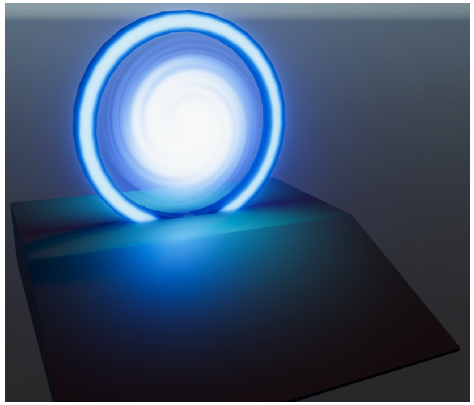
ImpactLab - Game Dev
Lecture 7: VI + Shader Graph

Summer 2024

School of Computing and Data Science

Wentworth Institute of Technology

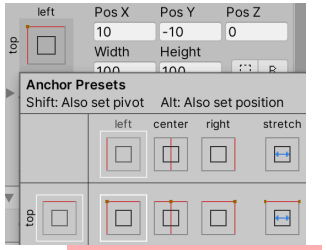


Wentworth
Computing & Data Science

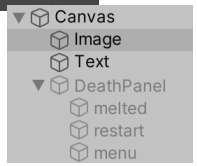
Health UI

Here's the Health UI on the same Canvas:

Note that the anchor is in the top left:



And the Image and Text are children of the Canvas (not the DeathPanel):



We'll update the text via code, for now I've entered "x3" as the default.

Code Hookup

Remember: public fields are updated in the inspector.

```
public Text healthText;
public GameObject DeathPanel;

public void updateHealthText(int health){
  healthText.text = "x" + health
}

public void deathPanelSwitch(bool state){
  deathPanel.SetActive(state)
}
```

GameManager.cs

These are the public methods that allow us to update the GUI information, i.e. show the DeathPanel and change the text for the health.

Wentworth
Computing & Data Science

Code Hookup

Now we can add the previous calls to the appropriate places in our current code:

GameManager.cs

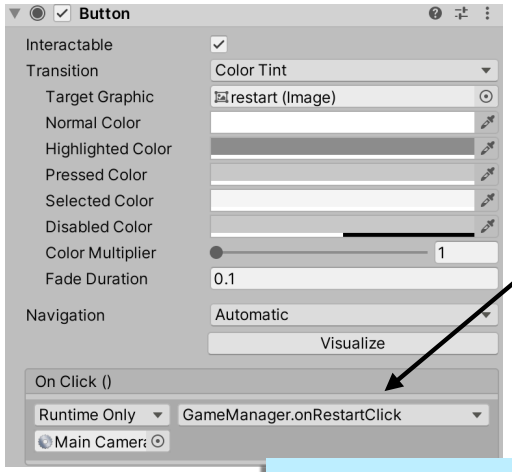
```
void Start(){
  deathPanelSwitch(false);
  ...
}

public void onRestartClick(){
  if(deathPanel.activeSelf){
    deathPanelSwitch(false);
    player.reset();
    spawner.reset()
  }
}
```

And create the method to call when the restart button is pressed

Wentworth
Computing & Data Science

Code Hookup



Easy to hook up function calls to button presses.

Restart Button

Back to Player and Spawner

Our onRestartClick() method resets the player and the spawner:

Position the player wherever you want, and ensure that the player gameObject is active.

```
public void reset(){
  health=3;
  this.transform.position=new Vector3(0f,0f,0f);
  GameManager.instance().updateHealthText(health);
  this.gameObject.SetActive(true)
}
```

Player.cs

Spawner.cs

```
public void reset(){
  timeBetweenSpawns=1.25f;
}
```

Reset spawner timing.

Back to Player and Spawner

When the player takes damage and/or "dies" we need to make sure we trigger the GameOver screen

```
public void takeDamage(int value){
  health-=value;
  GameManager.instance().updateHealthText(health);
  if(health<=0){
    //Instantiate death effect here if you want
    this.gameObject.setActive(false);
    GameManager.instance().deathPanelSwitch(true);
  }
}
```

This will make the player gameobject disappear (without destroying it in memory) and show the "DeathPanel"

Lots of Little Stuff

OK! There's always lots of little questions that we have to answer, for example:

Q: Do we want fireballs to continue spawning after the players dies?

A: Add a guard clause in the spawner script update() to check if the player is active.

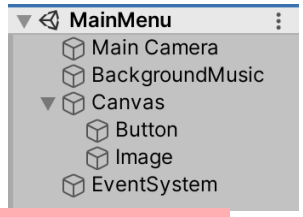
Q: How do we ensure the health is correct and the DeathPanel is off at the start of the game?

A: In the GameManager start(), call reset() on the player and spawner and call deathPanelSwitch(false)

Main Menu

Create a new Scene, call it MainMenu and put it into the scenes folder. Load this scene (double click it).

Ensure that there is a Camera in the scene and add a UI button and UI Image.



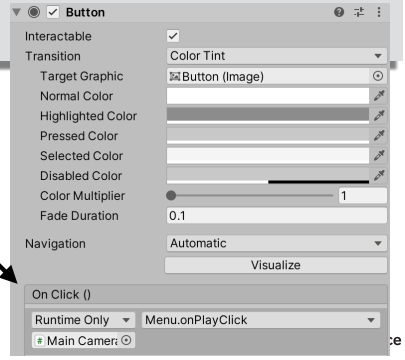
We'll add that BackgroundMusic gameobject in a moment

Main Menu

Create a new script called Menu.cs and attach it to the Main Camera

```
public class Menu : MonoBehaviour
{
  public void onPlayClick(){
    SceneManager.LoadScene("MainGame");
  }
}
```

Hook this up to the button that you just created. This loads the scene "MainGame" and unloads the current scene.



Back in GameManager and the MainGame Scene

Create a method to 'fire' if the user presses the menu button from the GameOver screen:

```
public void onMenuClick(){
  if(deathPanel.activeSelf){
    SceneManager.LoadScene("MainMenu");
  }
}
```

Hook this up to the menu button on the DeathPanel.

This is how you change scenes.

Scenes can represent many things: Menus, levels, cutscenes, etc.

Music and Sound

For now, we're just going to add the background music. Adding sound effects will be up to you in the homework.

We'll add the new gameObjects to the MainMenu scene.

Then, we'll set it to persist across scene transitions.

Sound effects will work similarly to what you see here, though they will likely just play once.

Feel free to use whatever software you want for creating music and sound effects.

Music:

- Bosca Ceoil
- LMMS

SoundFX:

- Audacity + your voice
- Bfxr

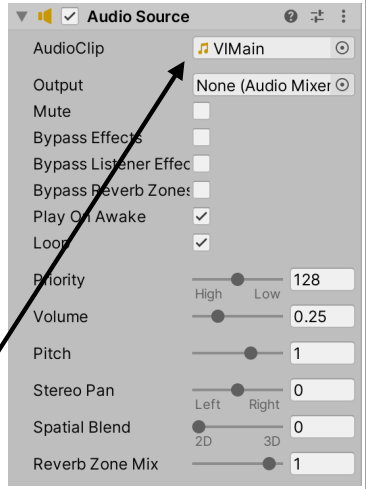
Background Music

Let's start by creating a new empty gameobject, name it "Background Music" and add the "Audio Source" component.

Make sure to have Play on Awake and Loop enabled.

Now create a new script, MusicManager and add it to the gameobject that we just created.

Since this will be our background music, drag your audio clip here



Background Music

We'll start by making the MusicManager class a singleton and set it so that changing scenes doesn't destroy the object (so the music keeps playing).

```
[RequireComponent(typeof(AudioSource))]
public class MusicManager : MonoBehaviour
{
  AudioClip backgroundClip;

  static MusicManager _instance=null;

  void Awake(){
    if(_instance==null){
      _instance=this;
    } else {
      Destroy(this.gameObject);
    }
    DontDestroyOnLoad(this.gameObject);
  }
```

MusicManager Class

Background Music

The start method is what grabs a reference to the AudioSource component and sets the backgroundClip. Then, if the clip is not playing, play it.

```
void Start()
{
  AudioSource source=GetComponent<AudioSource>();
  backgroundClip=source.clip;

  if(!source.isPlaying){
    source.Play();
}
```

MusicManager Class

Shaders

While shaders are a cool topic, a full intro is way beyond the scope of this class. We could have several semesters dedicated to shaders and learning a shader language.

Fundamentally, shaders are programs that *shade* 3D scenes.

They provide the appropriate levels of light, dark, and color to materials in the scene. These days, they also help with special effects and can be used as computational programs as well.

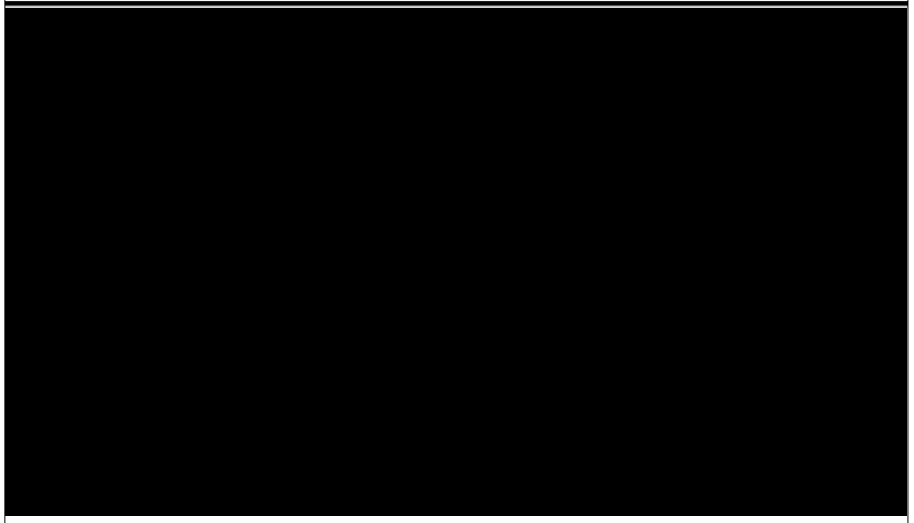
In essence, when you write a shader program, you are programing on the GPU.

Unity provides a few useful shaders for doing standard things, but for anything beyond typically needs a custom shader.

Resources

- If you want to learn about shaders, there are two standard resources:
- The Book of Shaders (for fragment shaders):
  - https://thebookofshaders.com/
- Shadertoy: interactive, web fragment shader creator
  - https://www.shadertoy.com/

CPU vs GPU



What are Shaders?

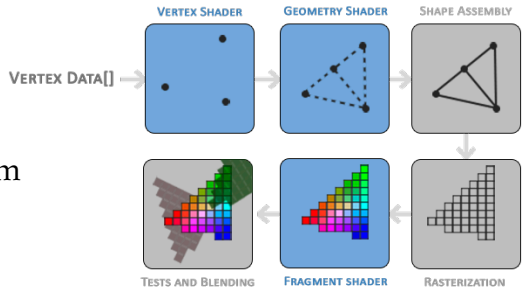
- Programs that run on the GPU
- Coded using GLSL or HLSL
  - These are C-like languages
- Compiled on the CPU (for your specific hardware) **at runtime**
- Come in two main types:
  - Fragment (colors on the screen)
  - Vertex (geometry information and manipulation)

In the parallel programming, I show a little bit of CUDA at the end of the semester

Shaders are similar, but are intended for drawing pixels on the screen rather than general purpose computing

Rendering Pipeline

- The entire rendering pipeline can be quite complex
- The boxes in blue are where we can program shaders
- Today:
  - The **fragment shader**



Example Shader

```
Shader "Unlit/SingleColor"
{
    Properties
    {
        // Color property for material inspector, default to white
        _Color ("Main Color", Color) = (1,1,1,1)
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            // vertex shader
            float4 vert (float4 vertex : POSITION) : SV_POSITION
            {
                return mul(UNITY_MATRIX_MVP, vertex);
            }

            // color from the material
            fixed4 _Color;

            // pixel shader, no inputs needed
            fixed4 frag () : SV_Target
            {
                return _Color; // just return it
            }
            ENDCG
        }
    }
}
```

Simple single color shader (ok, maybe not so simple).

The point is, you need to know the language and all the nuances to understand how this shader works.

However, Unity has *Shader Graph*, a way to write complex shaders with minimal or no coding.

Shader Graph: Getting Started

Make sure you have:

- At least Unity 2019 or later
- HDRP or URP

Ensure that Shader Graph package is loaded via the package manager.

Create a new Shader Graph:

Create → Shader Graph → URP → Lit Shader Graph

Typically, I create a **Shaders** folder to store them all in.

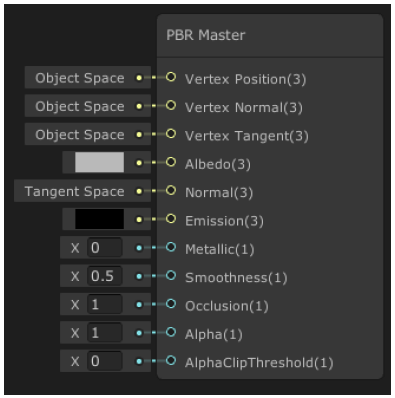
We will also need a material that the shader is applied to, right click on the shader: Create → Material (and give it a name)

Shader Graph

Here is our node: PBR Master

It contains many properties that we can modify to get the output that we want (on our material).

Each of the little circles on the left are inputs, and the boxes on the left are "default" values. We can either change them or hook other nodes into those values.



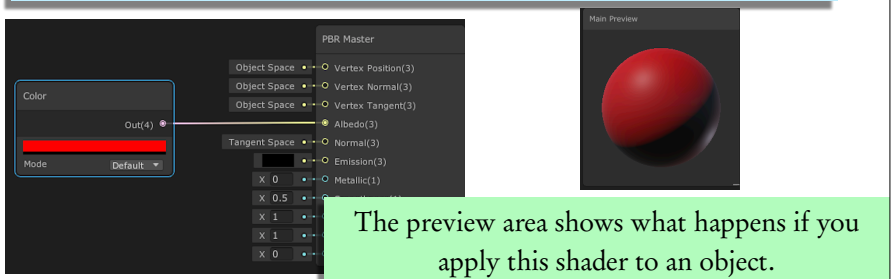
Shader Graph

Let's create a new node and hook it into the PBR Master: Right click, Create Node, type 'color', select Color.

We can drag the node around and adjust the color that this node represents.

The output circle can be dragged to the input of another node.

Drag the output to Albedo in the PBR Master.



The preview area shows what happens if you apply this shader to an object.

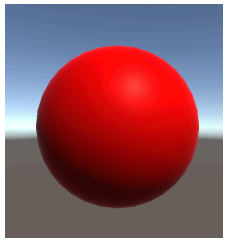
Shader Graph

Save the Shader! Upper Left, click '*Save Asset*'

Go back to Unity and create a 3D sphere:

Right click in the Hierarchy, 3D Object → Sphere

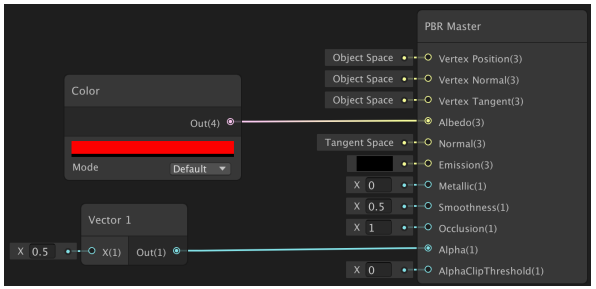
Drag the Material, that you created from the shader, onto the Sphere, either in the scene view or via the Hierarchy.



Now, if you change the color in the shader (and save it), that change will be reflected on our sphere.

Shader Graph

Add a Vector 1 node (this is just a single float value) and connect it to the alpha input of PBR Master. This controls the transparency of the object. Set the value of the Vector 1 to 0.5

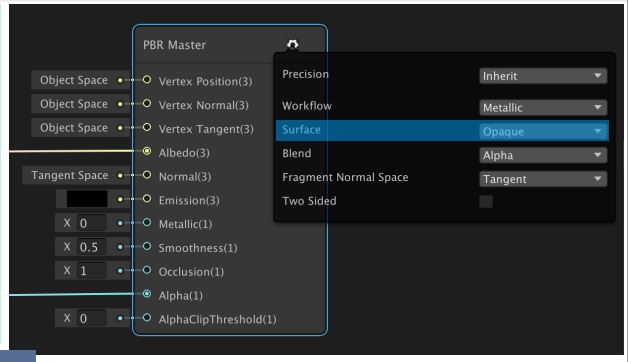


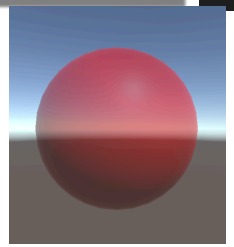
In principle, this should give us a partially transparent sphere. When you save and go back to the scene view, what happens?

Shader Graph

PBR Master is currently set to be an Opaque Shader.

Click on the gear on the PBR Master node, change it to Transparent and try again.





There are many little things like this that can have a very profound effect on your Shader.

Shader Graph: Couple of things

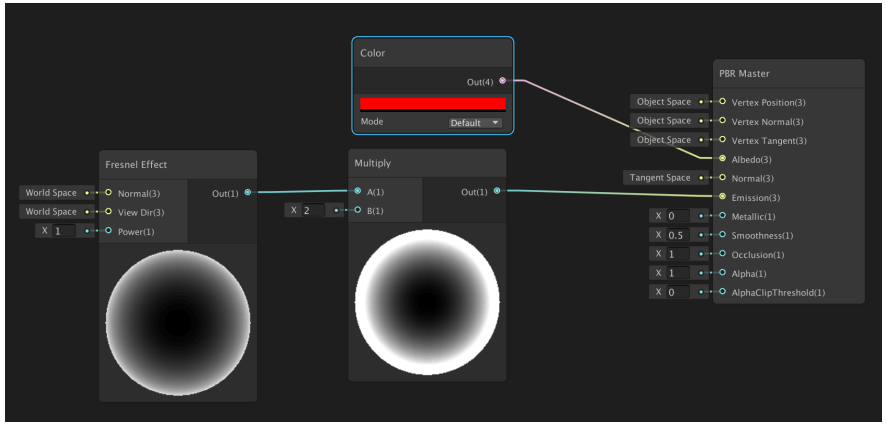
Here's a couple of useful things for dealing with Shader Graph:

- You can delete a connection by right clicking on it and selecting 'delete'
- You can delete nodes by right clicking on their top bar and selecting 'delete'
- Dragging and output line from a node to empty space will immediately allow you to create a new node
- The number (and color) next to the output/input represents the number of values associated with the output and the types of those values. Not all inputs can match to all outputs!
- An output can go to multiple inputs, but an input can only have one input

Shader Graph: Something More Interesting

Create a Fresnel Effect Node and a Multiply Node and hook them up as shown:

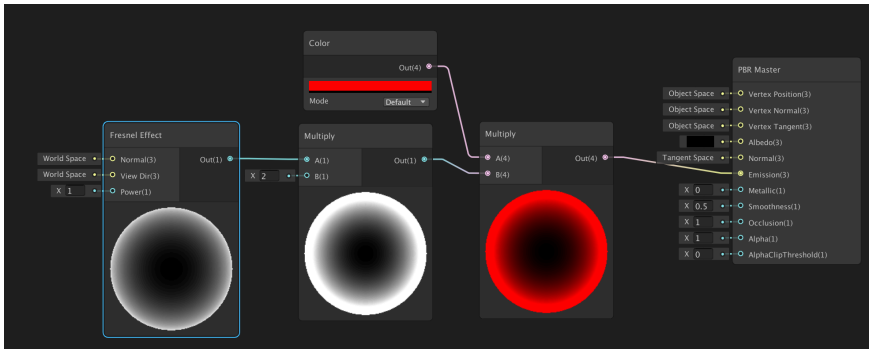
In the HDRP, the value to multiply by will need to be very large!



Shader Graph: Something More Interesting

The limb brightening effect is white, what if I wanted a different color?

We can multiply colors too!

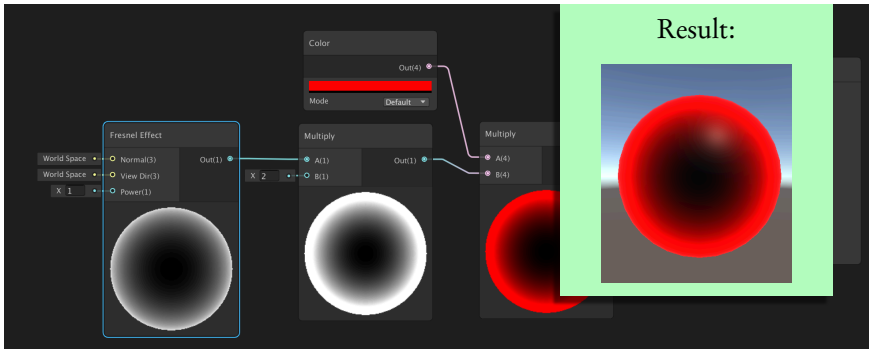


Wentworth
Computing & Data Science

Shader Graph: Something More Interesting

The limb brightening effect is white, what if I wanted a different color?

We can multiply colors too!

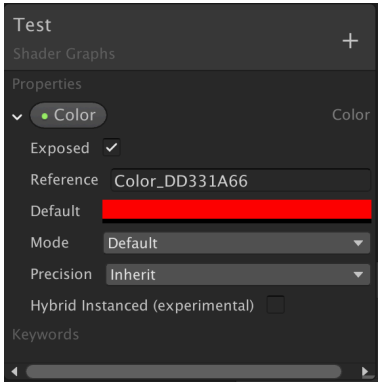


Result:

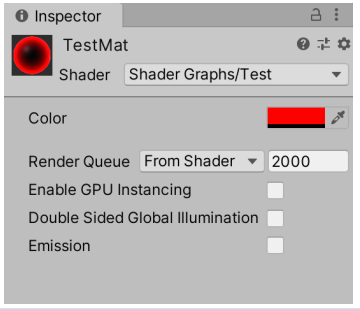
Wentworth
Computing & Data Science

Shader Graph: Properties

Right click on the Color node. Select 'Convert to Property'



Now it shows up in the 'Blackboard' and also:



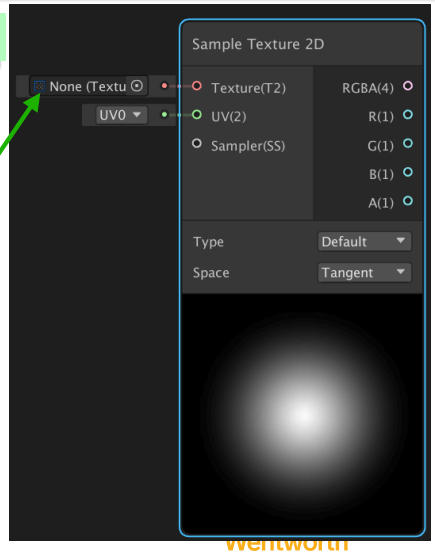
As an adjustable value in the material...We can now change the color without needing to edit the Shader Graph!

Shader Graph: Textures

Create a 'Sample Texture 2D' node:

The input is the actual texture image.

This can have its own node too (Texture Asset 2D) and can be converted to a property.

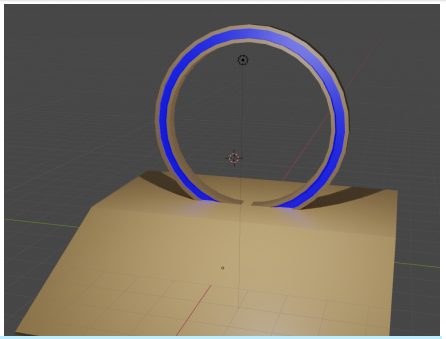


You can take the outputs individually or as a group of four.

If the texture you use is meant to represent the color of an object, you can input it directly into the Albedo channel in PBR Master.

Starter Scene

I've included a "stargate" mesh in the supplementary files.



We're going to recreate the portal in that scene.

I have many URP postprocessing effects enabled, which is a whole other topic entirely.

The main one is the Bloom effect, which gives the glow around emission materials.

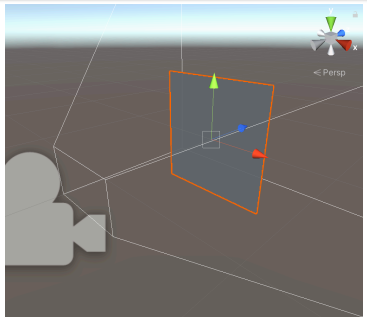
Create the Shader Graph

Create a new shader graph, name it whatever you want.

Create a material from that shader.

Create a new 3D object in the scene, a `Quad`.

Assign the material to the quad.



Add a "Time" node to the graph.

This is one of the most useful nodes for making interesting effects.

This allow us to change properties of the shader over time!

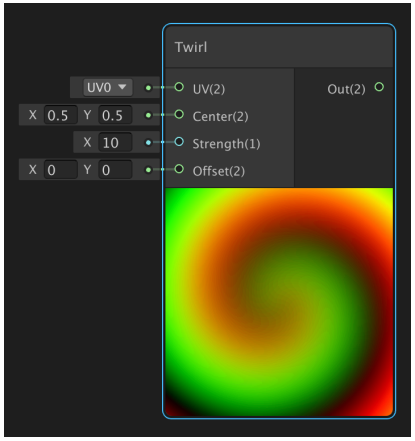
Twirl Node

The Twirl node skews any UV coordinate into a spiral like pattern.

Hook the Sine Time output into the Offset input and see what happens.

UV coordinates are usually represents by a 2D colored square:

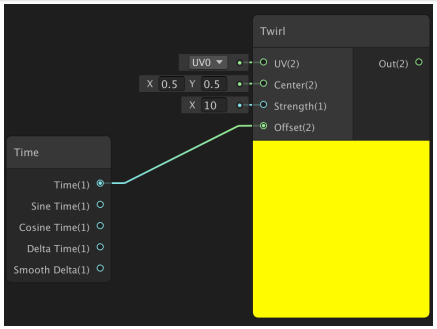
They are typically used to map textures to 3D models.



Note how the Twirl node distorts the typical UV square

Twirl Node

We will only be using the Time output, so hook that up.



The Twirl node goes completely yellow, but that's ok, the live preview is just reacting to the time input.

The Strength input allows us to control the amount of Twirl.

Create a new Vector 1, attach it to the Strength input and create a property out of it. This way we will be able to adjust the amount of Twirl without entering the Shader Graph.

Voronoi Noise

Shader Graph has a variety of noise nodes to help add randomness to your shaders.

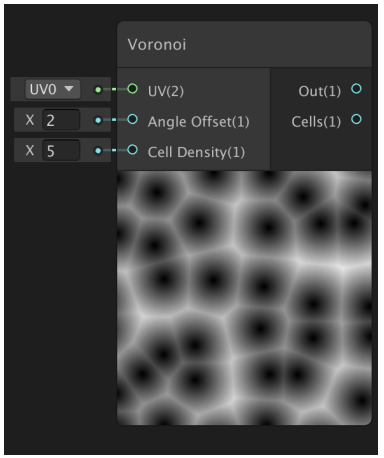
Voronoi Noise creates a random cell like pattern (and is actually easy to create yourself via code).

Note that it takes a UV input...

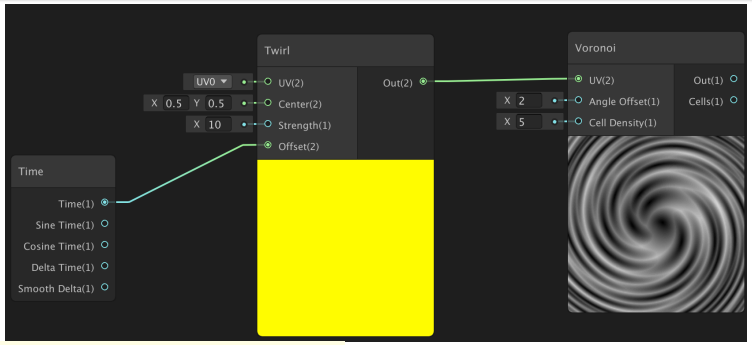
Our Twirl node distorts a UV...

Connect the Twirl output to the UV input...

What happens?



Twirly



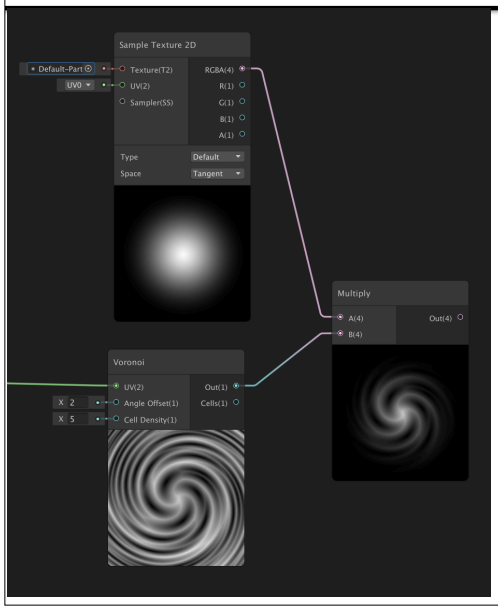
Now you can play around, what happens if you change the strength or center values on the Twirl Node.

What happens when you change the Angle Offset and Cell Density values on the Voronoi Node.

Hook the output of the Voronoi noise to the Albedo of PBR Master.

What happens to your Quad?

Not Done Yet...



Add a Sample Texture 2D and select the default particle texture for the Texture input.

Multiply the Out of the Voronoi and the RGBA output of the texture.

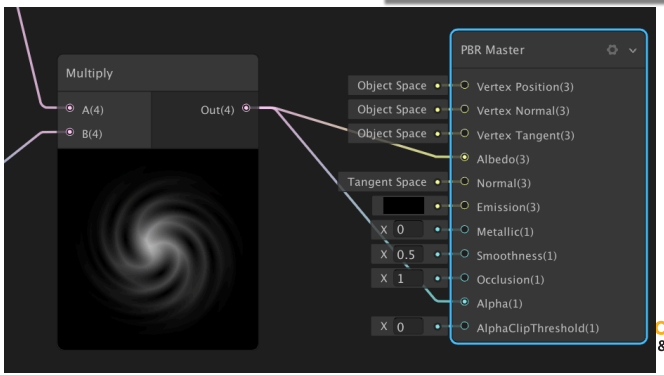
Examine the result. Make sure you understand why it looks this way.

Not Done Yet...

Connect the output of the multiply with the Albedo and Alpha of the PBR Master Node.

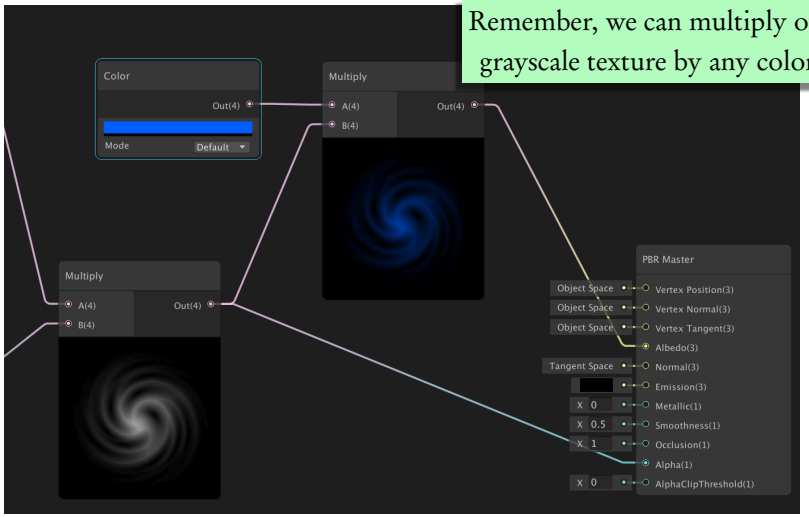
Remember, Alpha controls transparency (black is transparent, white is not).

Make sure PBR Master is set to Transparent too!



What About Color?

Remember, we can multiply our grayscale texture by any color!



Glow

This is getting closer, but it's not glowing like we want.

First, change the color mode to HDR, then go into the color picker and change the intensity (at the bottom).

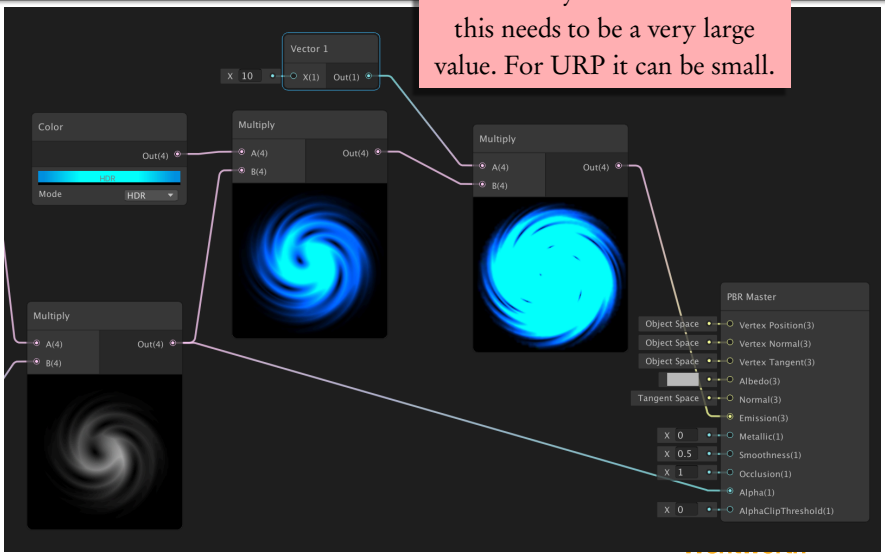
Add another multiply to the output of the color multiply. and hook up a Vector 1.

This allows us to enhance the Color by using a single value. Set the value to a very large number.

Instead of Albedo, hook the final output into the Emission input... What does it look like now?

Glow

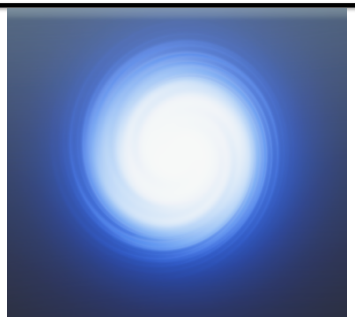
This Vector 1 helps us control the intensity. With the HDRP this needs to be a very large value. For URP it can be small.



Final Product

Now you can drag in the Stargate model and adjust the position of the quad to align with the gate.

You may need to scale the size of the quad to fit properly.



Keep in mind what we've done here:

There are only two triangles to render for the portal. The actual effect is handled by the GPU via the Shader program we wrote.

I've used the post processing stack to create the bloom effect, which is something we haven't talked about yet.

I also added lights to shine on the gate.

Feel free to make properties of more of the values, like the color.

What About the Twirly Speed?

Can you think of a way to increase or decrease the speed of the Twirliness?

Currently, we have no way to do it.

Finally, mess around with the values, add a different noise node or texture. At this point it's about what **you** want the effect to look like.

Lots of Resources

There are dozens of nodes to explore in Shader Graph and with them you can create very complex shaders that do some cool effects.

Unity has lots of great resources for learning about how to create your own shaders:

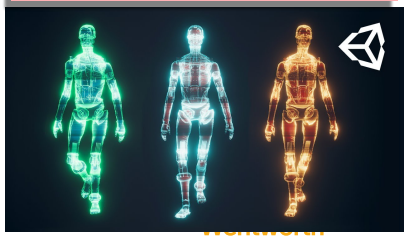
https://unity.com/shader-graph

Sample Shader Graph Project (a little old now, but still worth looking at):

https://github.com/UnityTechnologies/ShaderGraph_ExampleLibrary

And there are lots of other tutorials and resources online:

https://www.youtube.com/watch?v=KGGB5LFEejg



Coming Up:

- Simple Platformer