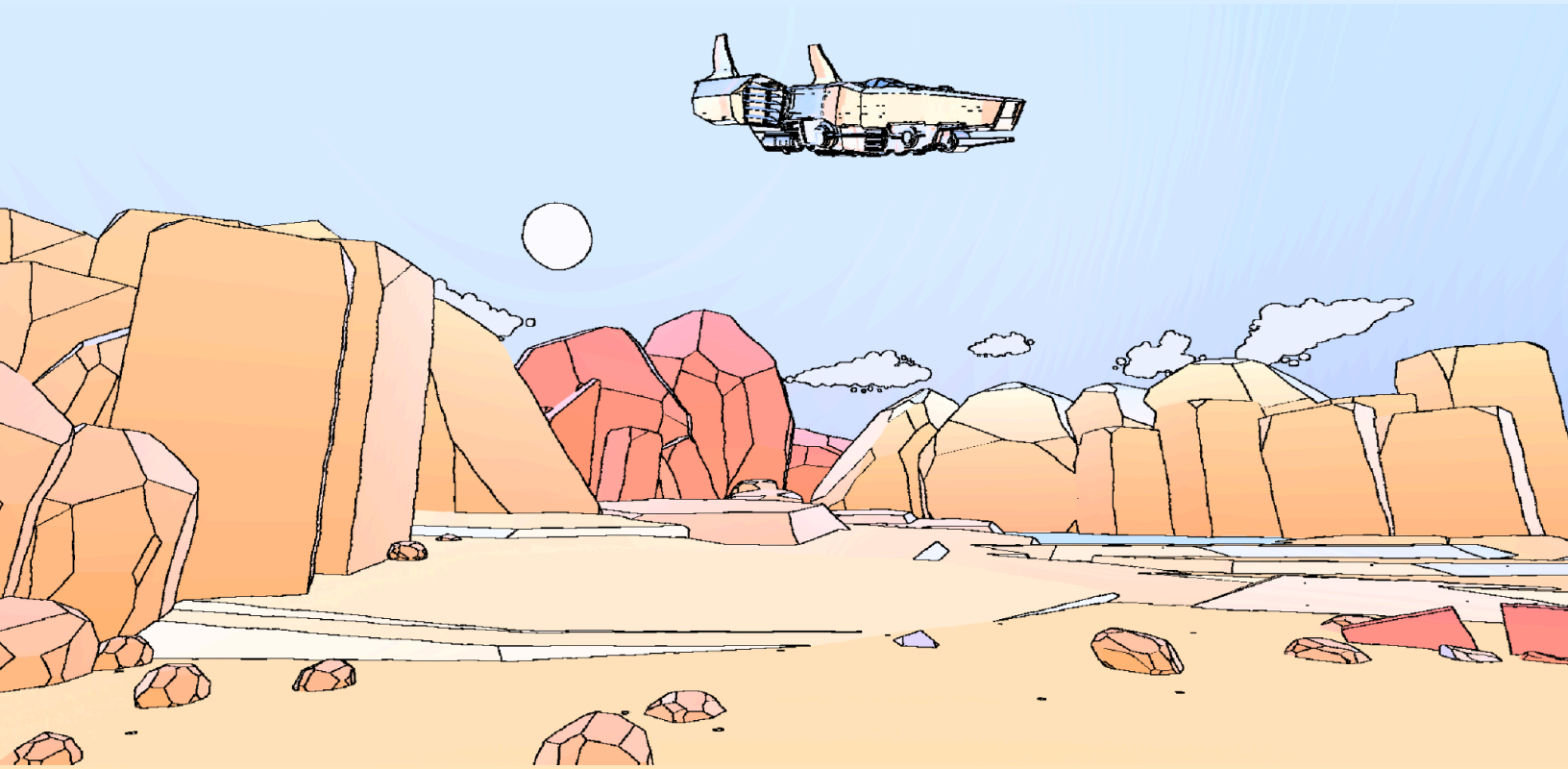


CREATING A LIBRARY OF SHADERS FOR A GAME ENGINE TO ACHIEVE A STYLIZED LOOK

Roshan Manojkumar - P2657789

Supervisor: Artur Machura

Download: https://github.com/rm20killer/Final-Year-Project

Table of Contents

Introduction

Driven by the increasing popularity of animated content and the trend of films and TV shows adopting a more stylized look with the release of shows like “Arcane”, and movies like “Spider-Man Into The Spider-verse”, and “Puss in Boots: The Last Wish”, all being successful and inspired many other creators from hobbyist artist to massive cooperation like Disney, with the release of “Wish”. I Expect that games will move away from realistic and adopt more stylized graphics in the future. Those Stylized looks can provide a unique and immersive experience for players and help differentiate a game from other games.

This project aims to get a better understanding of how to create those stylized looks inside a game engine. To achieve this aim better, I wanted to try different ways to do the same thing. The goal is to create a more distinctive look while also being visually appealing. Through experimentation with the game engine to change how it renders things while being performance enough to run in real-time with as little to no added latency.

This project has the potential to change the approach games use for art style because it will help streamline the pipeline, especially for procedural materials. This speed-up could speed up the asset creation process by shortening the time I spend making textures for each object, which can be quite time-consuming. It would also enable game developers to test multiple art styles without having to spend a lot of time on them.

This report will explore and explain the journey taken to develop diverse shaders, from research to final aesthetics. It delves into technical aspects while also showing issues faced by the project.

List of Used Software

Game Engine:

- Unity
    - It has one of the most open ways to edit how the game renders things while also including integration with Substance Designer

Procedural material:

- Substance Painter
- Substance Designer

Other software:

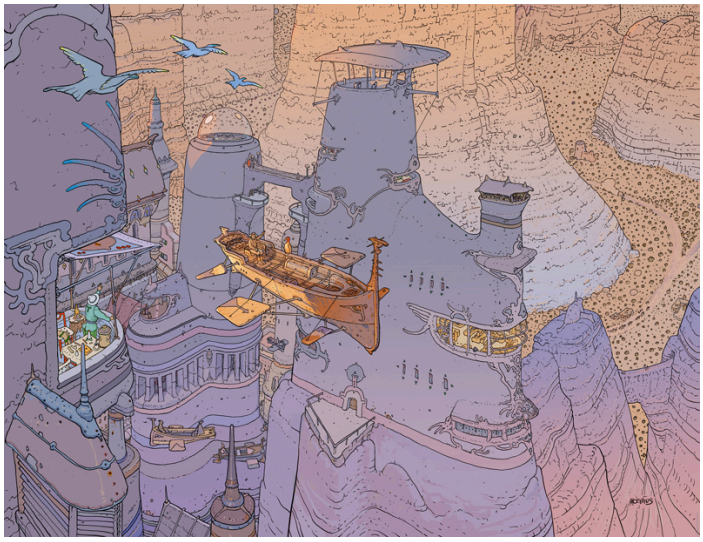
- VS code → JetBrains Rider *(Moved to Rider)*
    - *I moved to* Rider *as it offered better scriptwriting tools with unity compared to VS code and I have been using Rider for other projects when I moved over.*
- Blender
- Github
    - Version control
- Google Drive
    - Backup storage
- Obsidian
    - Note management
- Jira
    - Project management

Inspiration for art styles

It all started with Spider-Man: Into the Spider-Verse which revolutionised the movie industry by breaking all normal animation standards and ground-breaking art style. The movie recreated the comic book look with the use of lines, dots and cross-hashing. Every frame of the movie looks like it is straight from a comic book.

Another inspiration was the TV show Arcane which had amazing visuals and a unique style of animation that was not seen on any big TV shows yet it blends both 2D and 3D elements seamlessly. This was done with hand-painted textures on 3D models and a custom lighting system.

While researching this topic I found Moebius's artwork which I took inspiration from as well. Moebius used bright and bold colours alongside a gradient to create a unique look while being bold and pleasing to look at.



I decided to go with those three as inspiration as all of them have unique looks and have helped change the industry alongside the different techniques needed.

Project Management

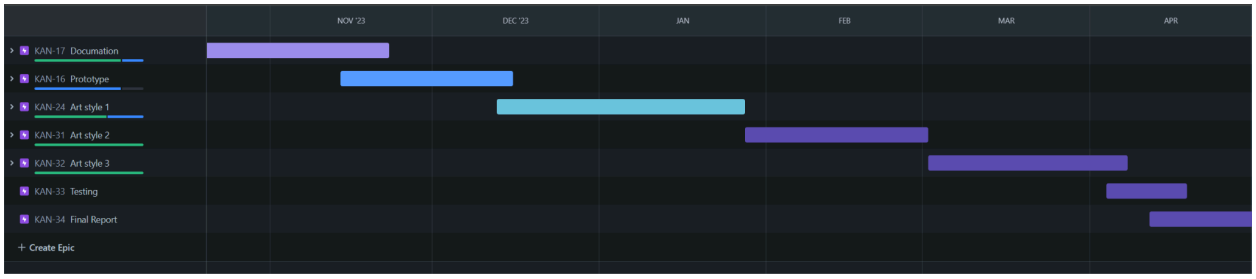
I used an agile development methodology for this project where I plan to develop, test and redo the cycle. I used this method compared to other methodologies like waterfall because it allowed me to prototype a lot faster and create more iterations.

Project Plan

My initial project plan was as follows.

Week	Date start	Week Ending Date	Task	
Week 1-2	02/10/2023	12/10/2023	Supervisor Allocation	
Week 3	16/10/2023	12/10/2023	Start working on initial Documation	
Week 4	23/10/2023	27/10/2023	Initial Documentation sign off	
Week 5	30/10/2023	03/11/2023	Finish off Gantt chart	
Week 6	06/11/2023	11/11/2023	Research start	
Week 7	13/11/2023	17/11/2023	Finish off Research	
Week 8	20/11/2023	24/11/2023	Start on prototype (First artstyle/shader)	
week 10	04/12/2023	08/12/2023	Get prototype ready for first deliverable	
Week 11	11/12/2023	15/12/2023	First Deliverable	
Week 15	08/01/2024	12/01/2024	Improve on prototype	
Week 16	15/01/2024	19/01/2024	Start working on new art style	
Week 20	12/02/2024	16/02/2024	Finish of art style	
Week 21	19/02/2024	23/02/2024	Start working on new art style if previous 2 are done and dont need improvement	
Week 26	25/03/2024	28/03/2024	Get 3rd set into a useable state	
Week 27	01/04/2024	05/04/2024	Put together testing scene	
Week 28	08/04/2024	12/04/2024	Test case	
Week 29	15/04/2024	19/04/2024	Work on Final Report	
Week 30	22/04/2024	26/04/2024	Prepare for viva	
Week 31	29/04/2024	05/05/2024	Final Deliverable	

However, I did go over certain time stamps, and my final Gantt chart looked way different as I went over the time allocated for the first style. This meant I could not spend as long on the other two styles.



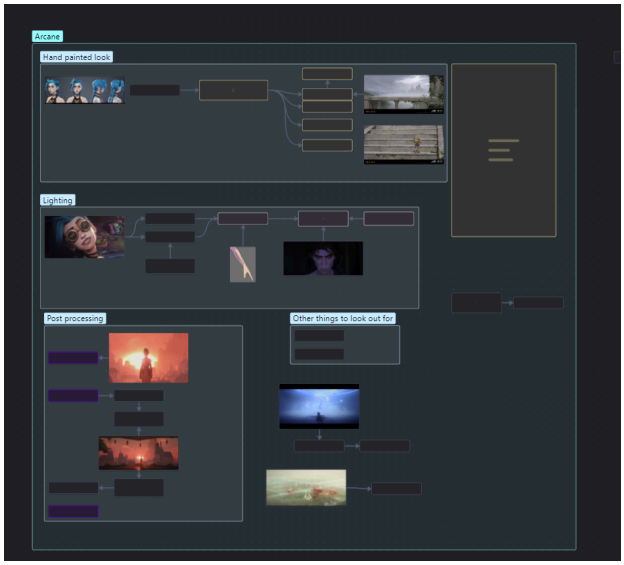
Project Backup/saving

I used GitHub as version control, but I had a bad habit of forgetting to commit my changes until I had finished something. But because I knew I might end up doing something like this, I also set up Google Drive to sync with the project files so all the files would automatically sync to my Google Drive storage as the main backup.

Research

Breaking down each style

Before jumping straight into making the shaders, I worked on creating a breakdown of all the components for each style. This would help when recreating the shader as the shader would be broken down into smaller tasks to recreate the look while also providing reference to look back to.



Arcane

To break down each component of Arcane, I looked at multiple images from concept art to the actual show. All had one thing in common which was that it looked like it was hand drawn. So the main goal was making it feel like it was painted with.

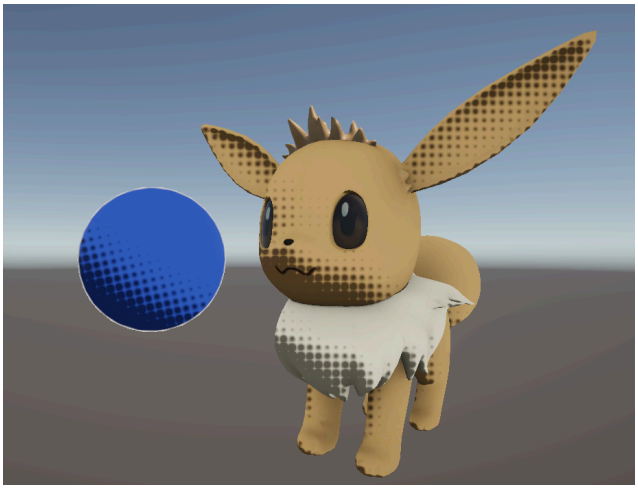
Spider-verse

Spider-verse went with comic book style using hashing and halftone. So, the shader would need to take in lighting and add hashing and halftone, depending on the brightness and shadows of the lights. It also has a different type of depth of field where objects not in focus would have an anaglyph filter applied to it. The black outlines on objects are key parts of the spider-verse movie. So, the main goal of recreating this style is to create the hashing and halftone while also applying the filter on distance objects while also adding an outline.

Moebius

Moebius had flat but bright colours using gradients on objects. It includes the use of black outlines with different thicknesses to add detail and to outline objects making each object stand out. The main task to recreate Moebius would have to be to procedurally create a gradient, making colours flat and the lines on objects.

Creating the first shader

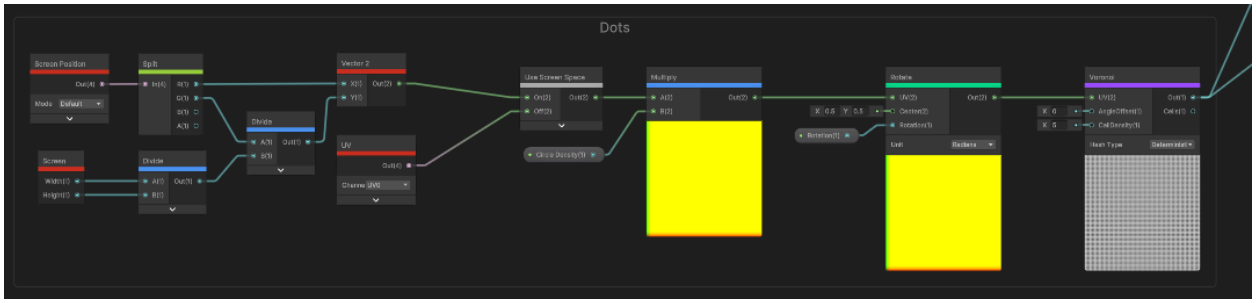


To get a better understanding of shaders inside Unity I created a simple half-tone shader, which would react to the change of the sun. Doing this would also provide the starting building blocks to create the reset of the shaders.

Create the dots

Texture

To create the dots, I used a Voronoi texture where I modified the UV to increase the number of dots per material and the rotation of the dots.



This was attached to a smooth step operation, which took the lighting data to apply the dots to the dark parts of the image.

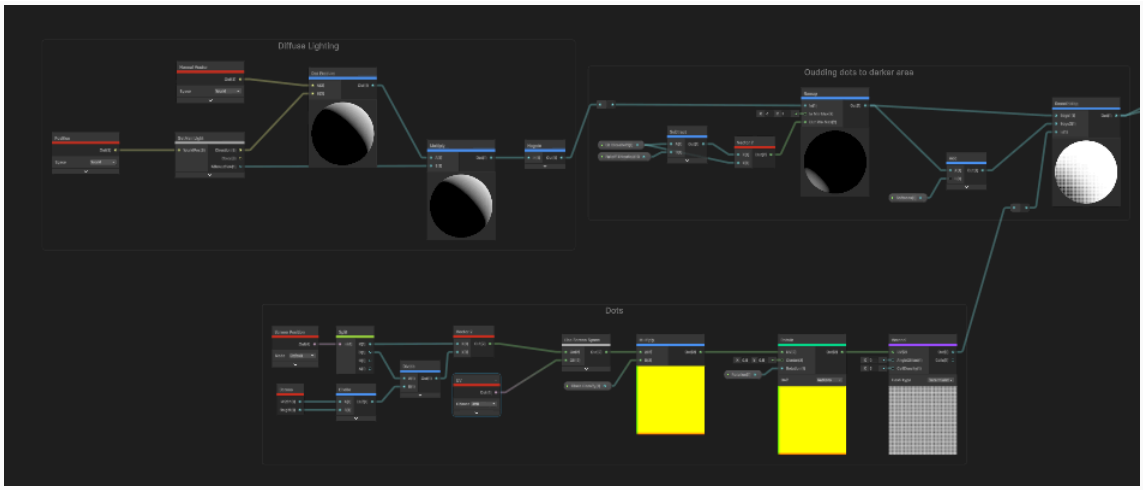
Getting light info

Unity does provide a function which gets the direction of the main light but I wanted to use this time to try out HLSL as I know I will need to use it to create much harder things that would need to be coded in HLSL and would take too much node inside unity or straight out not possible.

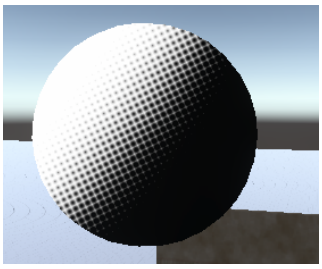
```
void MainLight_float(float3 WorldPos, out float3 Direction, out float3 Color, out float DistanceAtten, out float
ShadowAtten)
{
    #ifdef SHADERGRAPH_PREVIEW
        Direction = normalize(float3(0.5f, 0.5f, 0.25f));
        Color = float3(1.0f, 1.0f, 1.0f);
        DistanceAtten = 1.0f;
        ShadowAtten = 1.0f;
    #else

        #if SHADOWS_SCREEN
            half4 clipPos = TransformWorldToHClip(WorldPos);
            half4 shadowCoord = ComputeScreenPos(clipPos);
        #else
            half4 shadowCoord = TransformWorldToShadowCoord(WorldPos);
        #endif

        Light mainLight = GetMainLight(shadowCoord);

        Direction = mainLight.direction;
        Color = mainLight.color;
        DistanceAtten = mainLight.distanceAttenuation;
        ShadowAtten = mainLight.shadowAttenuation;
    #endif
}
```

This code will get the main light using the GetMainLight function and get the rest of the components from the main light. As there are multiple different types of shadow that Unity uses there is an if statement to see if it uses cascade shadows or not and get the right coordinates.

This is used to calculate which parts should be lit and added to the smooth step

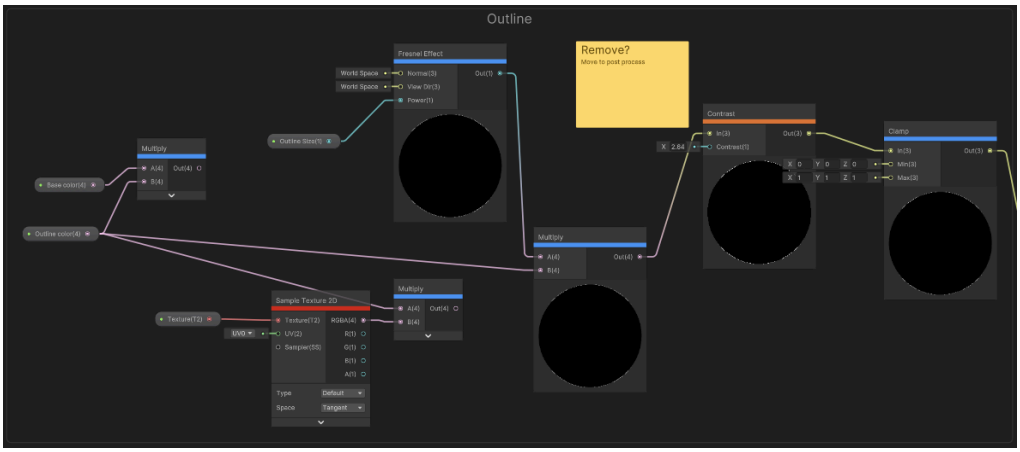


This is then added to the texture or colour dots as a mask where white will be replaced by the colour/texture.



Outline

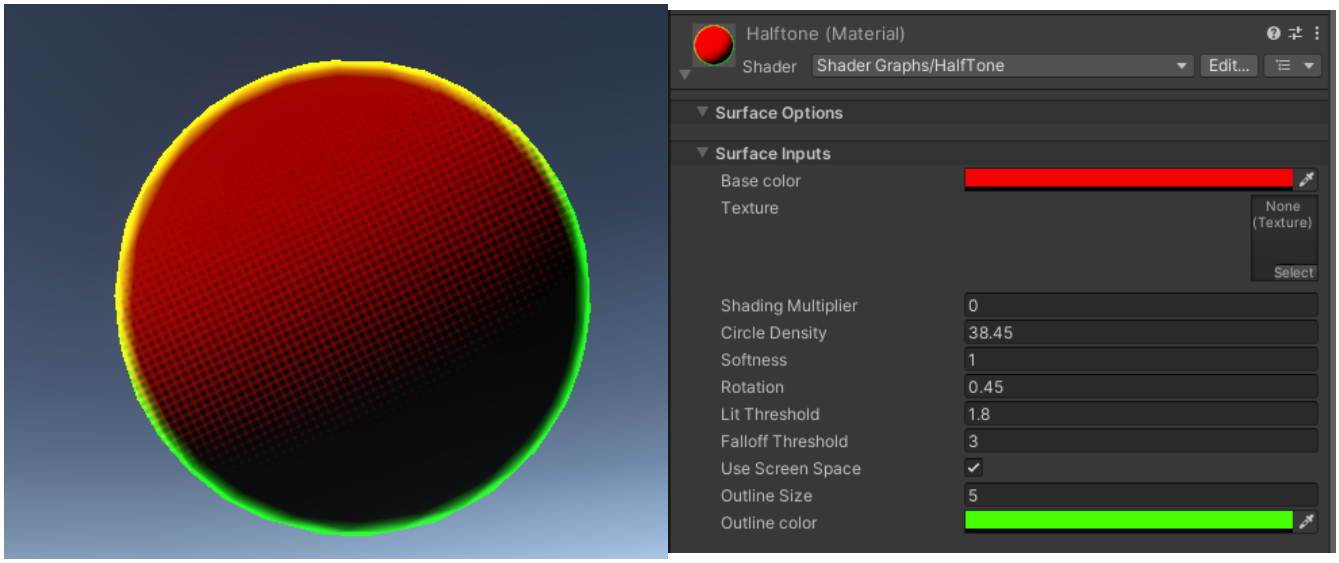
The outline was done using a fresnel effect node, similar to the trick used in blender edge detection, then applied colour to it which is then overlaid on the main object.



The downside of this method is that it does not look great on cubic objects. So I decided to not use this approach in the future, instead, I will be using render feature effects to add to the outline as this would work on all types of objects compared to the fresnel effect.

Final result

The end result was a simple material that can be applied to materials which create a different look from the standard rendering. Multiple parameters can be changed on the shader which would help control the look of the material.



This outcome was the primary building for the rest of the shaders which resulted in learning multiple things about shaders like the basic uses and simple HLSL scripting.

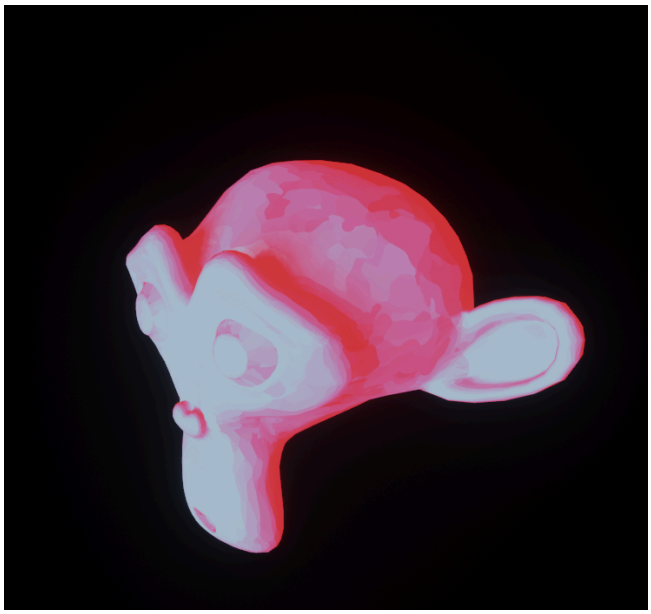
Recreating Arcane Style

The arcane style took longer than expected to create due to multiple issues which will be explained further along in the report.

Blender

I first attempted to use Blender to create a material which could be easily applied to any object, but the method used to do this only worked on the Eevee

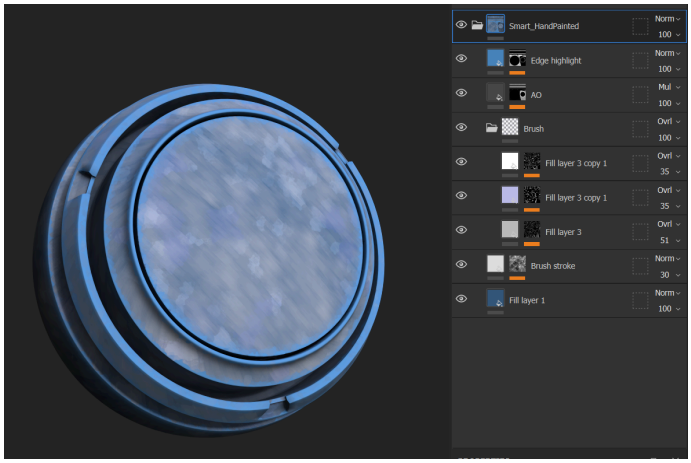
render, which is blenders real-time render, which is faster to render with but does not support texture baking like cycles. So, the use of this method was ruled out for games but understanding the basics of this method was useful in the final shader.



Substance Painter

Instead of doing it in Blender, I made a smart material inside of Substance Painter which did work and allowed for painted-looking objects with little effort from the end user.

This was achieved with multiple layers of noise with a blur slope filter each with a different colour to create brush strokes. On top of that extra detail was added to the edges with the curvature as a mask with another blur slope to add in the paint look.



For the end user, they would have to export the mesh into Substance Painter, then change the colours on the layers to the colours the artist wants. Once it is all done in substance, they need to export the texture. Those textures would then have to be brought into unity and attached to a material.

For this reason, Substance Painter would be faster than hand painting all the textures for every object. However, this method would require them to have an active substance licence, which is a 3rd party software and is expensive software.

The benefits:

- Faster than hand painting
- A lot of customisation
- A lot of control

Drawback:

- Paid software needed
- Middleman software

Unity Material

With my 3rd attempt at recreating the hand-painted style seen in Aracne, I made an in-unity shader which is used as a material for objects. With this, I wanted to create something similar to the blender shader where the colour is different if the main light is hitting it.

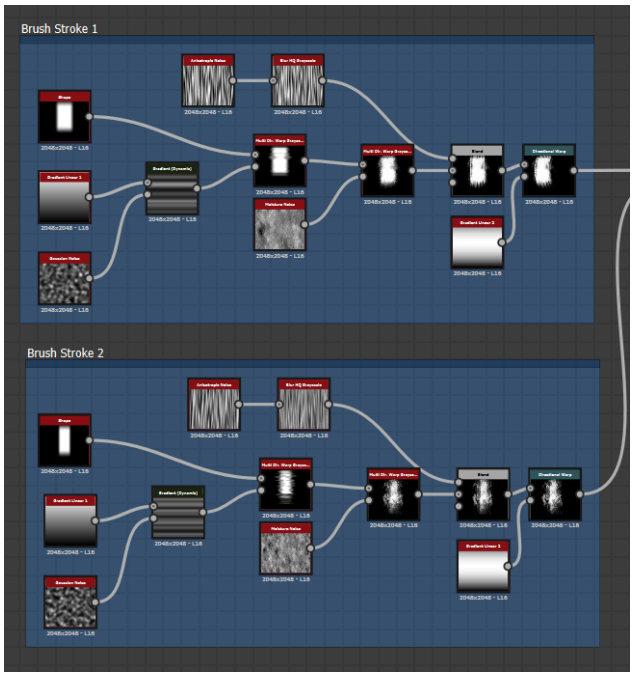
I first researched about this method to see if anyone had tried something like this, there was one person called “Technically Harry” I found who made something similar but using a surface shader, which is a .shader file which is written in all code, but I used the same concept but turned it into a node based shader graph which has a lot more control for the end user and is more straightforward to use.

The concept was quite simple where you create a black and white stroke texture which is used alongside the diffuse and specular highlights to make certain parts light and darker then blending that with the main object texture/colour.

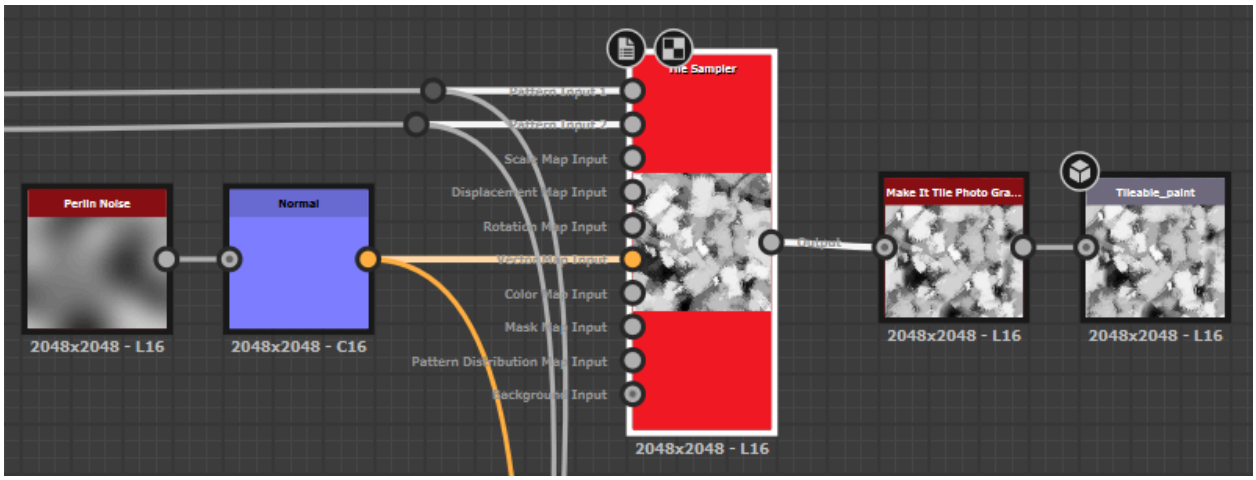
Creating the brush stroke

I created the brush stroke inside of Substance Designer to allow for easy control of all the parameters which will be useful later on.

The brush stroke is quite simple. The first step was to create a brush stroke shape by distorting the shape using warp nodes and noise to create more of a brush stroke.

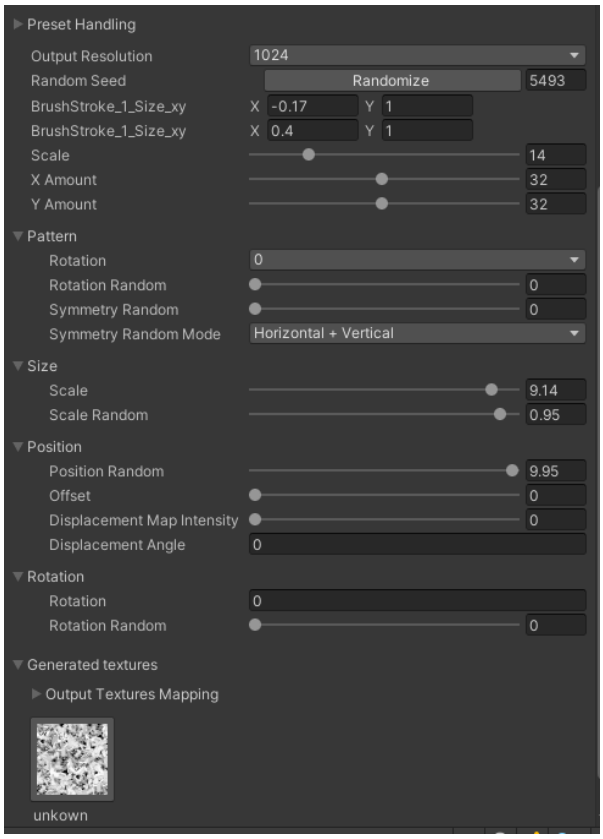


Those brush strokes get fed into a tile sampler node, which will spread the brush stroke out, filling the texture up alongside a node which will make the tile seamlessly tile that gets output.



Each node has different parameters that can be controlled inside of Substance Designer but by exposing the parameters and naming and organising all those parameters right. I am able to export this into a .sbsar file which Unity is able to read and able to generate the textures using a tool that can be installed on Unity called “Substance 3D for Unity”. This plugin lets Unity be

able to create texture from the .sbsar without having to have an active Substance licence and will allow you to edit those exposed parameters, including texture resolution if you need to get closer to a pixel density.



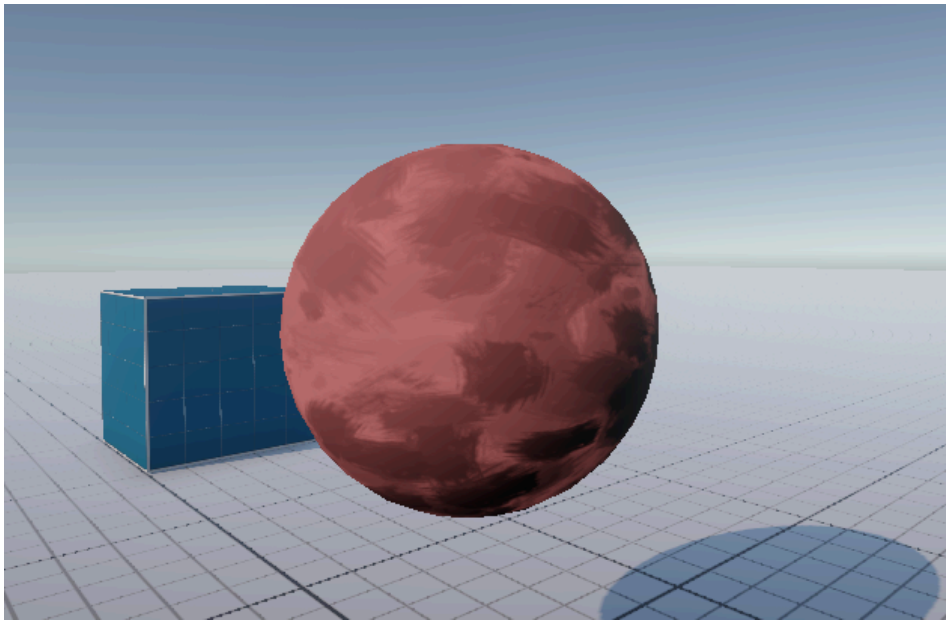
This allows for a lot of user controls while not having to leave the software.

Once you generate a texture you can either make a copy of the texture and move it into a different folder or if you need access to edit those parameters while seeing the changes happen in real time can make a copy of the .sbsar file which will generate a random brush stroke.

Shader Graph

Unlit shader

For the actual shader, maths was used. Using the diffuse lighting functions from my first shader attached. But by having 2 different smooth step nodes I can make certain areas a bit brighter by changing the smoothness on one of the smoothness nodes. This allows for the part that is directly in the light direction to be a much higher value than the edges. This results in a black and white texture.



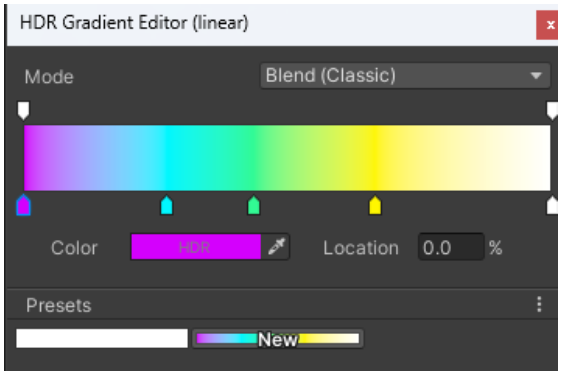
Plugging this into an alpha part of a Lerp node with B being the default colour and A being a darker version of the colour allowed it to dark areas based on the paint stroke texture. I then started to add a gradient using the black-and-white texture.

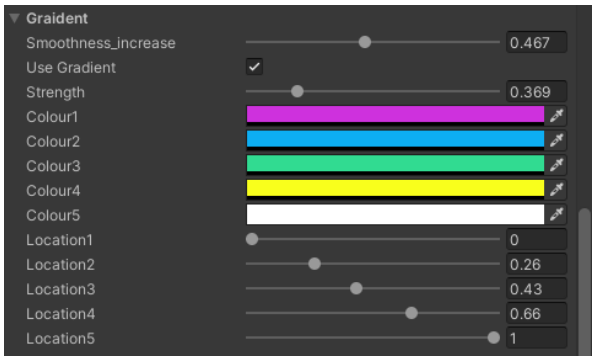
I assigned the colours using a sample gradient node with Unity's built in gradient property, but the gradient nodes could not be exposed which would mean that the user could not edit the colours outside of the shader graph. To overcome this issue I wrote a custom sample gradient function in HLSL which could take up to 5 colours and a value to sample it from the gradient. This is done with a lerp to the colour in between 2 colours.

```
void sampleGradient_float(float input,
    float4 colour1, float4 colour2, float4 colour3, float4 colour4, float4 colour5,
    float location1, float location2, float location3, float location4, float location5,
    out float4 Out)
{
    float4 output = 0;
    if(input < location1)
    {
        output = colour1;
        return;
    }
    if(input < location2)
    {
        float t = (input - location1) / (location2 - location1);
        output = lerp(colour1, colour2, t);
    }
    else if(input < location3)
    {
        float t = (input - location2) / (location3 - location2);
        output = lerp(colour2, colour3, t);
    }
    else if(input < location4)
    {
        float t = (input - location3) / (location4 - location3);
        output = lerp(colour3, colour4, t);
    }
    else if(input < location5)
    {
        float t = (input - location4) / (location5 - location4);
        output = lerp(colour4, colour5, t);
    }
    else
    {
        output = colour5;
    }
    Out = output;
}
```

This workaround does work but it does mean the gradient is limited to a maximum of 5 colours. It also makes editing the gradient not as

straightforward as the built-in gradient editor offers and takes more space up in the parameters window.

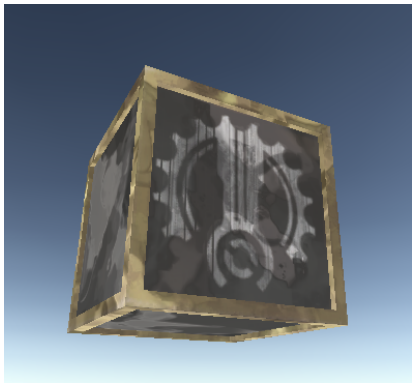


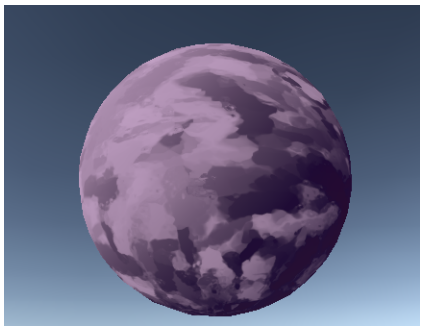


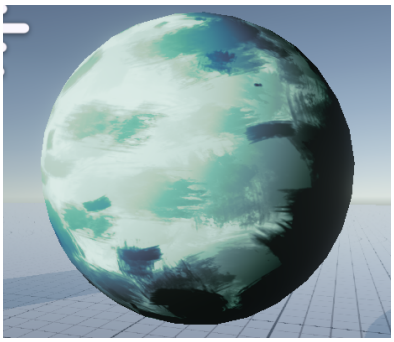
Once I got this working I made the colour brighter depending on if the object should receive any specular highlights. This was done using a built-in Unity shader function and making sure the values worked with the function.

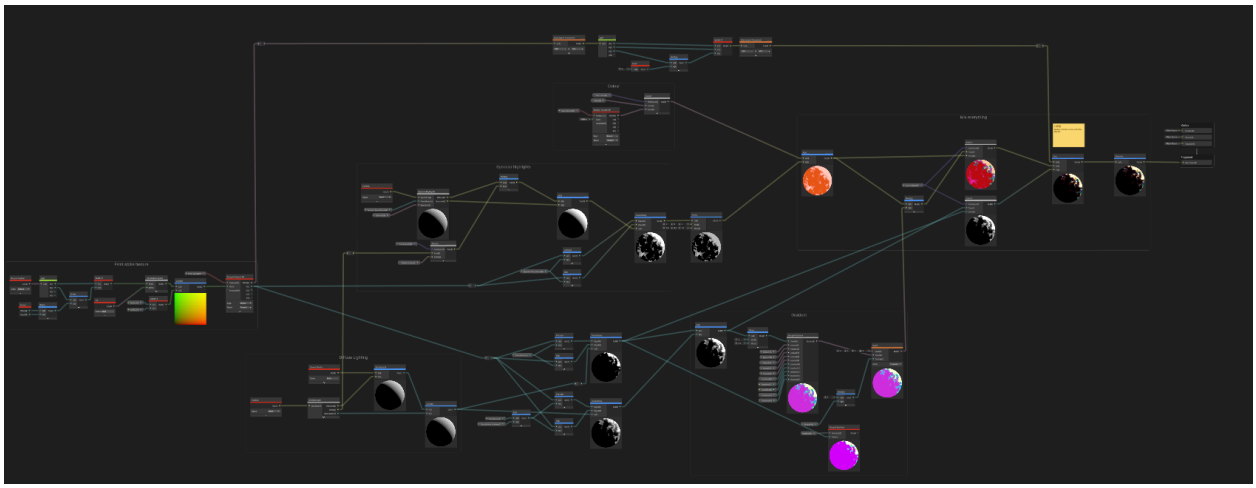
```
void MainSpecular_float(float3 Specular, float Smoothness, float3 Direction, float3 Color, float3 WorldNormal, float3 WorldView,
    out float3 Out)
{
    #if SHADERGRAPH_PREVIEW
        Out = 0;
    #else
        Smoothness = exp2(10 * Smoothness + 1);
        WorldNormal = normalize(WorldNormal);
        WorldView = SafeNormalize(WorldView);
        Out = LightingSpecular(Color, Direction, WorldNormal, WorldView, float4(Specular, 0), Smoothness);
    #endif
}
```

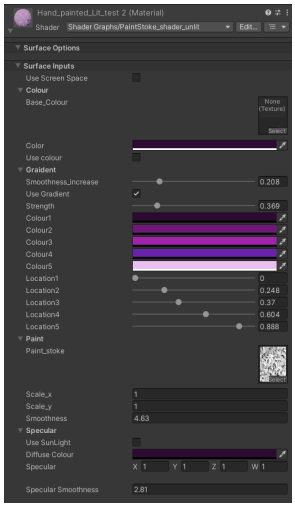
Once this was all done I added the ability to use a Texture as the colour instead of just a single colour.





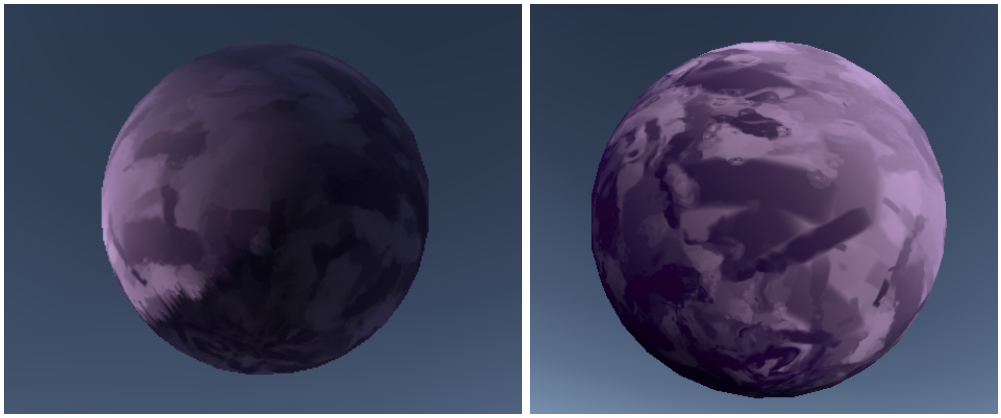




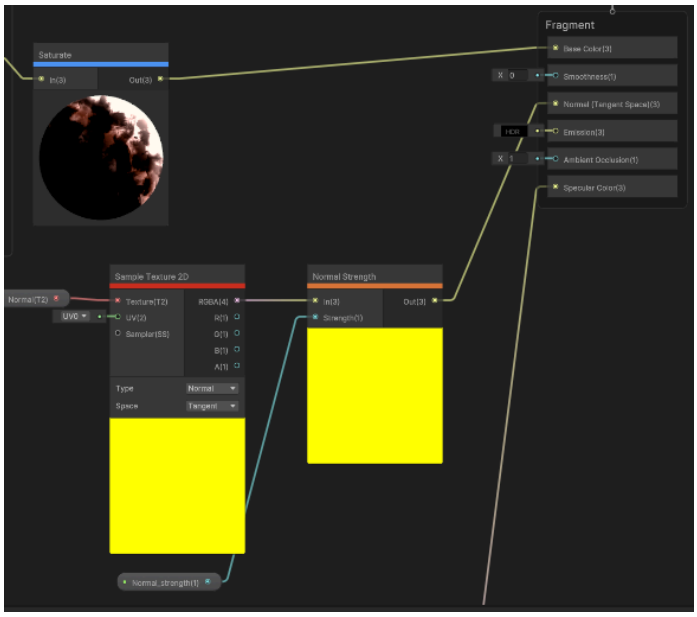


Lit shader

With the approach above, you can't use any normal maps so I made a copy and turned it into a lit shader, which reacts to light but results in a darker object because those shaders are affected by shadows. This makes it not look as great but allows for multiple light sources to affect it as well. Turning off shadows makes it look better.



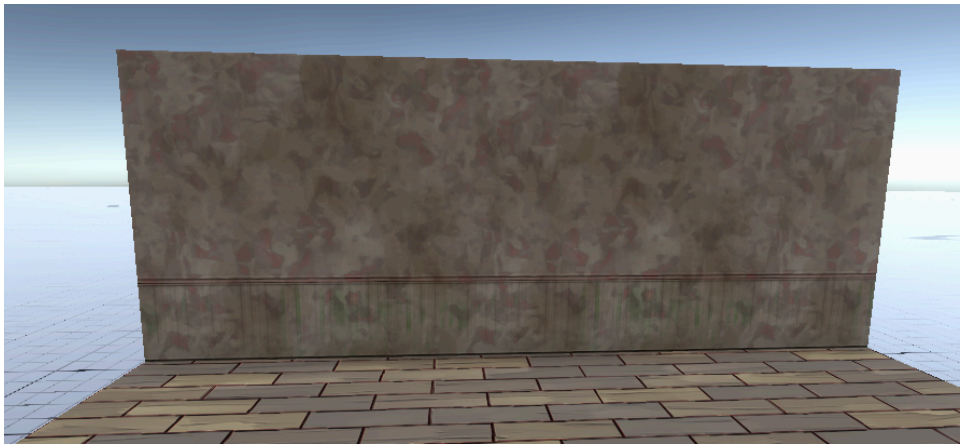
To add normal data I used a texture sample with a normal map as the input and attached that to the normal node. I used a normal strength to allow the user to change the strength level if they wish to increase or decrease it.

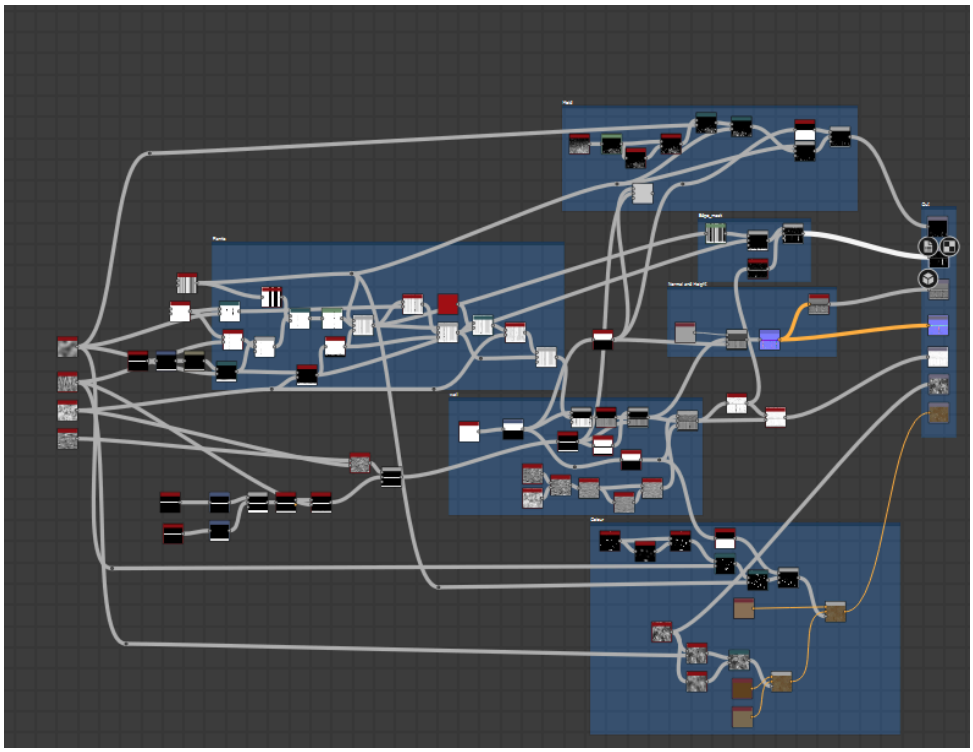


At the current moment, the unlit version of the shader does look much more likeable as the colours pop more, even though it does have many drawbacks, like no lighting and shadows.

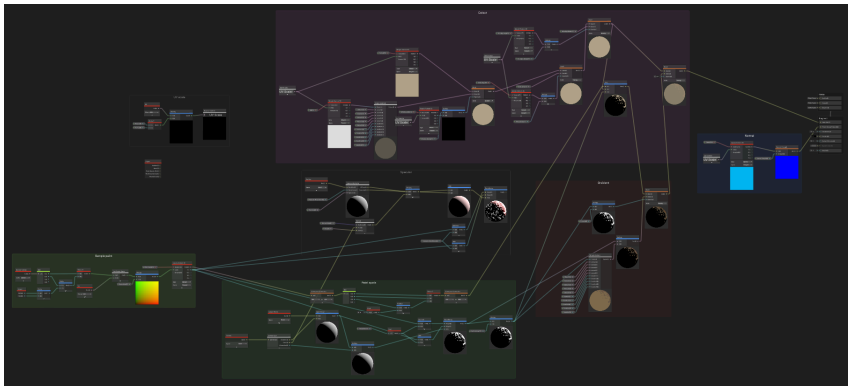
Substance Designer Textures

It would be a good idea to include some premade textures that can be used with the scene. So I made 2 node-based textures similar to the paint strokes but for walls and wooden floors. I wanted to make it all as procedural as possible, so I used Substance Designer to create them





I created separate outputs for all the different things like edges, mould and base colour then used a Unity shader graph to blend it all together. This was done to allow easy editing inside of Unity material inputs but required a new shader graph to do. It uses a copy of the lit shader but is modified to handle more inputs for the extra details.
Looking back at this I should have just blended the detail inside of Substance Designer and only have the colour texture and normal map as the export so the texture would work with the shader that has already been made.



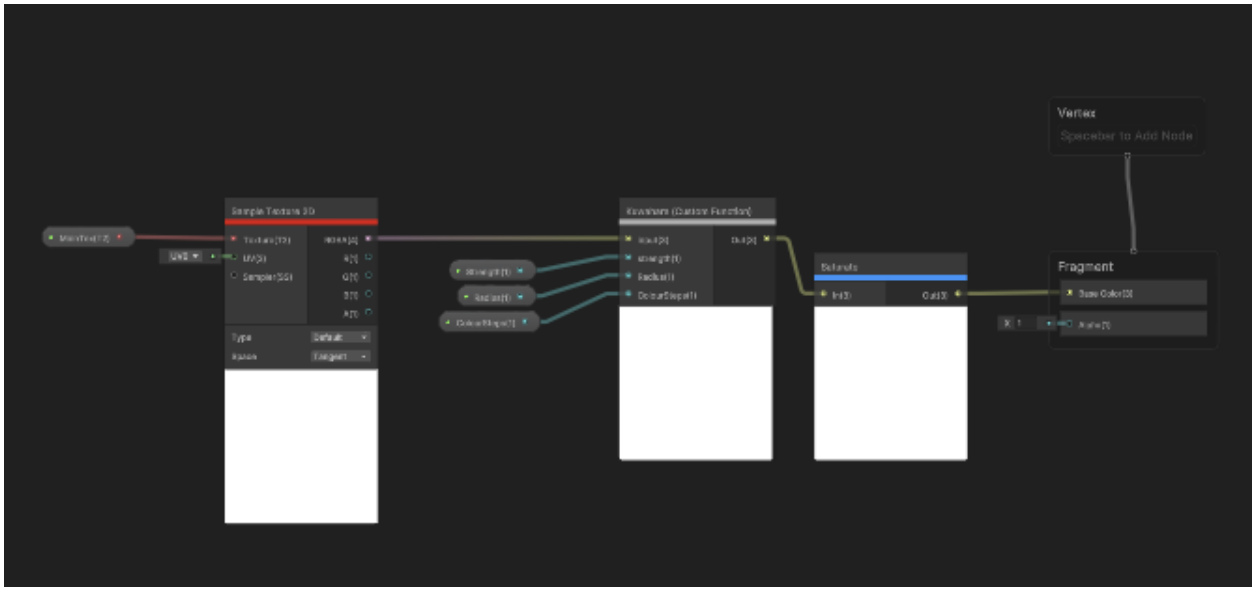
Kuwahara Render Feature

Once that was done I created a post-processing effect. The main goal of this effect was to add blur to the rendered image in the style of a Kuwahara filter which would blur it in such a way that it would look like it had been hand drawn. This was the hardest part of the shader as it required the most amount of work and a lot of maths to make.

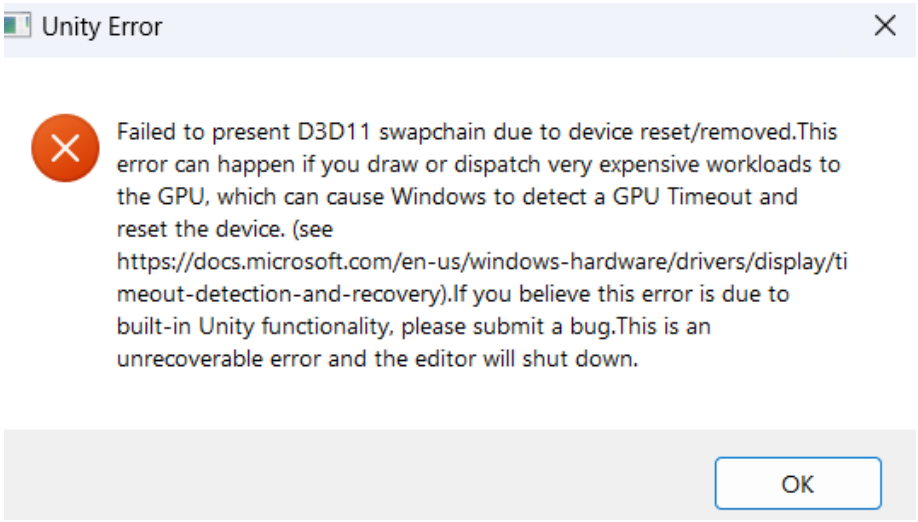
My attempts were done using a rendering feature which applies a shader over the screen. Those shaders are called fullscreen shaders.

Attempt 1

With my first attempt, I did a simple graph which just takes in the camera texture and gets sent to a custom function written in HLSL



I reversed a .shader script that does the filter but turned it into .hlsl and made it work with Unity's shader graph. The function took the texture and sampled the texture 4 times for left top, left bottom, right top and right bottom, while it is getting sampled, it gets the brightness of the quadrant. This is done twice to get 2 values which are used to value in between those 2 values and return it. But this would cause Unity to crash as the work needed to do this would cause too much of a performance impact as it is too expensive to run, making Windows think the GPU has timed out and reset the GPU. This caused Unity to be in a crash loop until I commented out the code.



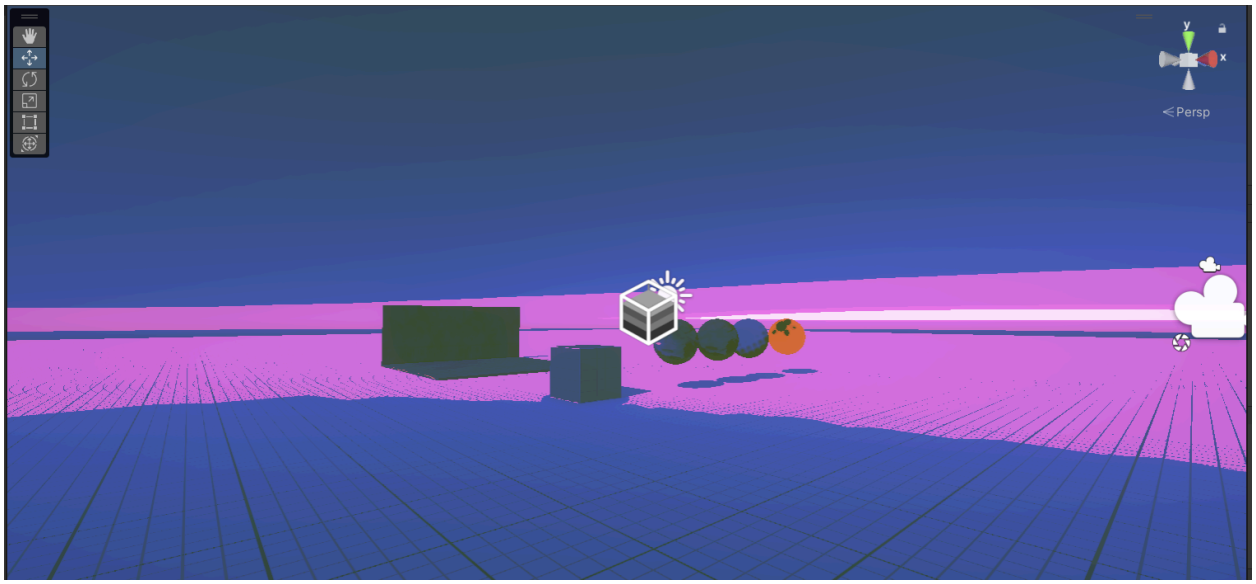
Any attempts to fix this issue would cause Unity to display errors, so instead I tried a different method.

Attempt 2

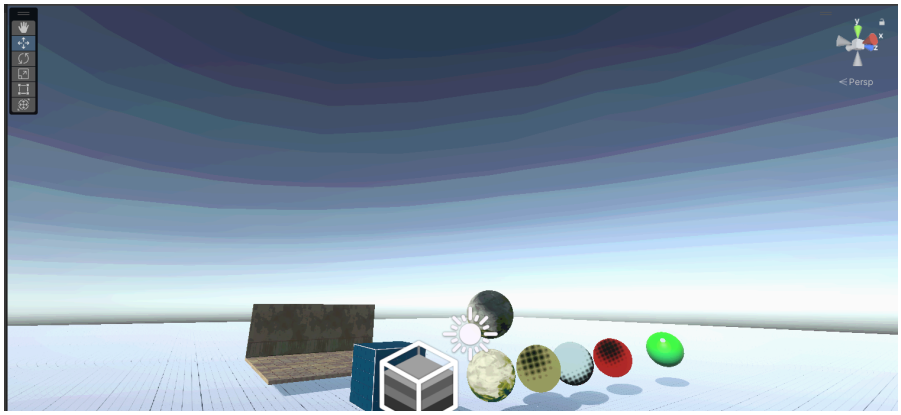
On the second attempt, I tried using a filter similar to how other blur filters work. This was done by sending an already sampled texture of the camera and using weights.

```
void Kuwahara_float(float3 input, float strength, float Radius, float ColourSteps, out float3 blur )
{
    float weights[5] = { 0.1f, 0.2f, 0.4f, 0.2f, 0.1f };
    float3 SampleBlur = float3(0.0, 0.0, 0.0);
    float2 blurDirection = normalize(float2(0.5, 1.0));

    for (int i = -2; i <= 2; ++i)
    {
        float offset = i * Radius * strength;
        float2 sampleOffset = offset * blurDirection;
        float3 sampleColour = input + float3(sampleOffset, 0);
        SampleBlur += sampleColour * weights[i + 2];
    }

    // float3 pass1 = gaussianblur(input, 0.5, 2);
    // float3 pass2 = gaussianblur(input, 0.5, 1);
    // float GaussianMix = lerp( pass1, pass2, 0.6);
    //mix GaussianMix with SampleBlur
    // blur = lerp(SampleBlur, GaussianMix, 0.1);
    blur = SampleBlur;

    int colourSteps = int(ColourSteps);
    if (ColourSteps > 0)
    {
        float3 posterized = floor(blur * colourSteps) / colourSteps;
        blur = posterized;
    }
}
```

At first, I thought it was not doing enough of a blur to see, so I added in a Gaussian blur and mixed it with the first blur, but then I noticed what it was actually doing. The blur was sampling blurring the colours but was doing it in such a way that it just destroyed the colours.



I wanted to salvage this and turn it into a new post-processing by adding a posterized filter on top of the blur which would limit the amount of colours and turn it into something different. At this stage, I was already way over my set schedule by a couple of weeks so I decided to move on without a working Kuwahara and come back to it when I finished my other art styles.



Attempt 3

I did this attempt once I finished the Moebius Style where I had the most experience and learnt the most. It was more of a continuation of the first attempt, where I would sample each region and do some maths operations to do the filter.

While I was doing more research on this filter I found that a lot use a gradient filter which would determine the direction of the blur. Saving this to a float would allow me to save a texture which would be used to determine the direction.

Inside of sampling the quadrants at different stages, I did it in the loop with the UV so there would be fewer function calls and operations.
But this caused Unity to crash or did not work on a lower radius. So I knew it was with the sampling or with the maths operations after sampling was done. So using this knowledge, I changed the code using UnityTexture2D and the Tex2D() operation, which caused a performance hit than using SHADERGRAPH_SAMPLE_SCENE_COLOR() multiple times.

To make sure I would not crash straight away I started with a low radius and made my way to a higher radius, with pixels slightly changing at certain areas but the performance impact after 8 was horrible. The performance made it unplayable.

By this time, I did not have enough to make another attempt at the shader, so I left it out. However, the second attempt did give me an idea for a future post-processing effect that I ended up making.

Final Result

The outcome of this art style was more of a shader material, allowing for a hand-painted look that is easy to use and does not take too much time to set up.

The only big issue with the style was with the Kuwahara filter which, if it did work, would have caused the style to look very close to how I wanted it to look. If I do work on this project again, I will spend some time learning how to write .shader files which would have provided better performance compared to the shader graph and would have better resources to get it working. I want to attach this to the depth buffer and make it so the further away an object is the objects will be more blurred.

Even with the render feature being a failure, I developed more knowledge about render features.

Recreating spider-verse style

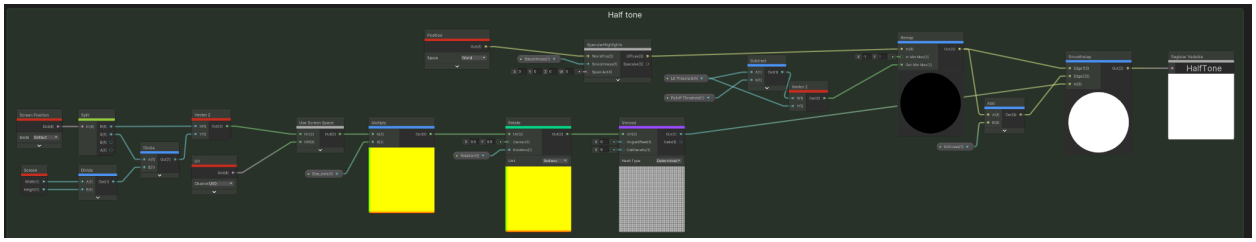
I did not spend as much time on the spider-verse style as I wish I could so those do not look as good as I hope they would. I was planning on using the Kuwahara filter and modifying it to add in a small amount of miss print detail.

At this point in the project, I learned about "Shader Graph Variables" by Cyanilux, which adds in get and set variable nodes to shader graphs and keyboard shortcuts for certain nodes, which helped the project.
This is also the time I learned about NedMakesGame's "RendererFeatureBlitMat" script, which is similar to the script I made beforehand. Still, instead of having multiple scripts for each render pass and coding the parameters that can be changed, the script takes in material instead of a shader graph. This makes the setup for each render pass a bit longer for the user but decreases development time as the parameters can be exposed on the shader graph to be edited.

Unity Material

Ben-Day Dots

I started by modifying my first shader and changed it so it applies to halftone on the light areas instead of the dark areas. This was done using the spectacular highlights custom function created for the arcane shader. This allowed for the dots to apply where specular highlights this was to recreate the Ben-Day dots that are used in the movies where the bright parts use them.



Cross-hashing

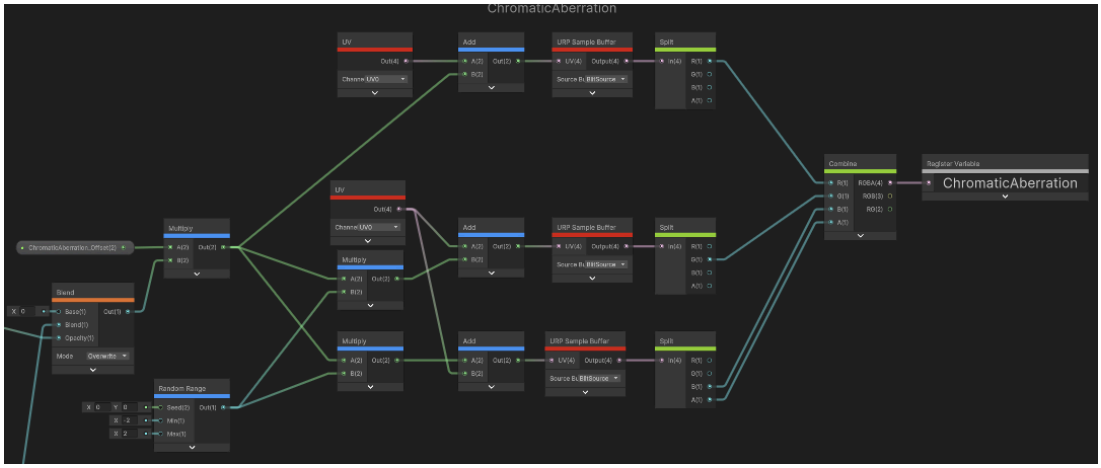
Once I had done the dots, I wanted to add the cross-hashing seen in the movies, where the shadows on the objects were presented with it.
I first tried to change the amount of hashing in the scene using two textures: one with the cross-hatching in the three colours (red, blue, and green) and a gradient texture. If the shadow is that shader of grey, it would apply the crosshatching from that gradient. But this did not work like I hoped it would, as it either failed or the output was not right.

Spider-verse Render Pass

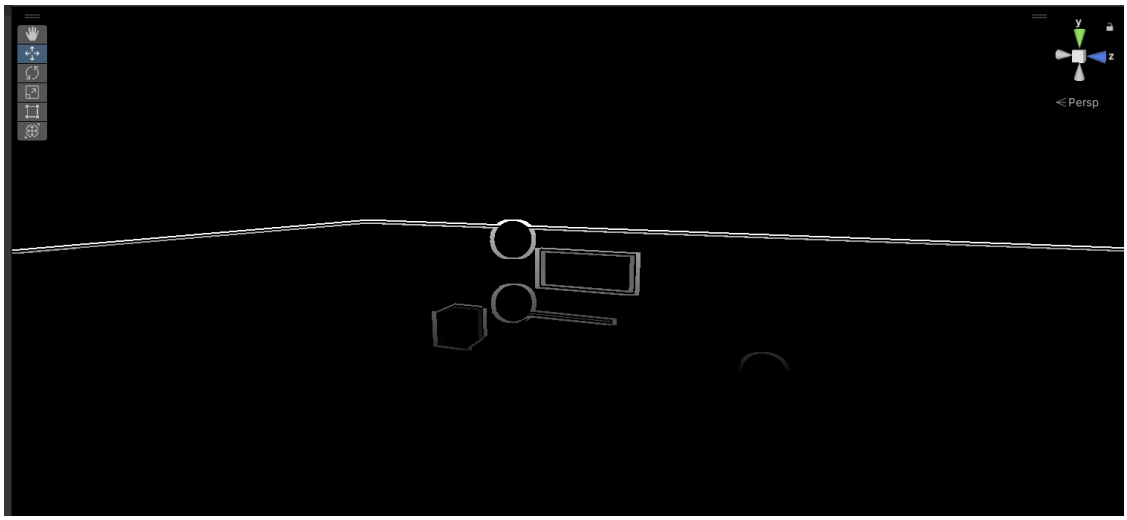
For the render pass, I wanted to create the outlines and the chromatic aberration. If I had time, I would also add a custom bloom that would replace the Ben-Day dots inside the material. This would allow for a more accurate-looking system that would affect the whole screen, including the sky, and would be more accurate to what it is like in the movies.

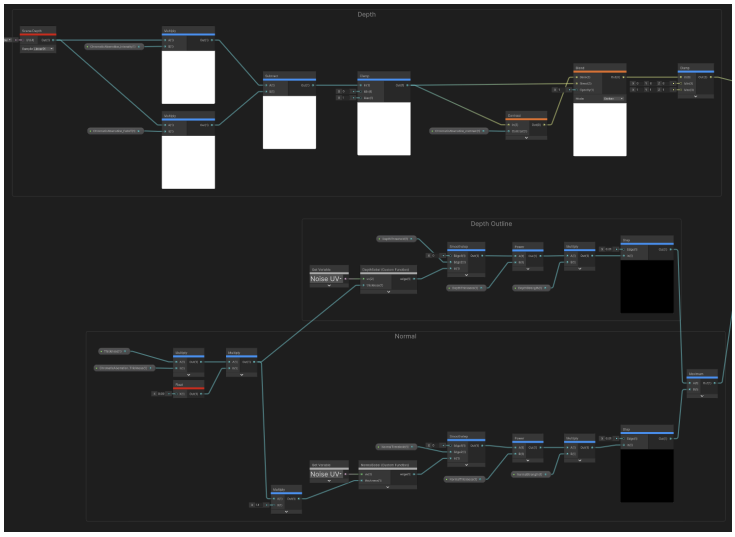
Create a Chromatic Aberration Effect

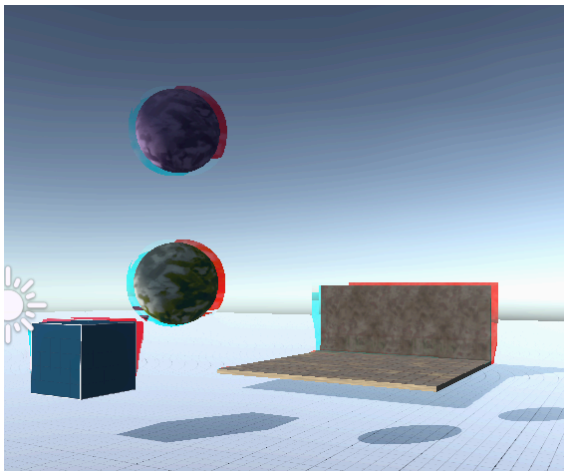
With the Chromatic aberration, I made multiple attempts until I was happy with it.
To make a chromatin effect, it is quite easy. You sample the camera buffer three times with a different UV offset and grab Red, blue, and green from each sample. Then, just combine those three into one new texture. But this would apply to the whole screen.



In the movies, chromatic aberration is only applied to objects that are not in focus and just on the edges. I started with using a depth sample which gives a black and white mask which gets lighter the further away the object from the camera. I then created an edge detection function using the depth buffer but this resulted in the chromatic effect to only apply on certain areas. I then did it with the normal data to create a better mask. Adding those together created the best mask . I then used the depth buffer alongside the edge detection to achieve a good-looking chromatic aberration effect.





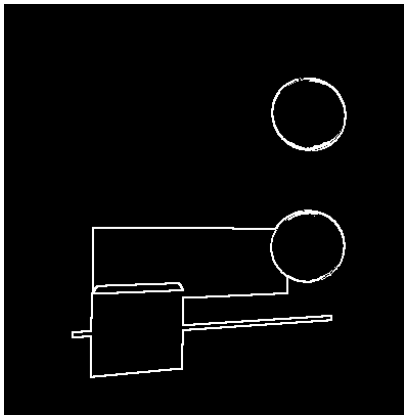


This looks weird when an object is next to the sky due to the high contrast between the sky and the object.

Creating the edge detection

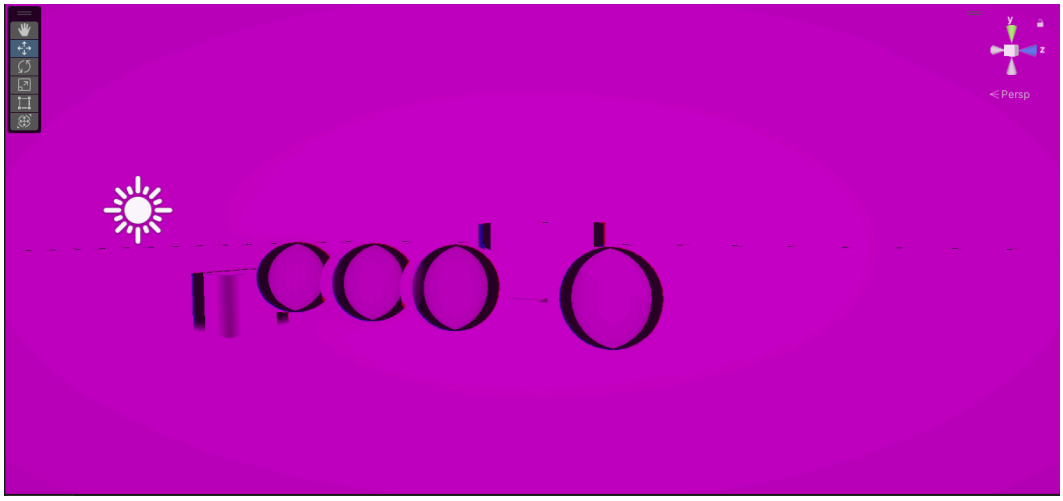
I used the same method as the chromatic aberration, but I will go more in detail here.

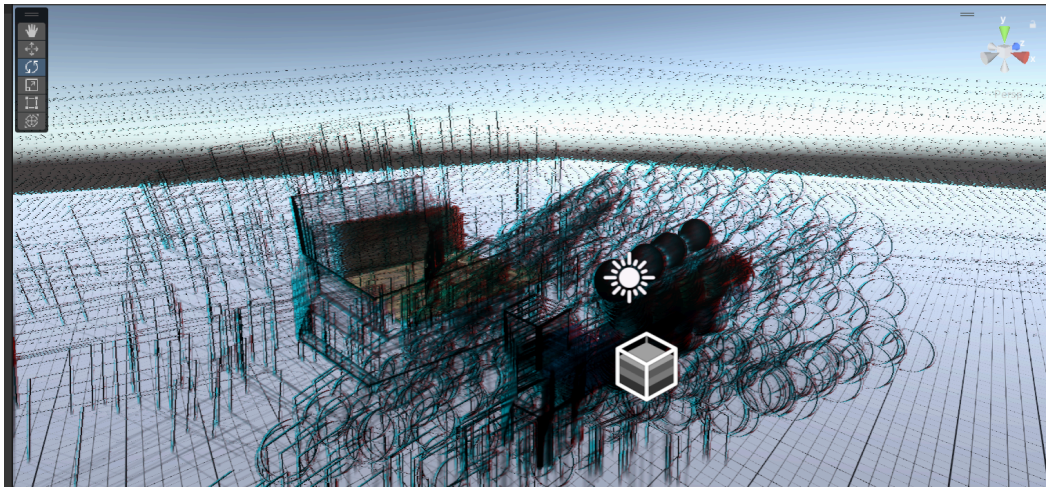
I used a custom function which samples the normal And depth buffer then used a Sobel operation to create an outline on the edge as a black and white texture. Those



```
/**
 * \brief Use a sobel edge detection on the depth of a scene to get the edges
 * \param uv The uv coordinate to sample the depth
 * \param thickness the thickness of the sample
 * \param edge the resulting edge value
 */
void DepthSobel_float(float2 uv, float thickness, out float edge)
{
    float2 sobel = float2(0,0);

    //Loop through the sobel matrix and sample the depth
    [unroll] for (int i = 0; i < 9; i++)
    {
        // float2 samplePoint = sobelSamplePoints[i];
        // float2 UV = uv + sobelSamplePoints[i] * thickness
        float depth = SHADERGRAPH_SAMPLE_SCENE_DEPTH(uv + sobelSamplePoints[i] * thickness);
        sobel += depth * float2(sobelXMatrix[i], sobelYMatrix[i]);
    }
    //Get the length of the sobel edge
    edge = length(sobel);
};
```

With my first attempt, I made a separate render pass, resulting in issues when chromatic aberration and outline were enabled. The first issue I had was with the background being pink like it was missing the texture. I did end up fixing this by messing with the settings but resulted in more issues like the outline and chromatic aberration frames not being cleared, resulting in overlapping from previous frames.





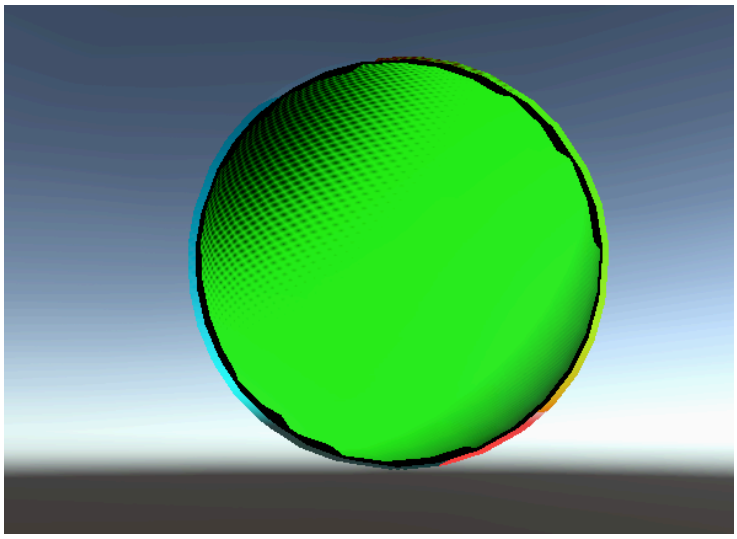
Combining the effects

To fix this issue, I combined the two effects into one render pass. I should have done this beforehand but having it separate would have allowed the user to disable one of the effects more easily, for example, if they want the outline but not the chromatic aberration.
Combining them was quite easy. It just required copying all the nodes and pasting them into a new graph, changing the base on the outline to the chromatic aberration effect, and increasing the thickness for the chromatic aberration to avoid it being the same. This resulted in a working spider verse shader.

At this point, I did not have time to add the planned feature of Ben-Day dots as the bloom.

Final Result

The outcome of this art style was a successful material that recreated the Ben-Day Dots and cross-hashed, along with a render pass that added an object outline and chromatic aberration. The user can change both quite easily.



If I had more time, I would have moved the Ben-Day Dots to the Render Pass using the bloom shader buffer to apply the effect, and the bloom would go through a dot filter to generate the Ben-Day dot.

Recreating Moebius style

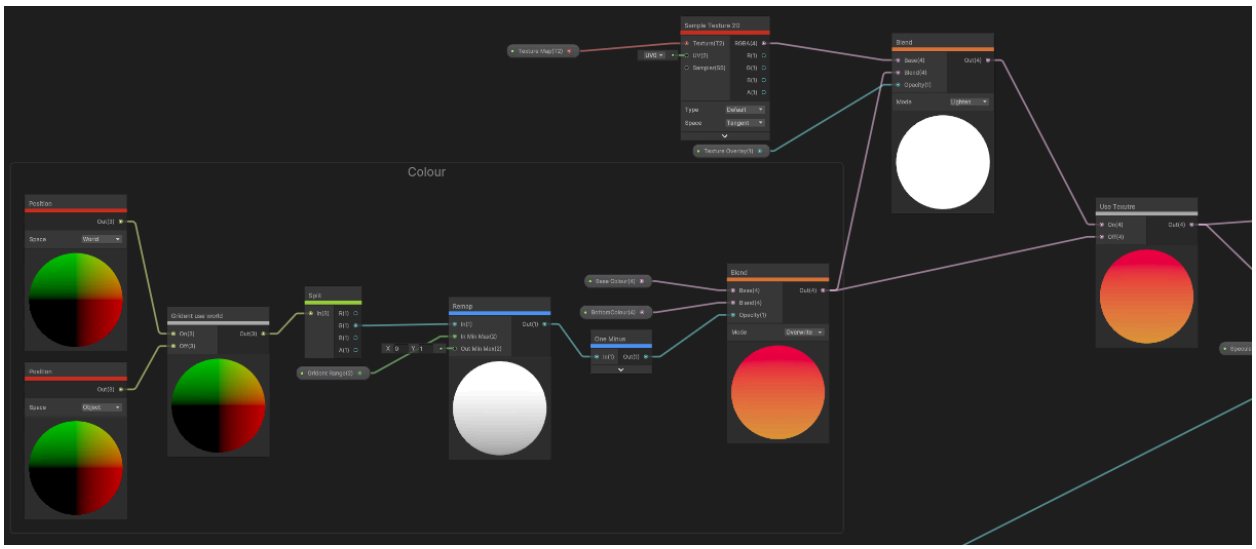
My third art style was based on the artist Jean Giraud, aka Moebius. He worked on many things, including Marvel comics and even some Spider-Man artwork.
To recreate his style. To recreate his style, I need to make a two-colour gradient material, which will also have cross-hashing similar to spider verse and specular highlights. He used bright and bold colours with black outlines.

Unity Material

2 Colour Gradient

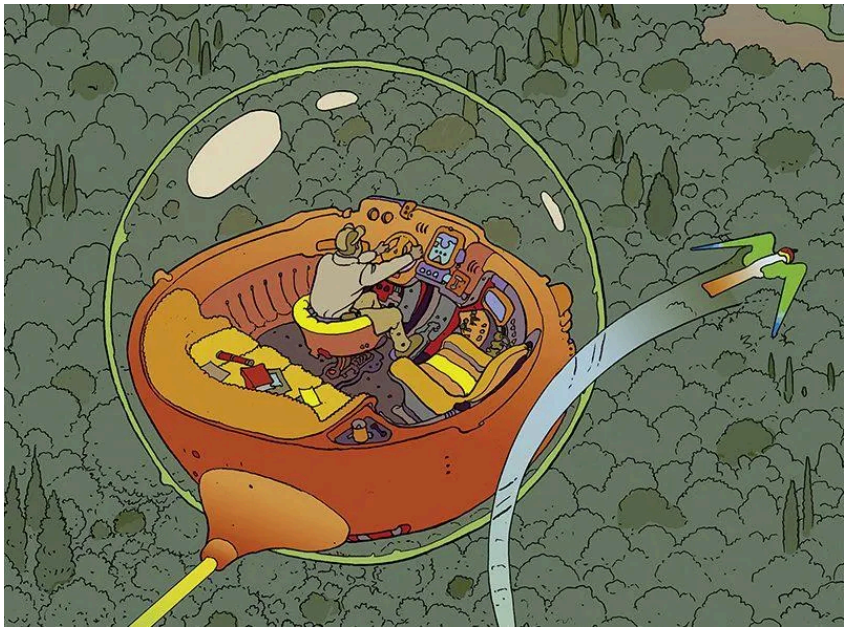
To create a gradient, I first made a mask using the position data of the object, then split this data to RGB, which for the position data would mean X Y Z. Getting the Y, I could remap it to user-defined levels, allowing for better control on when the change of the 2 colours happens. As this is only

for 2 Colours, I could use linear interpolation (lerp) between them, which I used previously for the code inside of my 5 colour gradient node. Instead, I used the position data as an alpha mask and blended two colours using the blend node. To allow for any texture to show, I then blend this on top of a texture map if one is provided.



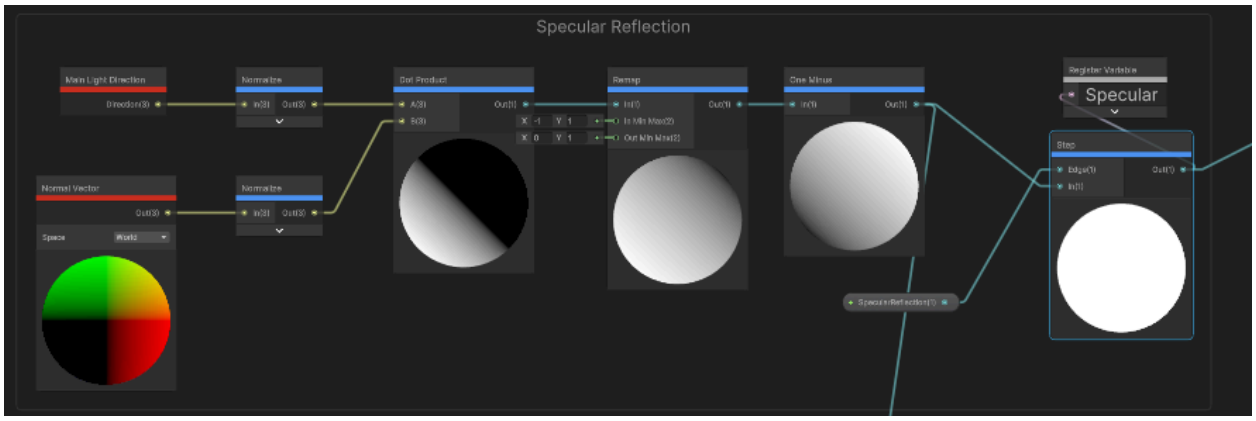
Specular Highlight

With Moebius, round objects have round specular highlights that are a different colour, ordinarily white.



To achieve this effect, I calculated the dot product between the light direction and world normal to get where light is hitting the object. This then goes through more nodes to set the values to between 0 and 1 and invert it

before going through a step node, which would make any values less than the edge to 0 and make everything else 1. This gives a texture that is mostly pure black with a pure white circle for the specular highlight.



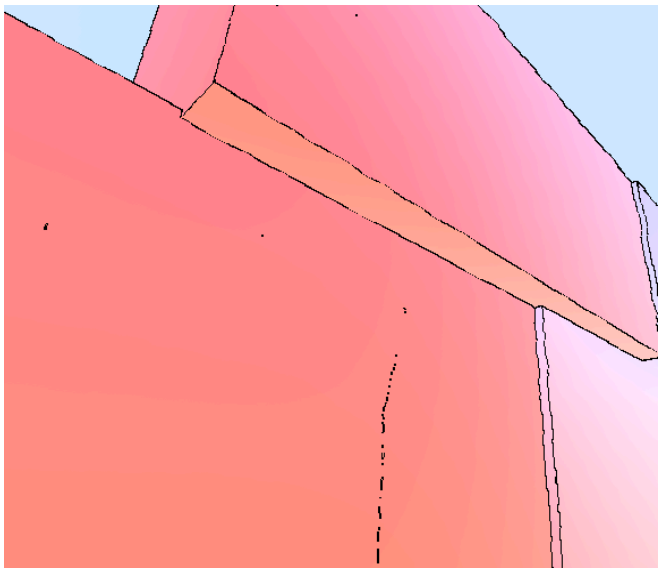
Cross-hashing

I went back on some of my cross-hashing from the spider verse material and tried to see if it looked good with this shader but like with the spider-verse. It did not look good, so I removed it from the final project.

Normals

For the normals, I took inspiration from Lenny, also known as Uselessgamedev, who had made a Moebius-style shader before which he used in his game, [Montrouge Grand Prix] and documented it with a YouTube video, to get some info on how I would be able to change the normals so it would affect the normal buffer differently for the render pass.

The basics of this are blending the black and red texture map, where red would be any details, with the object's normal vector. With the ability to change the Hue of the normal map to change the contrast of the object to other objects giving clear lines. This also works on normal normal maps that use all the values, like from premade assets.



To add an outline to the specular highlights, I took the specular highlights before blending them with the colour and adding that to the normals. I also tried this with the cross-hashing, but it still did not look nice, so I just removed it.

Sky material

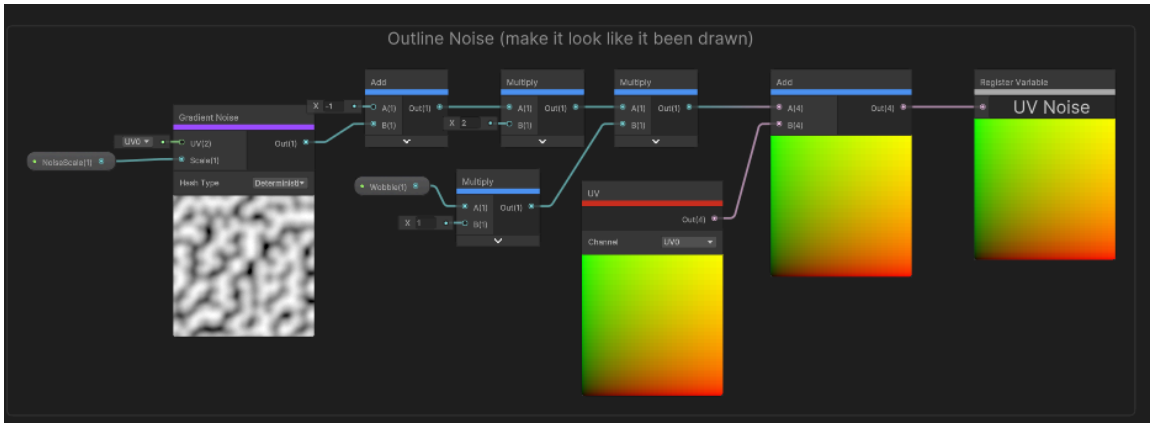
I modified my object material to create a sky material. This was done by getting rid of all the things I didn't need, like the normal textures, and modifying the colour to have a different-coloured sky, horizon, and bottom. This applies to a sphere that covers the whole level.

Moebius Render Pass

I used a Render Pass to make the colours bright and add to the outline.

Outline

For the outline, I used the [Spider-Verse edge detection]() but changed the UV to have some noise in it, using gradient noise and adding it to the UV. This effect makes the outline wobble and makes it feel more like it was drawn by hand.

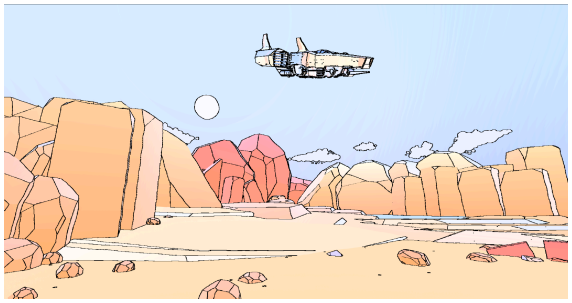


Making colours bright

To make the colours bright, I got the scene buffer and converted it from RGB to HSV. I then changed the value that would be blue when split into RGB to 1 before converting it back to RGB.

Final Result

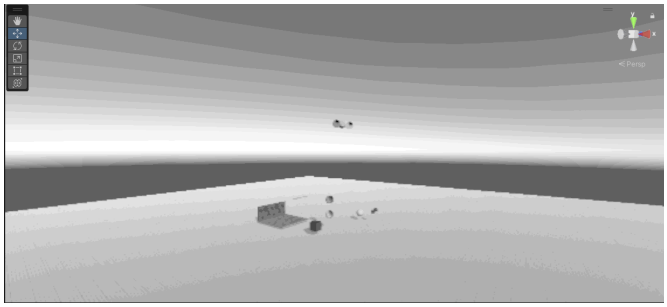
I am most happy with this shader. It's very much like an inspection. However, this shader does not show any shadows, which is what the cross-hashing was meant to help with. However, it did help make the colours bold and bright.

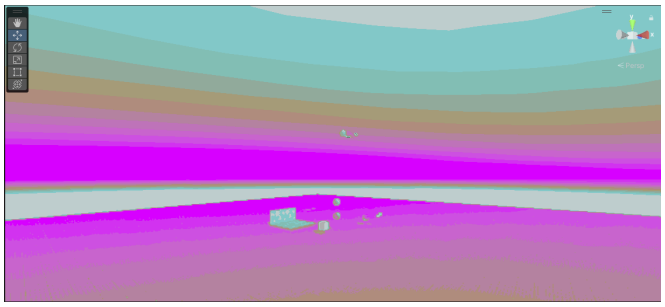


Other Render Features

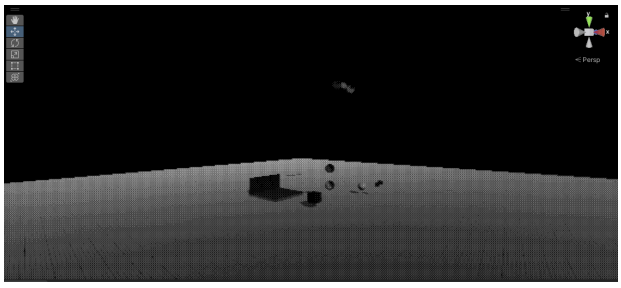
Colour Swap Render Feature

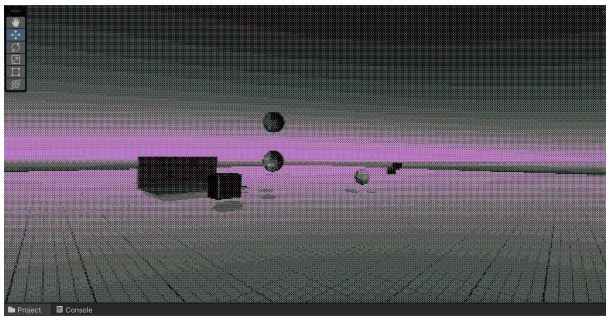
I created this render pass inspired by my 2nd attempt at the [Arcane style render feature].
I started with a black-and-white filter on the scene output, which I used as a mask input for the gradient sampler. I put the mask through a posterise node, which decreased the number of tones in the image.



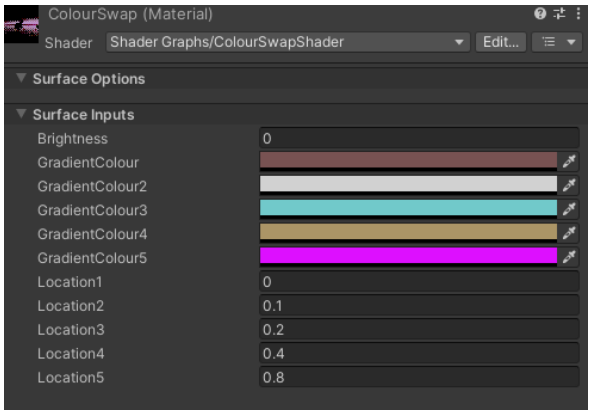


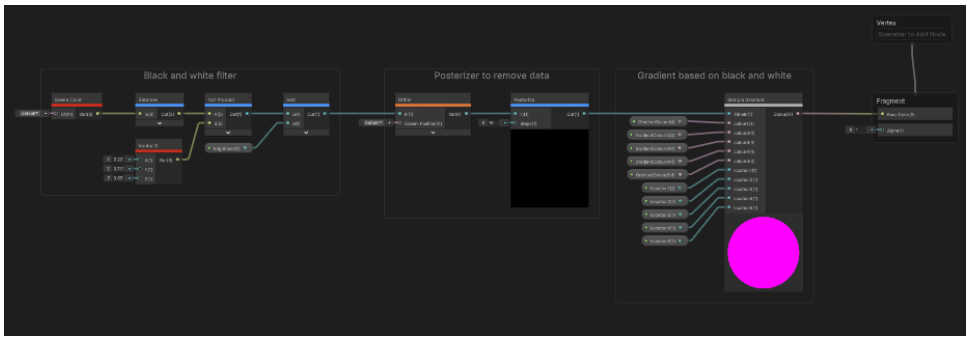
I then put it through a dither node, which broke a part of the mask to hide the visible colour banding that happens with the posterized node, making it easier on the eyes.





This resulted in a retro-looking shader that can be used to create an old-looking game.



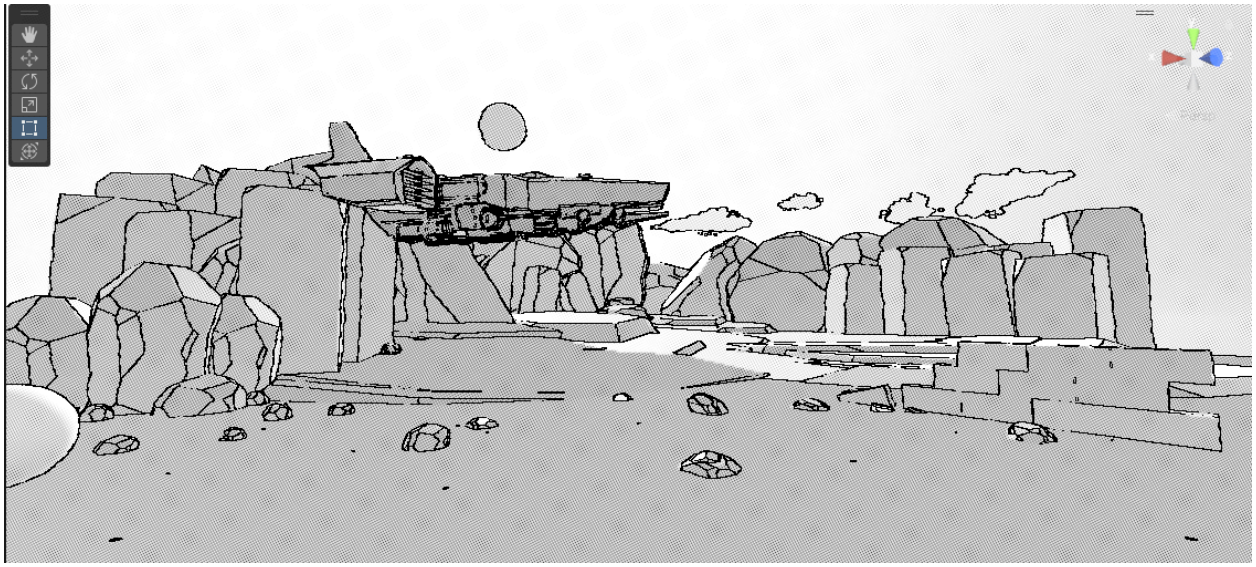


Line Art

I had this idea when I was creating the Moebius-style Render Pass. Instead of increasing the output value, I set the hue and saturation to 0 but kept the value, which resulted in a black-and-white image. I then used my dot generator from previous shaders and attached that to the black-and-white image scene. This was done similarly to the dither node for the Colour swap, where it helped break the image up but also added some texture to the image.

I then used the outline function from the Moebius render pass to create a drawn outline by overlaying it over the image.

This shader resulted in an art style that is more like hand-drawn line art. This could be useful for a game that wants to look like paper.

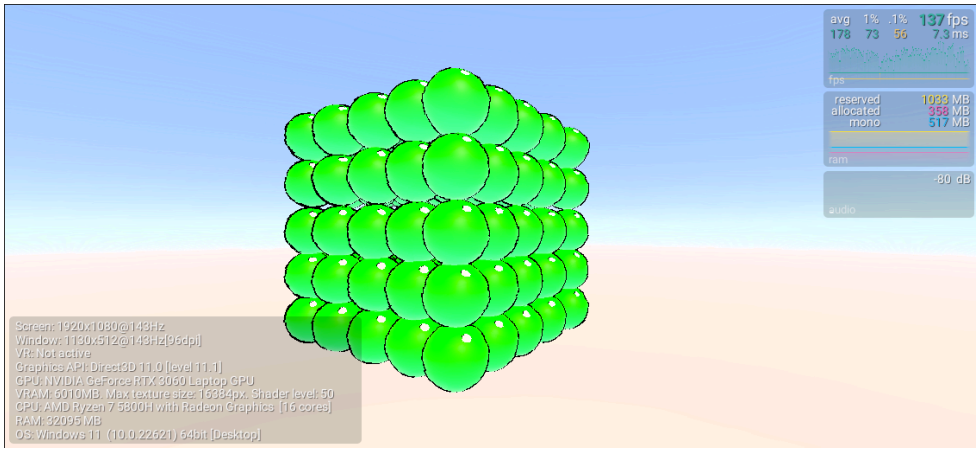


Testing

To ensure good performance, I created a scene with 125 spheres assigned to the materials. To see how well it holds up, I first tested it with the sun rotating, causing the lighting to change and, therefore, the values on the shader. It also changes the material's colour, which can be quite performance-intensive as it needs to render a different colour.

I installed a plugin called “[Graphy] - Ultimate FPS Counter - Stats Monitor & Debugger,” which shows the FPS and a graph I used to measure the performance. I also used the profiler to see if the FPS change was caused by rendering.

My findings showed that with just the material shaders, performance was the same but was better with any unlit shaders (Hand-painted look). I assume this was because it did not calculate any shadows or additional lights except for the main light.



Any of the render feature shaders had a performance drawback, especially if it does a lot of calculations. The worst performance came from the Kuwahara filter shader, which didn't even work, but the reset worked fine and would still be playable.

Material + style	FPS Without RF	FPS RF	Performance Hit	Performance Hit (RF)
Built-in URP	220	NULL	0.00%	NULL
Arcane Unlit	226	NULL	2.73%	NULL
Arcane lit	215	NULL	-2.27%	NULL
Spider-Verse	212	198	-3.64%	-6.60%
Moebius	208	182	-5.45%	-12.50%
LineArt (RF)	220	200	0.00%	-9.09%
Colour Swapper (RF)	220	190	0.00%	-13.64%

Here is the spreadsheet for my findings using the average FPS after 10 sec. RF refers to the render feature. Under “Material + style,” materials marked with RF are not materials but just render feature testing using the Built-in URP as the material on the object. Anything with NULL does not have any testing with it. The performance hit is the difference between the material and the Built-in URP shader. The RF column is the difference in FPS with the render feature turned off and on.

As shown by the data, there is not much of a performance impact due to the custom render shaders, which are a maximum of 20% lower than default URP materials and rendering.

Project outcomes

With the project, I learned a lot about shaders inside Unity and how to create them to fit a certain style. Some stuff I wish I could improve on was with the render pass, as I felt like with this project, I only touched the surface of what I could achieve with them. However, I achieved my aim of creating shaders that achieve a certain look, especially with the Moebius shader. The working shaders performed without too much of a performance drop.

If I had more time with the project, I would have spent learning more about .shader files and how to use them inside of Unity, alongside creating more complex shader graphs which could take in more than just colour, normal and depth textures like shadows and bloom, as those would have provided a larger amount of customisation which would not be possible to do without them.

References

Anisotropic Kuwahara Filtering on the GPU (2010). Available at: https://www.kyprianidis.com/p/gpupro/ (Accessed: 14 December 2023).

Anvil - Download Free 3D model by Arcane_designs (2023). Available at: https://sketchfab.com/models/17f739f328c14155838fd45bc9924280/embed?autostart=1 (Accessed: 11 January 2024).

ARCANE - Jinx - Modeling & Texturing, Thibaut Granet (2022) *ArtStation*. Available at: https://www.artstation.com/artwork/X1aWVw (Accessed: 1 November 2023).

Arcane - Netflix (no date). Available at: https://www.netflix.com/browse?jbv=81435684 (Accessed: 2 November 2023).

Boysen, J. (2021) 'Reaching for the stars ✨', *Medium*, 18 July. Available at: https://medium.com/@jannik_boysen/procedural-skybox-shader-137f6b0cb77c (Accessed: 31 March 2024).

Color Quantization and Dithering (2022). Available at: https://www.youtube.com/watch?v=8wOUe32Pt-E (Accessed: 4 March 2024).

Comic Book Style Drawing (2017) *ZBrushCentral*. Available at: http://www.zbrushcentral.com/t/comic-book-style-drawing/339837 (Accessed: 6 November 2023).

Cyan (2024) 'Cyanilux/ShaderGraphVariables'. Available at: https://github.com/Cyanilux/ShaderGraphVariables (Accessed: 28 February 2024).

Download Free 3D Models - Royalty Free (2017) *Sketchfab*. Available at: https://sketchfab.com/features/free-3d-models (Accessed: 31 March 2024).

Download Free 3D Models - Royalty Free (2020a) *Sketchfab*. Available at: https://sketchfab.com/features/free-3d-models (Accessed: 31 March 2024).

Download Free 3D Models - Royalty Free (2020b) *Sketchfab*. Available at: https://sketchfab.com/features/free-3d-models (Accessed: 31 March 2024).

Download Free 3D Models - Royalty Free (2022) *Sketchfab*. Available at: https://sketchfab.com/features/free-3d-models (Accessed: 31 March 2024).

Download Free 3D Models - Royalty Free (2024) *Sketchfab*. Available at: https://sketchfab.com/features/free-3d-models (Accessed: 31 March 2024).

Explore 3D Models (2022) *Sketchfab*. Available at: https://sketchfab.com/3d-models?date=week&features=downloadable&sort_by=-likeCount (Accessed: 11 January 2024).

Giraud, J. (no date) *Mœbius Production — Editions, artprints and exhibitions!* Available at: https://www.moebius.fr/ (Accessed: 14 December 2023).

Gooch, A. *et al.* (1998) 'A non-photorealistic lighting model for automatic technical illustration', in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98. the 25th annual conference*, Not Known: ACM Press, pp. 447–452. Available at: https://doi.org/10.1145/280814.280950.

[Graphy] - Ultimate FPS Counter - Stats Monitor & Debugger | GUI Tools | Unity Asset Store (no date). Available at: https://assetstore.unity.com/packages/tools/gui/graphy-ultimate-fps-counter-stats-monitor-debugger-105778 (Accessed: 1 May 2024).

Ilett, D. (no date) *Image Effects | Part 6 - Painting Joy, Daniel Ilett: Games | Shaders | Tutorials*. Available at: https://danielilett.com/2019-05-18-tut1-6-smo-painting/ (Accessed: 15 November 2023).

Jinx's Firelight bomb - Download Free 3D model by PHuttoncg (@rainbowbunny) (2022). Available at: https://sketchfab.com/models/b4c3a2fb87d6479fb4945663d70b263f/embed?autostart=1 (Accessed: 11 January 2024).

Kyprianidis, J.E. *et al.* (no date) 'Anisotropic Kuwahara Filtering with Polynomial Weighting Functions'.

Learn how to do stylized shading with Shader Graph - SIGGRAPH 2019 (2019). Available at: https://www.youtube.com/watch?v=4XxfiwNqU4I (Accessed: 31 March 2024).

Lu, J., Sander, P.V. and Finkelstein, A. (2010) 'Interactive painterly stylization of images, videos and 3D animations', in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. I3D '10: Symposium on Interactive 3D Graphics and Games*, Washington D.C.: ACM, pp. 127–134. Available at: https://doi.org/10.1145/1730804.1730825.

Moebius (2024) *Marvel Database*. Available at: https://marvel.fandom.com/wiki/Moebius (Accessed: 2 May 2024).

Moebius-style 3D Rendering | Useless Game Dev - YouTube (no date). Available at: https://www.youtube.com/watch?v=jlKNOirh66E (Accessed: 14 December 2023).

NedMakesGames/RendererFeatureBlitMat (no date). Available at: https://github.com/NedMakesGames/RendererFeatureBlitMat (Accessed: 2 May 2024).

Piltover Crate - Download Free 3D model by Zeruel (@792) (2023). Available at: https://sketchfab.com/models/3343a831330346a1a716d64e639824f0/embed?autostart=1 (Accessed: 11 January 2024).

Publishing, J., Espíndola, F. and Santalla, D. (2022) *The Unity Shaders Bible: A linear explanation of shaders from beginner to advanced. Improve your game graphics with Unity and become a professional technical artist.* Translated by M. Clarke. Erscheinungsort nicht ermittelbar: Independently published. Available at: https://www.jettelly.com/books/unity-shaders-bible/.

radiangames (2022) 'Using compute shaders to speed up your game while making it look a lot better too (as seen in Instruments of Destruction)', *r/Unity3D*. Available at: www.reddit.com/r/Unity3D/comments/spbntj/using_compute_shaders_to_speed_up_your_game_while/ (Accessed: 7 November 2023).

RendererFeatureBlitMat/Assets/Rendering/Desaturate/BlitMaterialFeature.cs at master · NedMakesGames/RendererFeatureBlitMat (no date) *GitHub*. Available at: https://github.com/NedMakesGames/RendererFeatureBlitMat/blob/master/Assets/Rendering/Desaturate/BlitMaterialFeature.cs (Accessed: 25 March 2024).

Sharda, A. (2018) 'How Image Edge Detection Works', *Medium*, 30 September. Available at:

https://aryamansharda.medium.com/how-image-edge-detection-works-b759baac01e2 (Accessed: 30 April 2024).

Spider-Man: Into the Spider-Verse - Netflix (no date). Available at: https://www.netflix.com/browse/m/genre/801362?jbv=81002747 (Accessed: 2 November 2023).

stevewhims (2021) *for Statement - Win32 apps*. Available at: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl-for (Accessed: 1 May 2024).

Stylized Smart Material 2.0 for Substance 3D Painter - Announcement and How to (2021). Available at: https://www.youtube.com/watch?v=6RPZG_-ZPFI (Accessed: 2 November 2023).

Substance Painter: Beginner Stylized Smart Material Tutorial [3 SIMPLE STEPS] (2019). Available at: https://www.youtube.com/watch?v=4X-4Elbyjww (Accessed: 2 November 2023).

Technologies, U. (no date) *Unity - Manual: Compute shaders*. Available at: https://docs.unity3d.com/Manual/class-ComputeShader.html (Accessed: 7 November 2023).

The Art of Moebius (2017) *Character Design References*. Available at: https://characterdesignreferences.com/artist-of-the-week-3/moebius (Accessed: 21 November 2023).

Toon Outlines in Unity URP, Shader Graph Using Sobel Edge Detection! ✔️ 2020.3 | Game Dev Tutorial (2020). Available at: https://www.youtube.com/watch?v=RMt6DcaMxcE (Accessed: 25 March 2024).

Torchinsky, A. (2020) *Cel-shading: some tricks that you might not know about*. Available at: https://torchinsky.me/cel-shading/ (Accessed: 2 November 2023).

Tyolie (2020) 'Trying to create my own shaders using Shader Graph', *r/Unity3D*. Available at: www.reddit.com/r/Unity3D/comments/kn9jaw/trying_to_create_my_own_shaders_using_shader_graph/ (Accessed: 7 January 2024).

Unity Outline Shader Tutorial at Roystan (no date). Available at: https://roystan.net/articles/outline-shader/ (Accessed: 28 February 2024).

Unity Shader Graph - Simple Edge Detection Shader (2024). Available at: https://www.youtube.com/watch?app=desktop&v=AtYhrsyxHB0&vl=en (Accessed: 28 February 2024).

Watch Spider-Man: Across The Spider-Verse - Bonus X-Ray Edition | Prime Video (no date). Available at: https://www.amazon.co.uk/gp/video/detail/amzn1.dv.gti.d5595418-c842-47be-99de-8244768719e1?autoplay=0&ref_=atv_cf_strg_wb (Accessed: 7 November 2023).

Zahed, R. (2018) *Spider-Man: Into the Spider-Verse: The Art of the Movie*. Illustrated edition. London: Titan Books Ltd.

Zahed, R. (2023) *Spider-Man: Across the Spider-Verse: The Art of the Movie*. 1st edition. New York: Abrams.

Meeting reports (continuing from first-hand in)

Meeting Report 7

Date: 06/03/2024

What we talked about?
With Artur and other classmates, we showed work at the current stage of development, gave feedback to each other, and talked about what we had done.

Meeting Report 8

Date: 24/04/2024

What we talked about?
We talked about the final report what we should include in it, how the viva will work and what we need to submit for the final year report.