Advanced Technologies: Water Simulation

Alex Feetham
17016942

University of the West of England

June 2021

This report details the process I went through to implement a simulation of water in a video game setting using shaders in Unity 3D.

1 Introduction

This task required me to research and develop a water simulation for use in games. To accomplish this, I selected to use shaders from the High-Definition Render Pipeline (HDRP) in the Unity game engine.

2 Related Work

2.1 Early Water in Games

The effectiveness of water rendering in early 2-dimensional games was greatly varied and was hard to get it to look accurate however for most the techniques used were the same. “Hardware registers would be manipulated during the frame refresh to change colours, effectively cutting the screen in two – above water and below water” (Linneman, 2012). Vice Project Doom used a clever trick to utilise mid-frame register writes to simulate the reflection of the light from the cityscape background (Figure 1). As hardware technology rapidly advanced, games were quickly able to start adding transparency. A great example of this was Donkey Kong Country which used a scrolling background to increase the depth in conjunction with a scanline effect to create the illusion of ripples (Figure 2). By choosing quite a saturated and vibrant colour palette it sells the illusion of being underwater.

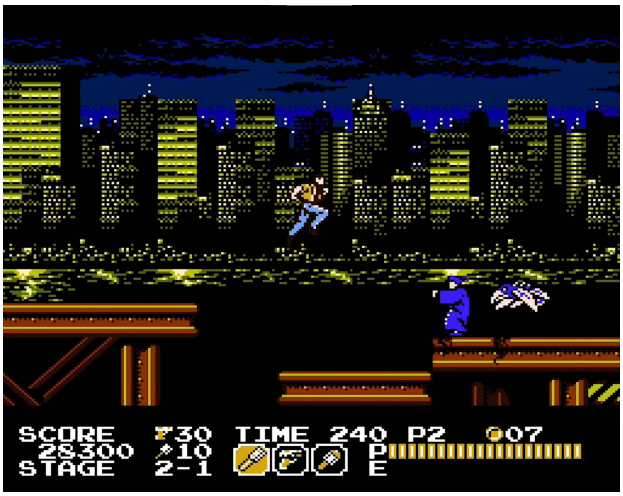

Figure 1: *Vice Project Doom*

Figure 2: *Donkey Kong Country*

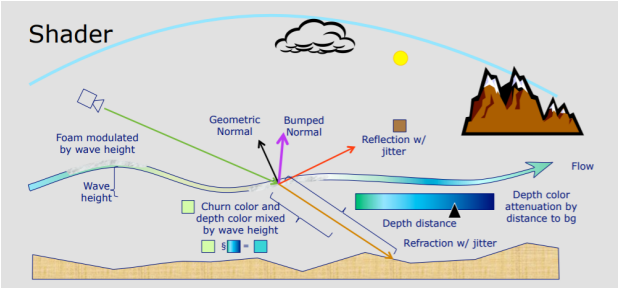

Figure 3: *Diagram illustrating how the shader works*

Figure 4: *Reflection layer without a Normal Map*

2.2 Modern Water in Games

(Gonzalez-Ochoa, 2012) explains how modern shaders are used to create a hyper realistic water in Uncharted 3. The shader is applied to a mesh and the elements of the shader are built up in layers. The first two layers handle the reflection and refraction of the water and are blended together using a Fresnel coefficient. A normal map is used to add height to the water and give it the appearance of a wave. Normal maps carry more data as they use RGB information which corresponds directly to the X, Y, and Z axis in the 3D space. In contrast, a bump map only uses greyscale meaning a lesser level of detail (*Difference Between Displacement, Bump and Normal Maps*). The normal map is scrolled across the surface of the mesh to simulate the movement of the wave. The colour of the water is affected by the depth, the deeper the water the darker the colour gets. Soft shadows can be used to simulate more murky water, foam can be added to the crests of the waves to make it appear more lifelike, and specular lighting can be applied to highlight reflections. Figures 3-8 showcase the process of adding the layers to the shader.

2.3 How can this be achieved?

Unity game engine can be used to create shaders in a similar fashion by using its High-Definition Render Pipeline (HDRP). “The HDRP is a high-fidelity Scriptable Render Pipeline built by Unity to target modern (Compute Shader compatible) platforms” (Unity Technologies, 2020). The HDRP is an additional package

Figure 5: *Reflection layer with a Normal Map*

Figure 6: *Reflection and Refraction layers with a Normal Map*

Figure 7: *Reflection and Refraction with Depth and a Normal Map*

Figure 8: *Final compiled shader*

which must be added to a project. The HDRP facilitates the creation of advanced/complex shaders and materials by utilising its shader graph editor.

3 Method

To create the water shader, a Physically-Based Rendering shader graph (PBR Graph) will be used. A material is then created from the graph and applied to the water plane.

3.1 Normal Maps and UV Panners

UV panners are used to add horizontal movement to the normal map, which is giving the height, thus creating the appearance of a wave. UVs are the 2D coordinates used to map a texture to a 3D model, the letters U and V represent the axis and were chosen as X, Y and Z represent an objects location in 3D space. Figure 9 shows how the effect of movement is created. Both halves of the shader graph have the same function however, the Vector2 properties at the start are different. The top has a Y value of 1 whereas the bottom has a value of -1. This allows me to move two normal maps in opposing directions to give a more realistic feel to the waves as they would otherwise both move in the same direction. Additionally, the bottom is offset slightly by 0.2. This is because as the two normal maps move towards each other, they will briefly overlap perfectly which makes it appear as though the simulation has frozen, and the waves stopped. A normal map is loaded in as a 2D texture and the outputs of the panners are used as the UV input to the texture nodes (Figure 10). The outputs of the nodes are blended and the result is used as the normal input on the PBR Master Node.

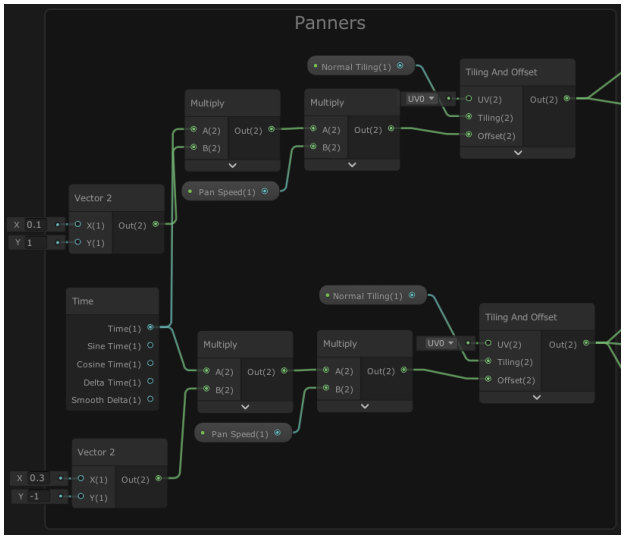

Figure 9: *Panners Section of the PBR Shader Graph*

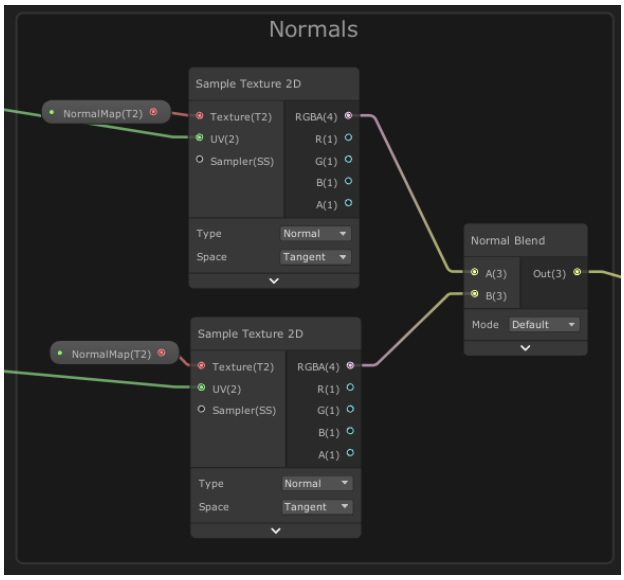

Figure 10: *Normals Section of the PBR Shader Graph*

3.2 Depth

Most of the visible light spectrum is absorbed within 10 meters (33 feet) of the water's surface, and almost none penetrates below 150 meters (490 feet) of water depth, even when the water is very clear (Encyclopedia, 2010). To create this effect within a shader, two colours need to be blended using a lerp node with a depth calculation used as the lerp's mask. The depth calculation can be seen in Figure 11, the scene depth is multiplied by the far plane from the active camera in the scene. The alpha channel from the screen position is multiplied by a Vector1 (Depth Distance = 0.65), the product of which is subtracted from the first multiply node. The result of the subtraction is multiplied by a Vector1 (Depth Strength = 0.25) and the result of which is passed through a Saturate node before being passed into the mask input of the lerp node.

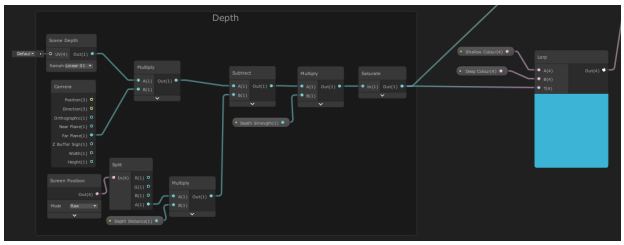

Figure 11: *Depth Section of the PBR Shader Graph*

3.3 Refraction

"When light travels from air into water, it slows down, causing it to change direction slightly. This change of direction is called refraction" (Figure 12) (Science Learning Hub – Pokapū Akoranga Pūtaiao, 2012).

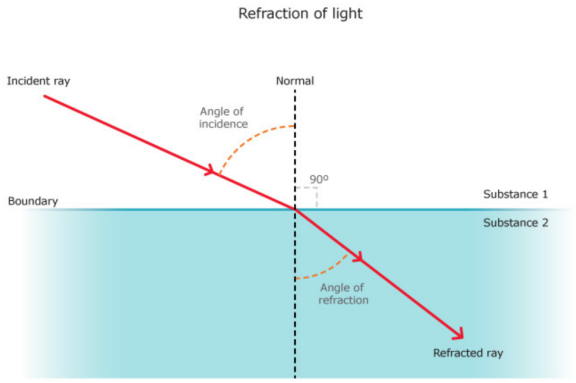

Figure 12: *Diagram Illustrating Refraction*

Refraction can be simulated in a shader graph by adding the red and green channels from the screen position to the product of the red and green channels from the normal map and a Vector1 (Refract Strength = 0.15). (Figure 13). The output of the refraction is passed into the B slot of a lerp node; the output from the lerp between the colours and depth is passed into the A slot, and the depth output is used as the mask. The output of this lerp is finally passed into the Albedo of the PBR Master node, via a Saturate node.

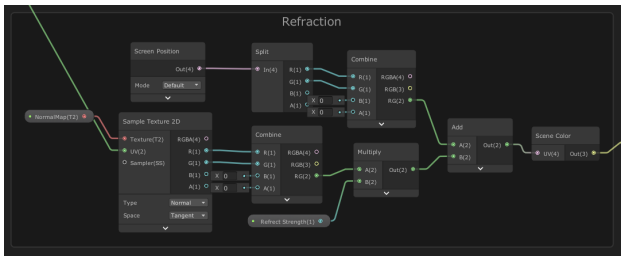

Figure 13: *Refraction Section of the PBR Shader Graph*

3.4 Displacement

To create an accurate displacement, it needs to be matched to the pattern of the normal map. To accomplish this, a matching heightmap needs to be used to alter the vertices of the water plane along the Y axis. The red and blue channels of the object's position are linked to the X and Z axis of a Vector3 to lock them to stay uniform with the horizontal bounds of the water plane. As with the normal map, the height maps need to be scrolled across the plane at the same rate, so the same UV Panner is linked into a Texture 2D node. The red channel of the displacement map for each direction are added and multiplied by a Vector1 (Displacement Strength = 0.75) and is passed into the Y axis of the Vector3 (Figure 14). This Vector3 is then linked to the Vertex Position input on the PBR Master.

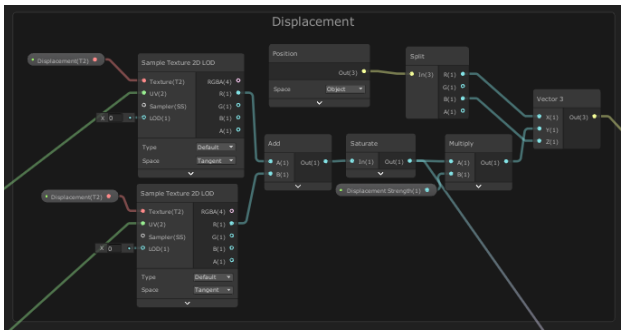

Figure 14: *Displacement Section of the PBR Shader Graph*

4 Evaluation

A key issue discovered during development was the vertex count of the water plane. A standard plane in Unity has 121 vertices. If the plane is kept at its default scale, the shader created applies appropriately and looks as intended. However, the shader is for water and thus if used in a game would cover a large area. Figure 15 shows two planes overlapped in a wireframe view to make pinpointing vertices easier; the white lines mark a standard Unity plane, scaled to 7 along the X and Z axis, and the black lines are a custom plane made in Maya 2020. The custom plane is able to be displaced to a much greater level of detail than the standard plane due to it having 197% more vertices with 16,641 in total.

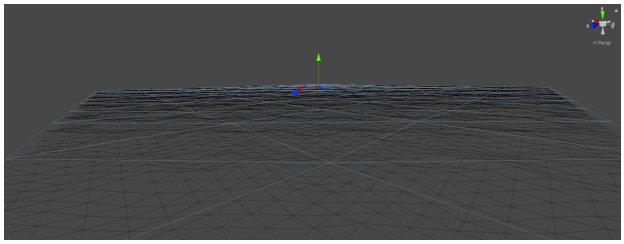

Figure 15: *Wireframe View of two Planes in Unity*

I was unable in integrate the effect of foam on the final shader presented however during an early iteration I did manage to get it to work, see Figure 16. The shader used for this water simulation was an Unlit Shader, whereas the final shader is a PBR Shader. The methods implemented in the Unlit Shader were not compatible with the PBR Shader and I was unable to find a suitable solution that could present itself in the same fashion.

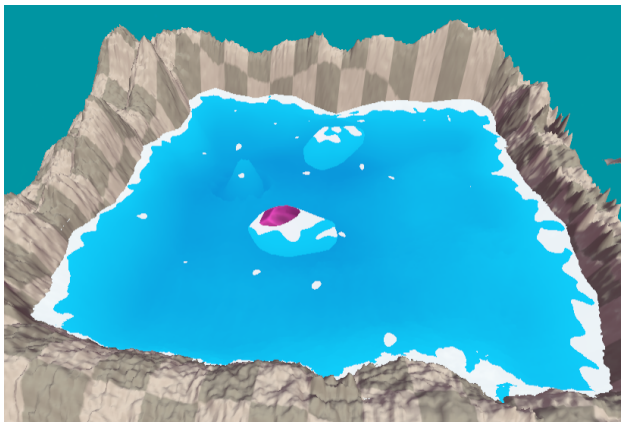

Figure 16: *Foam effect using an Unlit Shader Graph*

5 Conclusion

Figure 17 shows the final shader in a game scene. It has met the requirements of the task to create a water simulation. The shader is capable of reflection and refraction as ween by the distortions of the bridge in the water. It can simulate waves by scrolling a normal map and heightmap which manipulates the vertices of the plane.

Figure 17: *Final Shader*

Bibliography

Encyclopedia, Water (2010). *Light Transmission in the Ocean*. Accessed: 07/05/20A21. Science and Issues. URL: http://www.waterencyclopedia.com/La-Mi/Light-Transmission-in-the-Ocean.html.

Gonzalez-Ochoa, Carlos (2012). "Water Technology of Uncharted". In: Game Developers Conference, pp. 14–25. URL: https://www.gdcvault.com/play/1015517/Water-Technology-of.

Linneman, John (2012). *DF Retro: The history of water rendering in classic games*. Accessed: 07/05/2021. Eurogamer. URL: https://www.eurogamer.net/articles/digitalfoundry-2018-retro-the-history-of-water-rendering-in-classic-games.

Pluralsight. *Difference Between Displacement, Bump and Normal Maps*. Accessed: 07/05/2021. URL: https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps.

Science Learning Hub – Pokapū Akoranga Pūtaiao, University of Waikato (2012). *Refraction of Light*. Accessed: 07/05/2021. University of Waikato. URL: https://www.sciencelearn.org.nz/resources/49-refraction-of-light.

Unity Technologies, (2020). *High Definition Render Pipeline overview*. Accessed: 07/05/2021. Unity Documentation. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.1/manual/index.html.

Appendix A

Video Links:

1. Week 1
2. Week 2
3. Week 3
4. Week 4
5. Week 5