Water Shader Techniques

How to Create a Basic Water Shader - Project Documentation

Contents

Section 0 – Project Setup 2
Section 1 – Refraction 3
Step 1.1 – Adding Scene Colour 3
Step 1.2 – Adding Refraction 4
Step 1.3 – Adjusting Refraction Strength 6
Step 1.4 – Refraction Movement 7
Step 1.5 – Layered Refraction Normal Maps 9
Section 2 – Depth Mapping 10
Step 2.1 – Creating a Depth Mask 10
Step 2.2 – Re-Adding Refraction 12
Step 2.3 – Adding Color 13
Step 2.4 – Water Opacity 14
Section 3 – Normal Maps 15
Step 3.1 – Adding Reflections 15
Step 3.2 – Adjusting Reflection Strengths 16
Step 3.2 – Weighting Reflection Based on Depth 17
Step 3.3 – Full Shader Graph Reference 18
Section 4 – What's Next? 19
Next Steps 19
Extra Resources 19

Section 0 – Project Setup

Before we begin, you'll need to create a new Unity project. Any modern version will work, but make sure you **select the 3D (URP) template.**

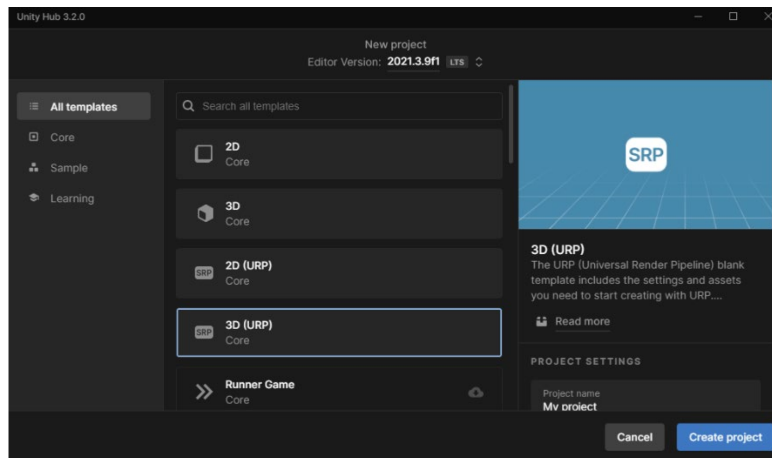


Figure 1 Unity Project Selection Screen. 3D (URP) is selected.

Once your project is created, right click in the "Assets" panel, and **go to Create > Shader Graph > URP > Lit Shader Graph**. Name it whatever you'd like, then double click to open it in Unity's Shader Graph editor.

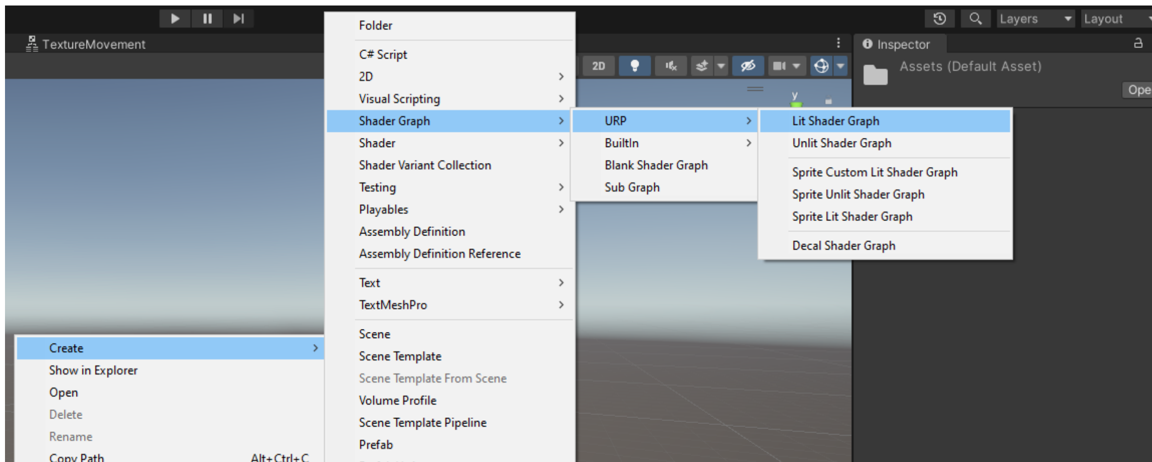
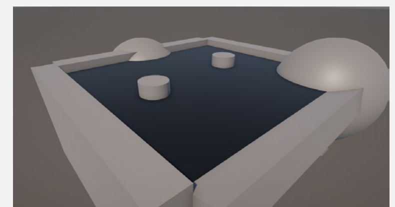


Figure 2 Menu path showing where to access Unity's "URP Lit Shader Graph".

Finally, right click on your shader in the "Assets" tab and click **Create > Material.** This will create a material that uses your newly created shader.

Pro Tip:

Apply your material to a test environment so you can check your progress as you go!



Section 1 – Refraction

Step 1.1 – Adding Scene Colour

Once your shader is open in Shader Graph, you'll have to change a few settings. In the "Graph Inspector" panel on the right, **set the "Surface Type" to Transparent**, and **Disable "Cast Shadows" as well as "Receive Shadows."**

Next, you'll need to create a couple of nodes by **right clicking and searching the node name.** Create a "Screen Position" node, as well as a "Scene Color" node. Connect the first node to the second, and connect the "Scene Color" node to the shader's "Base Color"

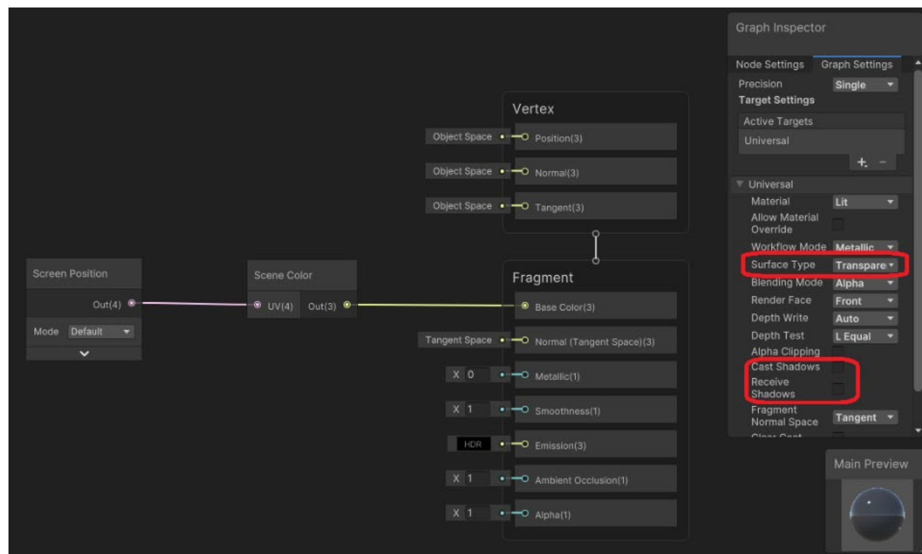


Figure 3 Basic Shader Graph that projects the scene colour onto the texture.

This will take the player's "Screen Position", grab the "Scene Colors" at that location, and then apply it to the material's texture. Basically, it makes the material appear transparent, as it's projecting whatever's behind it.

If you did everything correctly so far, your shader should look like the picture below:

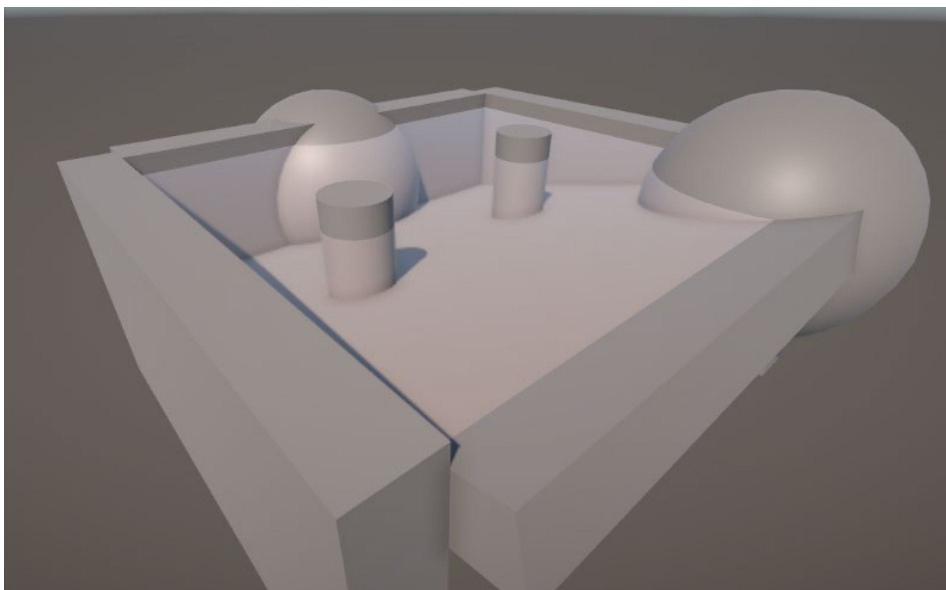


Figure 4 The shader projecting the scene colour behind it.

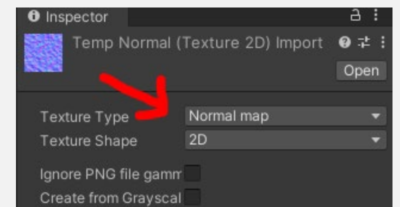
Step 1.2 – Adding Refraction

This is a good first step, but water isn't perfectly clear. It warps and bends light, making things underwater look distorted. This is called **refraction.**

We'll be using a normal map to simulate refraction. To do this, **add a "Texture2D"** variable to your Shader variables list. **Name it "Refraction Normals**", and make sure its **mode is set to "Normal Map".**

You'll need to connect a normal map to your "Refraction Normals" variable for refraction to work, either in the variable's "Default" slot, or in your created material. Any will do, however I'll be using this one found on catlikecoding.com.

Pro Tip:
When importing your normal map, make sure the "Texture Type" is set to "Normal Map"



Next, **add a "Sample Texture 2D" node**. **Plug your "Refraction Normals"** variable into its Texture(T2) slot, and make sure the **Type is set to "Normal."**

Then, **create an "Add" node**, and **add the normal map's RGBA values to the screen position**. This will add the offset of the normal map to the screen position's values, creating our desired distortion.

So far, your shader graph should look like this:

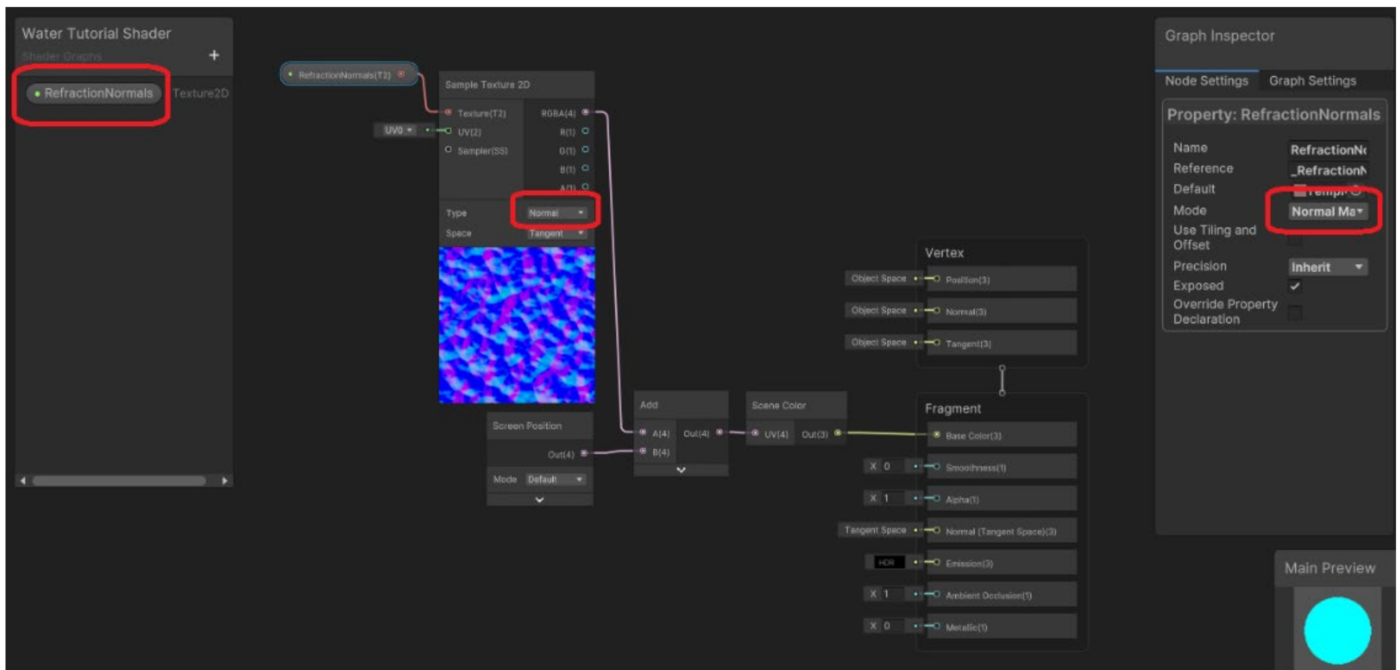


Figure 5 A normal map added to the shader graph in order to create refraction.

…and if everything is added correctly, your shader should look like this:

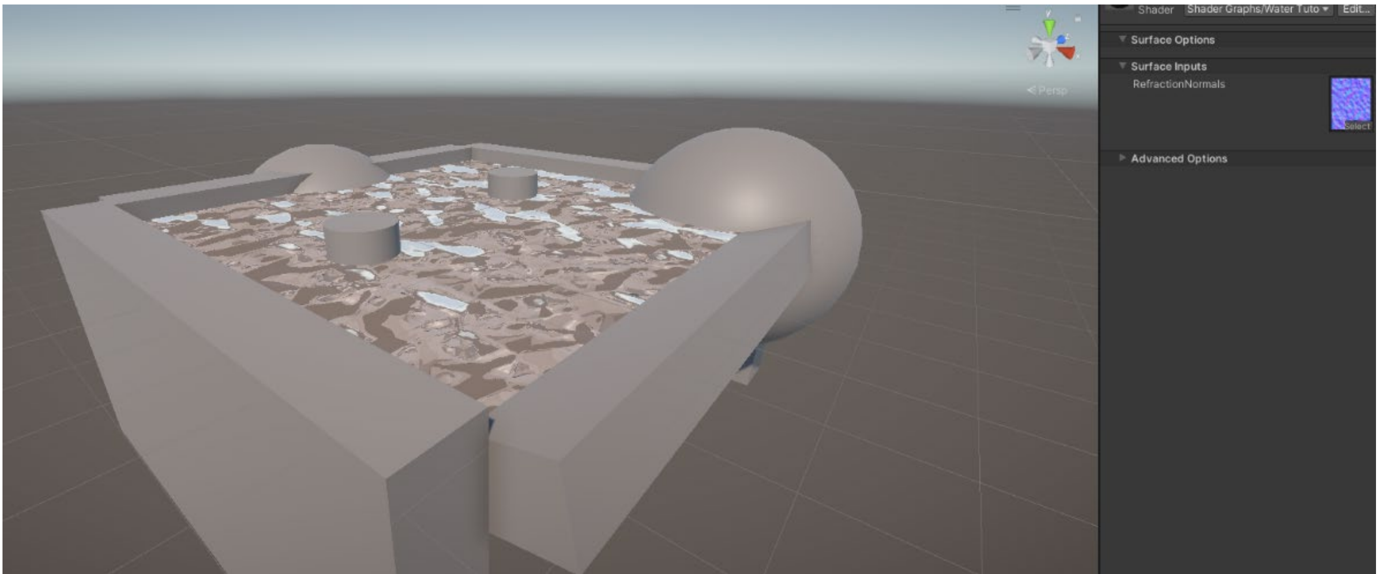


Figure 6 The shader with refraction now added. Without a control variable, the refraction is too strong.

Something's certainly changed! Our water now has refraction, but it's way too strong. To be able to adjust this, we'll need to add another variable

Step 1.3 – Adjusting Refraction Strength

To adjust the refraction strength, first add a **"float"** to your Shader variables list. Name it **"Refraction Strength"** and set its default value to 0.01.

Pro Tip:
For ease of use, you can also make variables display as a "Slider" in the Graph Inspector, with a minimum and maximum value. Try it out with the Refraction Strength!

Next, create a multiply node, and **connect the RGBA output of your "Sample Texture 2D" node,** as well as your **"Refraction Strength" variable.** Connect this to the "Add" input where your "Sample Texture 2D" used to go. This will multiply your normal map's strength by your variable, reducing it's effect.

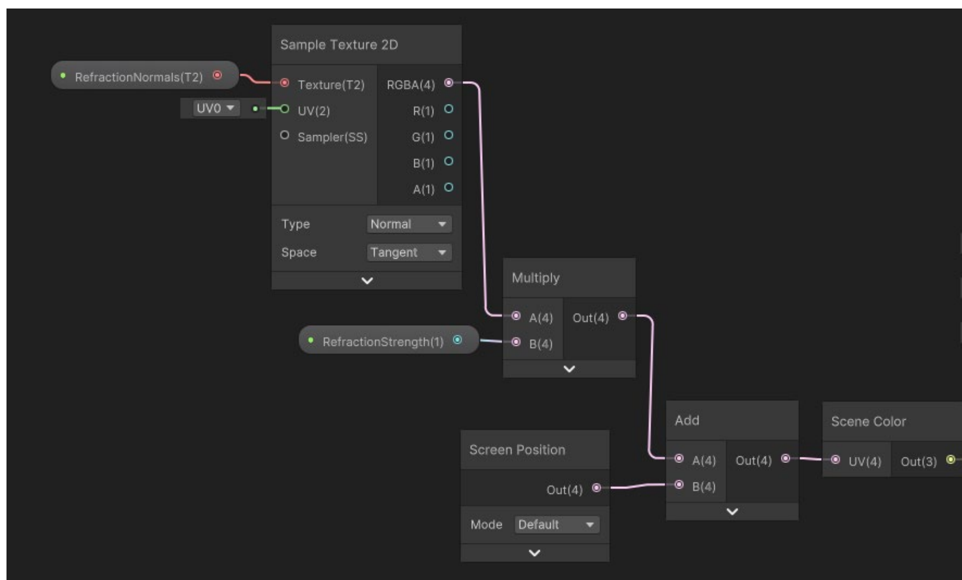


Figure 7 The shader graph with a "Refraction Strength" variable added.

Once you've done that, your shader should now look like this:

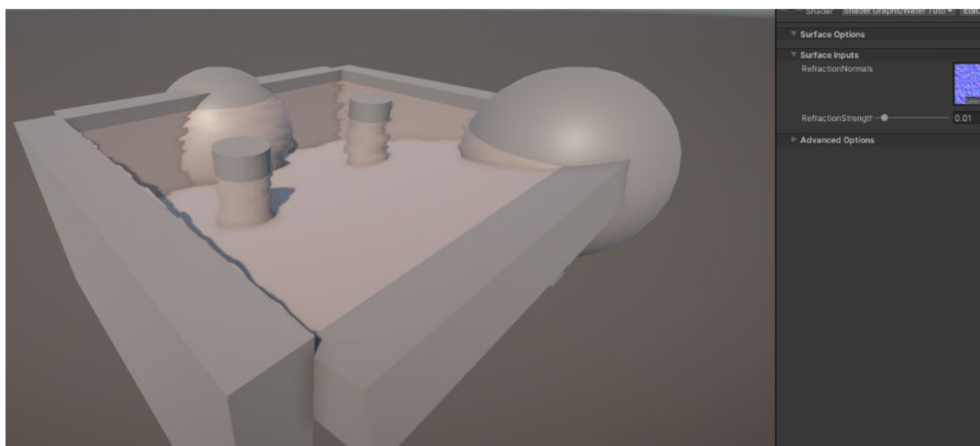


Figure 8 The shader with controllable refraction added.

That's better, but there's still a problem. You can't see it in this picture, but the refraction doesn't move. This makes it look more like ice than water, so we'll be adding some movement in the next step.

Step 1.4 – Refraction Movement

To add refraction movement, first **add two more "floats" to your Shader Graph's variable list.** Name one **"Refraction Speed"** and set its **default value to 0.1**. Name the other **"Refraction Strength"** and set its **default value to 1.**

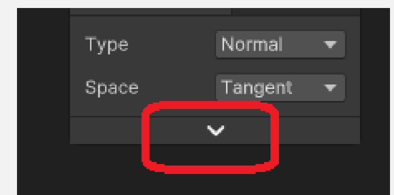
Next, create a **"Tiling and Offset"** node and plug it into the **"UV" input** of your **"Sample Texture 2D"** node.

Then, plug your **"Refraction Scale"** node into the **"Tiling(2)"** input of your **"Tiling and Offset" node.** Doing this will scale your normal map to the value of your "Refraction Scale" variable (ex. 0.5 = half as big, 2 = twice as big)

Finally, **create a "Time" and "Multiply" node.** Multiply the "Time(1)" output of your Time node with your "Refraction Speed", and **plug the output into the "Offset(2)"** of your "Tiling and Offset" node.

This will move the x and y value of your normal map by "Refraction Speed" every second. (ex. if the speed is set to 1, the image will fully repeat every second)

Pro Tip:
To preview the movement/scale of your normal maps, click the drop-down arrow on the "Sample Texture 2D Node"



If you've done everything correctly so far, your shader graph should look like this:

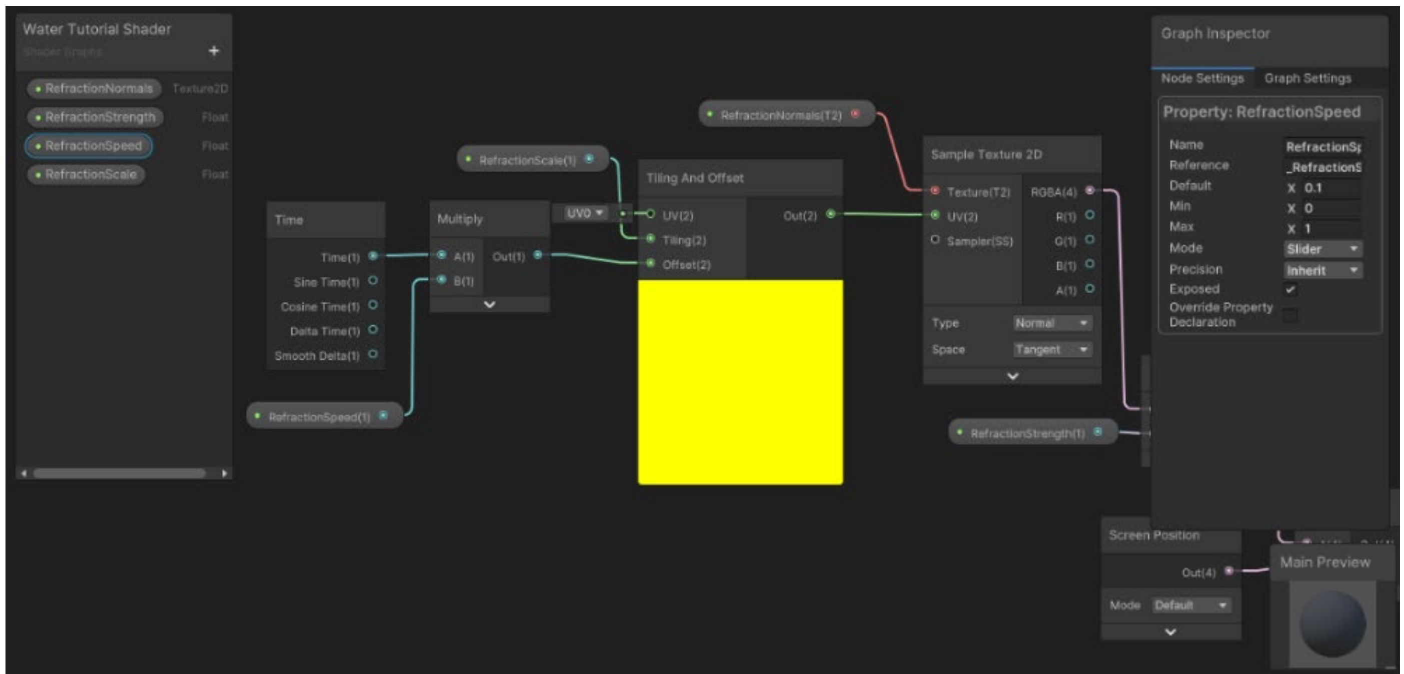


Figure 9 The shader graph with controls for animated refraction added.

…and your shader should look like this:

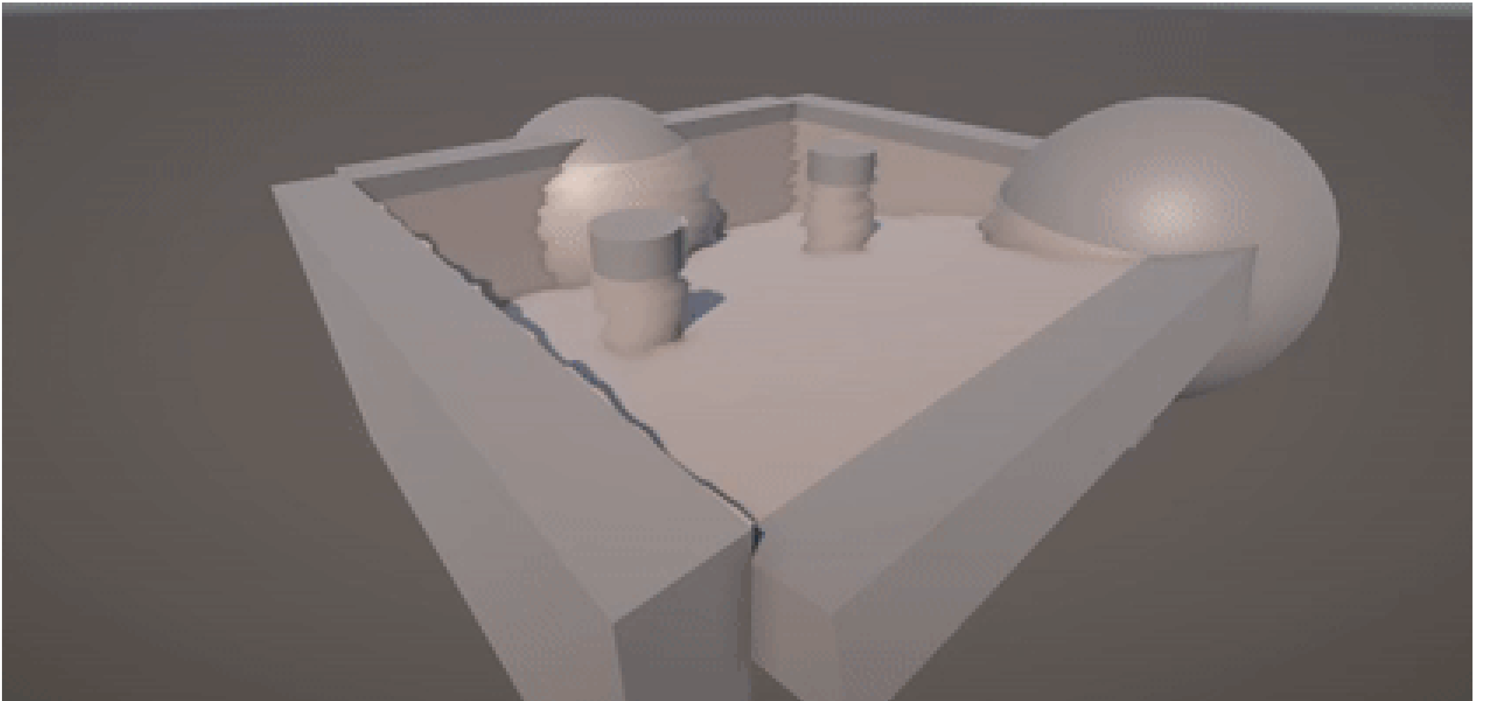


Figure 10 The shader with animated refraction implemented.

Hooray, we have movement! We’re almost done with refraction, but the direction of refraction movement is quite noticeable. To fix this, we’re going to add one more effect: **layered normal maps.**

Step 1.5 – Layered Refraction Normal Maps

Layering our refraction normal maps is quite simple. First, **duplicate your "Sample Texture 2D" node**, as well as all other nodes connected to its input. Move these duplicates below, so they don't get mixed up.

Next, **create another multiply node**, and connect it to the bottom copy of your "Refraction Speed" variable. **Multiply the "Refraction Speed" by -1**, and then connect the output to the multiply node where the "Refraction Speed" used to be connected.

You should see that the "Sample texture 2D" node's preview is now moving in the opposite direction. What we've just done is reversed the direction of the second normal map by multiplying it's speed by -1.

Finally, create a **"Normal Blend"** node and **connect the RGBA output of both "Sample Texture 2D" nodes** to its inputs. Then, connect the output to the "Multiply" node where the original "Sample Texture 2D" used to connect.

What we've now done is mixed the two normal maps together so that the direction of their movement becomes much less visible. If you've done everything correctly, your shader graph should look like this:

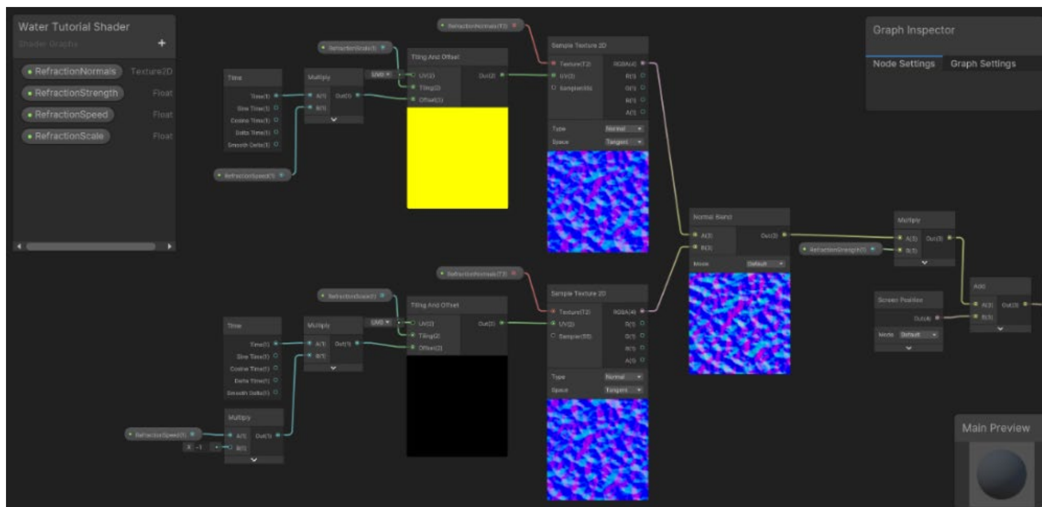


Figure 11 The shader graph with an additional normal map layer added.

…and your shader should look like this:

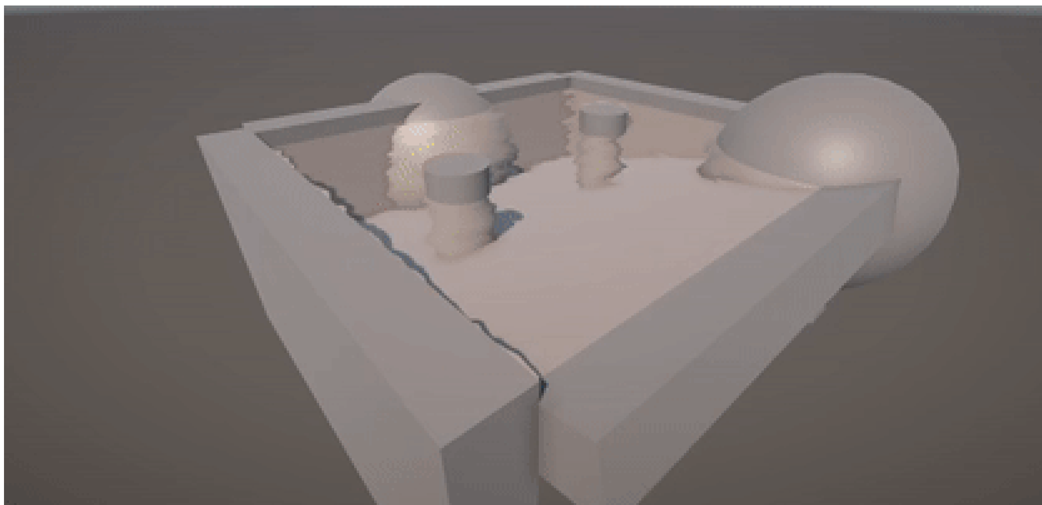


Figure 12 The shader with an additional normal map added for more complex movement.

Congrats, your refraction is now fully operational! Next, it's time to add some color to our water.

Section 2 – Depth Mapping

Step 2.1 – Creating a Depth Mask

To add color, we'll first need to get the depth of the scene. This way we can make the water change color the deeper it gets. We'll do this by creating a **depth mask.**

To do this, first **create two more floats**, "Depth Start" and "Depth End". Set the default value of "Depth Start" to 0, and the default value of "Depth End" to 5.

Next, create a **"Scene Depth"** node and a **"Screen Position"** node. Set the Scene Depth's Sampling to "Eye", and the Screen Position's mode to "Raw".

Then, create a **"Split"** node and a **"Subtract"** node. Connect the output of the Scene Depth node to the A input of the Subtract Node and connect the output of the Screen Position node to the input of the Split node.

Connect the A(1) output of the Split node to the B input of the subtract node. So far, your shader graph should look like this:

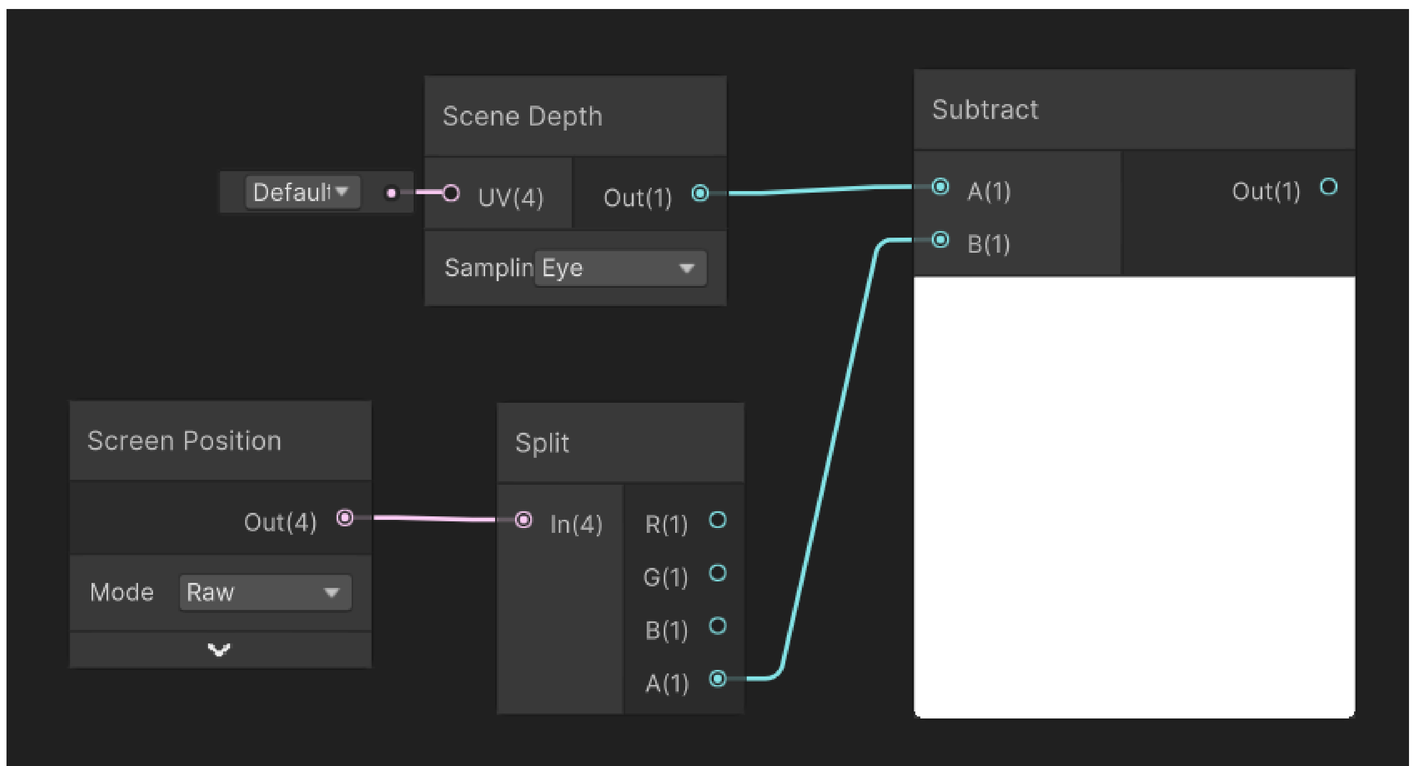


Figure 13 The first step of the depth fade shader graph.

After that, **create an "Add" node**, and add your "Depth Start" variable to the output of your subtract node. **Create a "Divide" node** and connect the output of your Add node to the A input and connect your "Depth End" variable to the B input.

Finally, **create a "Saturate"** node and plug the Divide node's output into its input. Then plug the Saturate node's output into your shader's Base colour. This will get rid of the refraction for now, but don't worry, we'll add it back soon.

If you've done all of that, your shader graph should look like this:

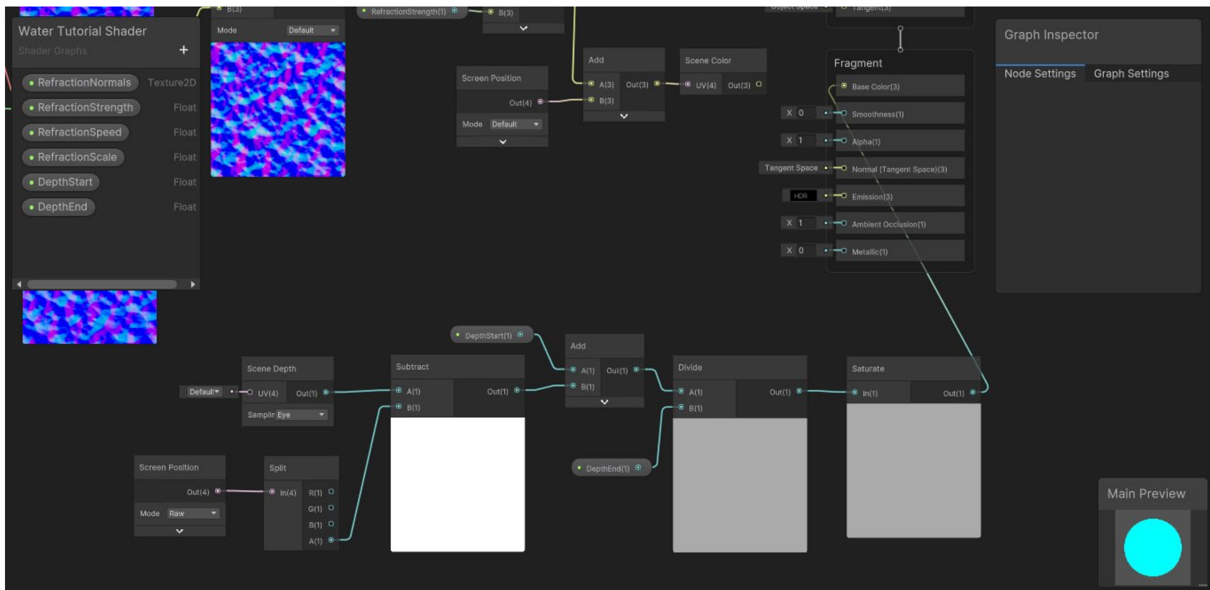


Figure 14 The remaining nodes of the depth fade shader graph.

That was a lot! But what does it do? Basically, **it grabs the depth** of everything you can see in the scene and **puts it on a gradient from black to white.** It then uses the Depth Start and Depth End to **squish that gradient into a range** closer to what we want.

This becomes more apparent if we look at the shader itself:

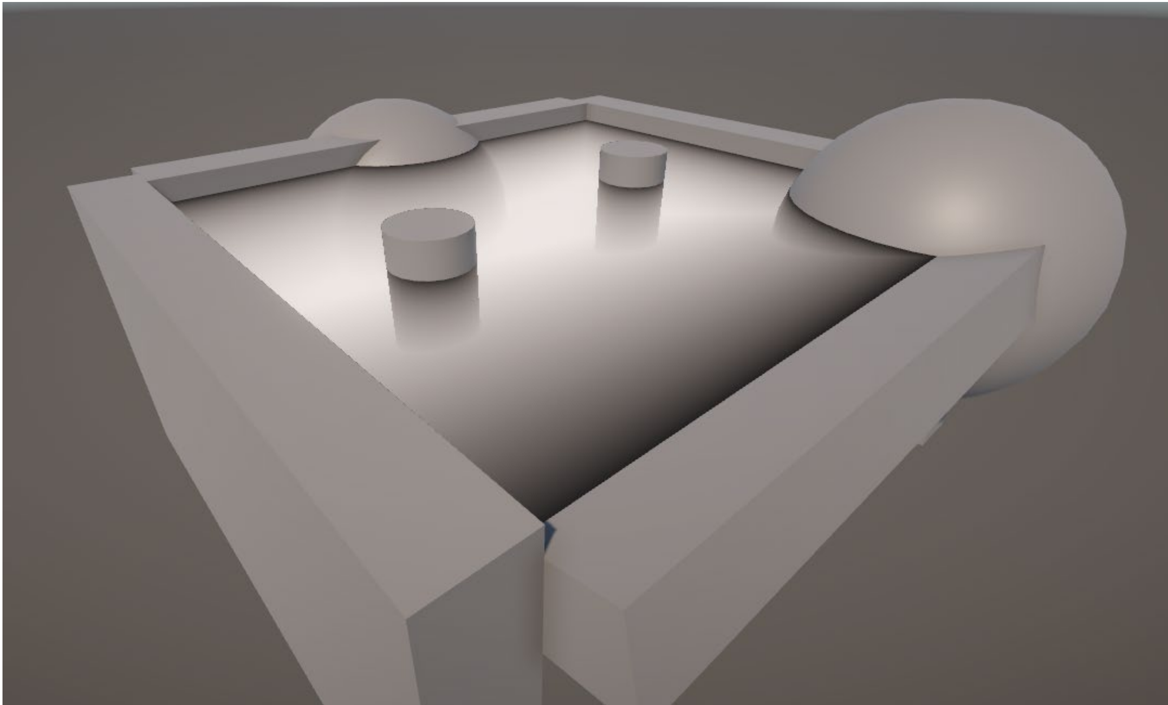


Figure 15 Depth fade implemented and displayed into the shader.

As you can see, things closer to the water's surface appear black, and things farther away appear white! We had to get rid of our refraction do this, so we'll be adding it back in the next step.

Step 2.2 – Re-Adding Refraction

To re-add refraction, simply **connect the output of the refraction's main "Add" node to the input of the Scene Depth node.**

This will feed in the distorted perspective that the refraction creates to the scene depth, overwriting the default view. Your shader graph should look like this:

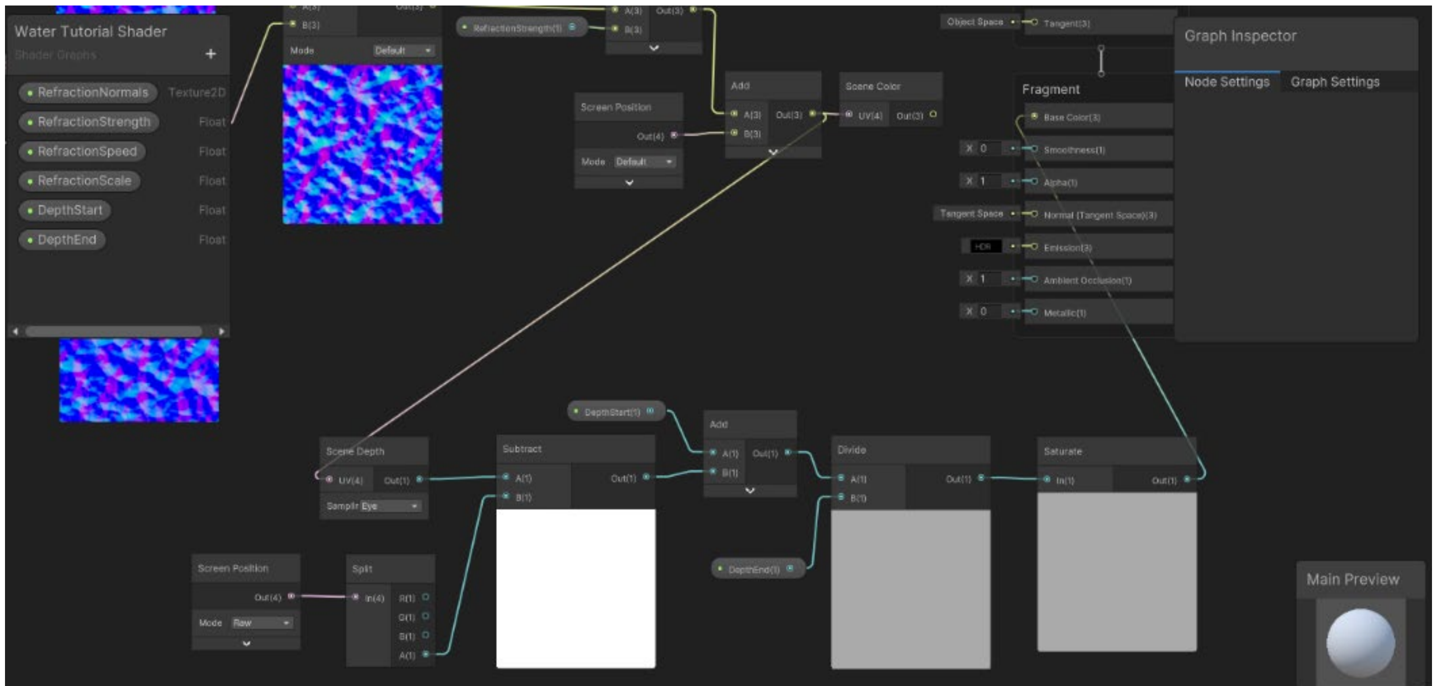


Figure 16 The shader graph with the depth fade and refraction effects combined.

…and your shader should look like this:

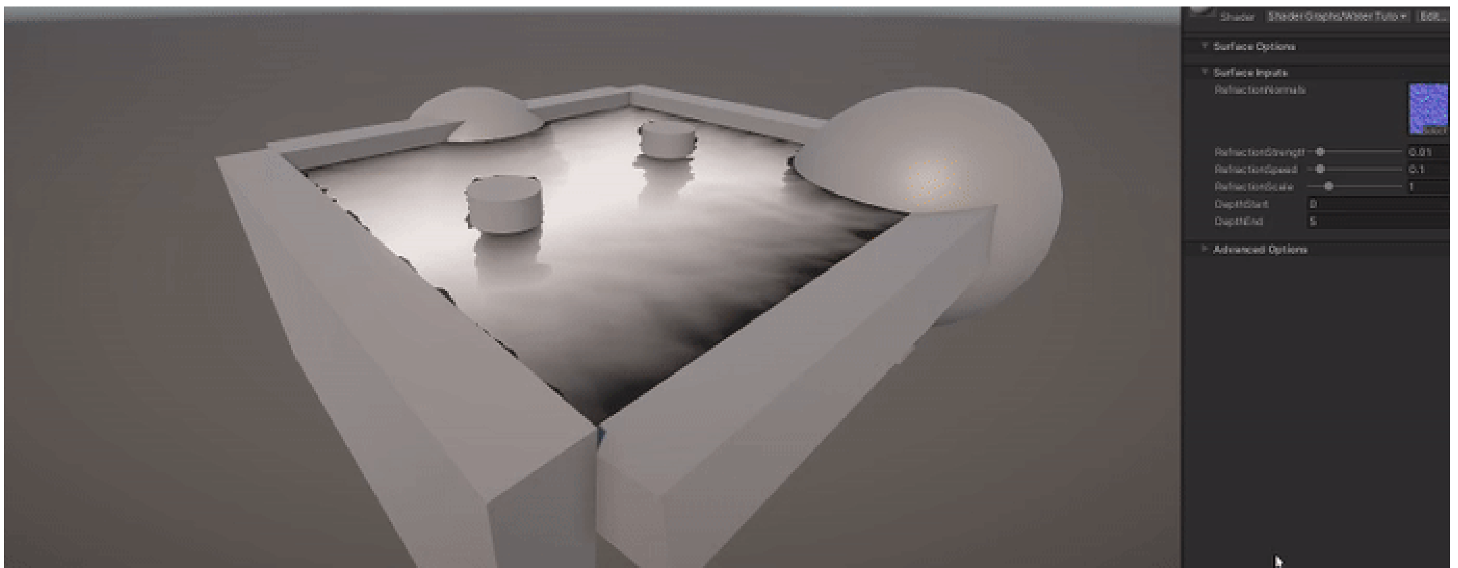


Figure 17 The shader displaying both depth fade and refraction.

And that's it! The scene depth and refraction have now been merged. Next, it's time to customize what color our water is.

Step 2.3 – Adding Color

To add color, you'll need two more variables. **Create two Color variables**, and name one **"Surface Color"** and the other **"Deep Color"**. You can set their default values to whatever you'd like, however in this example we'll be using two shades of blue.

Next, **create a "Lerp" node,** and **connect your two color variables**. Surface Color should connect to A, and Deep color should connect to B. Finally, connect the output of your "Saturate" node to T, and connect the Lerp nodes output to the shader's Base color. Your shader graph should now look like this:

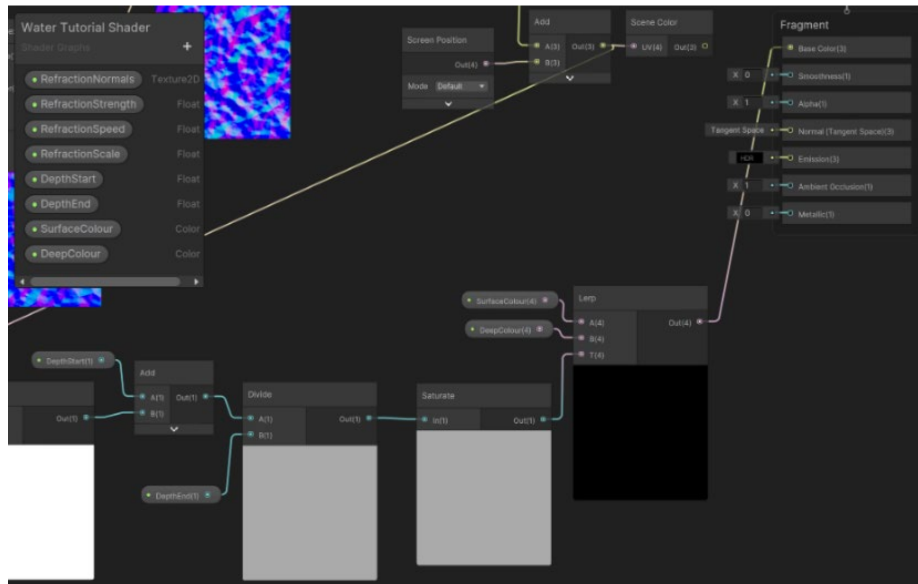


Figure 18 The shader graph with custom colours added.

…and your shader should look something like this:

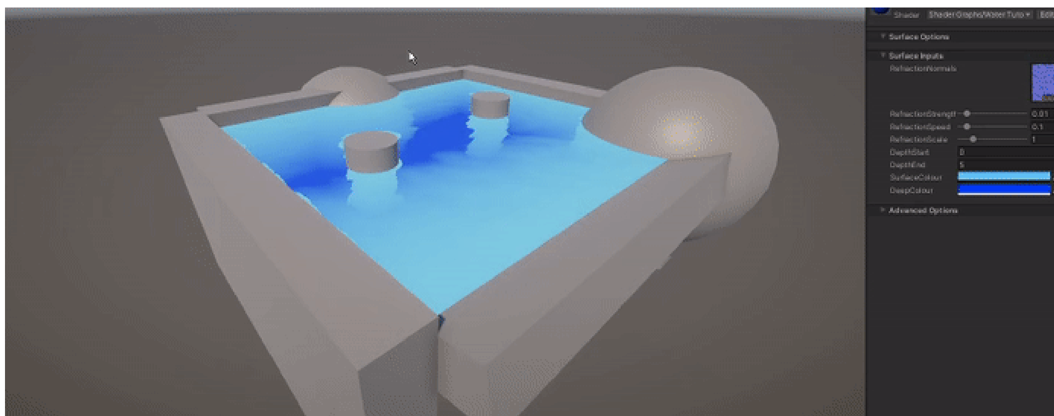
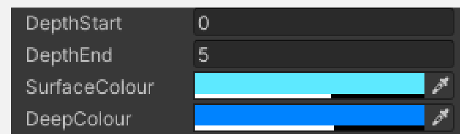


Figure 19 The shader with custom colours selected. Transparency is not yet supported.

Pro Tip:
If you don't like how the color gradient looks, play around with the "Depth Start" and "Depth End" values for a smoother look.



Although we have colour, you'll notice transparency isn't supported yet. We'll have to add that in the next step.

Step 2.4 – Water Opacity

To add transparency, we won't use the shader's "Alpha" value, but instead will be pulling a sneaky trick. **Remember when our water was fully transparent?** All we need to do to get transparency is **fade into that version** of the water depending on the color's alpha that the user picks.

First, **create a "Split" node** and connect the Lerp node we made last step to its input. This will let us **access the alpha value that the player selects** for their two colors.

Next**, create another Lerp node**, and connect the refraction's **"Scene Color"** node to its A input. Then, connect the output of the previous lerp node to the new lerp node's B input, and connect the A(1) output of our split node to the T input.

Finally, connect the new Lerp node's output to the shaders base colour.

This new lerp node will now **fade between the original refraction and the fully opaque refraction** based on the alpha values the player selects, creating the transparency effect we want! Your shader graph should now look like this:

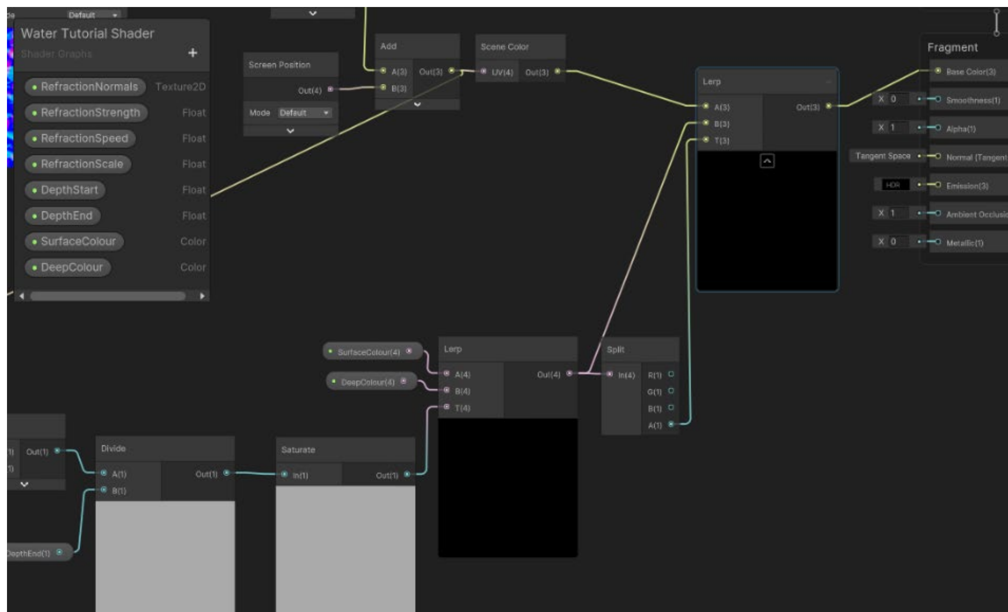


Figure 20 The shader graph with support for transparency added.

…and your shader should look something like this:

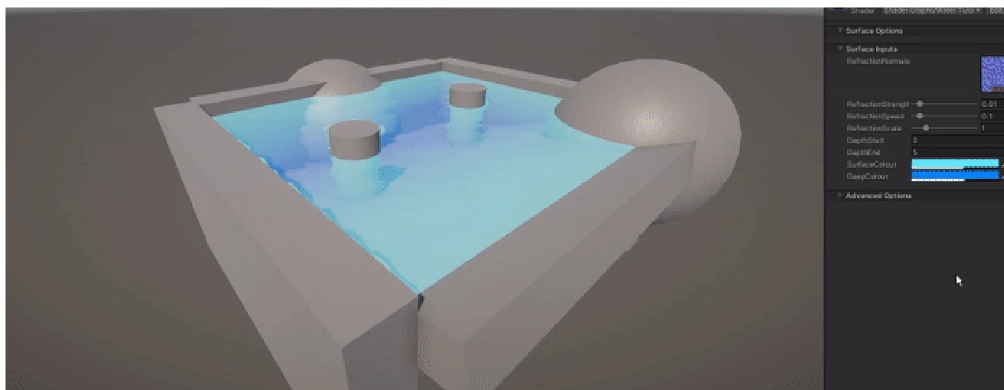


Figure 21 The shader with transparency added.

Section 3 – Normal Maps

Step 3.1 – Adding Reflections

Congrats, your water is now in full color! Next, **we'll be adding a normal map** to the water's surface to create a convincing shine.

The good news is that we already did a large part of the work when making the refraction, so all we need to do is add that same normal map to the water's surface.

First, make sure that your **shader's smoothness is set to 1**. This will make our water be extra shiny. Then, take the output of the "Normal Blend" node we used to create refraction before, and plug it directly into the shader's Normal slot. This will add the moving normal map to the shader's Normal property.

Figure 22 The shader graph with preliminary support for normal maps added.

Your shader should now look like this:

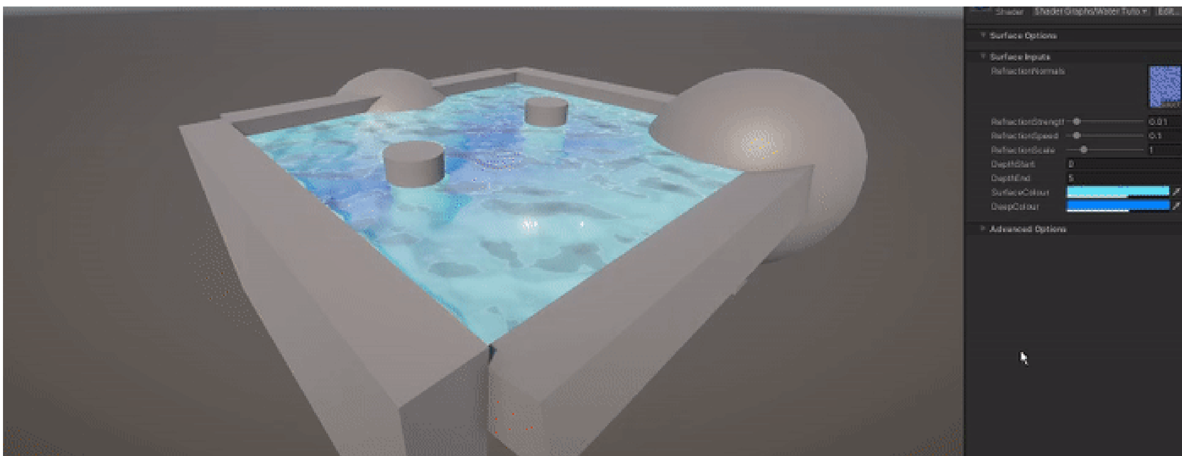


Figure 23 The shader with normal maps added. No way to control their strength has been added yet.

This is a step in the right direction, but the waves feel a bit strong. Next, we'll be adding a value to adjust the strength of the waves with.

Step 3.2 – Adjusting Reflection Strengths

To adjust the strength of the reflections, first **make a float called "Normal Strength",** and set its **default value to 0.5.**

Next, **create a "Normal Strength"** node and connect the "Normal Blend" node's output to the In(3) input. Then, connect the Normal Strength float value to the Normal Strength node's Strength(1) input.

Finally, connect the Normal Strength node's output to the shader's normal value. This will multiply the strength of your normal map by the Normal Strength float, lessening its effect. Your shader graph should now look like this:

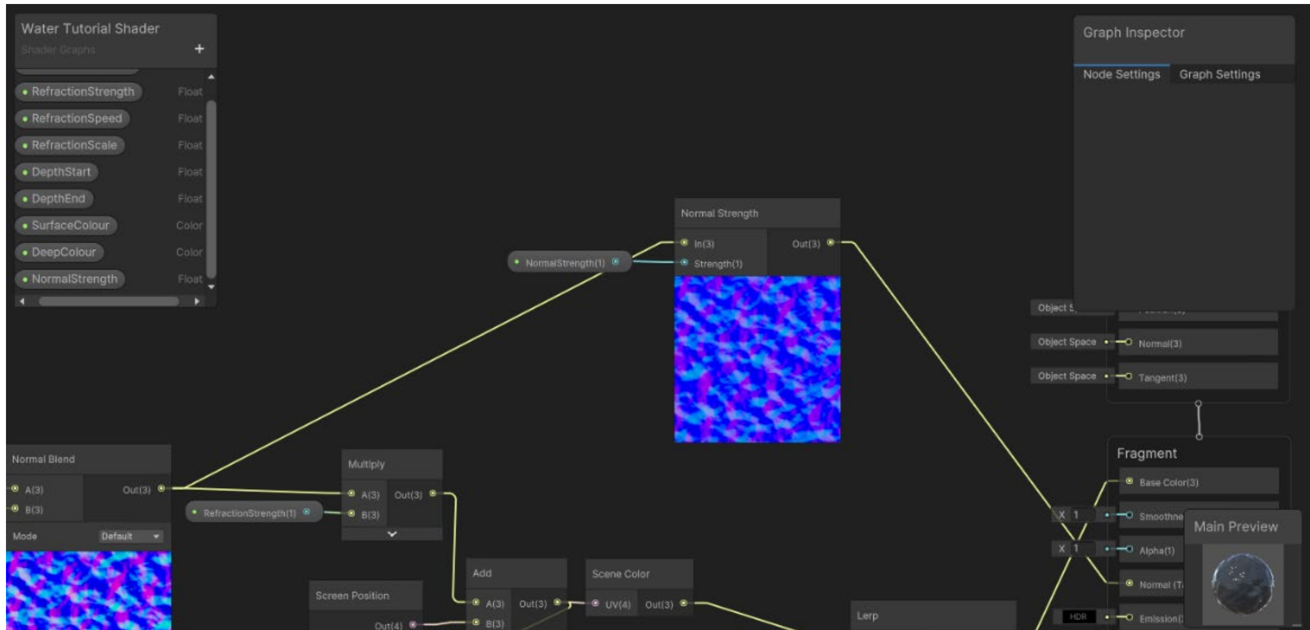


Figure 24 The shader graph with normal map strength control added.

…and your shader should look like this:

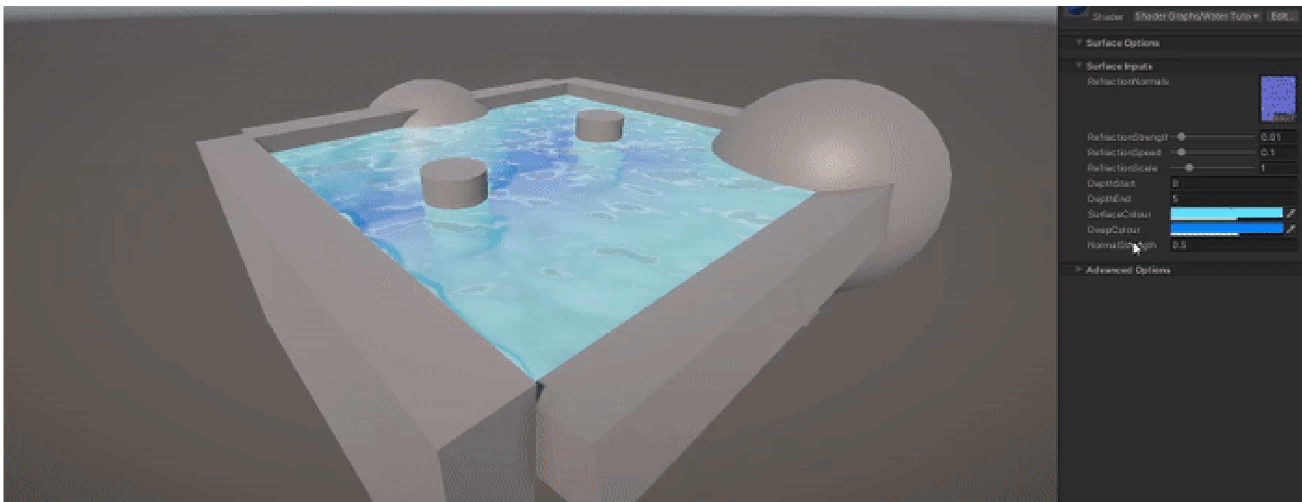


Figure 25 The shader with control options for normal map strength added.

Great! We're almost done, but there's one more effect we're going add. Right now, the waters shine is applied uniformly across the surface, but for a more convincing look we're going to make it appear more strongly in deep parts of the water.

Step 3.2 – Weighting Reflection Based on Depth

First**, create another "Lerp" node**, and set the A value to 0. Connect the Normal Strength float to the B value, and **connect the "Saturate" value of the depth map to T.** (You'll have to stretch across the graph for this!)

Then, **create a "Saturate" node** and connect the Normal Strength node's output to its input. Finally, **connect the Saturate node's output to the shader's Normal value.** This will apply the depth map onto the normal map like a mask, so only the deep parts are shown.

Your shader graph should look like this:

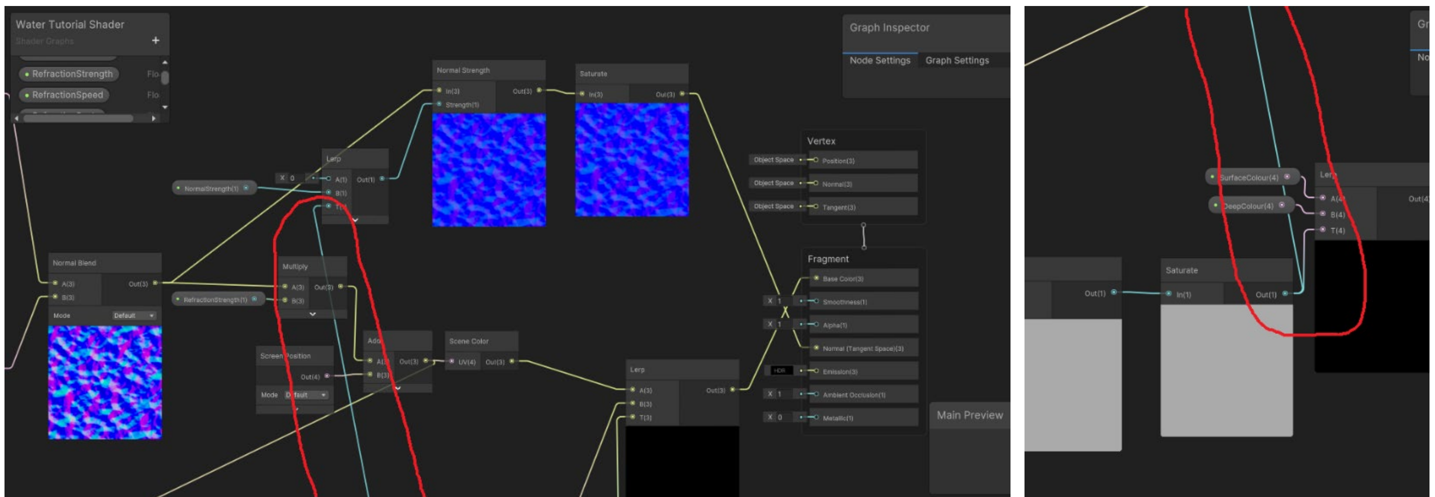


Figure 26 The shader graph with depth-based normal maps added.

…and your shader should look like this:

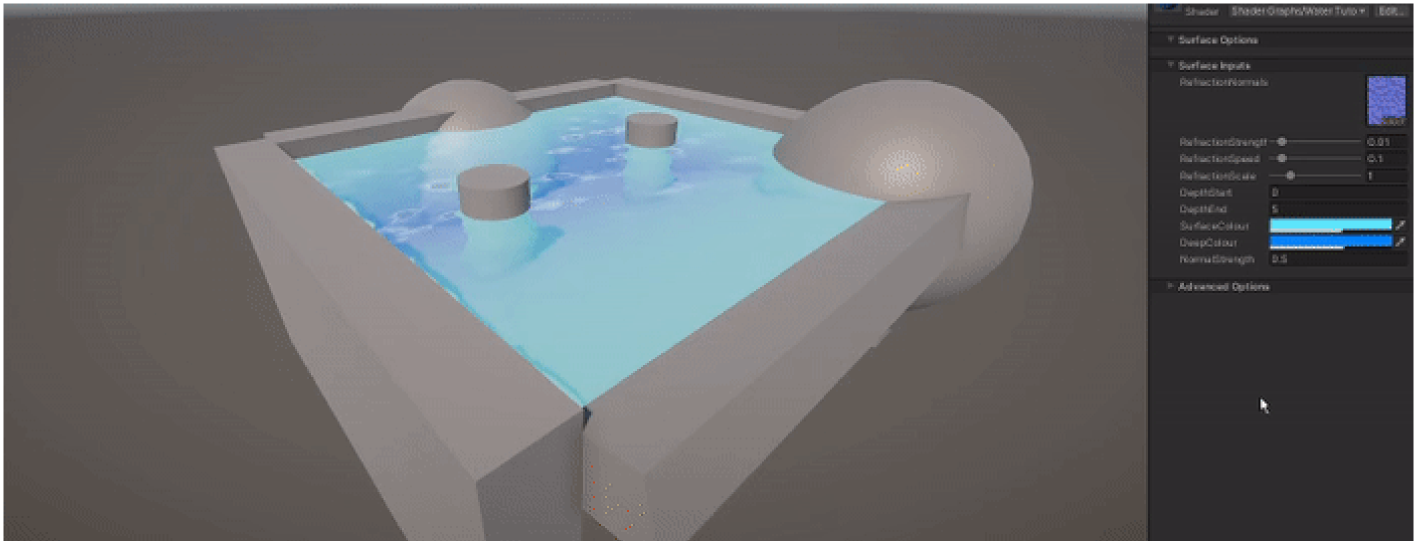


Figure 27 The shader with depth-based normal maps added.

Congratulations! You've made yourself a basic water shader!

Step 3.3 – Full Shader Graph Reference

In case your shader graph isn't working, or you want to reference the full code, here's an image of the complete shader graph:

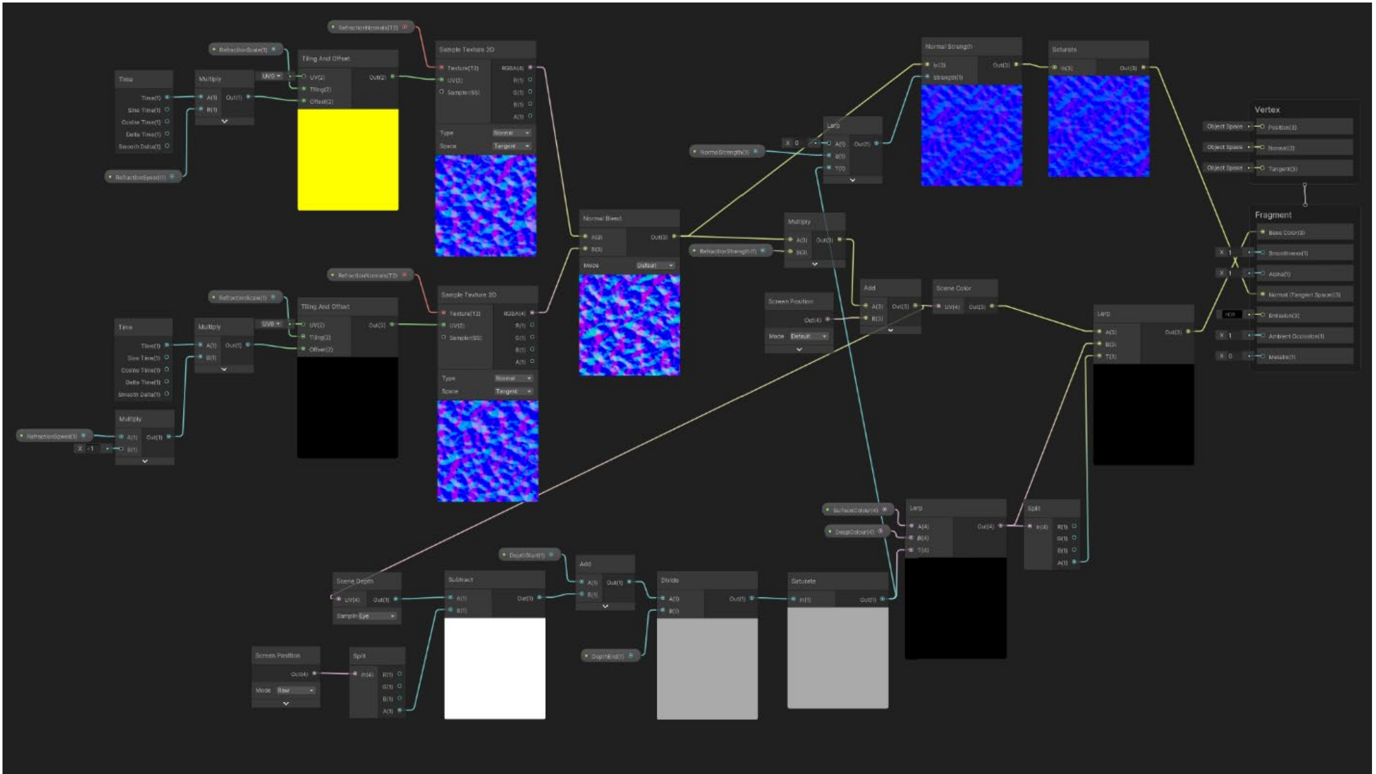


Figure 28: The completed shader graph.

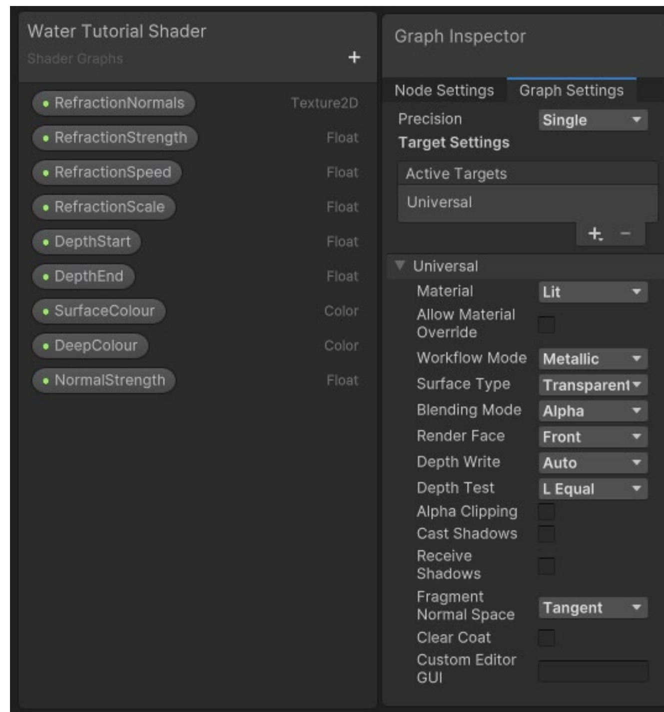


Figure 29 All variables in the completed shader graph.

Section 4 – What's Next?

Next Steps

This tutorial covered some of the basic techniques of water shaders that are used in most games, but don't treat this like the end of the road! Lots more techniques can be used to make your water shader stand out even more depending on the style and game you're creating.

Some things you might want to consider adding include:

- Vertex Displacement/Waves
- Foam Borders
- Emissive Textures
- Fresnel Effects

These techniques can really elevate your shader and can even create entirely new effects other than water!

Extra Resources

Below are some extra resources that I used when researching how to create a water shader. I recommend checking them out, as some go around these techniques differently, or even add different effects entirely!

Glass Shader Using Shader Graph in Unity3D

https://www.codinblack.com/glass-shader-using-shader-graph-in-unity3d/

Looking Through Water, Underwater Fog and Refraction

https://catlikecoding.com/unity/tutorials/flow/looking-through-water/

Refraction in Unity Shader Graph

https://www.youtube.com/watch?v=inht8WYX-A4&t=214s&ab_channel=DanielIlett

Stylized Water Shader Graph – Unity 2022 Tutorial

https://www.youtube.com/watch?v=1yevpCAA_rU&t=937s&ab_channel=BinaryLunar