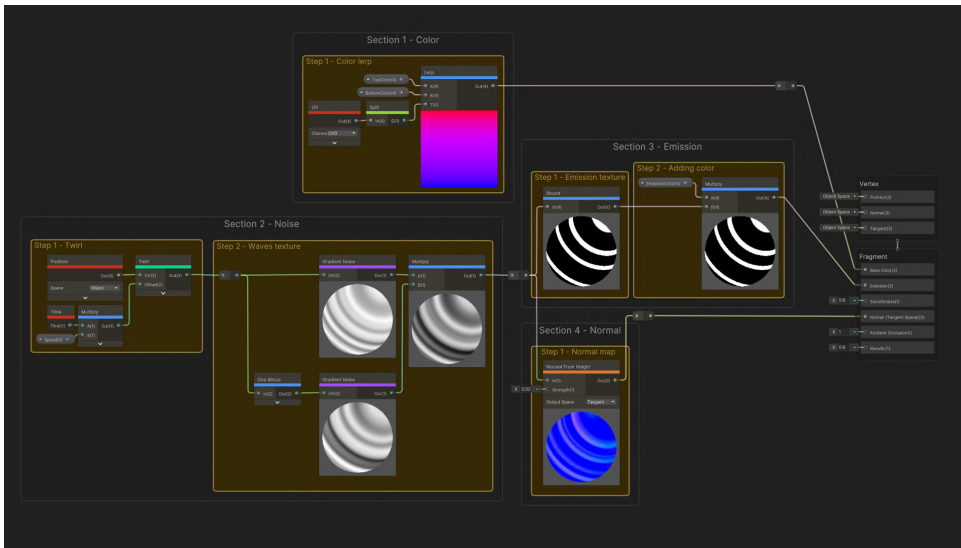


Figure 5.13: “Core“ Shader Graph.

In Unity	Name	Type	Description
<div style="background-color: #333; color: #eee; padding: 5px;"> core Shader Graphs + • TopColor Color • BottomColor Color • EmissionColor Color • Speed Float </div>	TOPCOLOR	Color HDR	Main color of the surface.
	BOTTOMCOLOR	Color HDR	Additional color of the surface for blending.
	EMITCOLOR	Color HDR	Swirling color.
	SPEED	Float [0, 1]	Speed of the twirling effect.

Table 5.5: “Core“ Shader Graph input parameters.

The first section (**Figure 5.14**) is responsible for the color. The combination of the UV and Split (Green channel) nodes creates a gradient, which is used as a mask for the Lerp node. The node is now responsible for the smooth transition between the two colors that the user can specify. The result of this group is sent to BaseColor.

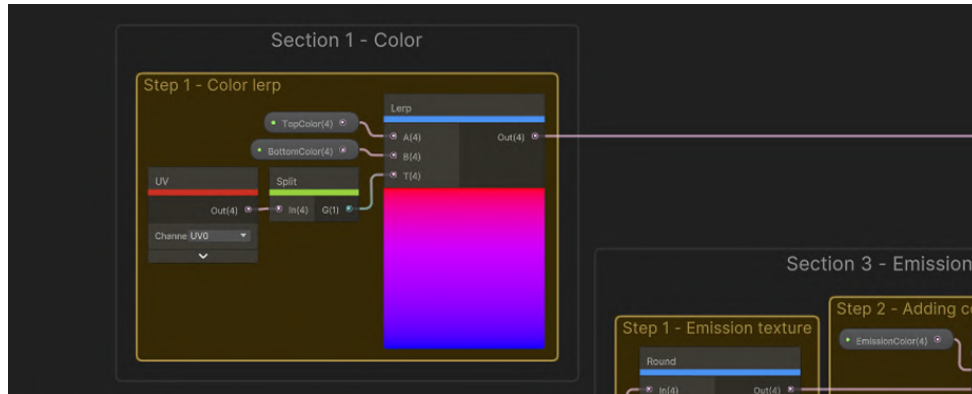


Figure 5.14: Section 1 of the “Core“ Shader Graph.

The second section (**Figure 5.15**) can be described in two steps: first, a dynamic Twirl is created using the Time node to change its shape constantly (users can control the speed of change); then, the modified UV of Twirl goes into Gradient noise to create a gray-scale wave texture. An identical noise is created below, reversed, and merged with the upper one for more variability in the waves.

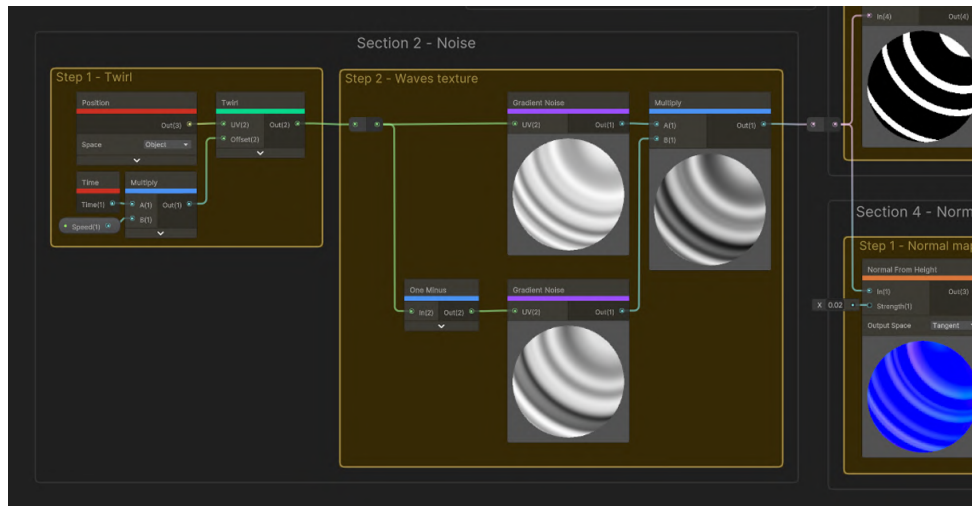


Figure 5.15: Section 2 of the “Core“ Shader Graph.

In the third section, the previously created wave texture is utilized to generate a mask for Emission through the Step node, which is then colored and fed into the Emission parameter. Moving onto the fourth section, the mask is used to generate a normal map using the Height to Normal node (as height maps are in gray-scale), intensifying the wave effect on the surface. The operation of these sections can be seen in **Figure 5.16** below.

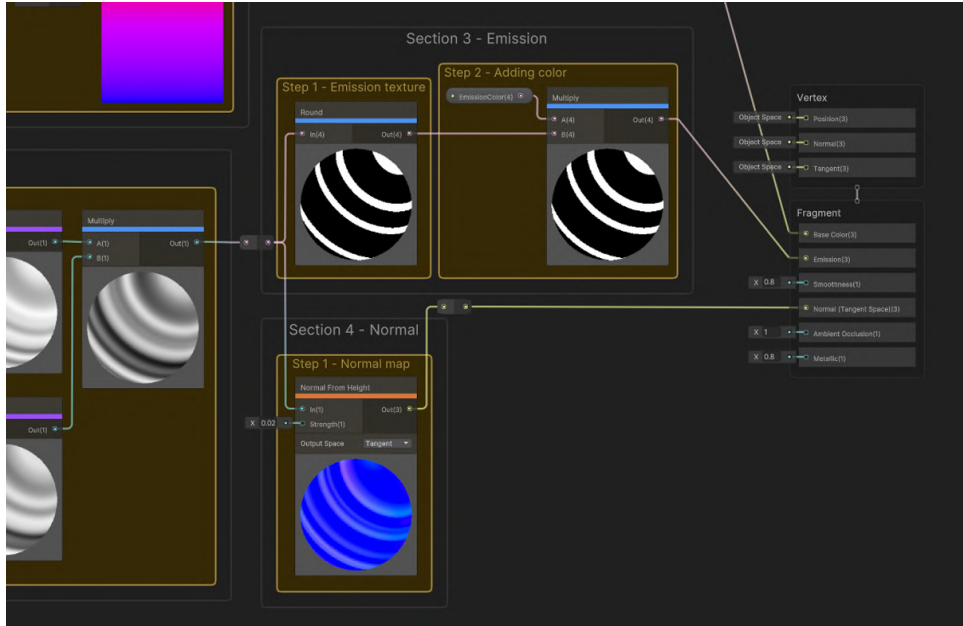


Figure 5.16: Section 3 & 4 of the “Core“ Shader Graph.

5.5 Dissolve

The Dissolve material can add dynamic and immersive elements. It can be used to create the appearance of objects dissolving or disintegrating over time. This material can be applied to a variety of objects, such as walls, floors, or even characters.

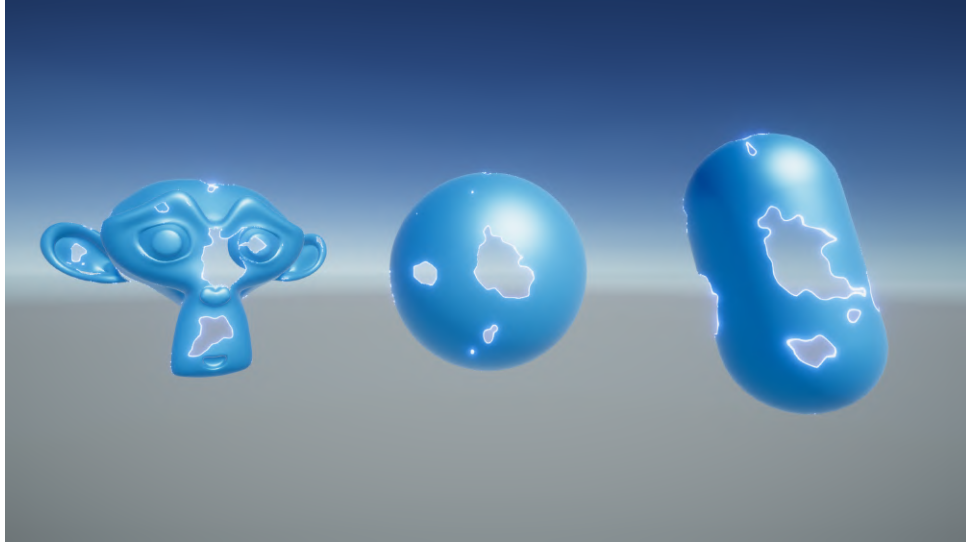


Figure 5.17: “Dissolve“ Material on a variety of objects.

In most cases, it is indeed responsible for the “evaporation“ of an object. For example, after killing an enemy (directly or post factum, as a “smooth“ removal of the object from the rendering field), the famous title *BioShock Infinite* (picture below **Figure 5.18**) can be remembered. A quality example can also be seen in *Genshin Impact*, where characters’ weapons appear/disappear with this effect.

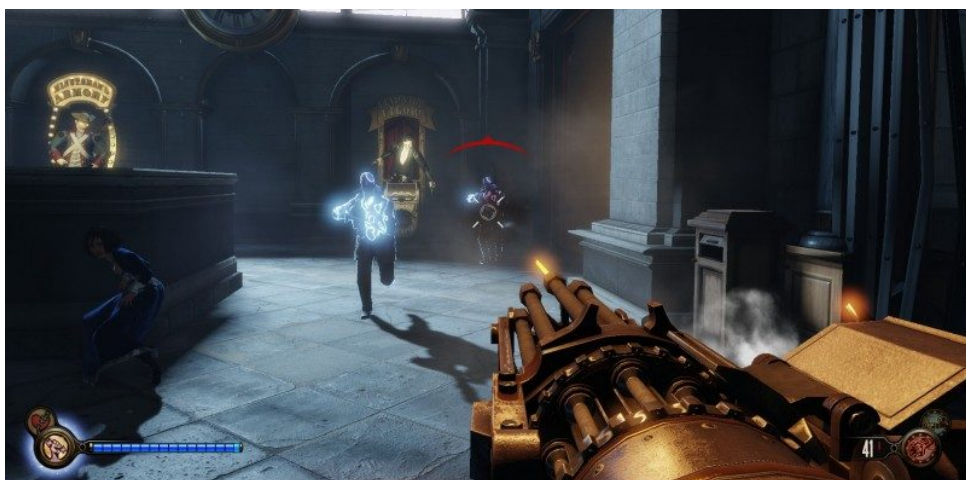


Figure 5.18: “Dissolve“ effect on corpses in *BioShock Infinite*.

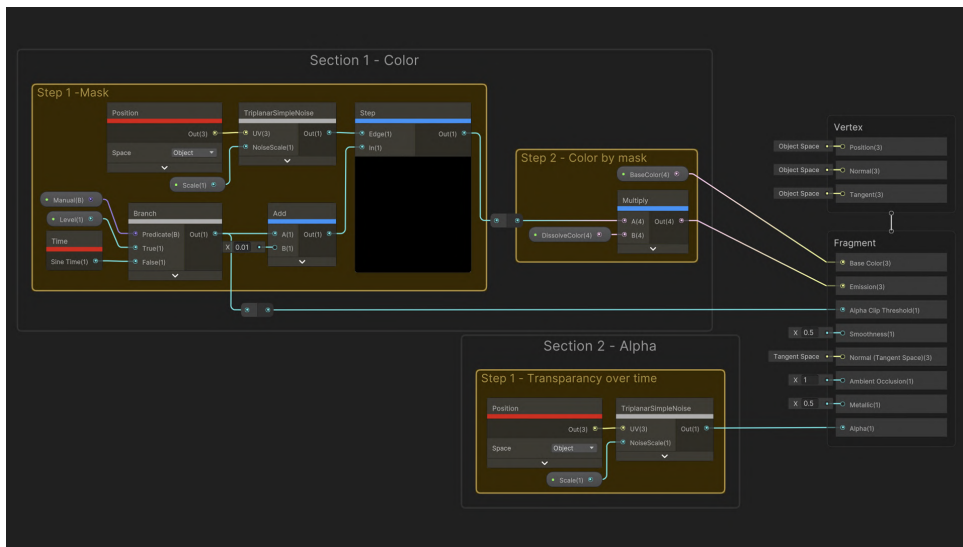


Figure 5.19: “Dissolve“ Shader Graph.

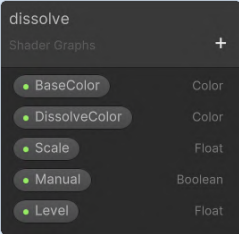
In Unity	Name	Type	Description
	BASECOLOR	Color	Main color of the surface.
	DISSOLVECOLOR	Color HDR	Color appearing on the edge of a disappearing area.
	SCALE	Float	Scale of dissolve generating noise.
	MANUAL	Boolean	Effect control switch, from automatic to manual.
	LEVEL	Float [0, 1]	Effect amount at manual control.

Table 5.6: “Dissolve“ Shader Graph input parameters.

5.6 Filler

The “Filler“ material is a versatile effect, that can be used to transform the look and feel of the surface. With its horizontal line animation, it creates an illusion of a “hoop“ that moves from bottom to top (and vice versa), gradually changing the color of the surface it is applied.

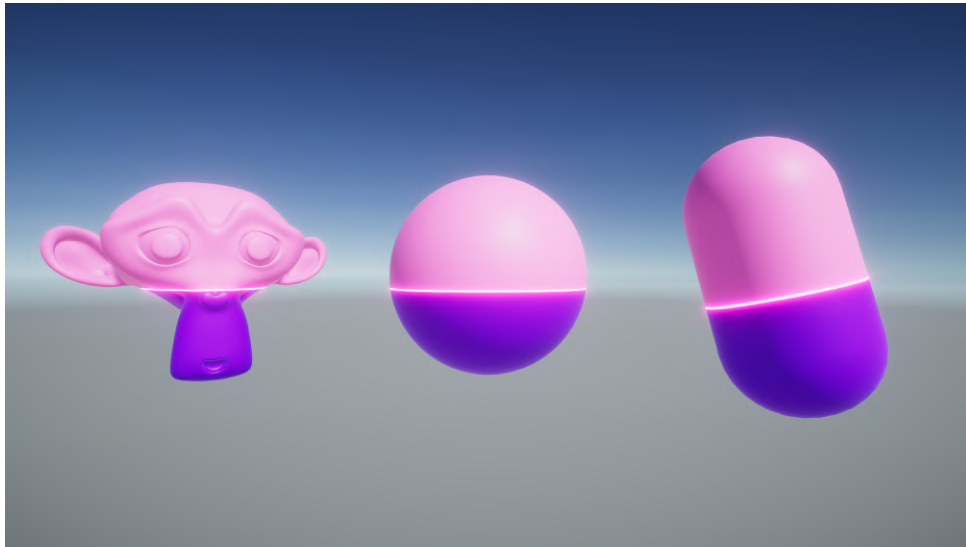


Figure 5.20: “Filler“ Material on a variety of objects.

Inspired by the work of TGA Digital (Figure 5.21 below) - their shader for the transition between 2 materials, the task to create a similar effect was supplied. This material can be used to create the effect of transformation or appearance/revealing. The “hoop“ can be controlled either over time or manually.

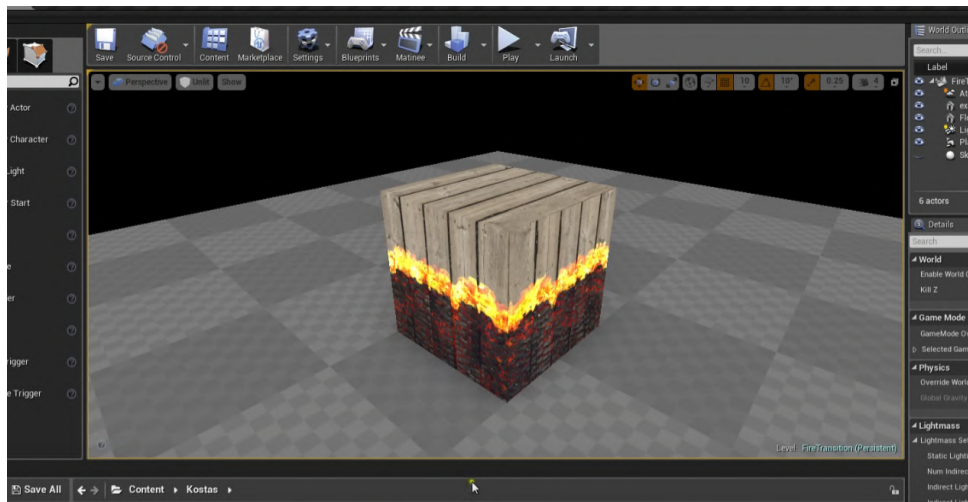


Figure 5.21: UE4 Transition Effect Shader by TGA Digital [20].

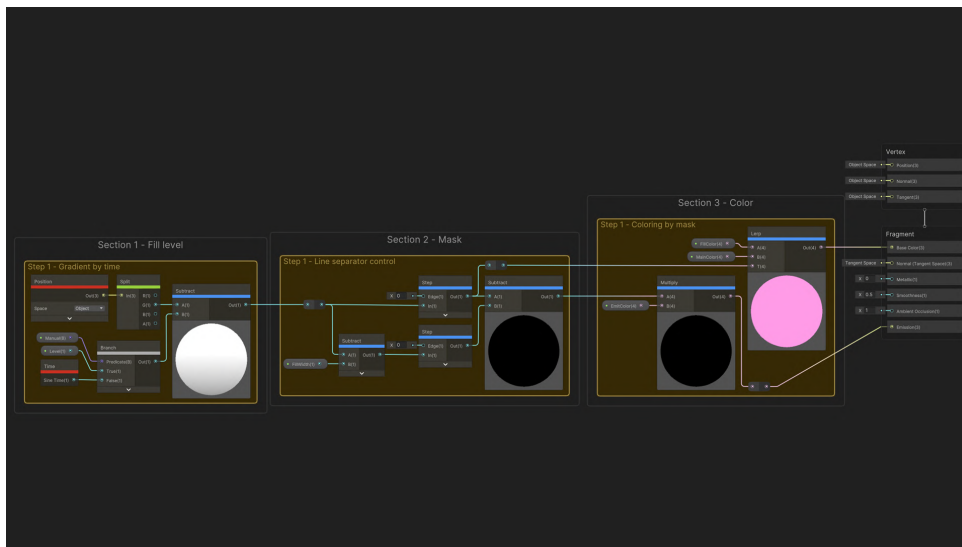


Figure 5.22: “Filler“ Shader Graph.

In Unity	Name	Type	Description
	MAINCOLOR	Color HDR	Main color of the surface.
	FILLCOLOR	Color HDR	Color of the surface after transition.
	EMITCOLOR	Color HDR	“Hoop“ transition color.
	FILLWIDTH	Float [0.01, 0.05]	Thickness of the “hoop“ transition.
	MANUAL	Boolean	Effect control switch, from automatic to manual.
	LEVEL	Float [-1, 1]	Transition level at manual control.

Table 5.7: “Filler“ Shader Graph input parameters.

5.7 Flame

The “Flame“ material is a unique effect that can be used to bring fire to a game. It creates a cartoon-like look that can be applied to objects, such as torches, fire pits, or even character abilities. The shader simulates a fire’s movement and color changes with its dynamic animation.

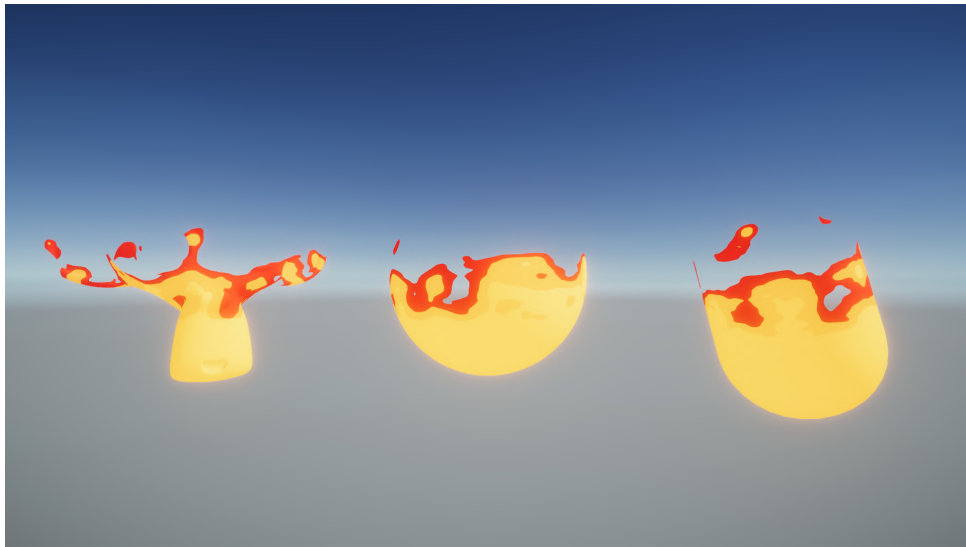


Figure 5.23: “Flame“ Material on a variety of objects.

When implementing a stylized version, fire can be defined as a pattern, with a clear transition from the base to the flames, sometimes with an additional passage. This kind of material would be good for stylized environments, like the fireplace in the **Figure 5.24**.



Figure 5.24: Example of using the “Flame“ material with a model of a fireplace. Models by Rocco Giandomenico (logs) [21], Michalina "Miszla" Gąsienica-Laskowy (fireplace) [22].

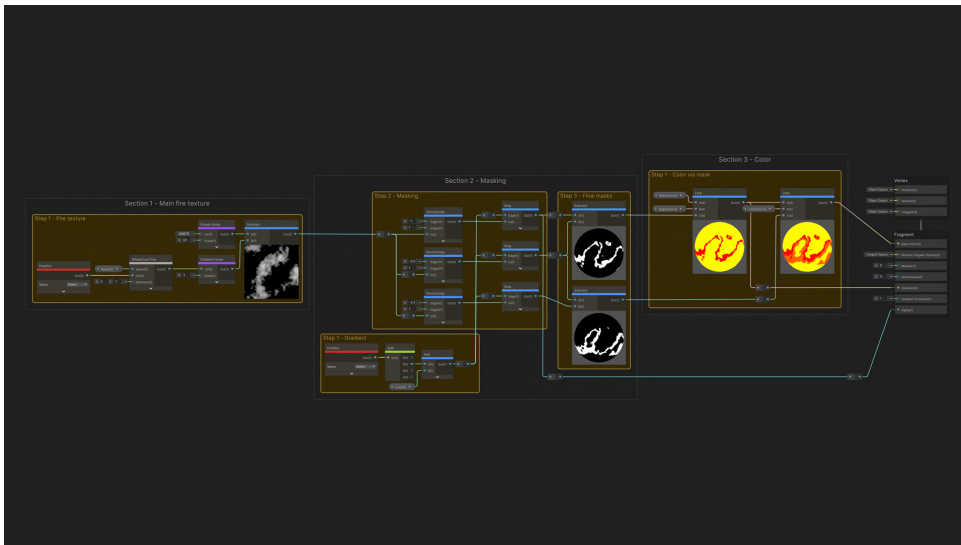


Figure 5.25: “Flame“ Shader Graph.

In Unity	Name	Type	Description
flame Shader Graphs + • MainColor Color • EdgeColor Color • LerpColor Color • Speed Float • Level Float	MAINCOLOR	Color HDR	Main color of the fire.
	EDGECOLOR	Color HDR	Color of flame tongues.
	LERPCOLOR	Color HDR	Color of transition flame part.
	SPEED	Float [0, 2]	Speed of change of flame tongues.
	LEVEL	Float [0, 1]	Level of flame tongues in relation to whole fire.

Table 5.8: “Flame“ Shader Graph input parameters.

The initial step (see **Figure 5.26**) aims to create a noise texture to serve as the basis for the flame tongues' masks. To achieve this, the custom subgraph *Offset Over Time* (detailed in **Section 4.2.1**, **Figure 4.7**) is used to move the UV of a Gradient Noise. The Gradient Noise is then subtracted from Simple Noise, resulting in a dynamic texture.

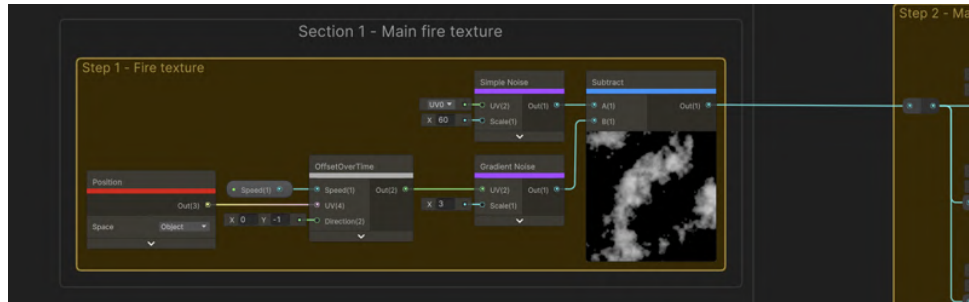


Figure 5.26: Section 1 of the “Flame“ Shader Graph.

In the following section two (**Figure 5.27**), masks for the color transitions of the flame tongues are created based on the noise texture obtained in the previous section. To achieve this, the Smoothstep node is used to generate 3 masks with different levels, where each new mask's upper bound differs from the next one by 0.4. These masks are then converted into a full black-and-white fire pattern using the Step node. The gradient from the Step 1 is used to determine the hardness (flame height). Finally, the 2 layers for coloring are created by subtracting the masks from each other.

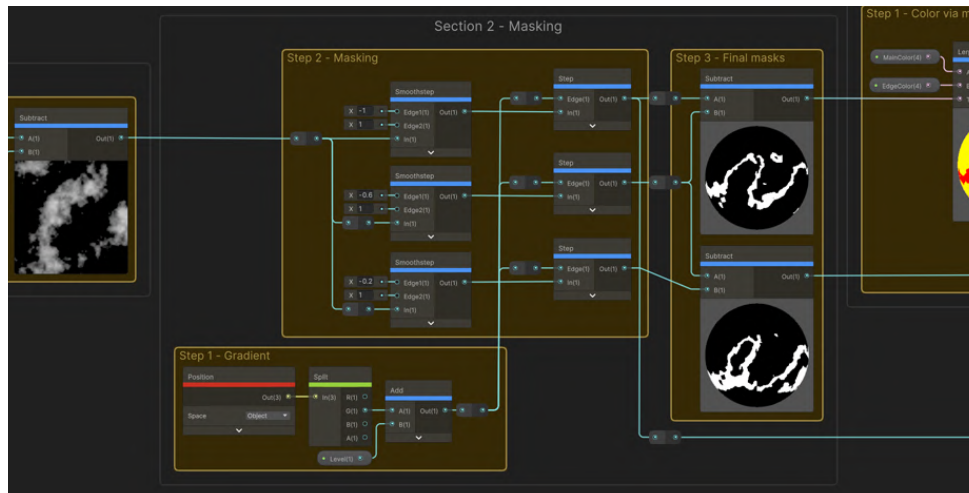


Figure 5.27: Section 2 of the “Flame“ Shader Graph.

The last section uses layers from the previous one (**Figure 5.28**). Both layers are used in the order in the Lerp nodes, the first to color the edges of the flame tongues and the base color, and the second to color the transition layer.

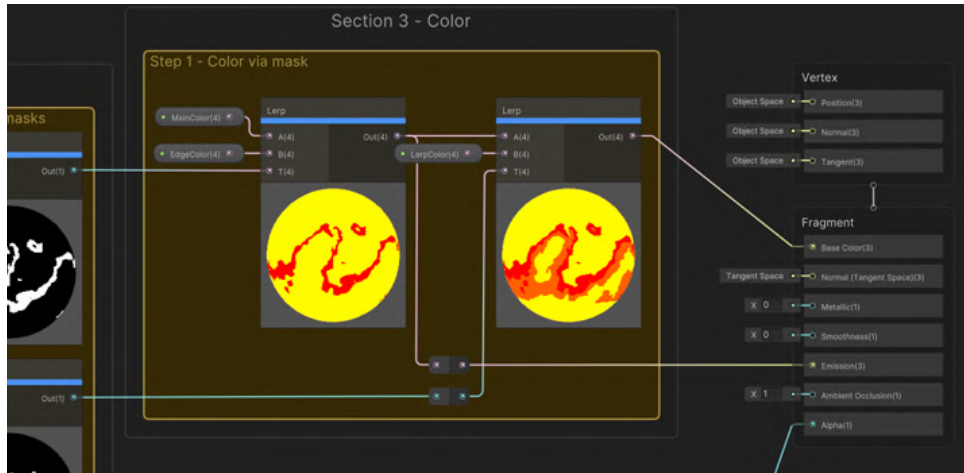


Figure 5.28: Section 3 of the “Flame“ Shader Graph.

5.8 Triplanar projection

Before proceeding to the next material, it is worth mentioning a small detail. Most of the markup nodes in the shader graph use Position or UV, which means that the only one independent of the unwrapping process is **Triplanar Projection**. This method is widely used in modern texturing programs, for example, Substance Painter (**Figure 5.29**).

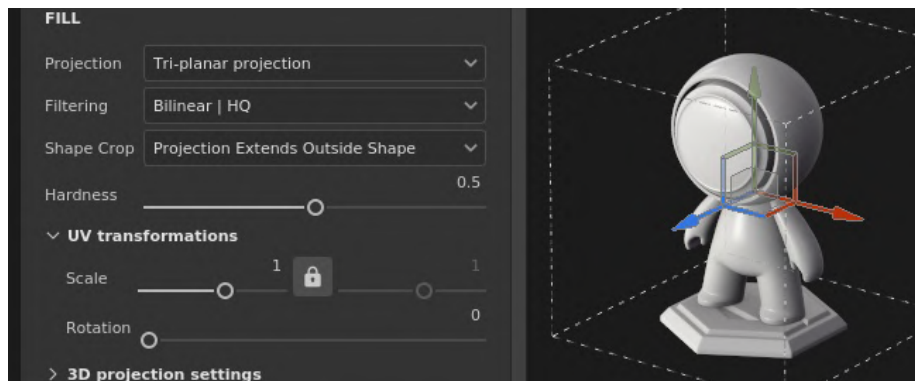


Figure 5.29: Triplanar Projection feature in Adobe Substance Painter. [23]

Triplanar projection is a method of mapping a 3D texture onto a 3D object without the need for UV coordinates. It works by projecting the texture onto the object from three different axes (X, Y, Z) and blending the results together. This allows the texture to be applied seamlessly to the object, regardless of its shape or orientation [24]. An example is well illustrated in **Figure 5.30** below.

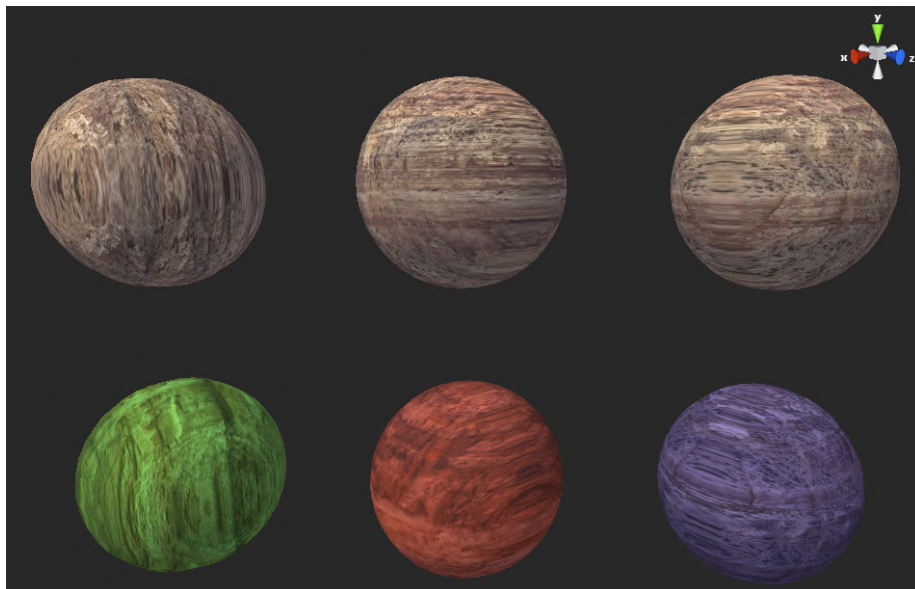


Figure 5.30: Triplanar maps over the surface from three different axes (X, Y, Z) by Martin Palko.

Triplanar projection can be useful in situations where UV mapping is difficult or impossible, such as with highly detailed or irregularly-shaped objects. The projected texture will look similarly good (an example of the projected texture in **Figure 5.31** can be seen below).

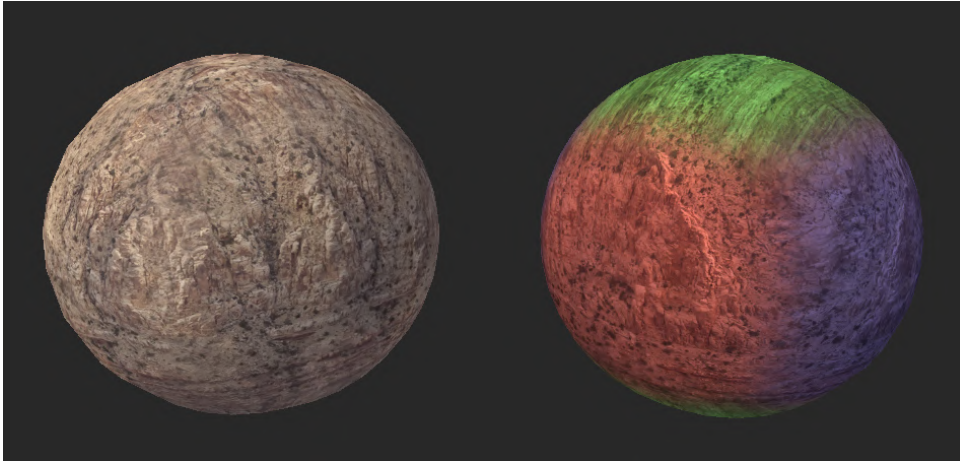


Figure 5.31: Triplanar mapped on a surface (left) and maps blending view (right) by Martin Palko.

The Unity Shader Graph features a Triplanar Texture node that could suffice, but it does not support process-generated noise, only pre-made textures. Fortunately, the Unity documentation [25] explains how projections are computed (**Figure 5.32**) within this node.

```
float3 Node_UV = Position * Tile;
float3 Node_Blend = pow(abs(Normal), Blend);
Node_Blend /= dot(Node_Blend, 1.0);
float4 NodeX = SAMPLE_TEXTURE2D(Texture, Sampler, Node_UV.zy);
float4 NodeY = SAMPLE_TEXTURE2D(Texture, Sampler, Node_UV.xz);
float4 NodeZ = SAMPLE_TEXTURE2D(Texture, Sampler, Node_UV.xy);
float4 Out = NodeX * Node_Blend.x +
            NodeY * Node_Blend.y +
            NodeZ * Node_Blend.z;
```

Figure 5.32: Example code represents the possible outcome of Triplanar Texture node (Default mode).

This approach simplifies the process of creating a Triplanar method for a required procedural texture, such as Simple Noise (shown in Figure 5.62). This can be achieved by using a sub graph. **A second sub graph was also created, differing only in the use of Gradient Noise.**

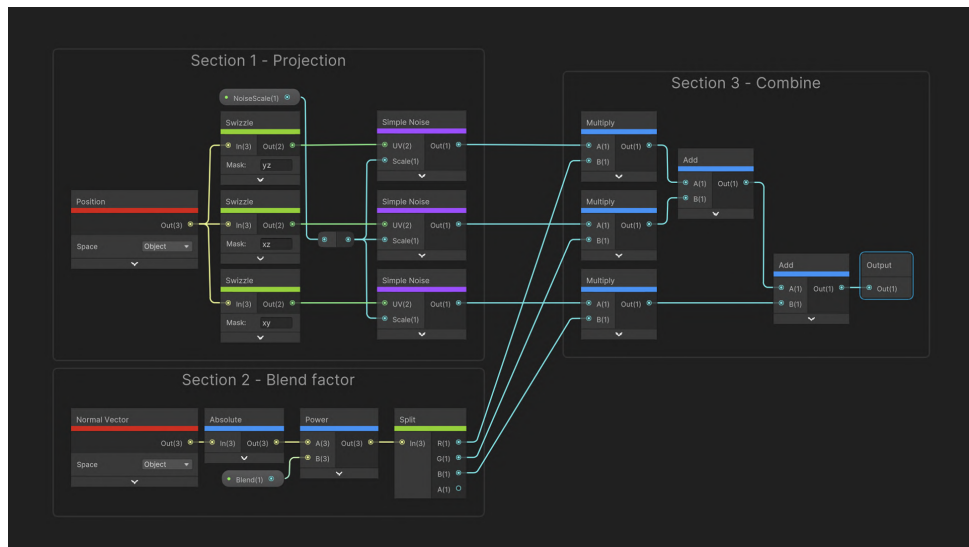


Figure 5.33: Triplanar Simple Noise sub graph.

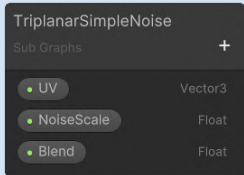
In Unity	Name	Type	Description
	UV	Vector3	UV coordinates for mapping.
	NOISESCALE	Float	Scale of the whole noise.
	BLEND	Float	The blending factor of all layers of noise.

Table 5.9: Triplanar sub graph input parameters.

5.9 Frost

The cold ice with “Frost” material adds a winter touch to the environment. It can be used on trees, rocks, and ice itself to give them a frosty appearance. Additionally, the material can be used to create a cohesive visual theme for a winter-themed level or event.

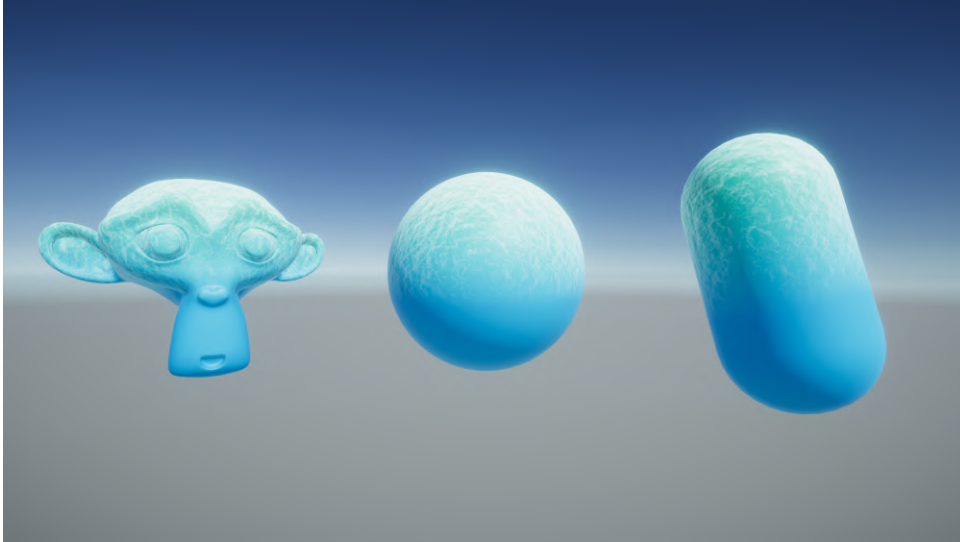


Figure 5.34: “Frost” Material on a variety of objects.

The inspiration came from my love of snow and several games with a winter (extremely harsh) theme, such as *Long Dark* and *Frostpunk*. There, icing up a storm is a severe challenge for the player. This material also interprets icing with crust, only in a more stylized format.



Figure 5.35: *Frostpunk* artwork by 11-bit Studios [26].

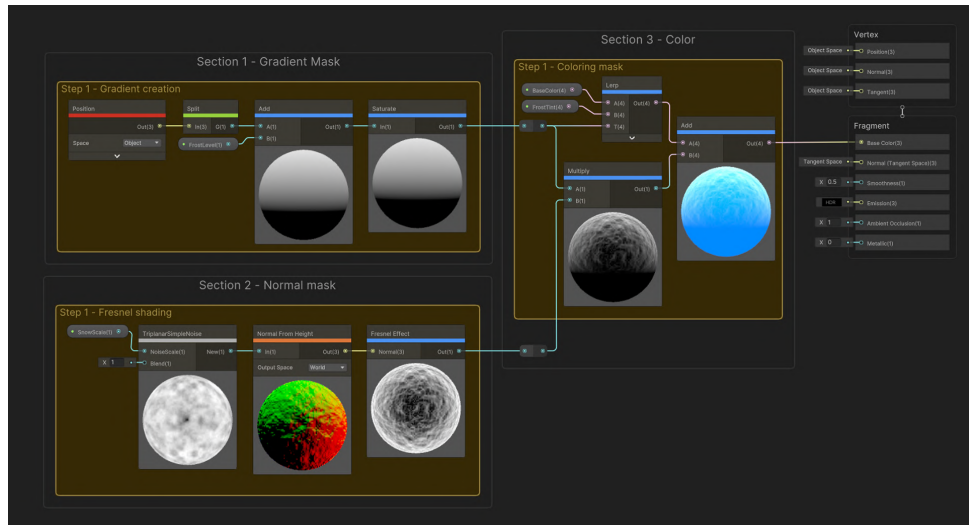


Figure 5.36: “Frost“ Shader Graph.

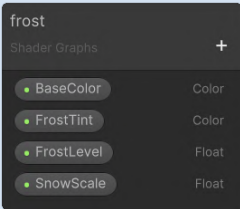
In Unity	Name	Type	Description
	BASECOLOR	Color HDR	Main color of the surface.
	FROSTTINT	Color HDR	Color of the frost part.
	FROSTLEVEL	Float [-1.5, 1.5]	Adjust the frost level mask vertically.
	SNOWSCALE	Float	Scale of the frost noise mask.

Table 5.10: “Frost“ Shader Graph input parameters.

The first section generates the icing level, which is oriented vertically through the gradient via the Green channel of the Split node (Figure 5.37). The level can be adjusted by the user. To avoid interference with future mask usage, all negative values are removed using the Saturate node.

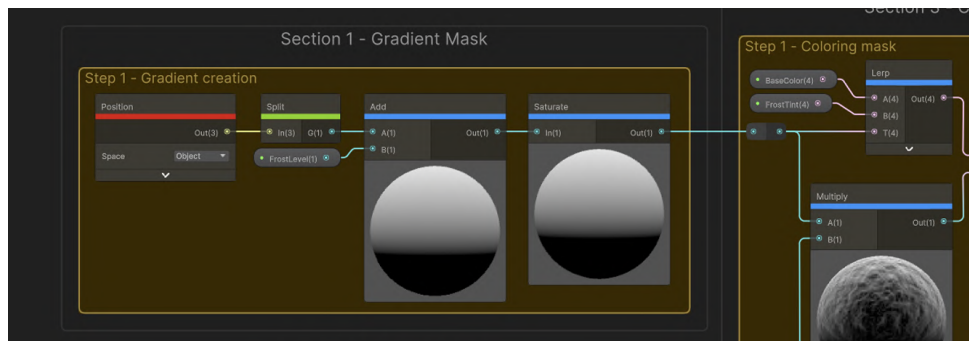


Figure 5.37: Section 1 of the “Frost“ Shader Graph.

The second section focuses on generating the icing texture, using a simple yet effective trick (**Figure 5.38**). Triplanar Simple Noise is utilized to create a gray-scale texture, so it can be transformed into a normal map with the Height To Normal node. By employing the principles of Fresnel’s work, using the normal map as the basis for the Fresnel effect adds more dimension to the material by taking into account the viewing angle.

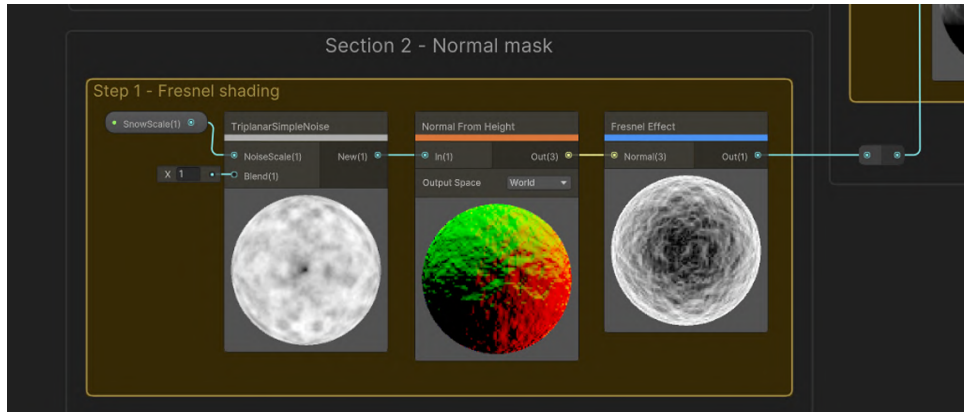


Figure 5.38: Section 2 of the “Frost“ Shader Graph.

In the final step (**Figure 5.39**), coloring is accomplished using a Lerp node that blends by using the combination of both masks: the level generated in the first section and the icing from the second one, with the user determining the primary surface color and the ice shade. The output is then sent to the Base Color channel.

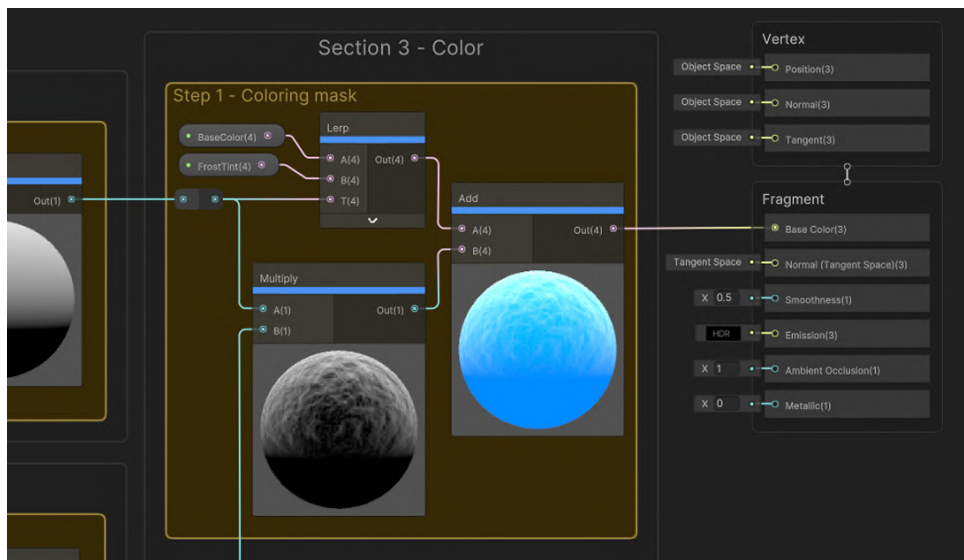


Figure 5.39: Section 3 of the “Frost“ Shader Graph.

5.10 Glitter

The spacey “Glitter“ can be used to add a touch of otherworldly charm to objects in a game. With its shimmering, glittering appearance, it can be used to add a touch of magic and wonder to objects such as weapons, armor, or other decorative elements. The material can also be applied to create a representation of the night sky or a space cube map.

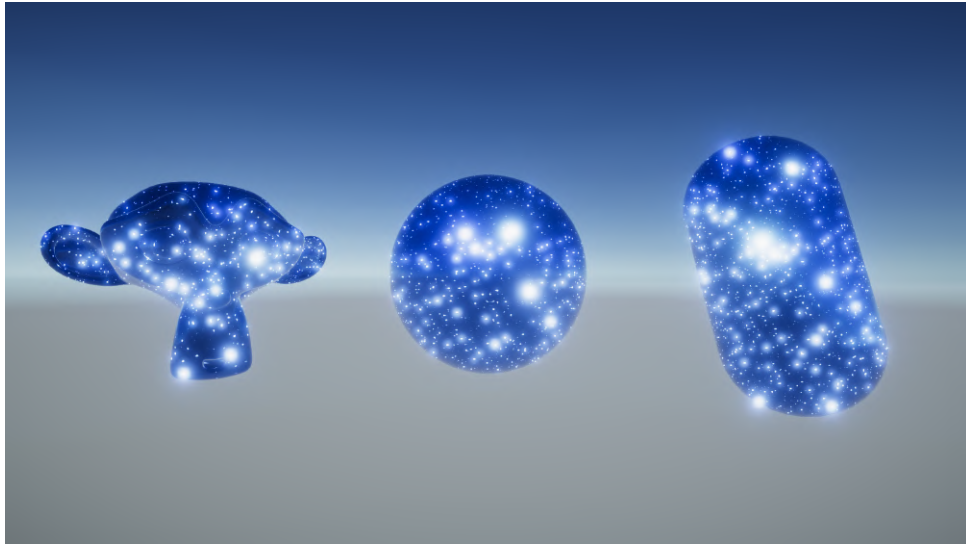


Figure 5.40: “Glitter“ Material on a variety of objects.

As inspiration were taken photos (or generated pictures) of the cosmos/-galaxies (like on a picture below). From them were borrowed the scatter of stars, the brightness of the “surface“ and the color transition.



Figure 5.41: Galaxy by rawpixel.com [27].

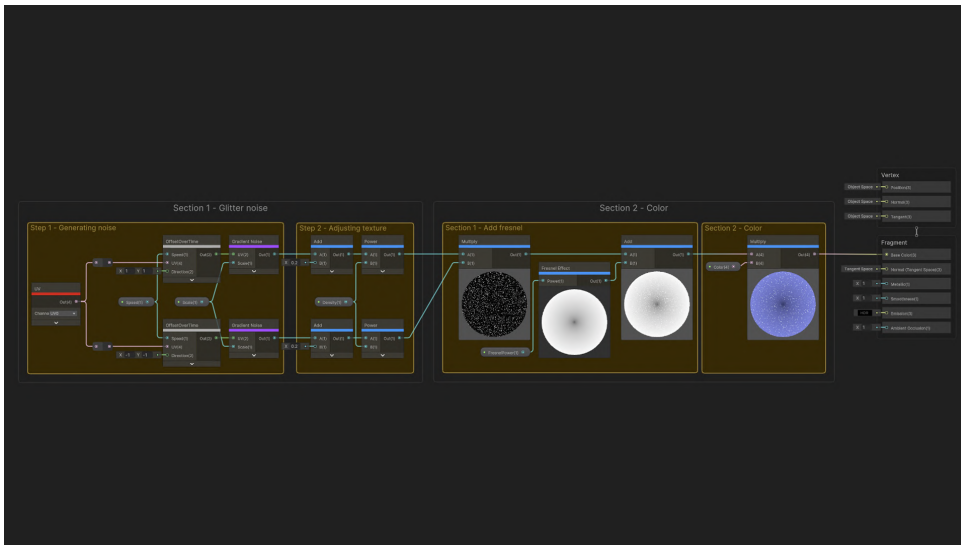


Figure 5.42: “Glitter” Shader Graph.

In Unity	Name	Type	Description
glitter Shader Graphs	COLOR	Color	Main color of the surface.
Color	SCALE	Float	Particles noise scale.
Scale	DENSITY	Float	Particle density.
Density	SPEED	Float [0.01, 0.1]	Speed of glare of partials on the surface.
Speed	FRESNELPOWER	Float [0, 2]	Fresnel effect strength.
FresnelPower			

Table 5.11: “Glitter” Shader Graph input parameters.

5.11 Grid

The “Grid“ material is a vibrant and eye-catching material that can be used to create a retro-futuristic or cyberpunk aesthetic in a game. By adjusting the color and intensity of the grid lines, it can give the impression of a computer interface or data visualization. Additionally, it can be used as a background element to add visual interest and depth to a scene. The grid lines can be made to pulsate or animate to create a sense of movement and energy.

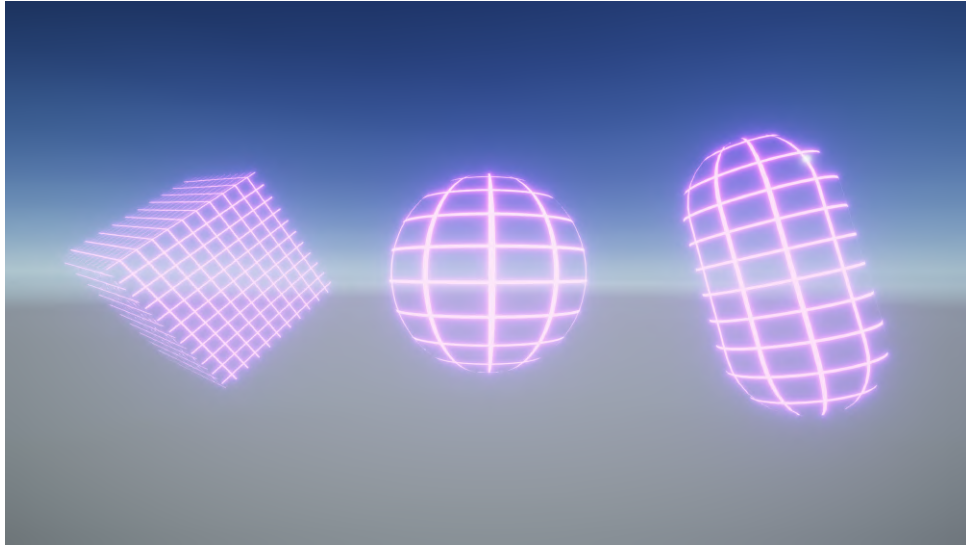


Figure 5.43: “Grid“ Material on a variety of objects.

The material features a purple grid pattern that is reminiscent of the popular retro wave style, which is characterized by a blend of 1980s nostalgia and futuristic elements (like on picture below).



Figure 5.44: Retrowave Neon 80's Background by Rafael-De-Jongh [28].

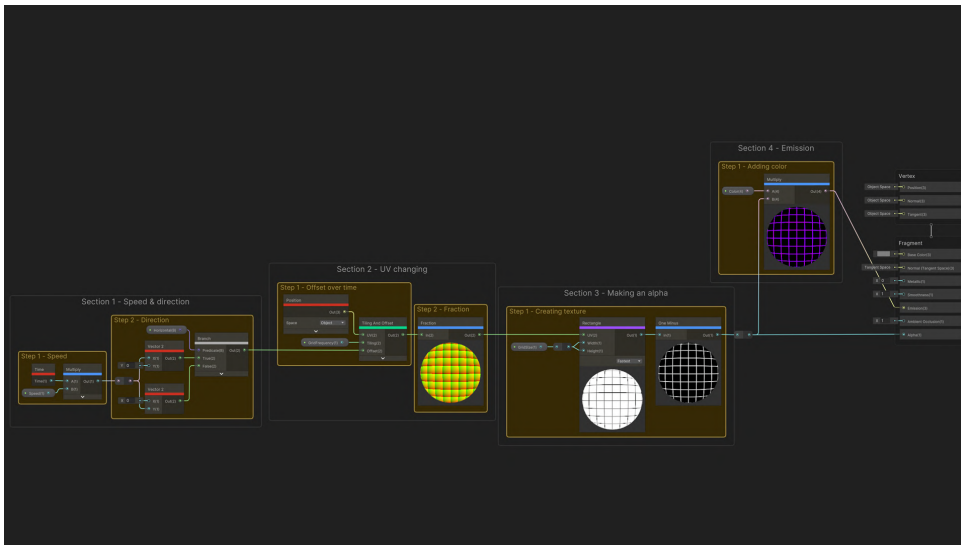


Figure 5.45: “Grid“ Shader Graph.

In Unity	Name	Type	Description
	COLOR	Color HDR	Main color of a grid.
	GRIDSIZE	Float [0.7, 0.9]	Size of a quad.
	GRIDFREQUENCY	Float	Grid quad frequency.
	SPEED	Float [1, 5]	Speed of an offset.
	HORIZONTAL	Boolean	Direction of an offset switcher (horizontal/vertical).

Table 5.12: “Grid“ Shader Graph input parameters.

In the first section, the grid is controlled with a time offset and a switch for horizontal/vertical rotation. By creating a corresponding vector in UV with a forced X or Y excision, the grid can be rotated. In the second section, the offset from the first section is used, and the grid size is controlled using an input parameter. When a value greater than 1 is present, the node Fraction node is utilized to repeat the UV instead of stretching it. The above-described process can be seen on **Figure 5.46**

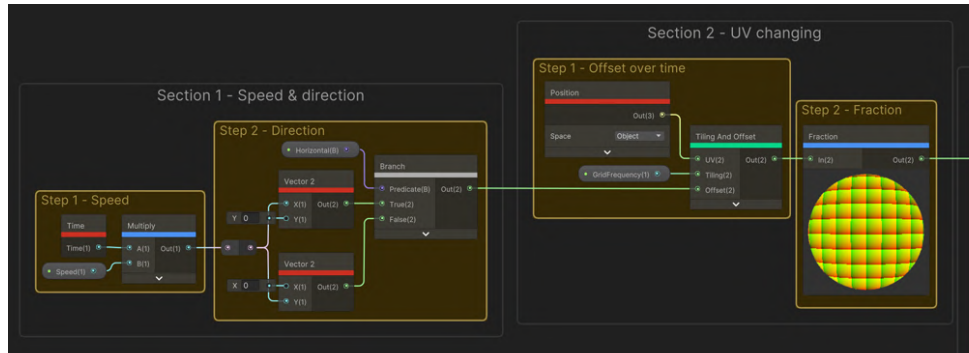


Figure 5.46: Sections 1 & 2 of the “Grid“ Shader Graph.

In the third section, a transparency mask is created while the fourth section handles coloring. The UV generated in the second section is utilized to call the Rectangle node, producing a grid mask of the user-defined size that is always square, because a single parameter determines both height and width. The resulting mask is applied to the Alpha channel and is also recolored to the desired color before being sent to Emission (**Figure 5.47**).

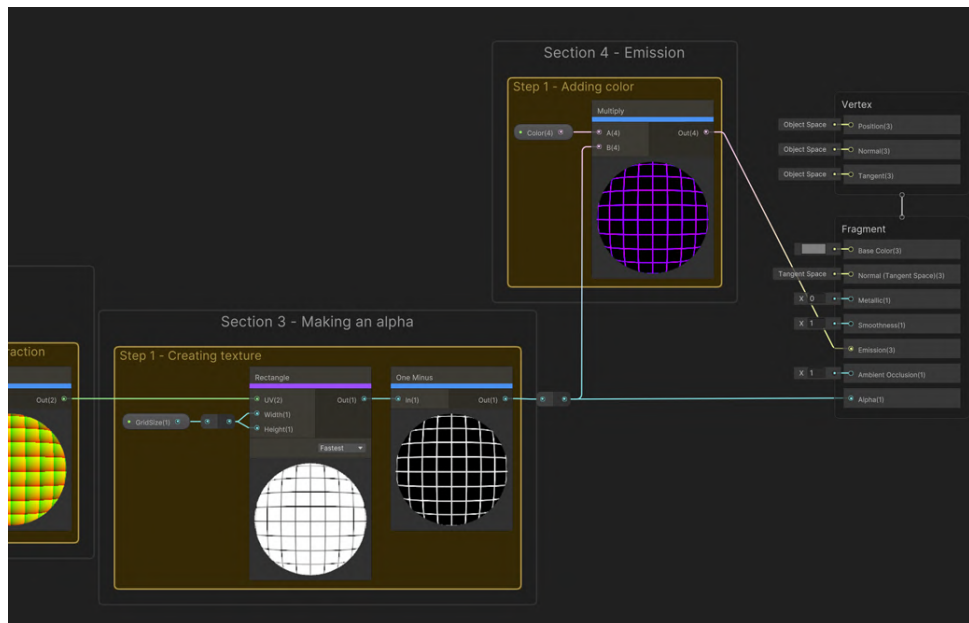


Figure 5.47: Sections 3 & 4 of the “Grid“ Shader Graph.

5.12 Halftone

The “Halftone” is a special kind of material that is used to simulate a halftone printing effect on a 3D object. The Halftone can be adjusted using various parameters such as dot size, spacing, and angle, to fine-tune the final look. As an example, check out the Kirby games (e.g. Kirby Battle Royale[?])

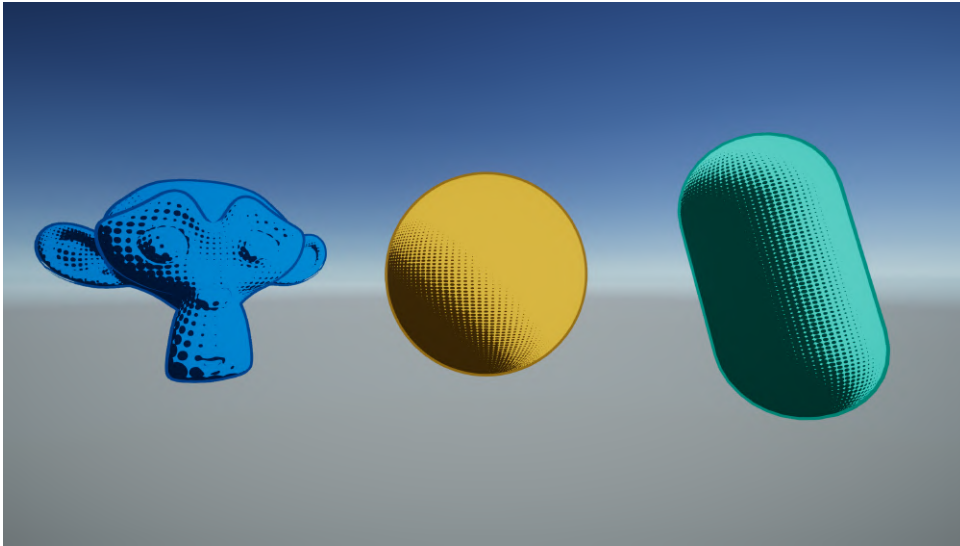


Figure 5.48: “Halftone” Material on a variety of objects.

The Halftone effect is a technique used in graphic design and digital art to simulate the appearance of a printed image or photograph. It works by breaking the image down into a series of tiny dots, with the size and spacing of the dots varying to create shading and depth (such as in the picture below).

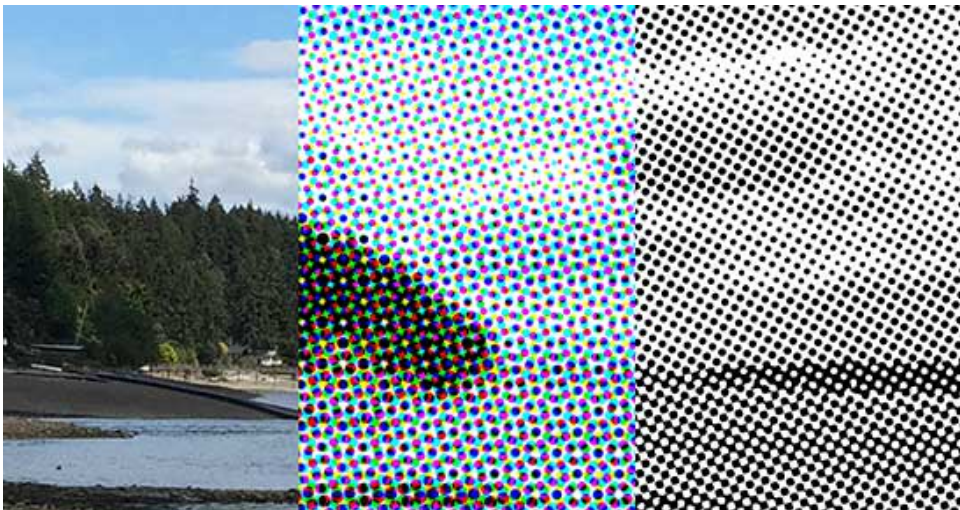


Figure 5.49: Halftone effect example (source image on the left, final transition on the right) [29].

You may also have noticed the **outline** effect that often goes along with halftone. The outline effect is used in games to create a visible line around the edges of an object (as in **Figure 5.50**). It is typically used to create a visual distinction between different objects in the game world or to draw attention to certain objects.



Figure 5.50: Outline effect in Left4Dead game is used to distinct player’s allies.

The effect works by rendering a duplicate of the object, slightly scaled up, and drawing a colored line around the edges of the duplicate. The result is an outline that gives the appearance of a border around the original object. The thickness and color of the outline can be adjusted. The graph of this effect is quite small, it can be observed below in **Figure 5.51**.

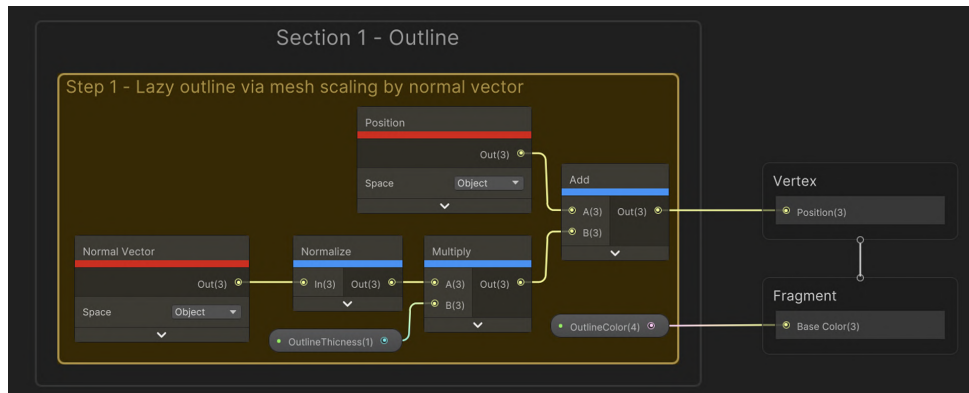


Figure 5.51: “Outline“ Shader Graph.

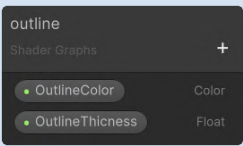
In Unity	Name	Type	Description
	OUTLINECOLOR	Color	Main color of the outline.
	OUTLINETHICKNESS	Float [0, 1]	Regulates thickness of the outline.

Table 5.13: “Outline“ Shader Graph input parameters.

Before looking at the Halftone graph in more detail, it is worth mentioning the custom nodes that will be used in it.

■ GetMainLight

It is responsible for obtaining basic information about the main light source of the “Directional” type, for subsequent use in filling the surface (Attenuation parameter) and direction, for obtaining the shadow (Direction parameter) [30]. The code generating this code can be observed below:

```
void GetMainLight_float(out half3 color,
                       out half3 direction,
                       out half attenuation)
{
    #ifdef SHADERGRAPH_PREVIEW
        direction = half3(0.5, 0.5, 0.5); // default light direction
        color = half3(1, 1, 1); // white color
        attenuation = 1.0;
    #else

        // URP
        #if defined(UNIVERSAL_LIGHTING_INCLUDED)
            Light mainLight = GetMainLight();
            color = mainLight.color;
            direction = mainLight.direction;
            attenuation = mainLight.distanceAttenuation *
                          mainLight.shadowAttenuation;
        #endif
    #endif
}
```

Figure 5.52: GetMainLight custom function node source.

■ ColorValueReduce

This node is a sub graph and does nothing supernatural. The principle of its operation is as follows: convert RGB to HSV, reduce Value (green channel in split), and convert HSV to RGB. The graph can be seen on **Figure 5.53**.

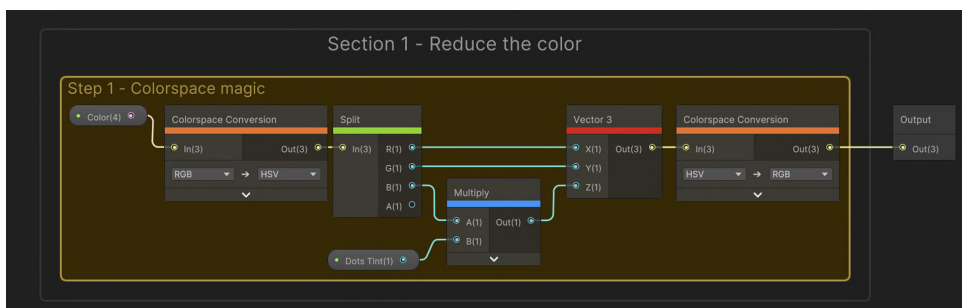


Figure 5.53: “ColorValueReduce” Shader Graph.

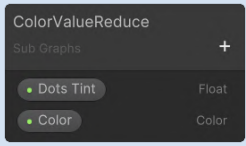
In Unity	Name	Type	Description
	COLOR	Color	Main color of the surface.
	DOTSTINT	Float	Adjusts how much the dots lose color in relation to the main color.

Table 5.14: “ColorValueReduce“ Shader Graph input parameters.

Now we can take a more in-depth look at the main graph itself.

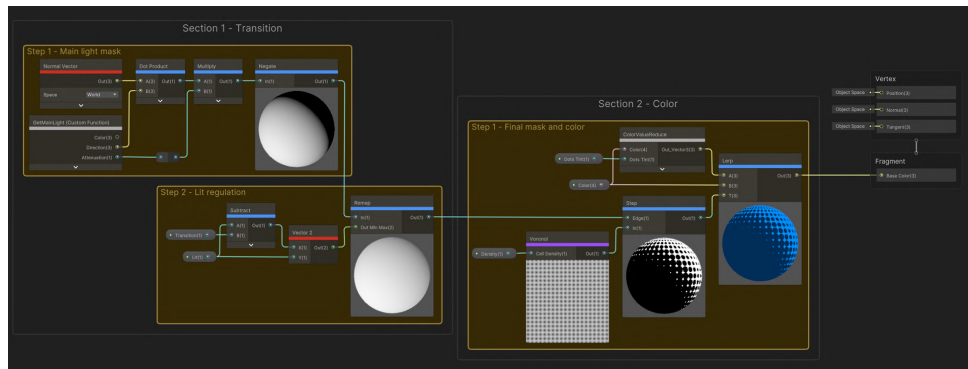


Figure 5.54: “HalfTone“ Shader Graph.

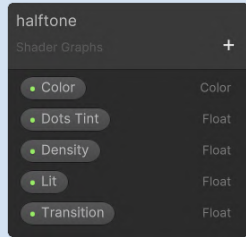
In Unity	Name	Type	Description
	COLOR	Color	Main color of the surface.
	DOTSTINT	Float [0, 1]	Adjusts how much the dots lose color in relation to the main color.
	DENSITY	Float	Determines the density of the dots.
	LIT	Float [0, 1]	How much the dots condense into the shadow.
	TRANSITION	Float [1, 3]	Determines the level of transition.

Table 5.15: “HalfTone“ Shader Graph input parameters.

Section one generates a shadow mask using the custom node Get Main Light (**Figure 5.55**). First, information about the main light in the scene is obtained, and then the Dot Product aligns the surface and light coordinates to calculate the shadow. The resulting value must be adjusted to fit within the range of -1 to 1. However, the user can adjust the output stream to modify the shadow transition and dot frequency.

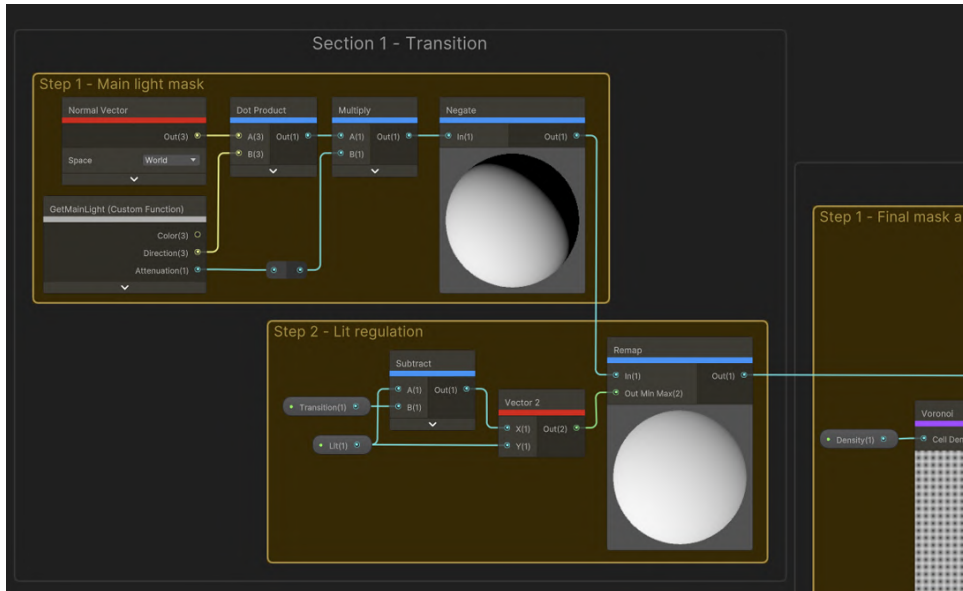


Figure 5.55: Section 1 of the “Halftone” Shader Graph.

The next section (**Figure 5.56**) is responsible for generating dots in the shadows by using a dot pattern, such as Voronoi, with the offset angle removed, and applying it as the Step parameter. The resulting mask is then used for coloring through Lerp, with the main color being darkened beforehand through custom node Color Value Reduce for the shadow effect.

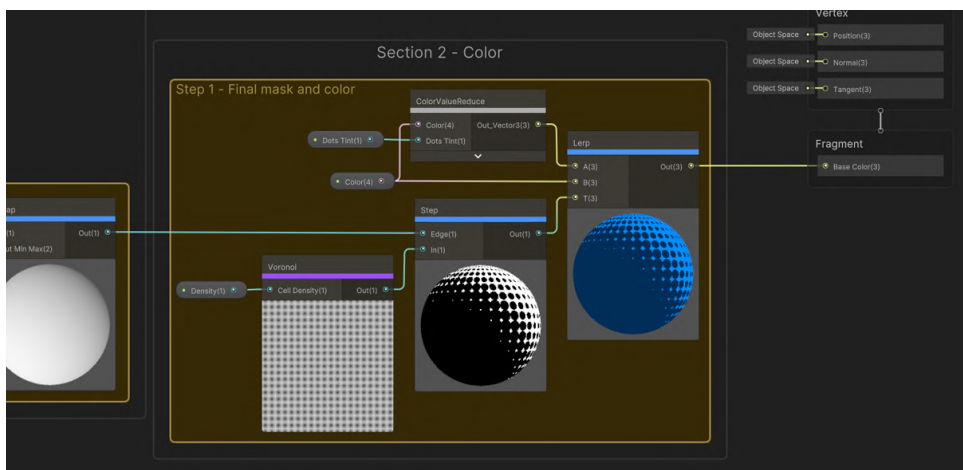


Figure 5.56: Section 2 of the “Halftone” Shader Graph.

5.13 Hologram

The “Hologram“ material is an effect that can be used to bring a futuristic or sci-fi element to a game. It can be applied to a wide range of objects, from characters and vehicles to environmental elements and props. It can help to create a sense of technology or advanced capabilities.

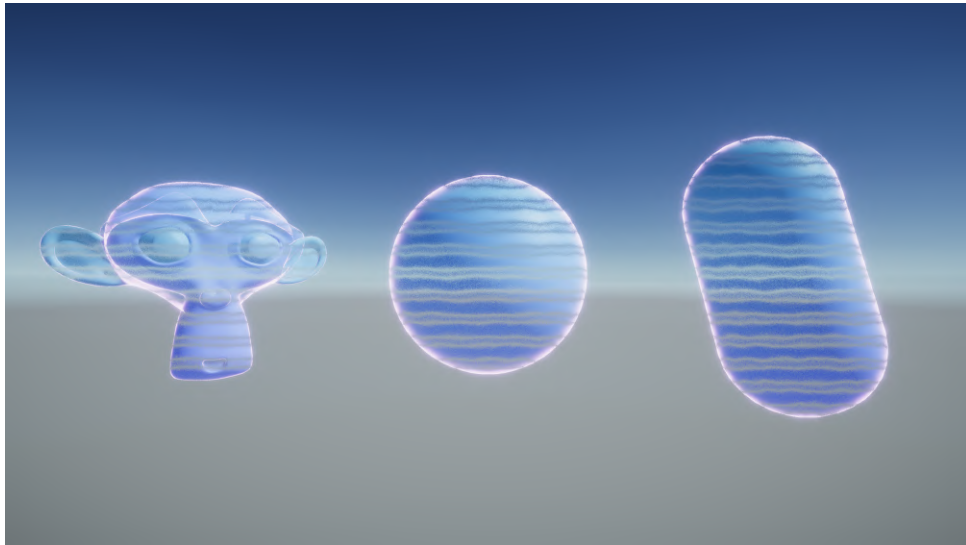


Figure 5.57: “Hologram“ Material on a variety of objects.

Holograms are an integral part of the sci-fi theme. They can be seen most often in movies, such as in Star Wars. In games they are also very common, for example in the game series Halo there is a character who is represented by a hologram - the AI “Cortana“ (Figure 5.37).



Figure 5.58: Cortana as a hologram in Halo 5.

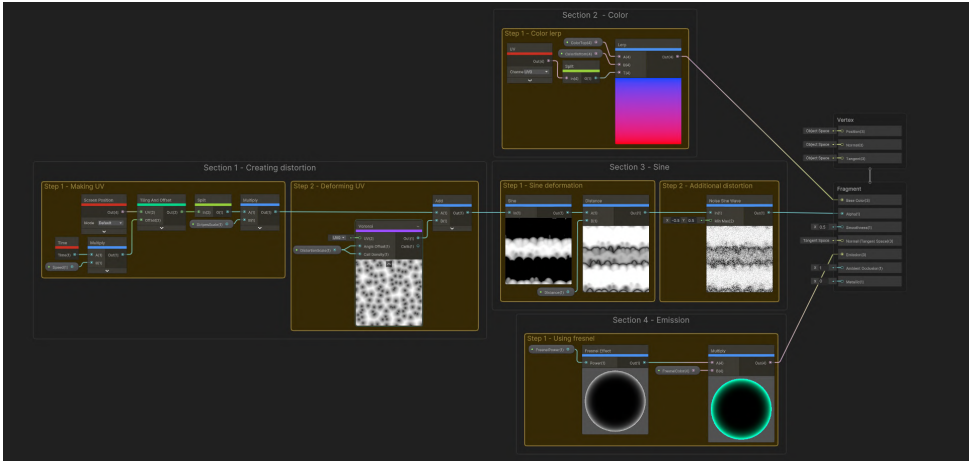


Figure 5.59: “Hologram“ Shader Graph.

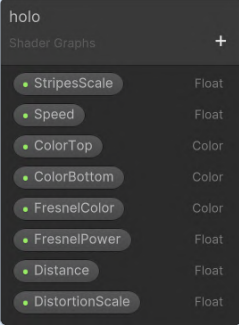
In Unity	Name	Type	Description
	STRIPESSCALE	Float	Stripes thickness.
	SPEED	Float [-0.1, 0.1]	Hologram update speed.
	COLORTOP	Color HDR	Main color of the surface for the gradient transition.
	COLORBOTTOM	Color HDR	Additional color of the surface for the gradient transition.
	FRESNELCOLOR	Color HDR	Fresnel effect color.
	FRESNELPOWER	Float [5, 10]	Fresnel effect intensity.
	DISTANCE	Float [0, 1]	Distance between stripes.
	DISTORTIONSCALE	Float [0, 20]	Hologram distortion amount.

Table 5.16: “Hologram“ Shader Graph input parameters.

5.14 Liquid

The “Liquid“ material can be used to simulate the appearance of a liquid substance within a game. The material’s wave effect can add an element of movement and dynamic motion to the game environment, creating the illusion of waves on the surface of the liquid.

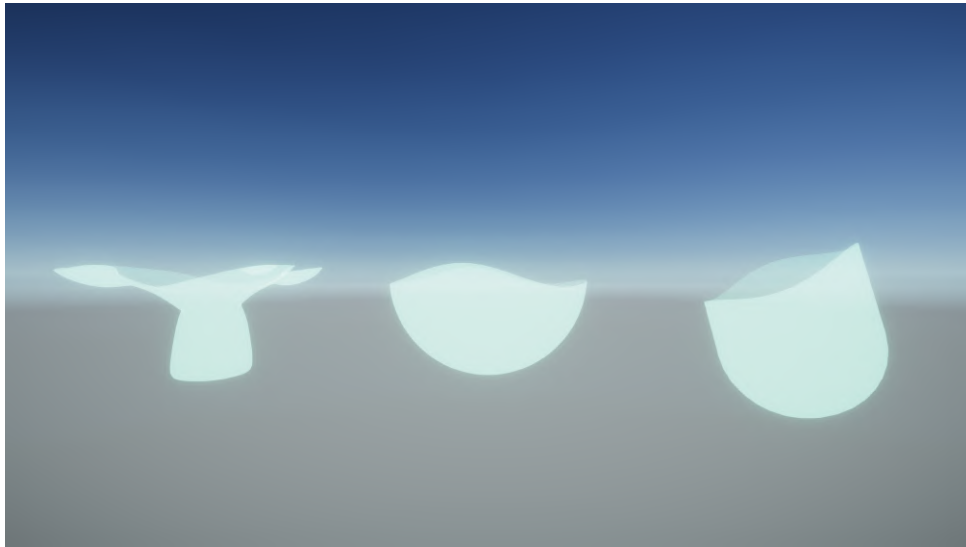


Figure 5.60: “Liquid“ Material on a variety of objects.

The fluid was once a stumbling block in game graphics. It is a very complex computational process. However, games prefer simulation, so this material only simulates the motion of waves based on a Sine wave. However, such a shader can be modified, for example by adding a tilt or velocity dependence.



Figure 5.61: Liquid in the bottle in Half-Life: Alyx (waves and the movement of the liquid depends on the movement of the bottle).

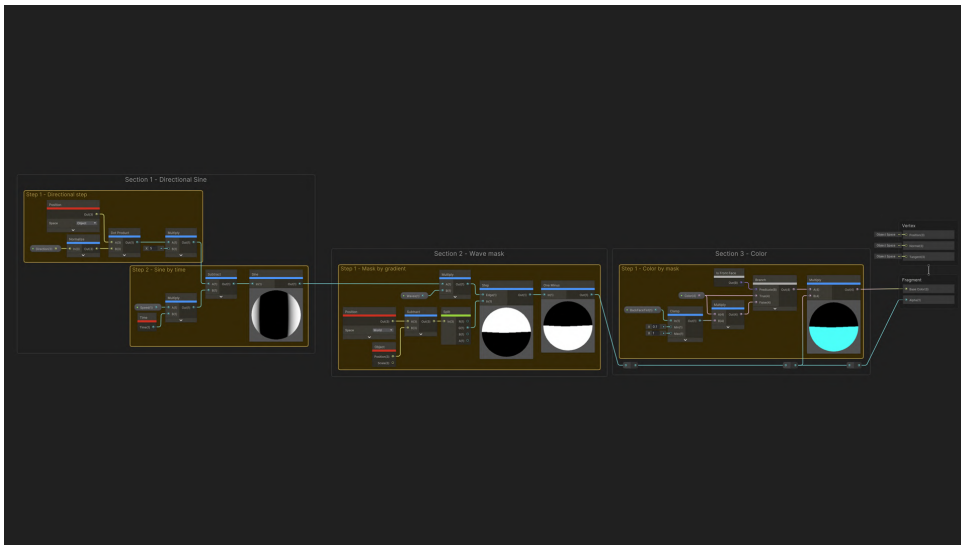


Figure 5.62: “Liquid” Shader Graph.

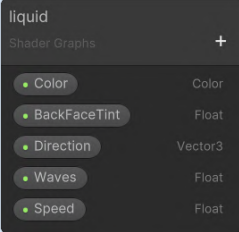
In Unity	Name	Type	Description
	COLOR	Color	Main color of the liquid.
	BACKFACETINT	Float [0, 1]	How much color fades when viewed “through” the liquid.
	DIRECTION	Vector3	Direction of the waves in world space.
	WAVES	Float [0.01, 0.1]	Waves strength.
	SPEED	Float [1, 10]	Controls speed of waves.

Table 5.17: “Liquid” Shader Graph input parameters.

5.15 Minecraft

This material is a posterized effect that can be used to create a pixelated appearance similar to that of the popular video game, Minecraft. It can be useful for creating games that have a voxel-style aesthetic.

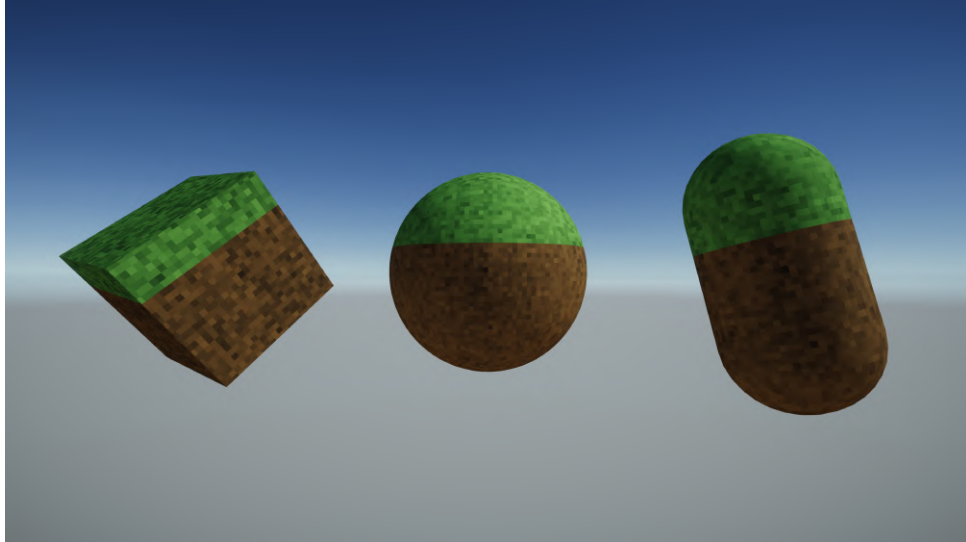


Figure 5.63: “Minecraft“ Material on a variety of objects.

The inspiration is obvious - a block of grass in Minecraft (Figure 5.42). However, this material can be customized to a different palette, thus covering several more blocks of the famous game.



Figure 5.64: Grass block in Minecraft.

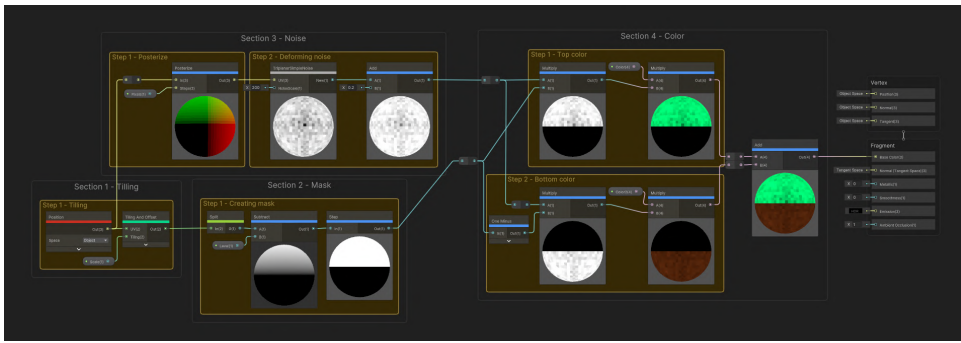


Figure 5.65: “Minecraft” Shader Graph.

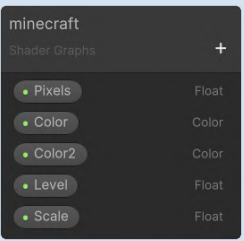
In Unity	Name	Type	Description
	PIXELS	Float	Pixel density.
	COLOR	Color	Main color.
	COLOR2	Color	Additional color.
	LEVEL	Float [0, 1]	Transition between both colors.
	SCALE	Float	Noise tiling.

Table 5.18: “Minecraft” Shader Graph input parameters.

This material will be broken down in detail in the following snapshots describing the generation process step by step.

The first and second sections are shown in **Figure 5.66** (below), where the first section is focused on controlling the texture’s scale. The output is then passed on to section two, where a level mask is created to determine the ratio of grass to ground. This is achieved by adjusting the gradient’s density. To remove the gray-scale effect, the Step node is utilized, resulting in a clearer transition.

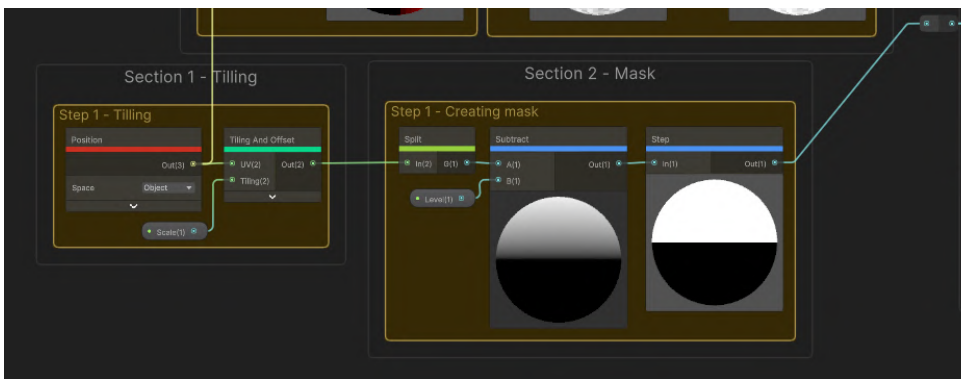


Figure 5.66: Sections 1 & 2 of the “Minecraft” Shader Graph.

In the third section (**Figure 5.67**), the focus is on pixelating the UV texture. This is achieved using the Posterize node, which allows the user to adjust the level of pixelation as needed. The pixelated UVs are then fed into the Triplanar Simple Noise node to generate the well-known noise effect. Finally, the Add node is used to slightly increase the overall brightness.

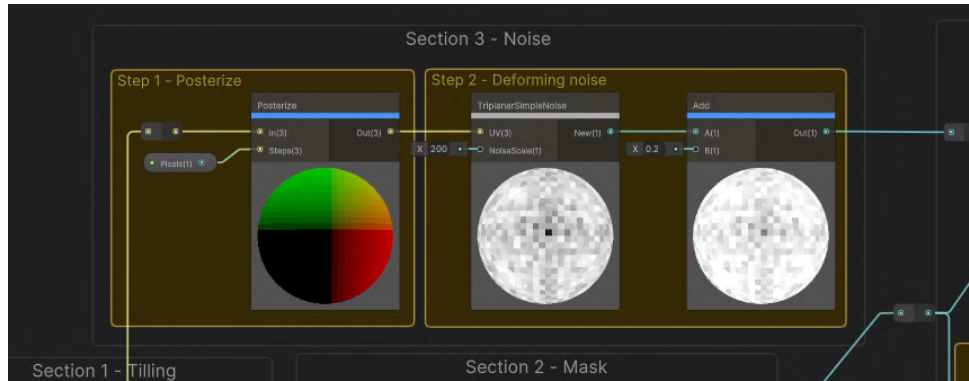


Figure 5.67: Section 3 of the “Minecraft” Shader Graph.

In the final section (**Figure 5.68**), the texture is colored according to the user’s preference. The level mask from the first step is used to divide the texture into two parts - the top (grass) and the bottom (ground). Once these parts are colored, they are merged and sent to Base Color.

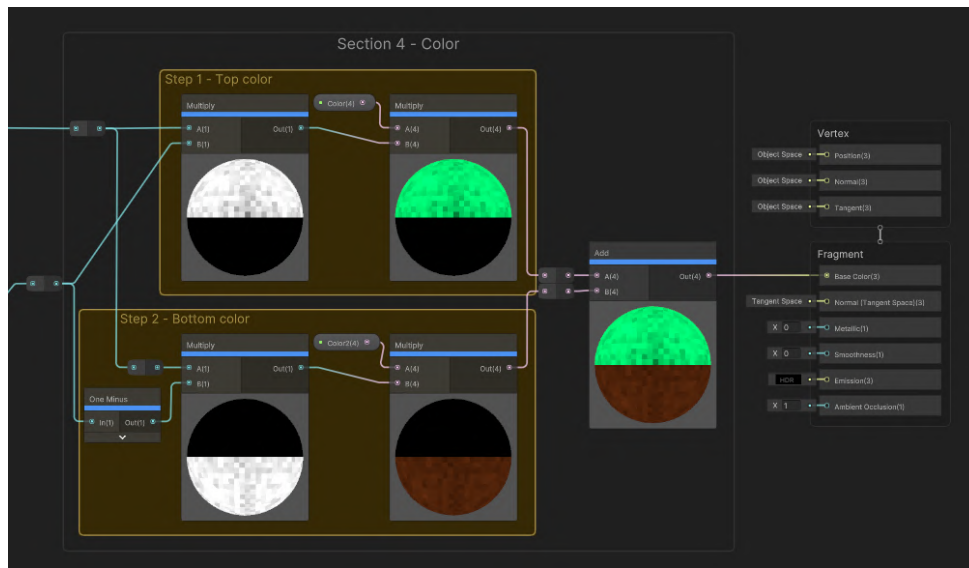


Figure 5.68: Section 4 of the “Minecraft” Shader Graph.

5.16 Mirage

The “Mirage“ material is an effect that can be used to create a visual distortion similar to a heat haze or mirage. It can be applied to environmental elements, such as deserts or hot urban environments, to create a sense of heat and distortion. The material features a controllable effect that can be adjusted to fit the desired look and feel of the scene (e.g. plane engines or fire).

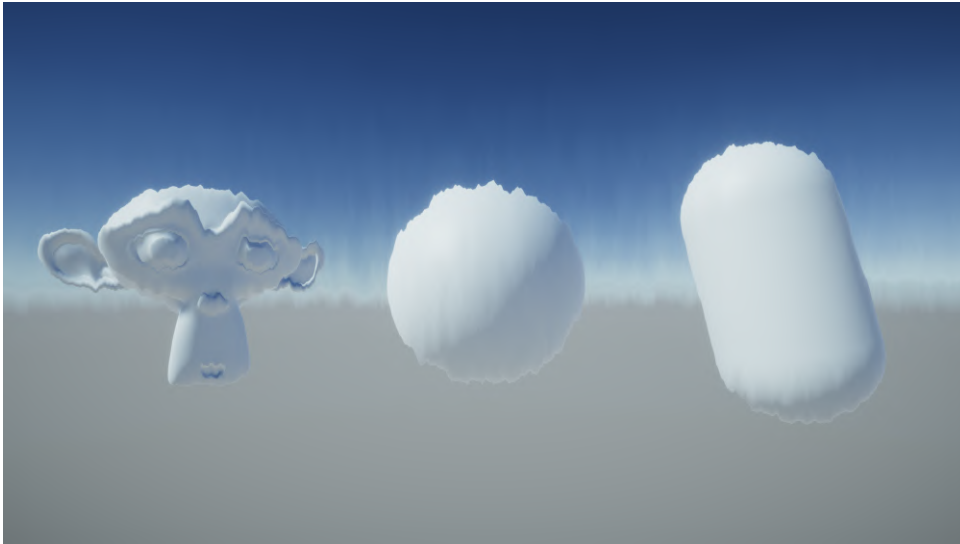


Figure 5.69: “Mirage“ Material applied on a plane near the camera, so the whole view is distorted.

The inspiration came from the frequent flying - each plane has this obscuration when the engines are on, about the same as in Figure 5.53. When the engines accelerate, the heat is released faster, thus the distortion changes its appearance, which can also be adjusted in the created material.



Figure 5.70: Heat Haze near plane engines by Scott Barbour [31].

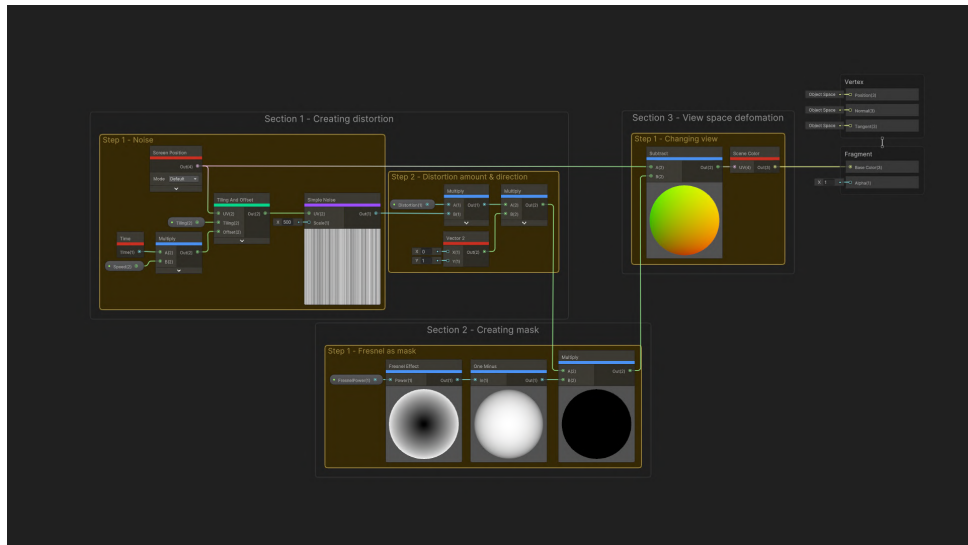


Figure 5.71: “Mirage” Shader Graph.

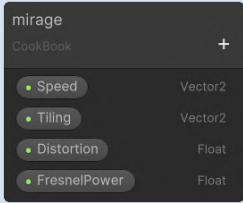
In Unity	Name	Type	Description
	SPEED	Vector2	Main color of the surface.
	TILING	Vector2	Tiling of the distortion texture.
	DISTORTION	Float	Distortion amount on heat haze.
	FRESNELPOWER	Float	Adjusts the mask transition (to control transparency at the edges of the mesh, when using material on objects placed in the world).

Table 5.19: “Mirage” Shader Graph input parameters.

5.17 PewDiePie

The “PewDiePie“ material is inspired by the iconic red-black striped pattern (figure 5.55 below) often associated with the popular YouTuber Felix’s Arvid Ulf Kjellberg (known as PewDiePie [32]). The material’s repeating stripes can add a touch of personality and pop culture reference to a game environment.

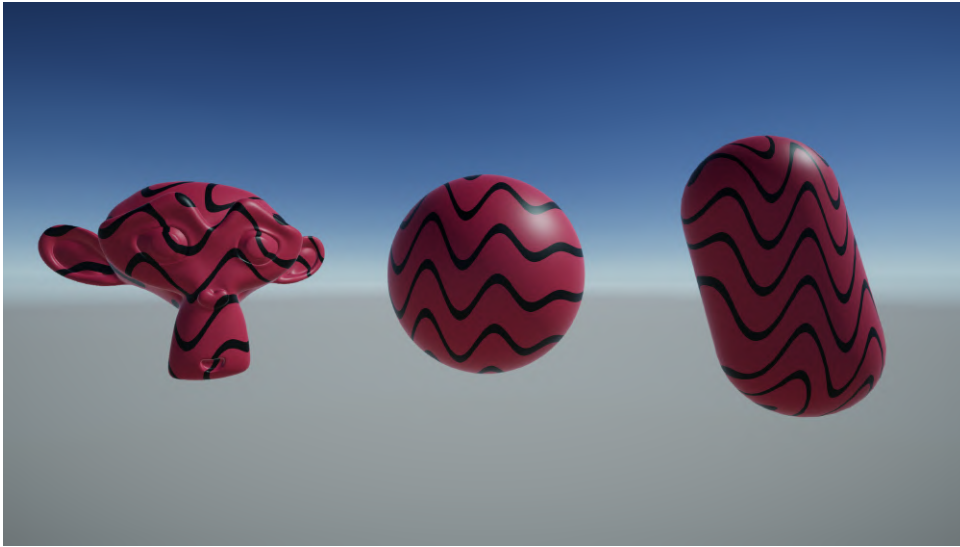


Figure 5.72: “PewDiePie“ Material on a variety of objects.

The material’s bold colors and high contrast make it easily recognizable (**Figure 5.73**). It changes over time, being distorted by sinus waves, and does not depend on the point of view, which makes it quite stand out from its surroundings.



Figure 5.73: PewDiePie’s YouTube channel pattern. Used as a YouTube banner image, as a screen saver while waiting for the start of the stream, and as a recognizable texture for branded merchandise [33].

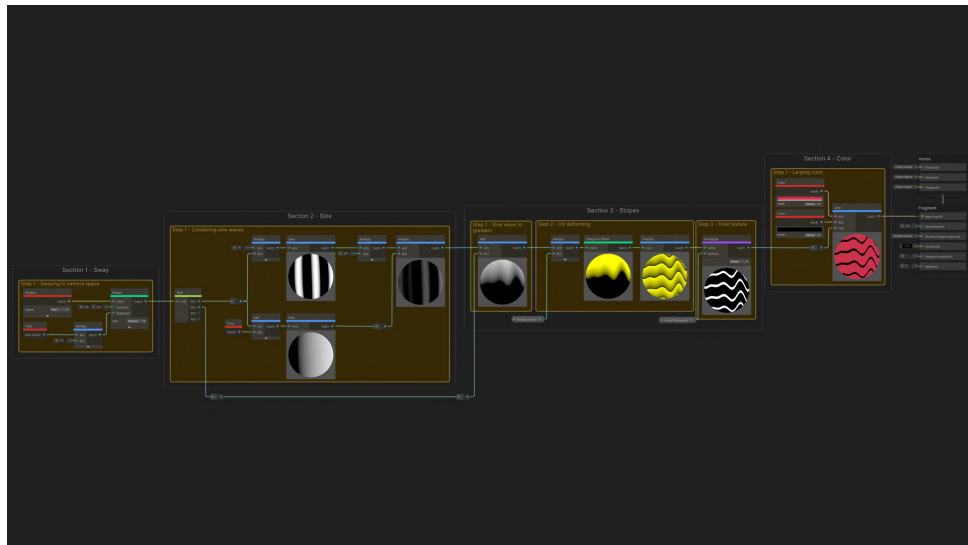


Figure 5.74: “PewDiePie“ Shader Graph.

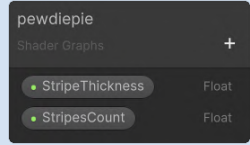
In Unity	Name	Type	Description
	STRIPETHICKNESS	Float	Controls the thickness of individual stripe.
	STRIPESCOUNT	Float	Regulates the count of stripes.

Table 5.20: “PewDiePie“ Shader Graph input parameters.

5.18 Soap

The “Soap“ material is a transparent, reflective material that can be used to add a touch of realism to objects such as bubbles or other translucent objects. Reflective properties allow it to catch and reflect light in a way that accurately simulates the way a real soap bubble would behave.

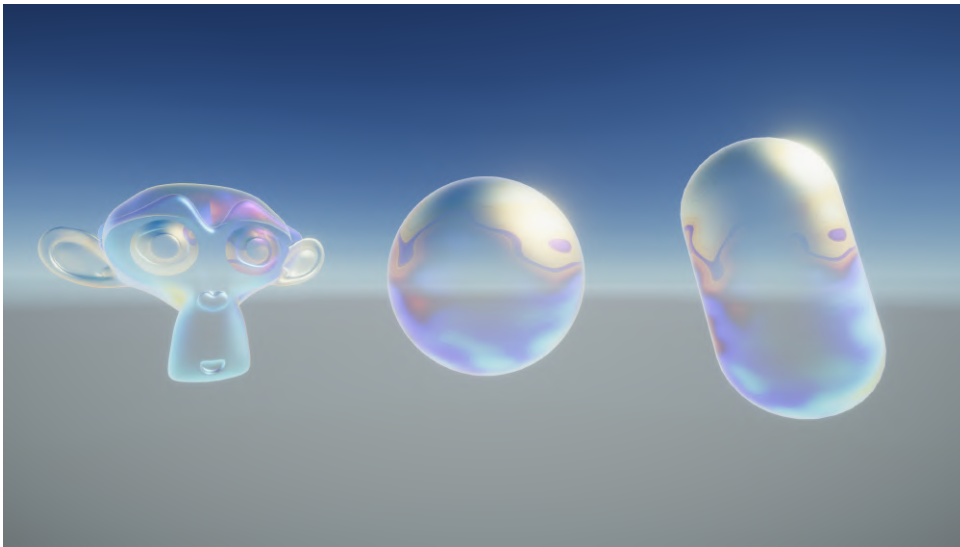


Figure 5.75: “Soap“ Material on a variety of objects.

The inspiration for a graph came from the ever-changing iridescence texture of a common soap bubble (such as in **Figure 5.76**). The iridescence on soap bubbles arises due to the interference of light waves. When light passes through the thin layer of soap film, some of it is reflected from the top surface of the film, while some are refracted and reflected from the bottom surface. The reflected waves from both surfaces interfere with each other, producing a pattern of colors that vary based on the thickness of the film [34].



Figure 5.76: Group of soap bubbles with good visible film [35].

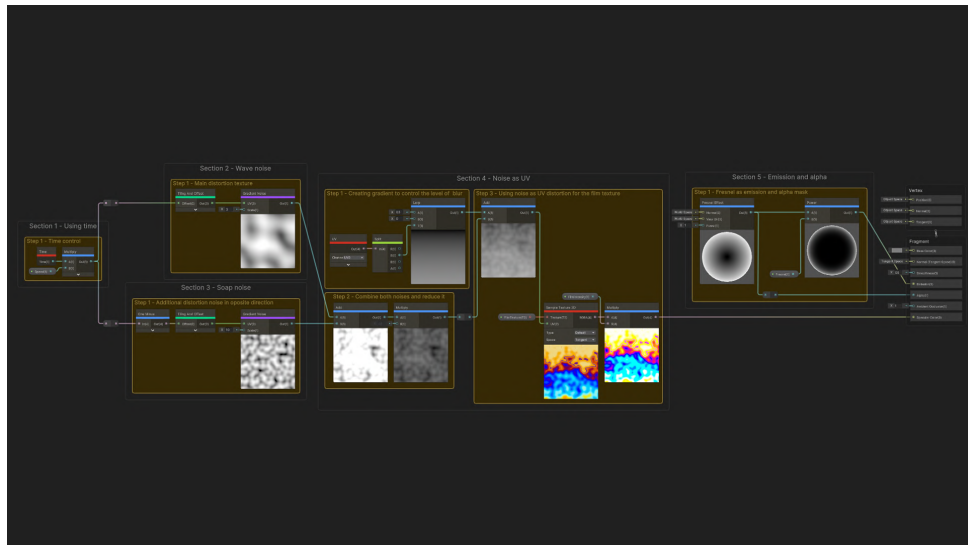


Figure 5.77: “Soap“ Shader Graph.

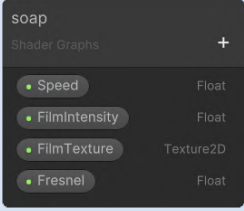
In Unity	Name	Type	Description
	SPEED	Float [0, 1]	How quickly the noise texture will change over time.
	FILMINTENSITY	Float [5, 10]	Primary color intensity.
	FILMTEXTURE	Texture2D	The texture that will be used as the main color.
	FRESNEL	Float [2, 6]	Glow intensity.

Table 5.21: “Soap“ Shader Graph input parameters.

5.19 Toon

A “Toon“ material gives objects in a game a cartoon-like appearance. It is characterized by its simplified lighting and shading techniques, which typically include bright, flat colors and sharp outlines (commonly named *Cel shading*). The Toon shader is often used in games that want to evoke a particular visual aesthetic or create a specific mood (e.g. in a lighthearted tone).

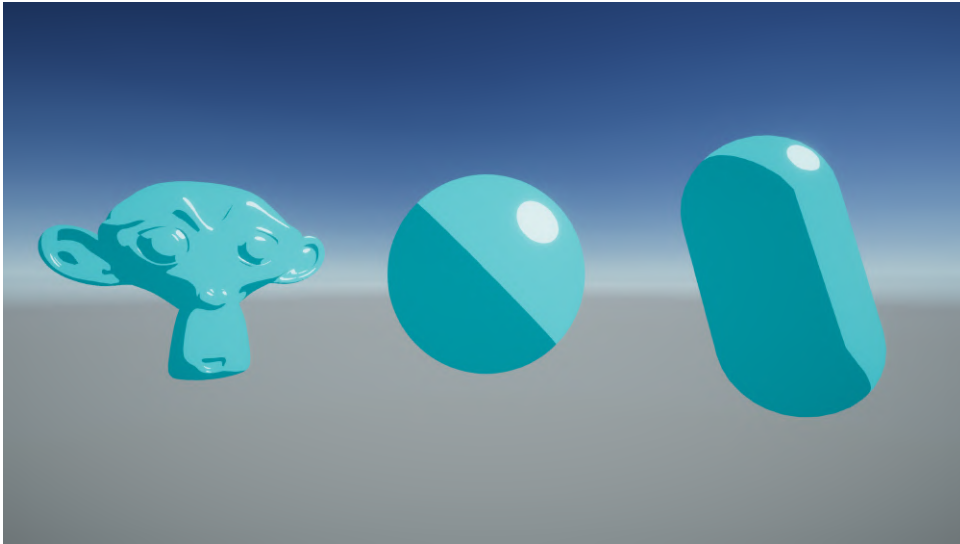


Figure 5.78: “Toon“ Material on a variety of objects.

The inspiration for this material can be gained from anywhere, thankfully there are plenty of games with this kind of visuals. Personally, I’ve most often seen the description of Toon shader based on his version in *The Legend of Zelda: Breath of the Wild* (**Figure 5.79**).



Figure 5.79: Cel shading in *The Legend of Zelda: Breath of the Wild*.

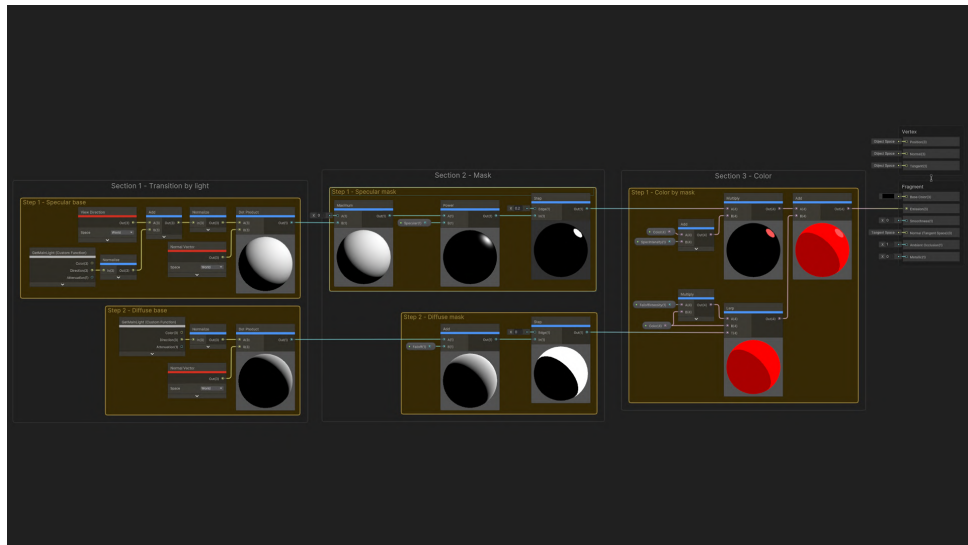


Figure 5.80: “Toon“ Shader Graph.

In Unity	Name	Type	Description
toon Shader Graphs + • Color Color • Falloff Float • FalloffIntensity Float • Specular Float • SpecIntensity Float	COLOR	Color	Main color of the surface.
	FALLOFF	Float [0, 1]	Cel shading transition level.
	FALLOFFINTENSITY	Float [0, 1]	How wide the stripes are.
	SPECULAR	Float	Specular part highlight.
	SPECULARINTENSITY	Float [0, 1]	Specular part color intensity (from slightly main color tint to white).

Table 5.22: “Toon“ Shader Graph input parameters.

5.20 Water

The material “Water“ is a stylized material that simulates waves on the surface of water. In contrast to the usual water surface, this one has a flow whose speed can be controlled.

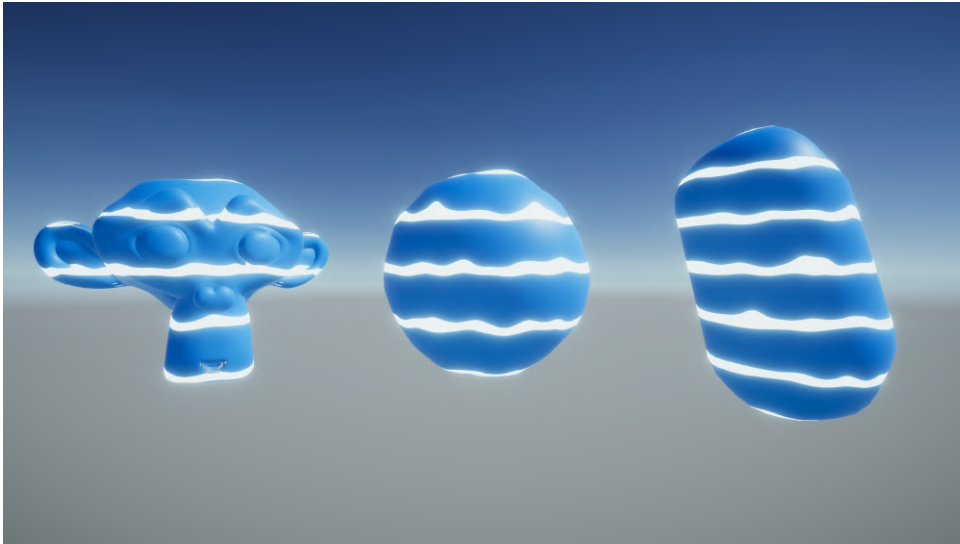


Figure 5.81: “Water“ Material on a variety of objects.

The inspiration came from several games, but the most interesting material was the water from *Zelda: Wind Waker* (see the **Figure 5.82** below). Often foam and waves are made with a Voronoi noise. However, the goal was to convey a simplified effect, keeping in mind that the potential user would use the material at, say, a river or waterfall, where the flow is clearly visible.



Figure 5.82: Water surface in *Zelda: Wind Waker*.

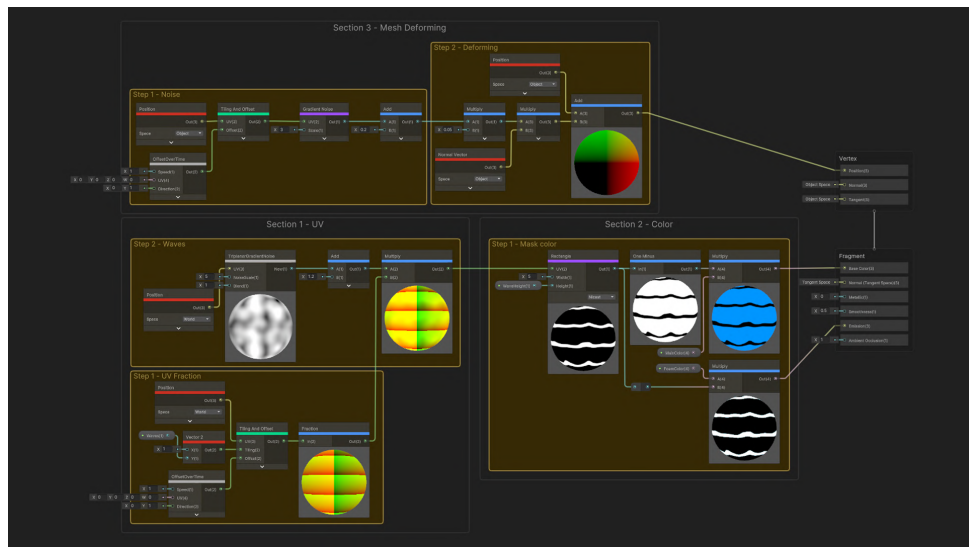


Figure 5.83: “Water“ Shader Graph.

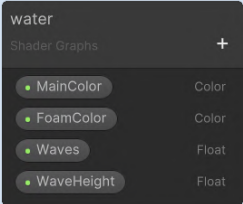
In Unity	Name	Type	Description
	MAINCOLOR	Color	Main color of the water surface.
	FOAMCOLOR	Color HDR	Color of the foam stripes.
	WAVES	Float [1, 10]	Controls the count of the waves.
	WAVEHEIGHT	Float [0.1, 0.5]	Regulates the wave strength.

Table 5.23: “Water“ Shader Graph input parameters.

In the first section (**Figure 5.84**), the focus is on generating the UV. The process utilizes a custom node named *Offset Over Time*, which creates dynamic waves that are dependent on the world location. A *Vector2* is also created to control the number of waves along the Y-axis. Finally, the information is transferred to *Fraction*. The UV can be distorted by multiplying it with *Triplanar Gradient Noise*, as shown in step 2.

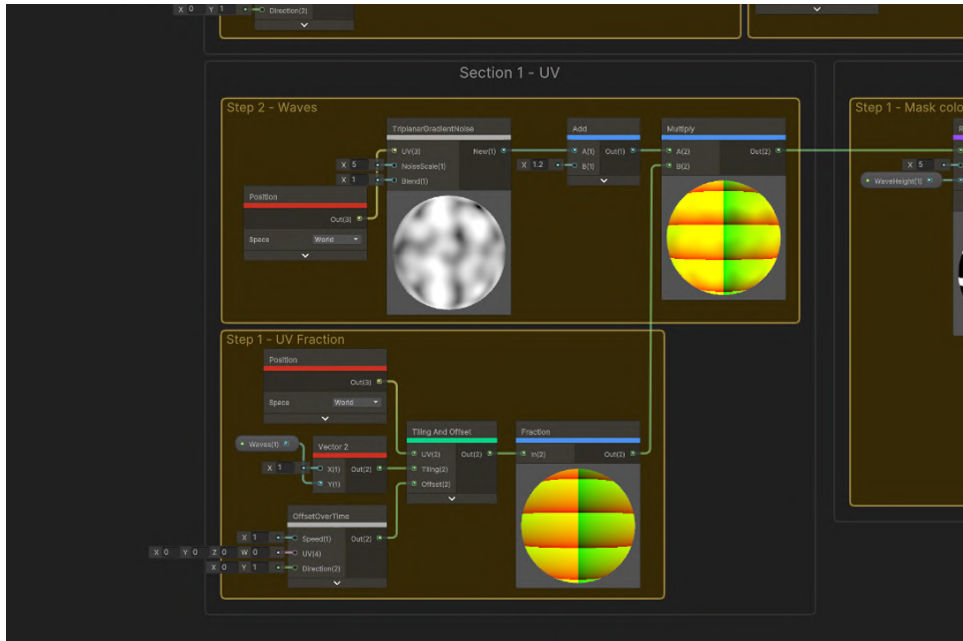


Figure 5.84: Section 1 of the “Water“ Shader Graph.

Section two creates the wave texture directly (**Figure 5.85**). The *Rectangle* node is called, stretching it by width and leaving control over the wave thickness to the user. Then, the created UV is applied, and the wave texture is ready. The resulting mask is used for *Emission*, making the foam waves brighter. Additionally, a mask for the main color is created using *One Minus*.

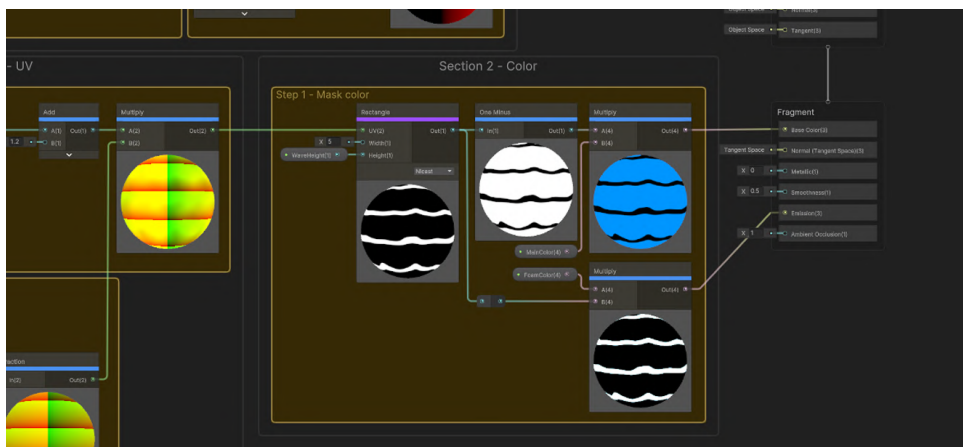


Figure 5.85: Section 2 of the “Water“ Shader Graph.

In the last section (**Figure 5.86**), the mesh is deformed to resemble waves. This effect is achieved by utilizing noise, similar to the method used with UV. However, in this case, the noise is applied to the normal vector of the vertex. By adding these changes to the vertex position in the object space, the shape of the surface is altered, creating a wave-like appearance (with a magnitude of 0.05 in step 2). The resulting output is then applied to the Position.

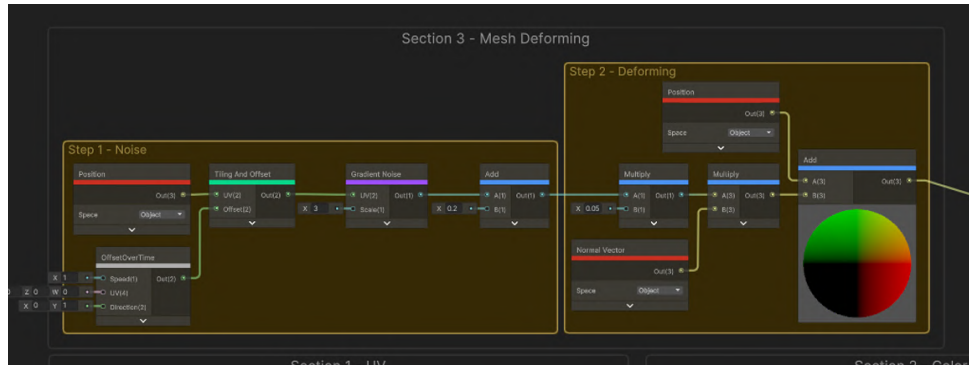


Figure 5.86: Section 3 of the “Water“ Shader Graph.

5.21 Wood

The “Wood“ material is a wood texture that can be applied to objects in a scene to add a touch of authenticity. This material is particularly well-suited for use on objects that are meant to resemble wooden surfaces, such as furniture, doors, or floor.

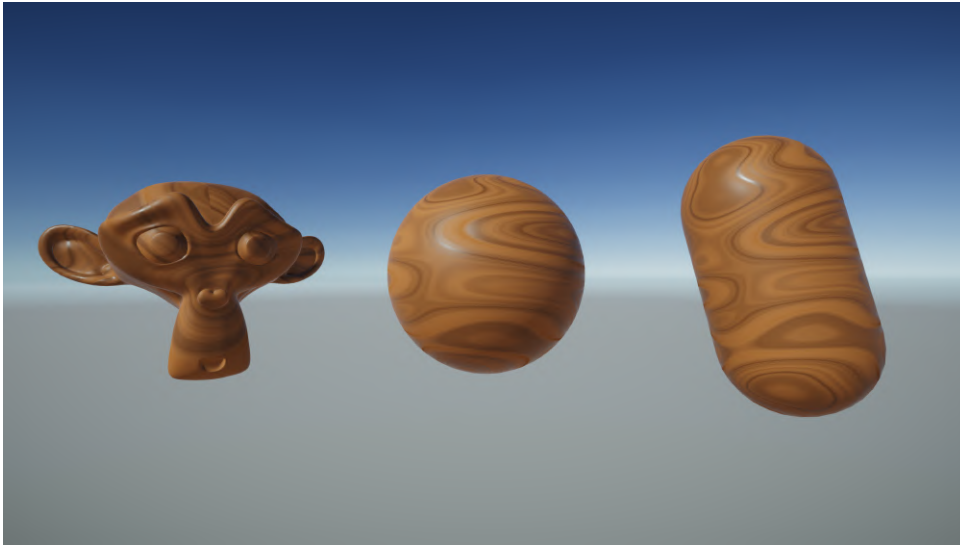


Figure 5.87: “Wood“ Material on a variety of objects.

There was no special inspiration here, the goal was to create a texture with noise that would look the most believable when a gradient was applied to it. Most wood (treated) has noticeably blurred rings on the surface with a distinct color transition (as in the **Figure 5.88** below).



Figure 5.88: Photo of the wooden surface from close up by FWStudio [36].

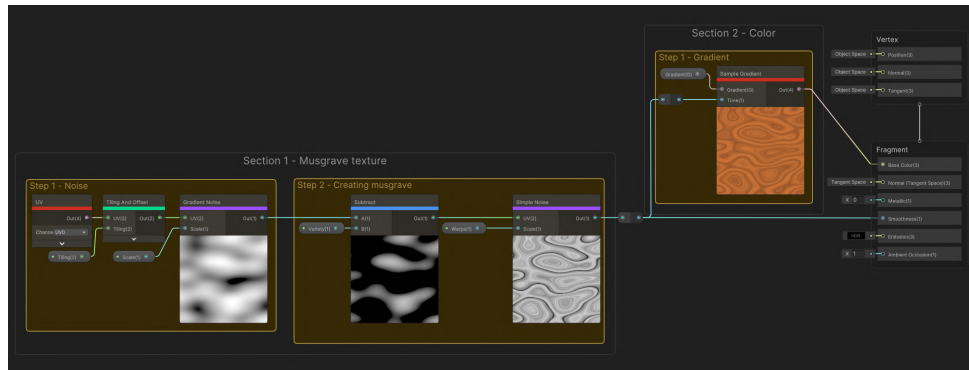


Figure 5.89: “Wood“ Shader Graph.

In Unity	Name	Type	Description
	GRADIENT	Gradient	The basic color palette for covering Musgrave texture ([144, 42, 10] has been chosen).
	TILING	Vector2	Noise texture tiling.
	SCALE	Float	Overall scale of a texture.
	VARIETY	Float [0, 1]	Transition in curvatures parts.
	WARPS	Float	Noise distortion adjustment.

Table 5.24: “Wood“ Shader Graph input parameters.

The first section creates noise of the Musgrave type [37] by calling Gradient Noise, subtracting some number of transitions from it, and finally using it as the UV coordinates of Simple Noise (Figure 5.90). The result is a texture with circular transitions, similar to wood. This texture will also be responsible for Smoothness, which will give more impact.

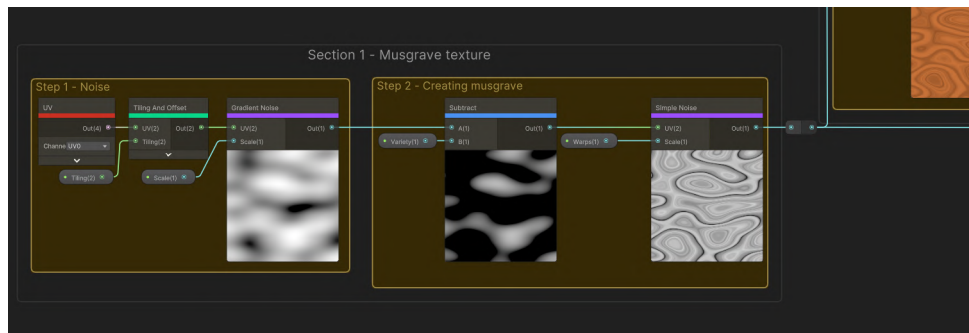


Figure 5.90: Section 1 of the “Wood“ Shader Graph.

Section two (**Figure 5.91**) only takes care of the color, here applies the Gradient and Mask, the texture as a mask that was made in the first section.

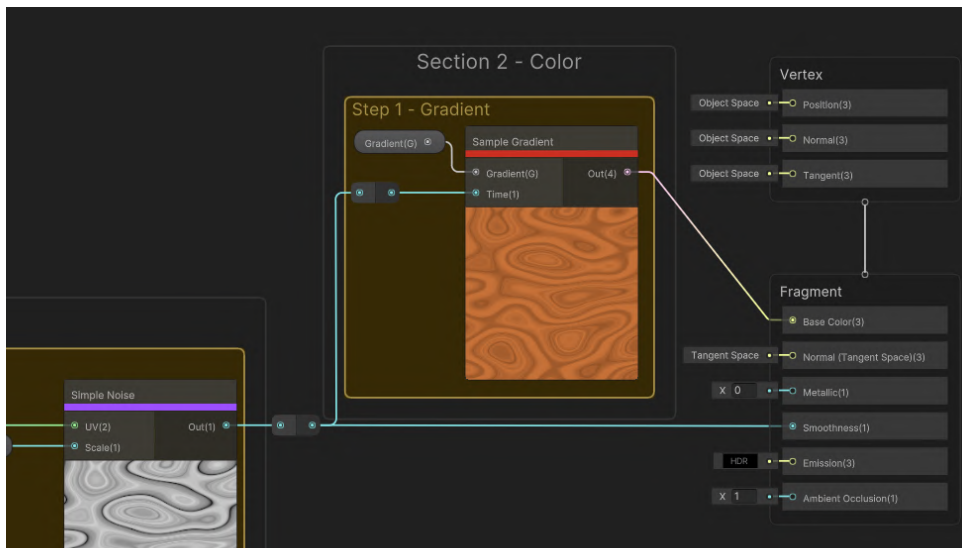


Figure 5.91: Section 2 of the “Wood“ Shader Graph.

5.22 Metal

The material with the uncomplicated name “Metal“ was created without a focus on stylizing, but on creating a physical/chemical property inherent in certain metals.

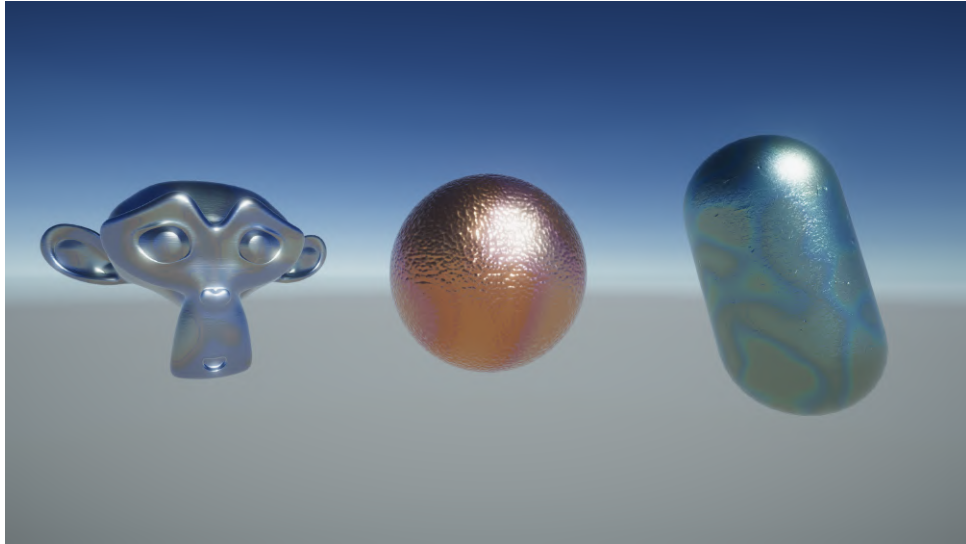


Figure 5.92: “Metal“ Material on a variety of objects (steel, copper, titanium).

The effect that can be seen in the metals shown above is “heat coloring“, which is a phenomenon that occurs when a metal surface is heated (**Figure 5.93**) and the temperature causes the metal to oxidize, resulting in a color change [38]. This effect is commonly seen in metals such as steel, copper, and titanium.



Figure 5.93: Examples of heat coloring on metals: steel by Jeffrey H Dean [39], copper by Mari [40], titanium by Robert Lopez [41].

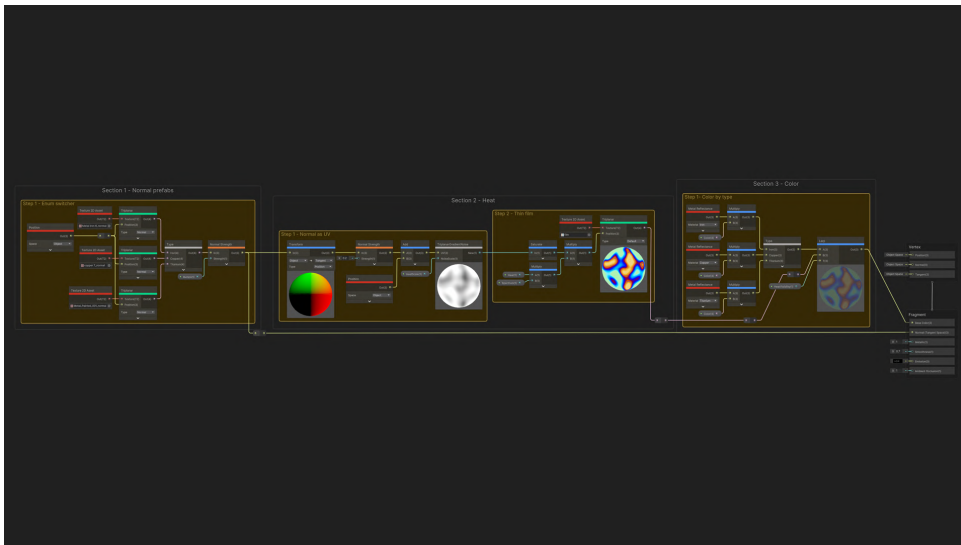


Figure 5.94: “Metal“ Shader Graph.

In Unity	Name	Type	Description
metal	COLOR	Color	Main color of the surface.
Heat	HEAT	Float [0, 1]	The intensity of the heat spectrum.
HeatVisibility	HEATVISIBILITY	Float [0, 1]	Visibility of the heat.
HeatScale	HEATSCALE	Float	Controls the scale of the noise.
Spectrum	SPECTRUM	Float [0, 1]	How much spectrum will be used for heating.
Bumps	BUMPS	Float [0, 1]	The intensity of the normal map.
Type	TYPE	Enum	The keyword responsible for the type of metal.

Table 5.25: “Metal“ Shader Graph input parameters.

Section one generates presets of triplanar normal maps for three different types of metals (top to bottom: steel, copper, and titanium) using the Enum switch (**Figure 5.95**). The user can select the desired metal type and control the intensity of the normal maps with an input parameter via the Normal Strength node. The output is connected to the next section and directly to the Normal channel.

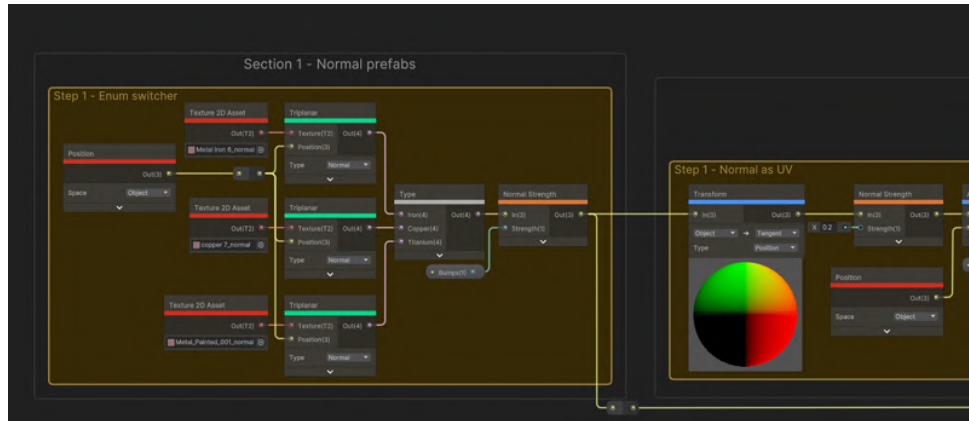


Figure 5.95: Section 1 of the “Metal“ Shader Graph.

The next section (**Figure 5.96**) is dedicated to the creation of noise that simulates the oxidation of metal when exposed to heat. Firstly, the normal map is converted into Tangent space to ensure accurate and consistent shading on the surface. This is because Tangent space aligns with the surface geometry, ensuring the normal map represents surface details regardless of the viewing angle. The output is then combined with the Position Object and used as UV coordinates for the Triplanar Gradient Noise. This approach utilizes several surface properties to generate the noise, improving the material’s perception. In the final step, a heating spectrum is obtained by utilizing noise as UV. This spectrum can be customized by the user through two input parameters. Before that, any negative values are removed via Saturate node to prevent any overlap in the resulting color palette.

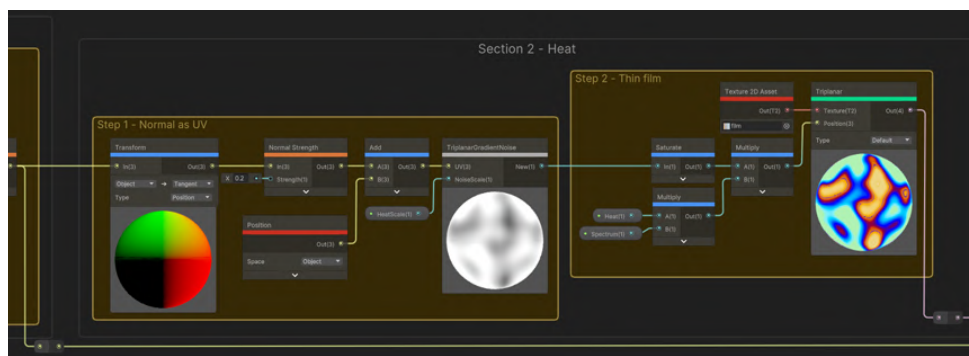


Figure 5.96: Section 2 of the “Metal“ Shader Graph.

The last section allows the user (again with the Enum) to select the main color of the metal (**Figure 5.97**), which will be affected by the Metal Reflectance node, which outputs the colored specular highlights on certain metals. Now it only remains to use spectrum and ready color through Lerp, the visibility parameter available to the user is used as a mask (by adjusting the value from 0 to 1).

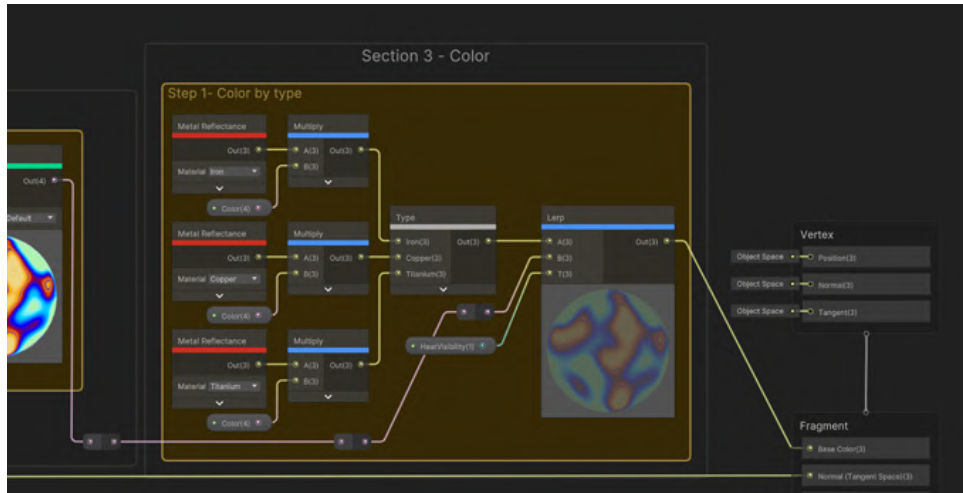


Figure 5.97: Section 3 of the “Metal“ Shader Graph.

5.23 Pearl

The “Pearl” material in Unity aims to replicate the basic properties of real pearls, such as their iridescence effect, smooth (or bumpy) surface, and unique color. Upon observing the surface of a pearl, it becomes evident that it possesses iridescence properties that were taken into consideration during the development of the Soap material (**Section 5.9**).

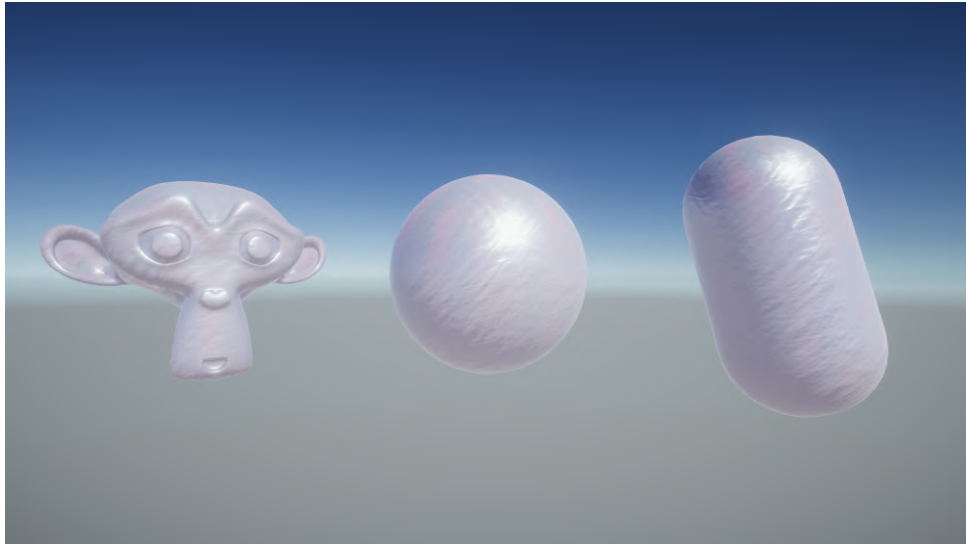


Figure 5.98: “Pearl” Material on a variety of objects.

Another term relevant to pearls is Pearlescence. The key difference between Iridescence and Pearlescence is in how light interacts with the surface of the material. While iridescence is caused by the interference of light waves and creates different colors depending on the viewing angle, Pearlescence is caused by the reflection and refraction of light within the material, giving it a pearly sheen (such as the one in the **Figure 5.99** below) [42].

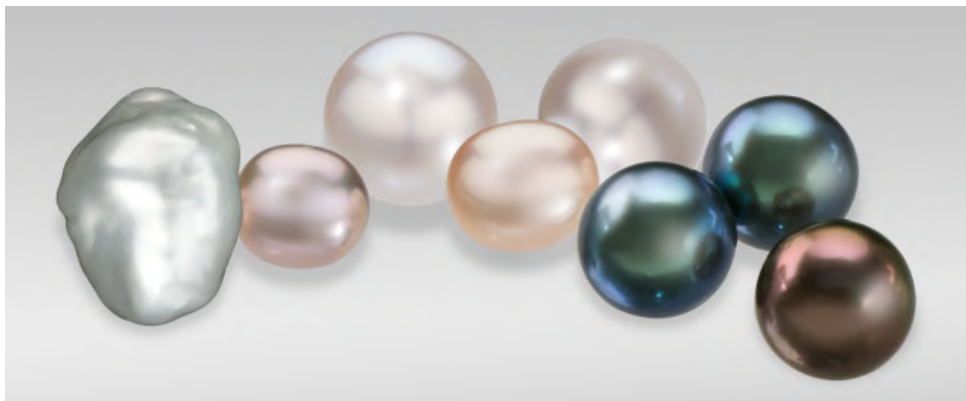


Figure 5.99: Different types of pearls by GIA.

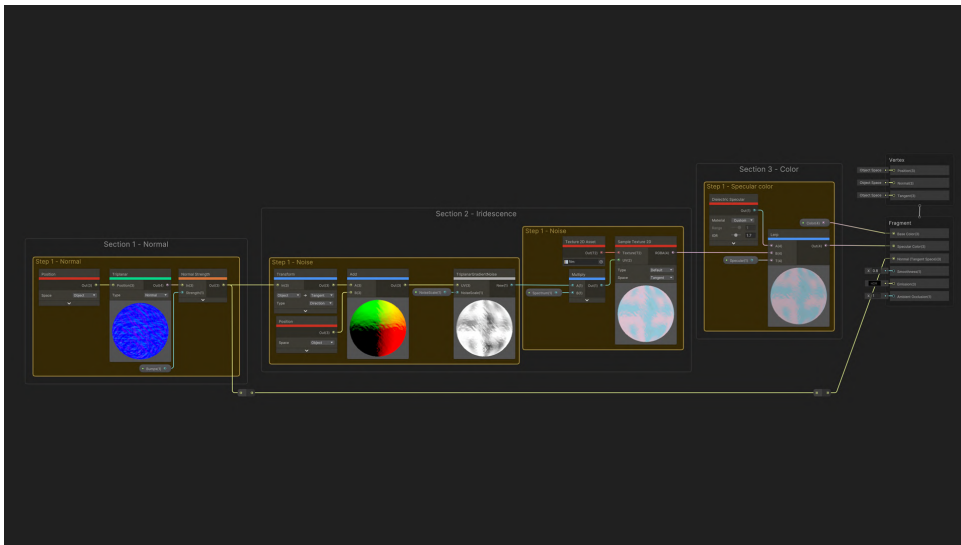


Figure 5.100: “Pearl” Shader Graph.

In Unity	Name	Type	Description
	COLOR	Color	Main color of the surface.
	SPECULAR	Float [0, 1]	Adjusts the mask level of the specular shade.
	SPECTRUM	Float [-0.1, 0]	How much spectrum will be used for iridescence.
	NOISESCALE	Float	Controls the scale of the iridescence noise.
	BUMPS	Float [0, 1]	The intensity of the normal map.

Table 5.26: “Pearl” Shader Graph input parameters.

5.24 Glass

The “Glass“ material is designed to mimic the look of real-life distorted glass. This type of glass can be found in various places, such as old buildings, and is characterized by its wavy or uneven surface.

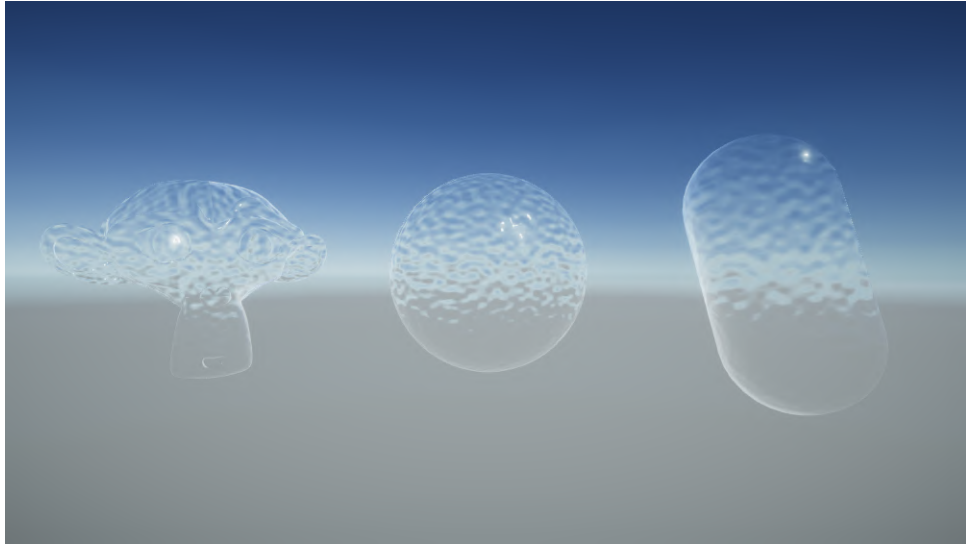


Figure 5.101: “Glass“ Material on a variety of objects.

When observing distorted glass up close (such as in **Figure 5.102** below), its pattern appears as a series of cavities that can be replicated using a gray-scale texture like Gradient Noise. The material is created using a combination of procedural noise textures and refraction effects, which give the appearance of light being bent as it passes through the glass.



Figure 5.102: Photo of the distorted glass by Image*After [44].

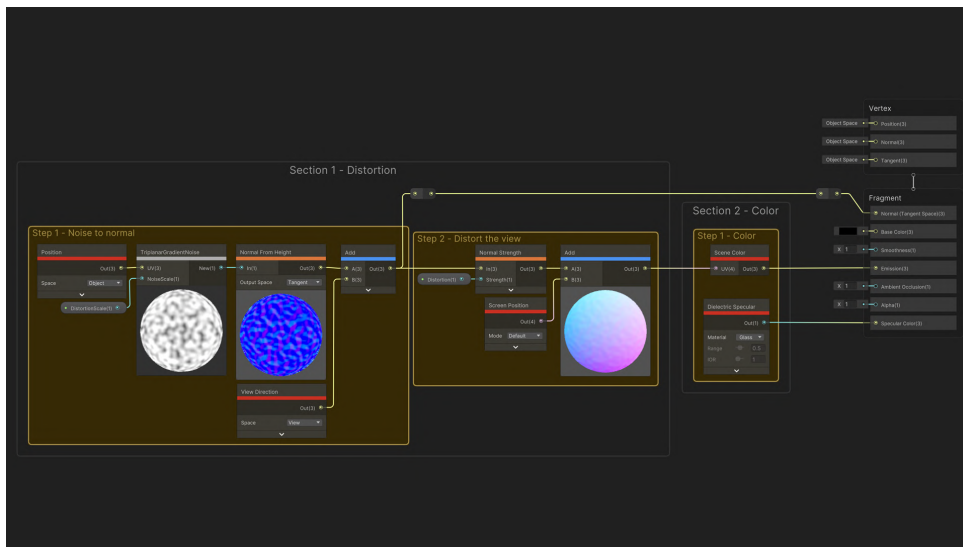


Figure 5.103: “Glass” Shader Graph.

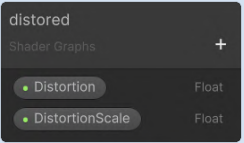
In Unity	Name	Type	Description
	DISTORTION	Float [0, 1]	Regulates the distortion intensity.
	DISTORTIONSCALE	Float [10, 20]	Noise scaling.

Table 5.27: “Glass” Shader Graph input parameters.



Chapter 6

Conclusion

The various issues involved in simulating the appearance of materials, with a focus on physically based models, have been explored. The support for creating materials using Shader Graphs in Unity has been described, and the possibilities of creating specialized, dynamically changing materials have been highlighted.

A sampler of basic, frequently used materials has been created using a Shader Graph. High-quality detail simulation has been achieved through the use of detail textures, noise functions, normal maps, and height maps.

To demonstrate the flexibility and versatility of the materials created, a test scene has been modeled where materials can be easily previewed and modulated.

Through this work, a solid understanding of the tools and techniques available in the Materials Library in Unity for creating high-quality materials and textures for use in games has been gained.

In the future, there are plans to expand on the materials created using Shader Graphs in Unity. Specifically, there is a desire to explore more advanced techniques for creating materials, such as the use of subsurface scattering. Additionally, there is a plan to incorporate more complex custom functions, as well as explore the use of procedural generation to create truly unique materials.

Furthermore, there is a goal to integrate the materials created into a larger game project, where they can be tested and refined. This will involve exploring how the materials behave in different environmental factors and conditions.



Bibliography

- [1] *We have you covered with the Measured Materials Library*, Edward Martin, Luc Vo Van, 08.02.2019. <https://blog.unity.com/manufacturing/we-have-you-covered-with-the-measured-materials-library>
- [2] *Materials*, Joey de Vries, Learn OpenGL. <https://learnopengl.com/Lighting/Materials>
- [3] *Blender shader nodes*, Blender 3.5 Manual. https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html
- [4] *Types of textures*, Matuchnov I., Black VR, 2021. <https://blackvr.org/tpost/a23mydd5bc-tipi-tekstur>
- [5] *Render pipeline feature comparison*, Unity Technologies, 2022. <https://docs.unity3d.com/2021.3/Documentation/Manual/render-pipelines-feature-comparison.html>
- [6] *Textures and Materials for Rainbow6*, Alina Ivanchenko CG Artist. <https://foxfrombox.com/projects/A1qAz>
- [7] *Blender 2.8 Procedural PBR Textures for beginners*, Learn OpenGL, @Jayanam, 2019. <https://www.youtube.com/watch?v=2Ea7JKvTYhg>
- [8] *Sub Graph*, Unity Technologies. <https://docs.unity3d.com/Packages/com.unity.shadergraph@7.1/manual/subgraph.html>
- [9] *Real-Time Rendering*, Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michal Iwanicki, Sebastien Hillaire, 1999.
- [10] *Unity ShaderGraph Cookbook vol.1*, Kamosoba, 2021. https://zenn.dev/r_ngtm/books/shadergraph-cookbook/viewer/lookup-of-recipe
- [11] *Phong reflection model*, Wikipedia. https://en.wikipedia.org/wiki/Phong_reflection_model

- [12] *Illustration of the components of the Phong reflection model (Ambient, Diffuse, and Specular reflection)*, Brad Smith, 2006.
https://commons.wikimedia.org/wiki/File:Phong_components_version_4.png
- [13] *About Shader*, Unity Technologies.
<https://docs.unity3d.com/Manual/Shaders.html>
- [14] *Metallic Vs Specular*, Unity Technologies.
<https://docs.unity3d.com/Manual/StandardShaderMetallicVsSpecular.html>
- [15] *URP_Materials_Library Unity Project*, GitHub.
https://github.com/ScotTishCyclopSS/URP_Materials_Library
- [16] *Biological Waste Processing Plant*, The Half-Life & Portal Encyclopedia.
https://half-life.fandom.com/wiki/Biological_Waste_Processing_Plant
- [17] *Green Canes*, Stardew Valley Wiki, 2021.
https://stardewvalleywiki.com/Green_Canes
- [18] *Claymation*, Adobe.
<https://www.adobe.com/creativecloud/animation/discover/claymation.html>
- [19] *Claymation style animation in Blender 2.81*, J Middleton on YouTube.
<https://www.youtube.com/watch?v=omXGa9AJHPY>
- [20] *UE4 Transition Effect Shader*, TGA Digital on YouTube.
https://www.youtube.com/watch?v=_vGLVXHEQDQ
- [21] *Stylized Low-poly Wooden beam*, Rocco Giandomenico on Sketchfab.com.
<https://sketchfab.com/3d-models/stylized-low-poly-wooden-beam-01b16abea26646c8ba334c905374f83e>
- [22] *Stylized Modular Fireplace*, Michalina "Miszla" Gąsienica-Laskowy on Sketchfab.com.
<https://sketchfab.com/3d-models/stylized-modular-fireplace-696bfa98c2b34bc3b962151ba3304928>
- [23] *Triplanar Projection*, Adobe (Substance Painter), 2022.
<https://substance3d.adobe.com/documentation/spdoc/tri-planar-projection-180191954.html>
- [24] *Triplanar Mapping*, Martin Palko, 20/03/2014.
<https://www.martinpalko.com/triplanar-mapping/>
- [25] *Triplanar Node*, Unity Technologies.
<https://docs.unity3d.com/Packages/com.unity.shadergraph@8.1/manual/Triplanar-Node.html>

- [26] *Frostpunk Posters*, 11-bit Studios.
<https://displate.com/11bit/frostpunk>
- [27] *Space galaxy background*, by rawpixel.com.
https://www.freepik.com/free-vector/space-galaxy-background_4413893.htm#query=spacebackground
- [28] *Retrowave Neon 80's Background - 4K*, Rafael-De-Jongh, 16/08/2017.
<https://www.deviantart.com/rafael-de-jongh/art/Retrowave-Neon-80-s-Background-4K-699082710>
- [29] *How to Make Amazing Halftone Effects with Photoshop*, David Blatner, 10/06/2015.
<https://creativepro.com/how-to-make-amazing-halftone-effects-with-photoshop/>
- [30] *GetMainLight()* source, Unity Technologies.
<https://github.com/Unity-Technologies/Graphics/blob/master/Packages/com.unity.render-pipelines.universal/ShaderLibrary/Lighting.hlsl>
- [31] *Aircraft Engines*, Scott Barbour, Getty Images.
<https://www.gettyimages.in/detail/news-photo/exhaust-emits-from-the-engines-of-a-passenger-jet-as-it-news-photo/57214655>
- [32] *@PewDiePie user's profile*, YouTube.com.
<https://www.youtube.com/@PewDiePie>
- [33] *Pewdiepie Stream Asset*, dspall.
<https://www.dspall.work/portfolio/pewdiepie>
- [34] *A Practical Extension to Microfacet Theory for the Modeling of Varying Iridescence.*, Laurent Belcour, Pascal Barla, ACM Transactions on Graphics, 2017, 36 (4), pp.65. doi:ff10.1145/3072959.3073620ff.fhal-01518344v2
- [35] *Soap bubbles*, Rapeepong Puttakumwong, Getty Images.
<https://www.gettyimages.nl/detail/foto/the-rainbow-soap-bubbles-from-the-bubble-blower-royalty-free-beeld/1319471891>
- [36] *Brown Wooden Surface*, FWStudio on www.pexels.com.
<https://www.pexels.com/photo/brown-wooden-surface-129733/>
- [37] *Musgrave Texture Node*, Blender 3D.
https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/musgrave.html
- [38] *Heat Tint (Temper) Colours on Stainless Steel surface Heated in air*, British Stainless Steel Association.
https://bssa.org.uk/bssa_articles/heat-tint-temper-colours-on-stainless-steel-surface-heated-in-air/

- [39] *Heat Coloring Steel: Heat Coloring Tips and Metal Art Course & Workshop*, Jeffrey H Dean.
<https://www.jeffreyhdean.com/heat-coloring-steel/>
- [40] *How to Flame Paint Durable Colors on Copper*, Mari on basketofblue.com.
<https://www.basketofblue.com/how-to-flame-paint-durable-colors-on-copper/>
- [41] *Heat Coloring A Titanium*, Robert Lopez on YouTube.com.
https://www.youtube.com/watch?v=uk7oZycB6_4
- [42] *Color Science in the Examination of Museum Objects: Nondestructive Procedures*, Ruth Johnston-Feller, 2001.
- [43] *Pearl Description*, Gemological Institute Of America.
<https://www.gia.edu/pearl-description>
- [44] *Distorted glass texture*, Image*After.
<http://www.imageafter.com/image.php?image=b19glass014.jpg>