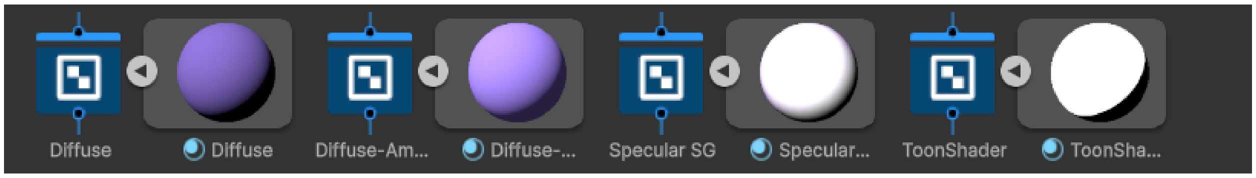
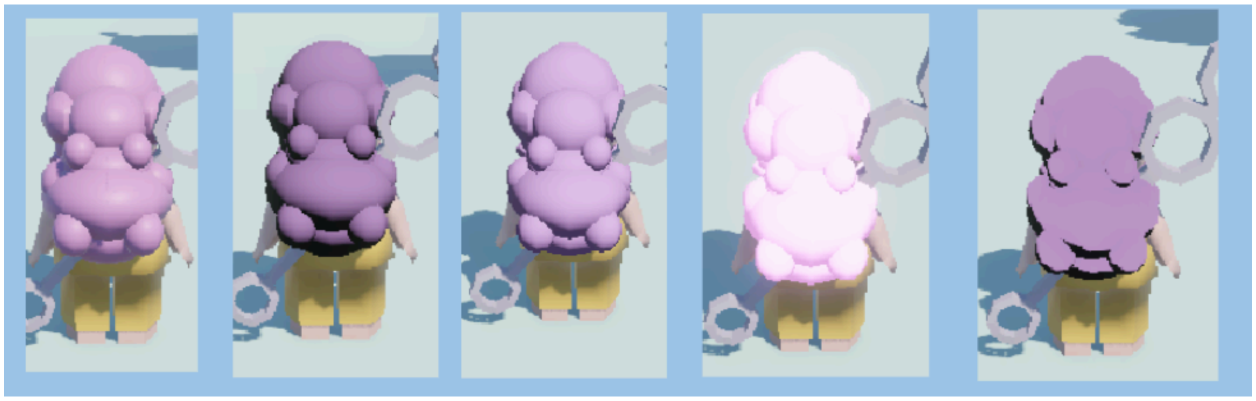
Hi there - We're Ashley Klahm and Katie Palmerio, and this is our progress report on our Computer Graphics Project. We're here today to show you what we've been working on for our project, called Bubble Floaty Mc3D. For context this is a play on words relating to a game I made in programming called "Bubbly Floaty McGee" and we brought it back by popular demand.

Here's the outline of our presentation; we will be starting with a brief overview of the base deliverables that we've implemented, then an overview of various illumination shaders that we've implemented, then showcasing the colour grading that we've added, and then closing out with demonstrating the required shader implementations we've done.

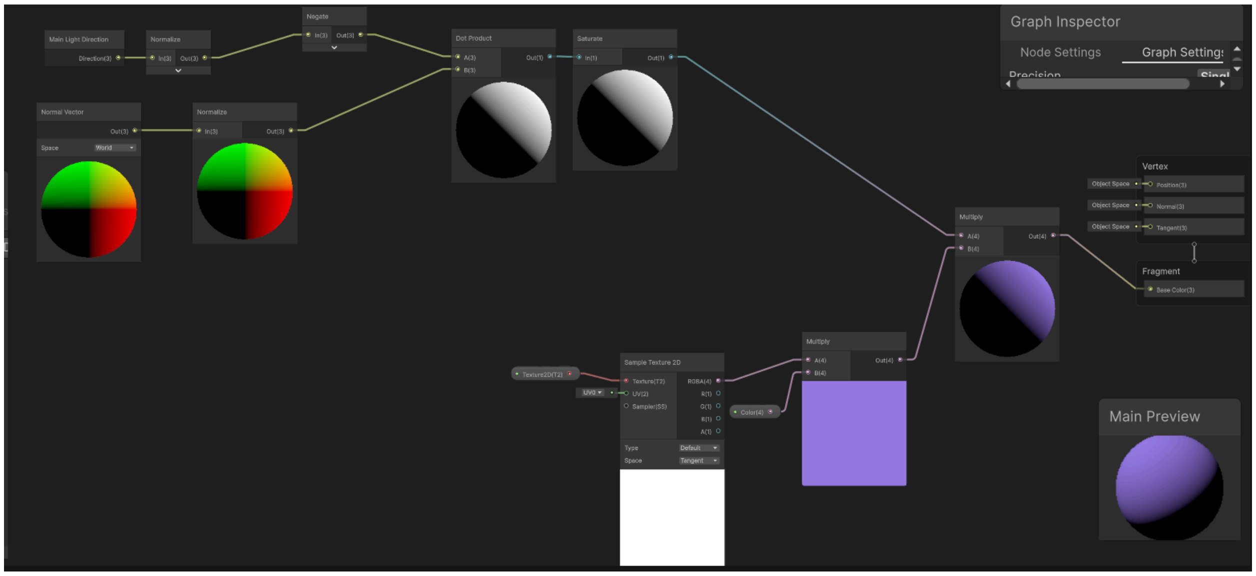
Let's start with a quick rundown of the base deliverables. For our dynamic game objects we have bubbles, the objects that you must collect to win the game, the player character, Bubbly Floaty McGee, and Bog the Bubble Dog who will say dialogue for the final submission. Our playable main character must go from platform to platform collecting bubbles scattered around the scene to fit the theme of the game. For our games win/lose condition you are able to win when you collect all of the bubbles, and you lose when you fall out of the map and get teleported back to spawn. As for the overall game aesthetic we have a simple pastel visual style for everything in the scene to keep it simple and fun!

For the illumination deliverables, we implemented the three required shaders plus a toon shader by using Shader Graph. The changes in illumination are showcased through changing the shader for our protagonist Bubble Floaty McGee's hair.

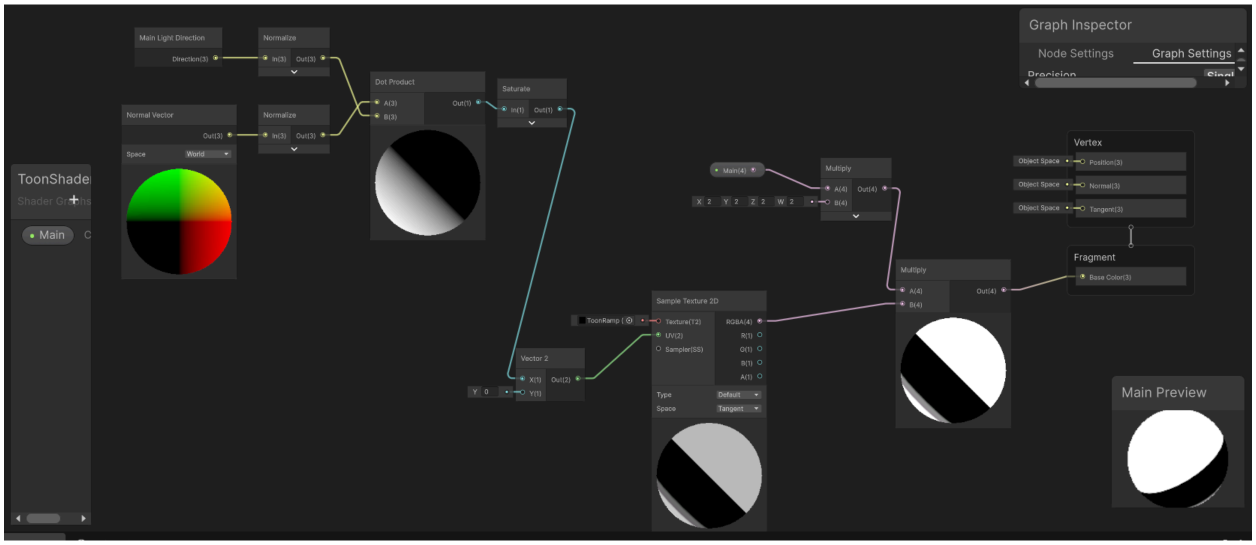




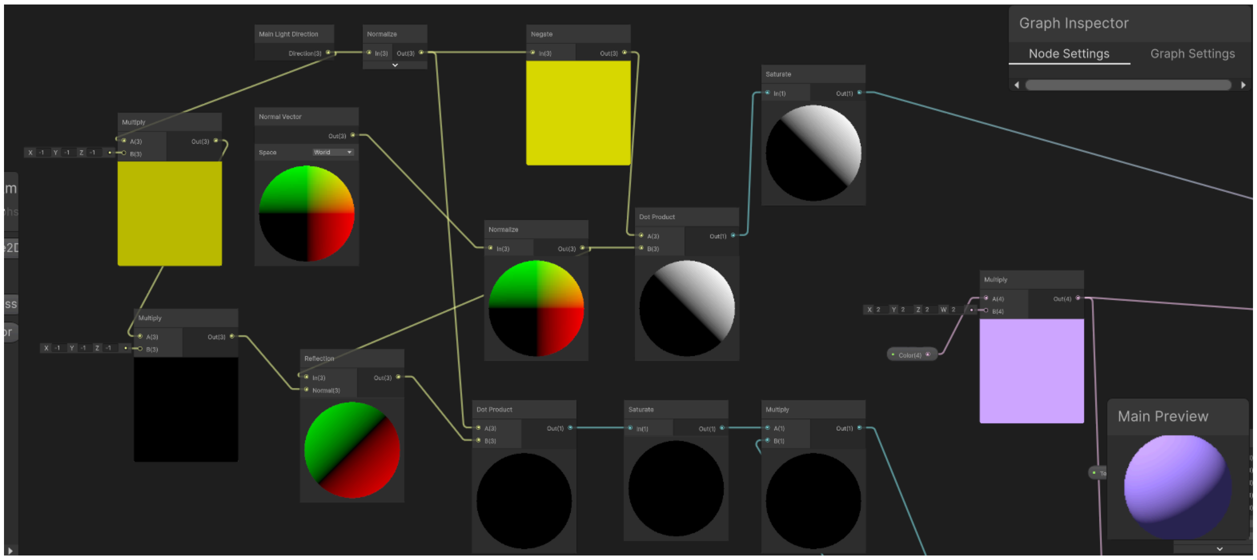
We decided to go with a toon shader for our extra shader, as it fit the simplistic and cartoony aesthetic of our project and just overall looked neat. Implementing each of them through Shader Graph was easier to work with than writing shader code, so that's why we went with this choice.
Diffuse Lighting was simple to implement, as we were just taking the main light direction and normal vector and normalizing both, then doing the dot product of both and saturating before multiplying with the texture.

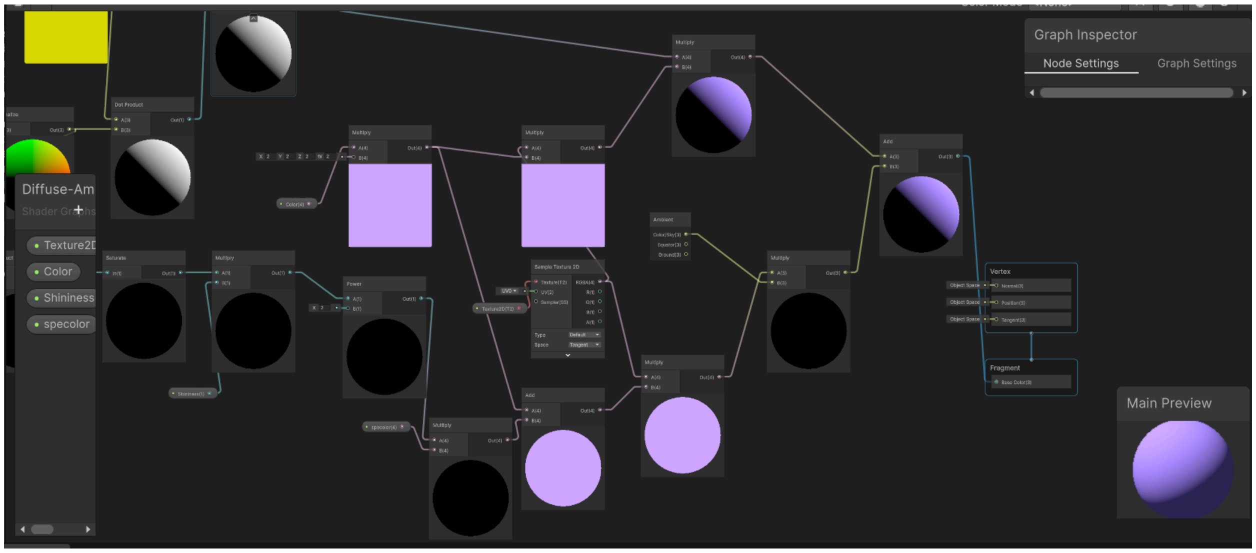


The toon shader was simple as well, having a similar process with the main light direction and normal vector normalization, dot product, and saturation, but after that we ran it through a Vector2 node, the sample texture we were using, and then multiplying it with the main colour.

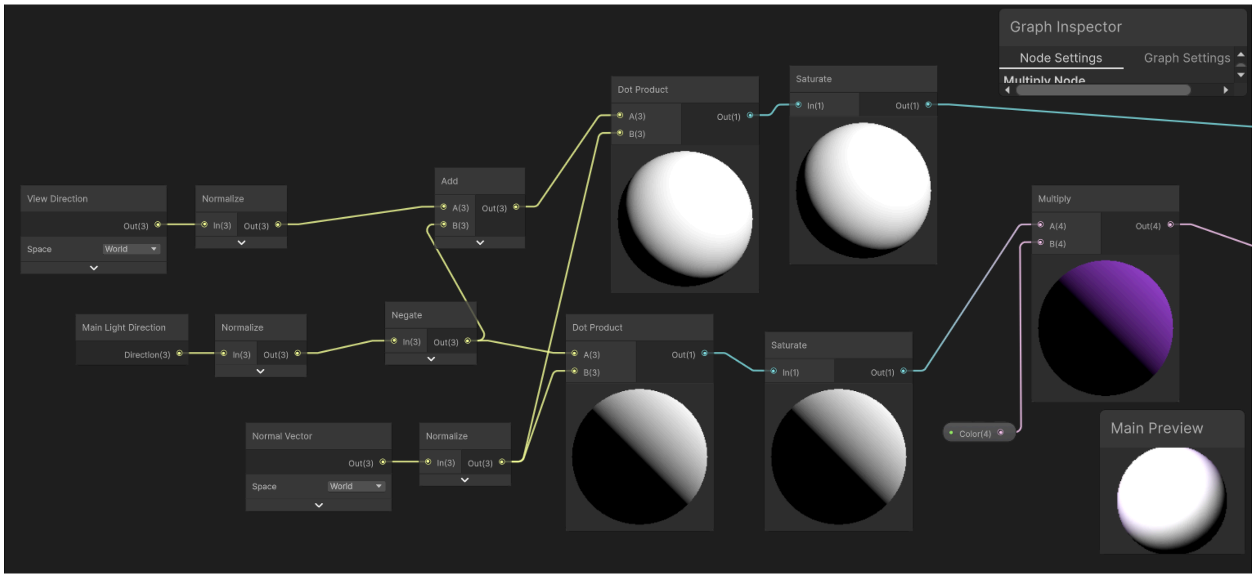


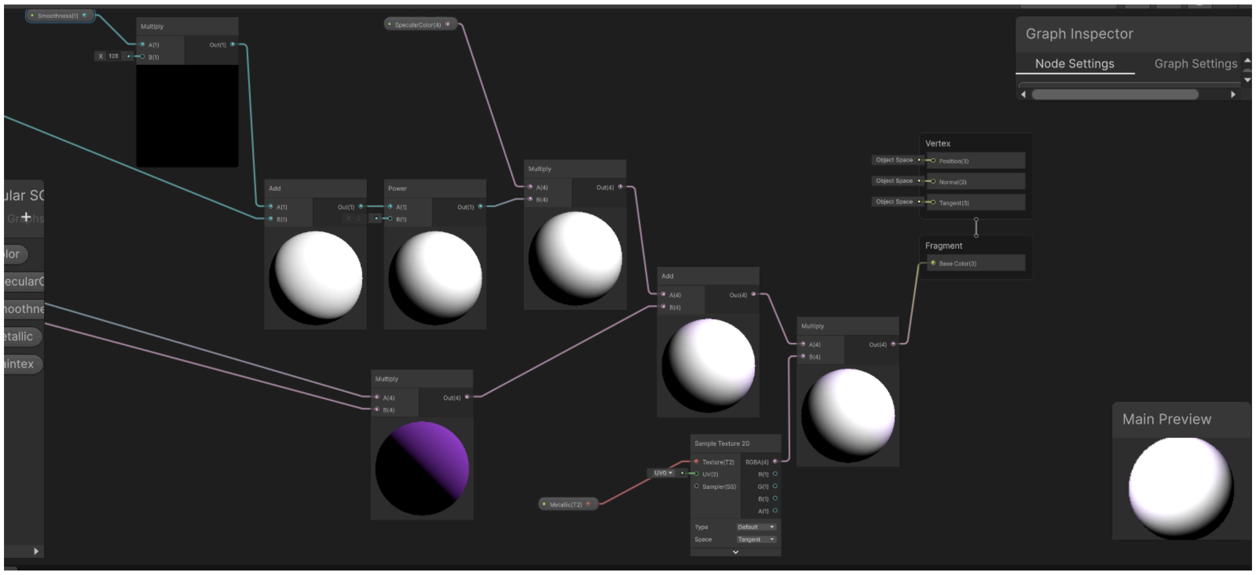
The Diffuse-Ambient shader builds off of the Diffuse shader, and just adds some more extra steps to it. One key change is reflecting the normalized normal vector and the multiplied main light direction before taking the dot product of them, which reflects the entire shader and changes it drastically. Alongside extra multiply, add and power nodes, all of this combines together to create a nice shader effect.





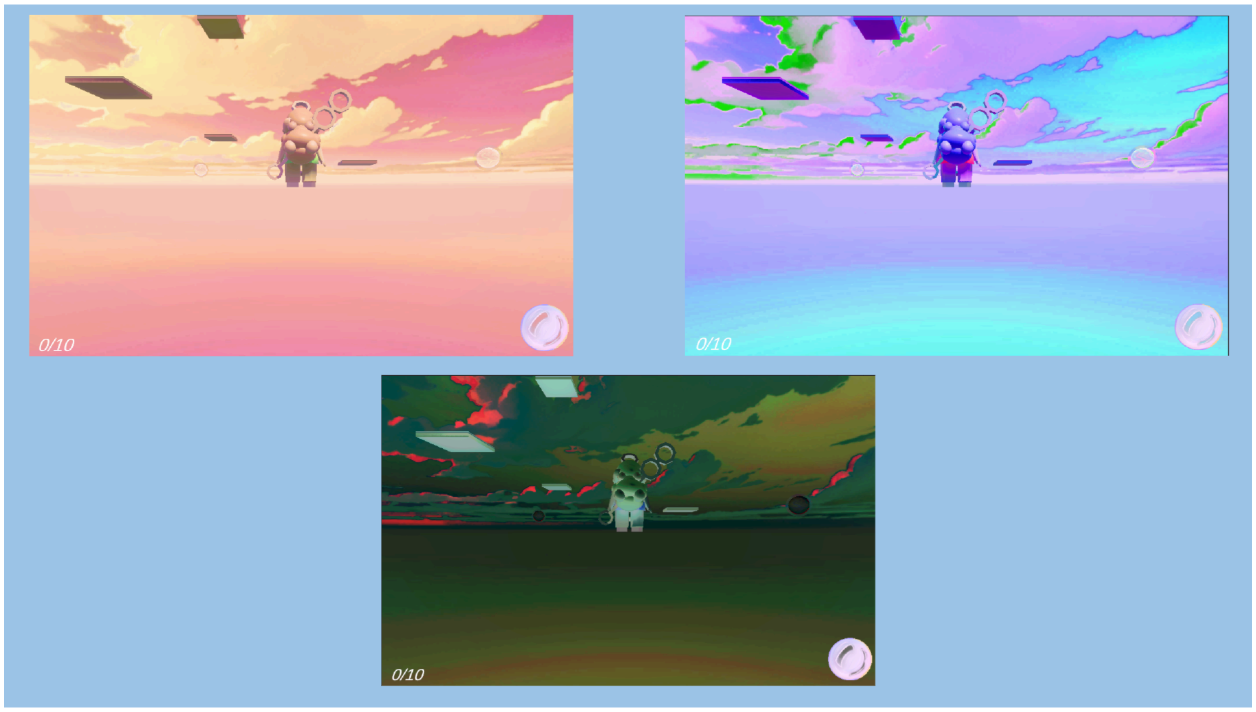
The Simple Specular shader utilizes view direction, main light direction, and the normal vector as its foundations. The important part here is the smoothness value multiplied and added to the shader after the dot product and saturation have finished, as this value smooths down the harsh light produced by the shader and makes it more pleasing to the eyes. After that, more power, multiply, add and texture nodes are added to the shader in order to complete it. A metallic texture is also applied here to add more flair to the final shader.



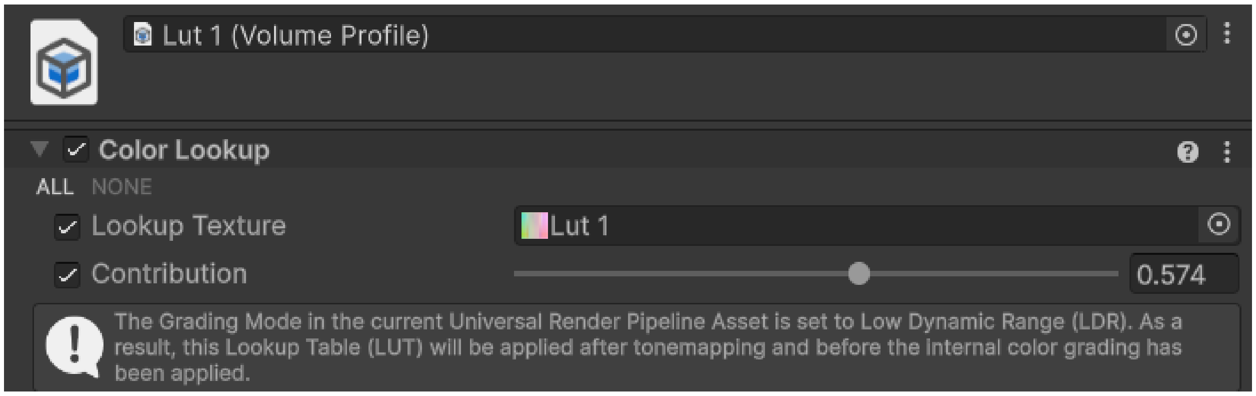


For all the illumination the negate node was included to better work visually with the scene and the main character's hair.

For colour grading, we used Volume Profiles to easily and effectively implement toggleable colour grading. We chose to implement three colour gradings, one with soft warm lighting, one with harsh neon lighting, and one that inverted the colours onscreen.



We did this through the Volume Profile asset's Color Lookup function, where you could assign an LUT texture and it would overlay it across the entire screen. We then assigned each colour grade a game object and then had its display be toggleable depending on which number key you pressed. This method of implementing colour grades was both fast and efficient, and was interesting to discover on our own.


Lut 1 (Volume Profile)

Color Lookup

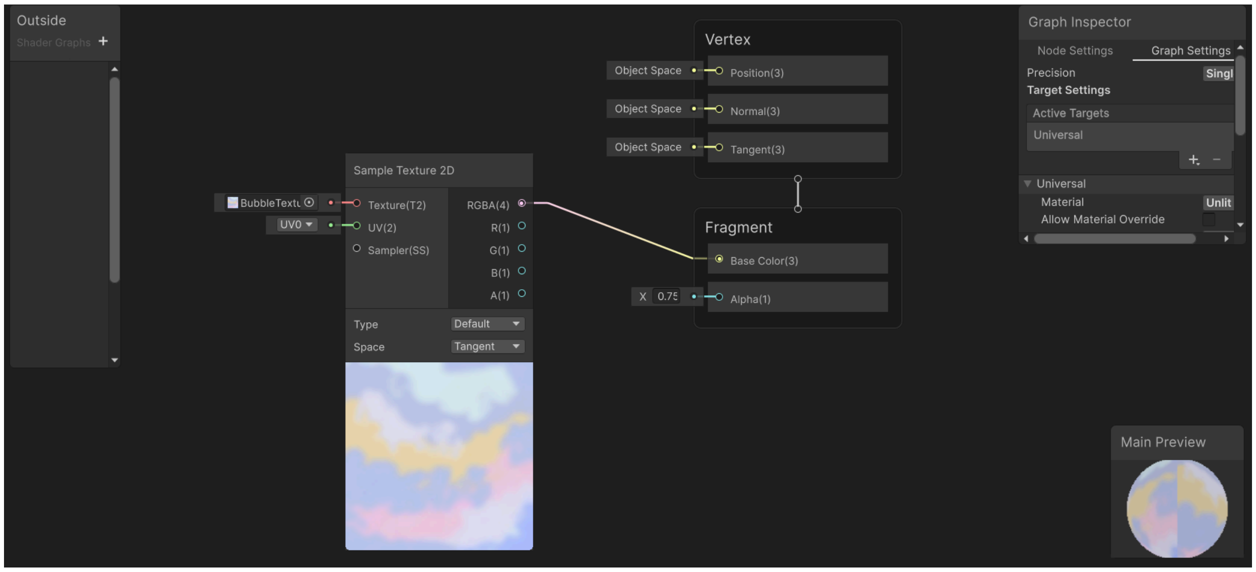
ALL NONE

Lookup Texture: Lut 1

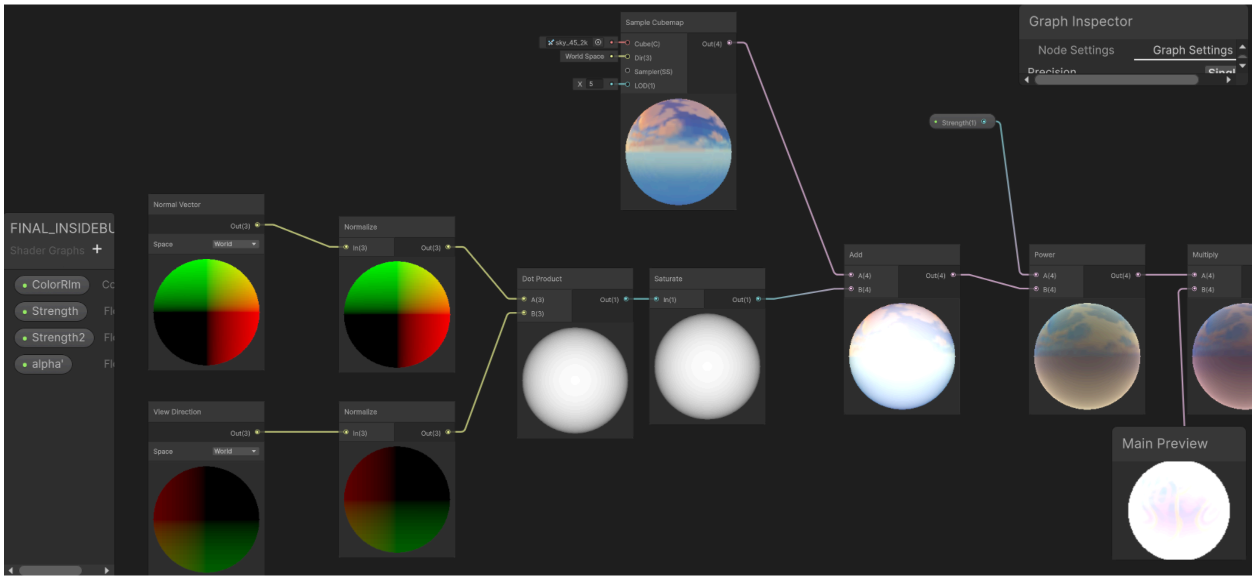
Contribution: 0.574

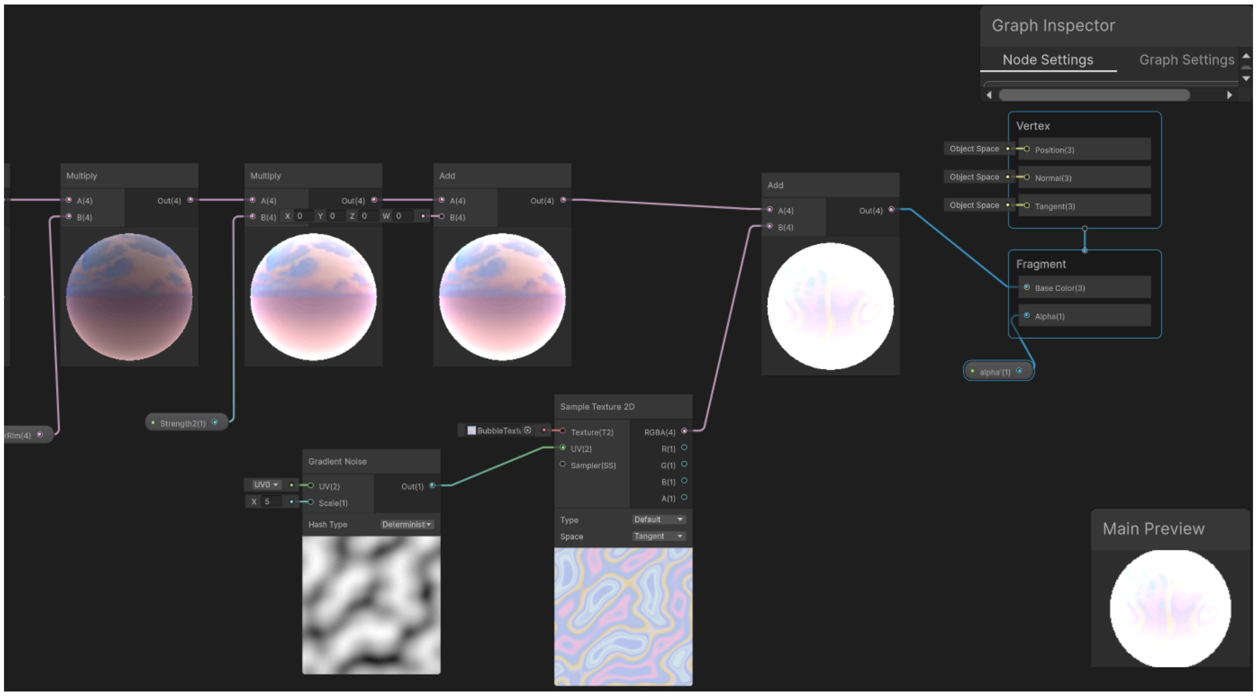
The Grading Mode in the current Universal Render Pipeline Asset is set to Low Dynamic Range (LDR). As a result, this Lookup Table (LUT) will be applied after tonemapping and before the internal color grading has been applied.

For the three other shader effects that were required, we decided to apply them to the bubbles and platforms. The bubbles are actually made up of two different shaders in order to achieve its look. The first shader used here is a simple texture shader at 75% opacity applied to the outside layer of the bubble.

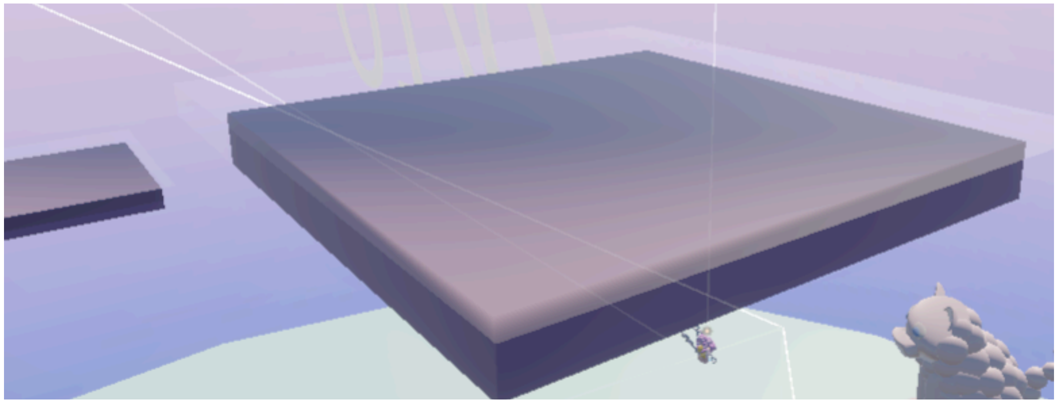
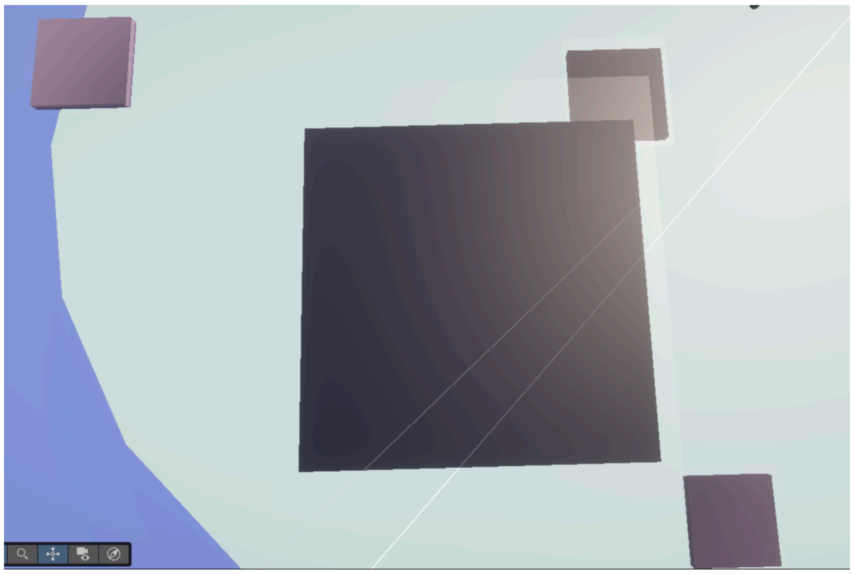
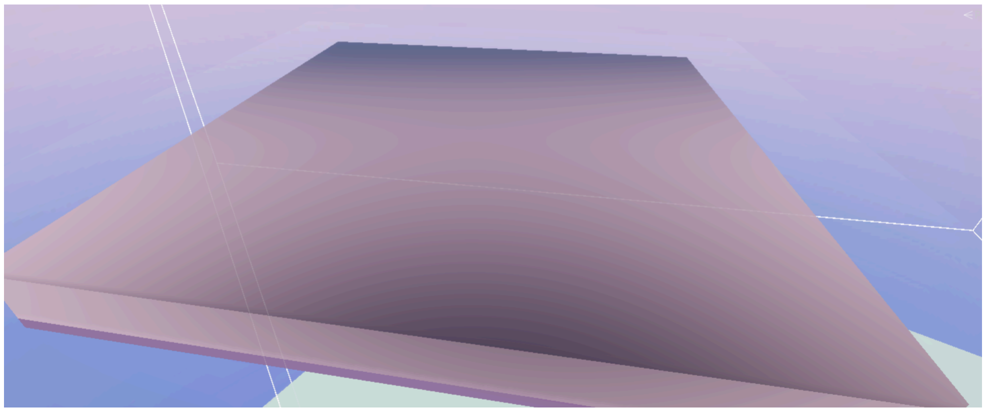


This is done to achieve a slightly transparent effect, much like bubbles in real life. The second shader used here is a modified rim lighting shader, where once the normal vector and view direction are combined using the dot product and saturated, a texture is added before the shader is multiplied by the power of 0.2, multiplied by the rim colour, and then multiplied by 4 before a gradient noise and second texture are added to complete the shader.

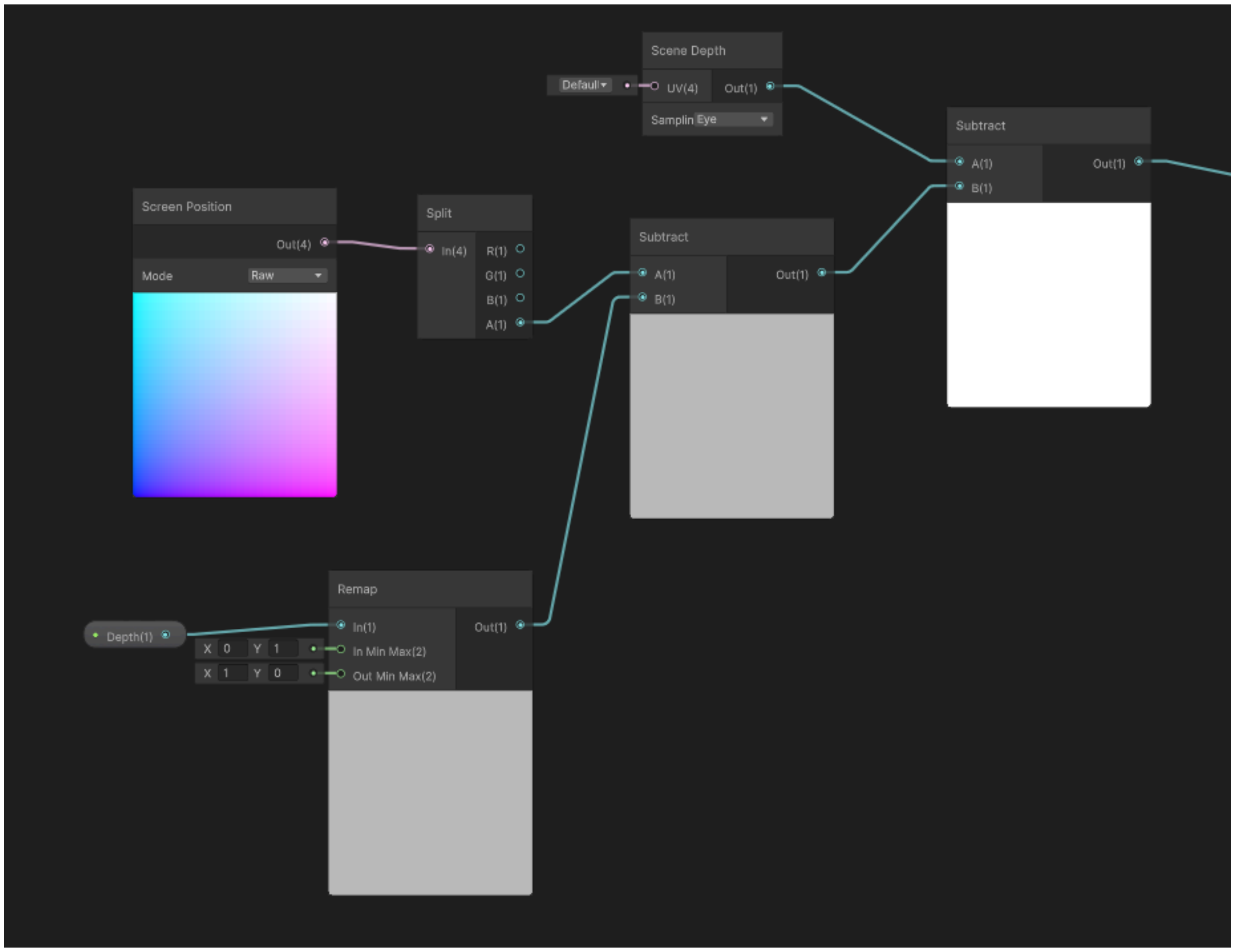




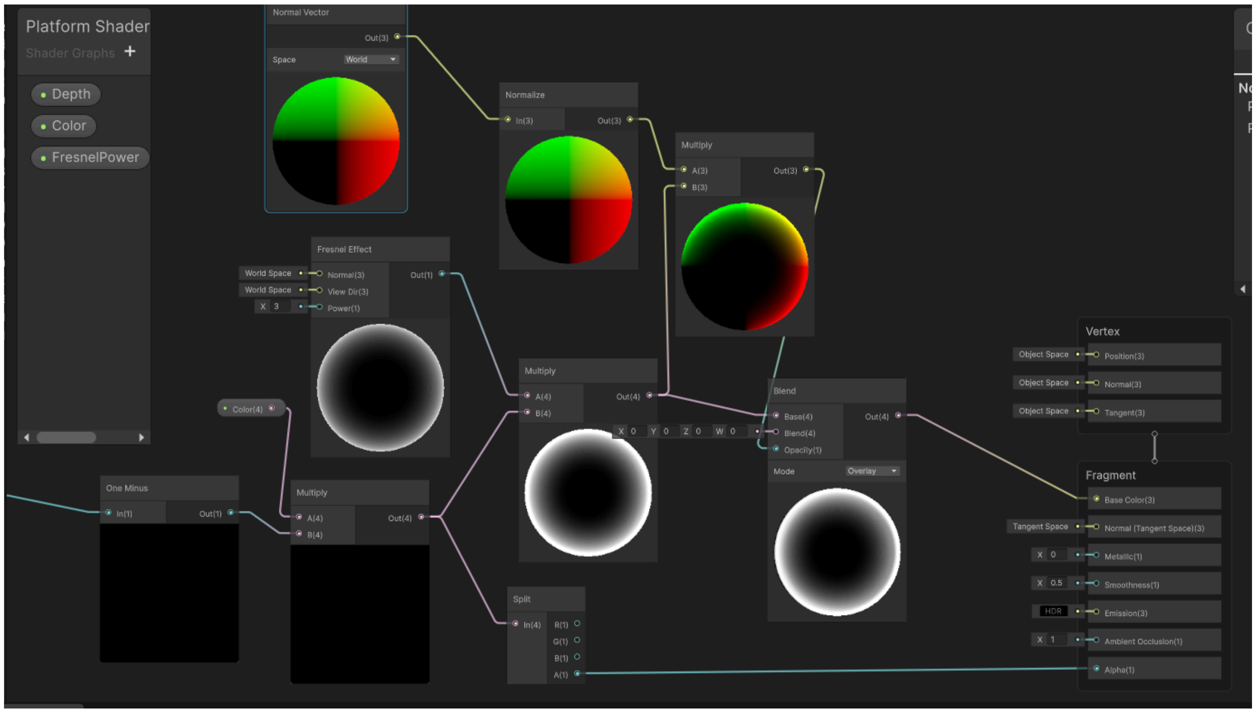
For the platforms, the shader used here is a modified intersection shader with blended Fresnel effect and normal vector added to create a soft gradient light for the tops of the platforms.




(This shader was not seen in class, so we based it off of the one from this video (https://www.youtube.com/watch?v=Uyw5yBFEoXo) and made some adjustments to make it unique.)

The shader utilizes an intersection between two objects to apply an effect, in this case a soft light. It's done by taking each object's respective positions and depths and then comparing them against each other to detect where to apply the shader effect.



In this case, the second object is transparent to highlight the original object. The base intersection shader is multiplied against the Fresnel effect to achieve a softer light, and then that in turn is multiplied by the normalized normal vector to achieve a more unique gradient effect.



(For transparency's sake, the code in the script ThirdPersonCamera.cs is from this video: https://www.youtube.com/watch?v=o7O28SFGWS4. We made the decision together to use a guide for a Cinemachine third person camera as we were both unfamiliar with how Cinemachine works and wanted to try and use it for this project in order to try and get the third person camera movement we wanted. As the camera movement needs improvement, this will only be a placeholder for this project progression update and will be replaced for the final build.)