Unity Shader Graph Workshop
VR Jam 2025

Dr. Micah Schuster

Spring 2025

School of Computing and Data Science

Wentworth Institute of Technology

GitHub Repo with all the slides from today and the starter and finished versions of the scenes and shaders:

https://github.com/mdschuster/SGWorkshop

Shaders

While shaders are a cool topic, a full intro is way beyond the scope of this workshop. 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.

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

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

Art, Science and GPU's

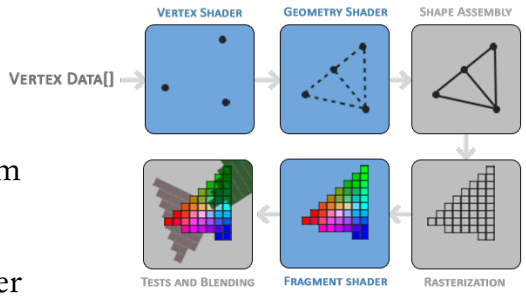
Adam Savage & Jamie Hyneman

Explain Parallel Processing

NVIDIA

Rendering Pipeline

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



Shader Graph: Getting Started

Make sure you have:

- At least Unity 2019 or later
- HDRP or URP

Our example project for today uses the URP, but this will also work with HDRP.

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

Create a new Shader Graph:

Create → Shader Graph → URP → Unlit 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)

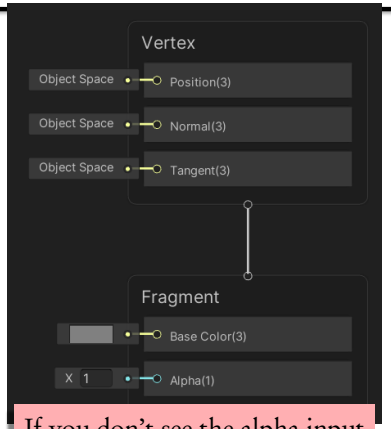
Shader Graph

Here is our Master node

It contains two parts, the vertex shader and fragment shader.

We will only be using the fragment part: color and alpha

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.



If you don't see the alpha input in the fragment shader, you'll need to set the shader to "Transparent" in the global properties.

Concepts: UV Distortion

UV coordinates are typically used to map a 2D image unto a surface.

- When you bring a texture into Unity, that texture is mapped to a square in UV space.
- Any distortion to the texture's UV coordinates will distort the texture.
- This is the basic concept for the shaders we will create throughout this workshop.

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

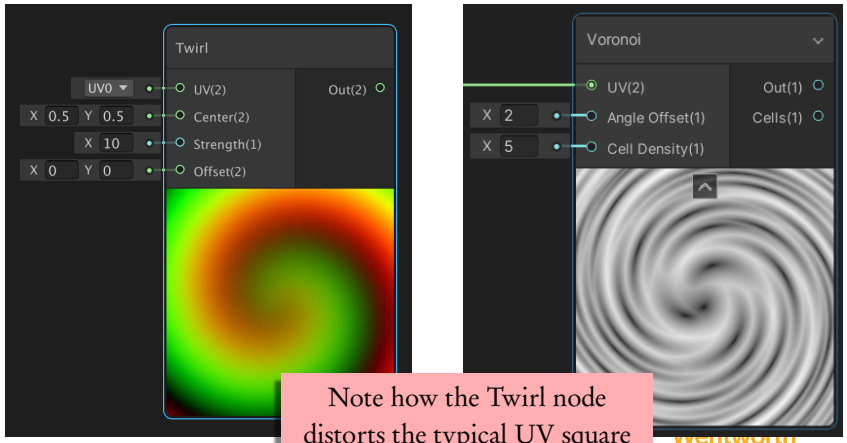
The colors are (Red, Green) mapped to (x,y) on the image.

For example, (0,0) is the lower left, black, and (1,1) is the upper right, yellow.

Concepts: UV Distortion Example

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

Hook the output of the spiral node into the input UV of a texture:

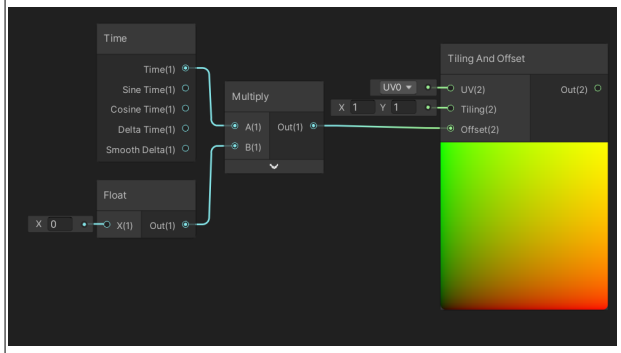


Note how the Twirl node distorts the typical UV square

Concepts: Multiply

Multiplying the output of two nodes can be done using the Multiply node

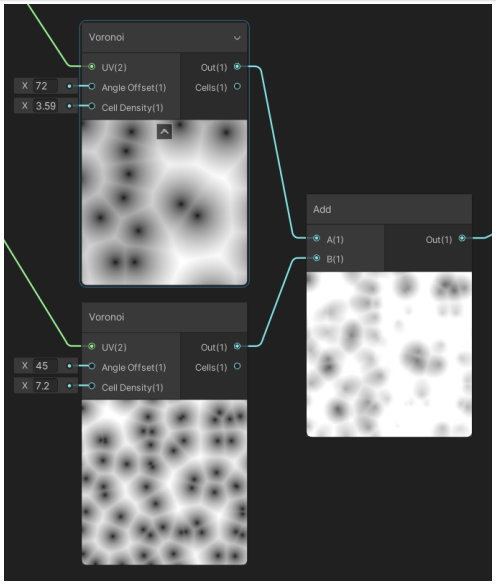
- The inputs can be nearly any output from another node: floats, vectors, textures, UVs, etc.
- Different inputs have different effects: For textures, pixel values are multiplied



Here, adjusting the value of the float now changes the speed of the scroll effect

Concepts: Add

- For a shader today, we'll use the sum of two noise textures that scroll at different speeds
- Adding and multiplying textures is very intuitive, the operation is done to each input channel. Use the preview to help get the effect you want.



Project 1: Portal

Our portal effect is fairly simple, but shows some of the basic Shader Graph nodes and their usage.

In the Starter project, Open the Portal Scene to see the portal model and a quad to apply our effect to

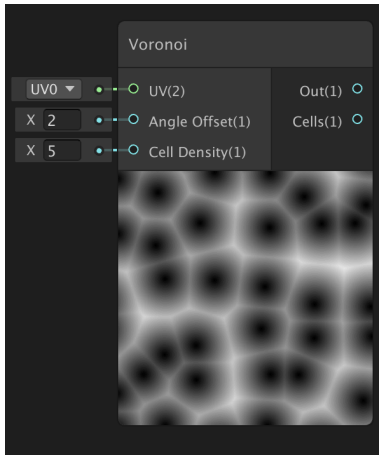
Wentworth
Computing & Data Science

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...



Wentworth
Computing & Data Science

Twirly

Hook up Time to the Twirl Offset and the UV output into the Voronoi UV input

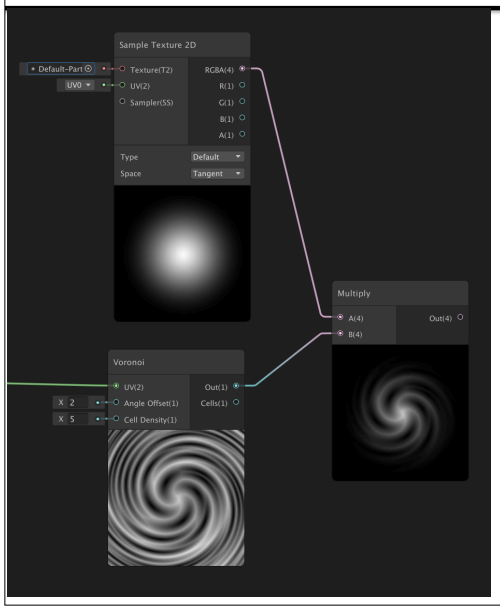
Take the out of the Voronoi and put it into the Base Color of the fragment shader

What happens to your Quad?

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.

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.

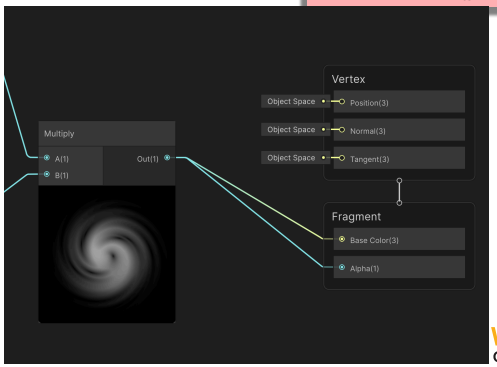
Wentworth
Computing & Data Science

Not Done Yet...

Connect the output of the multiply with the Base Color and Alpha of the Fragment Shader.

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

Make sure your settings are set to Transparent too!



What About Color?

We can multiply our grayscale texture by any color!

Glow

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

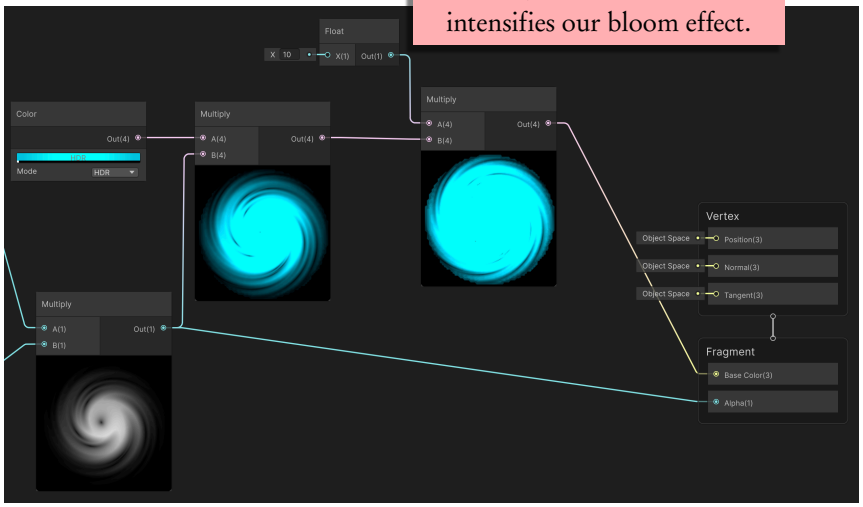
First, change the color mode to HDR, then go into the color picker and change the intensity (at the bottom).

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

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

Glow

This float helps us control the intensity of the color, which intensifies our bloom effect.



Final Product

Now you can 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 in the project to create the bloom effect.

At this point, you can add lights/particle effects to make it look even better.

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

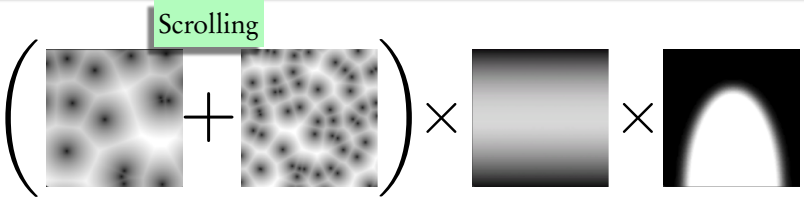
Project 2: RiME Fire

I've based this effect on a talk by Simon Trümpler (3D/VFX artist) about the VFX of the game RiME:

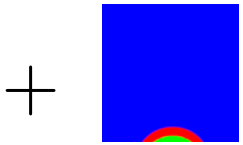
https://www.youtube.com/watch?v=fwKQyDZ4ark

Wentworth
Computing & Data Science

Basic Shader Outline



The resulting texture is then added to the RGB texture UV:

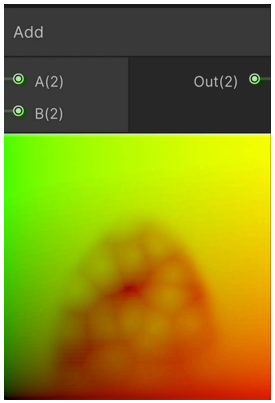


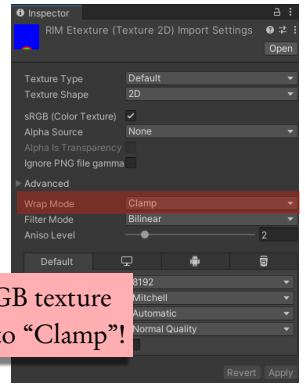
RGB UV

Wentworth
Computing & Data Science

What Does This Actually Do?

The RGB texture UVs are changed based (mostly) on the noise texture movement. Thus, different parts of the RGB image are shown in different places (this is the UV distortion).





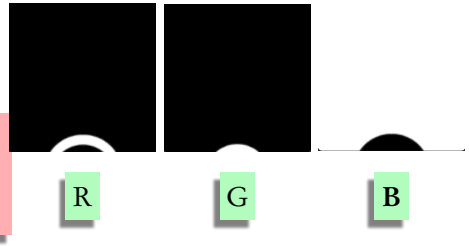
Note: Your RGB texture needs to be set to "Clamp"!

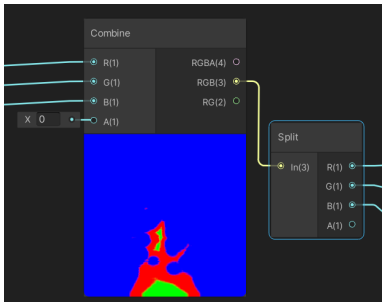
Wentworth
Computing & Data Science

Texture Packing

While a rainbow fire might be appropriate for some games, let's adjust the colors

Here, I've packed separate grayscale information into the same texture using the RGB channels

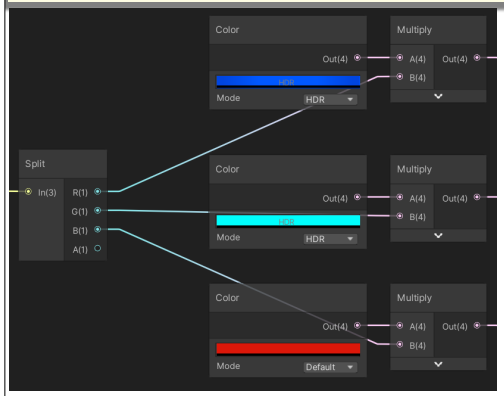


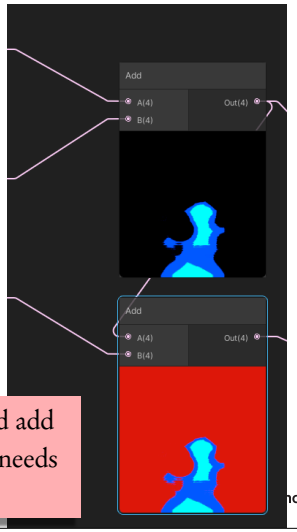


Using a Split node, we can separate each component to use independently.

Adding Color

Create a Color node and multiply it to one of the split RGB channels.



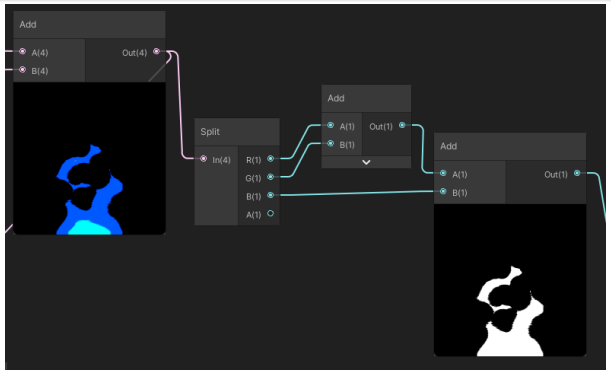


Do this for each channel and add them all back together (this needs two Add nodes)

Adding Alpha

For transparency, we need to do two things:

- Make sure our shader is able to be transparent
- Create an alpha "texture" to use in the alpha input of the Fragment shader



I create the input to the alpha by adding all the color channels from the "texture" created from the R+G channels.

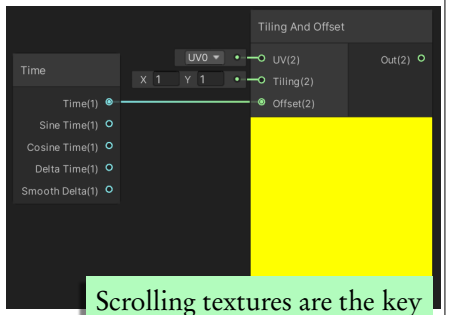
Time node for Scrolling Textures

Add a "Time" node to the graph.

This is a very useful node!

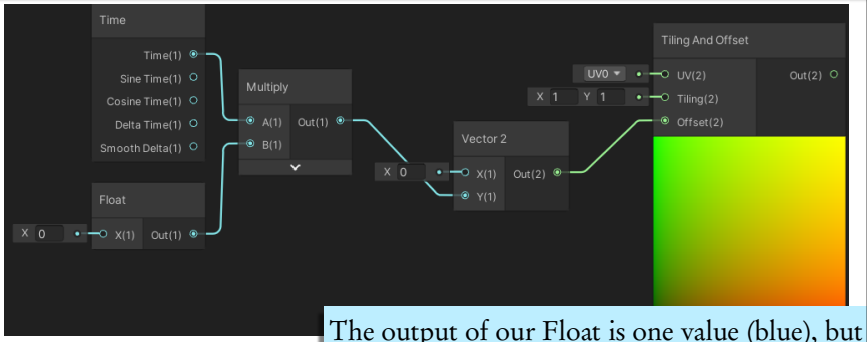
Remember: This allow us to change properties of the shader over time

Using a Time node with a Tiling and Offset node allows us to "scroll"our UV coordinates, thus making our texture appear to scroll over time.



Scrolling textures are the key to many interesting effects

Texture Scrolling



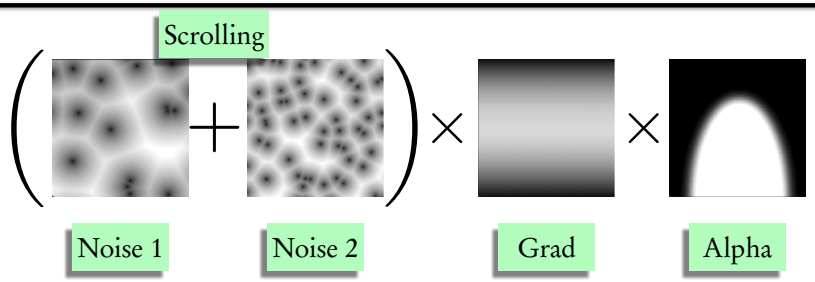
If we want to reverse the direction, we can just multiply by a -1 before input into the Tiling and Offset Note

The output of our Float is one value (blue), but it is plugged into a two value (green) input in the Tiling and Offset node.

This applies the float to both inputs (x,y).

To change this, we'll use a "Vector 2" node and input the float to the y input only.

Reminder: Basic Shader Outline



The resulting texture is then added to the RGB texture UV:

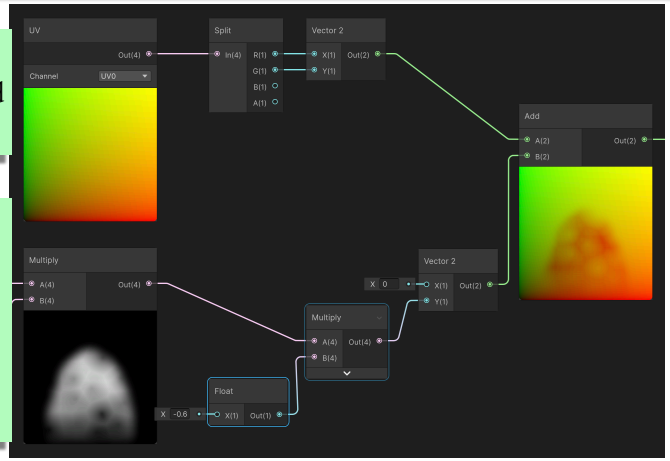
RGB UV

Wentworth
Computing & Data Science

UV Magic

Grab a UV node and take the R and G channel

Multiply the animated texture by a negative and put it into the y component of a Vector 2

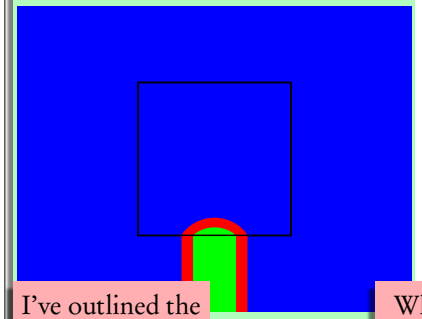


Now, add the texture (in the Vector 2) to the UV node. This new UV is then input into the packed texture.

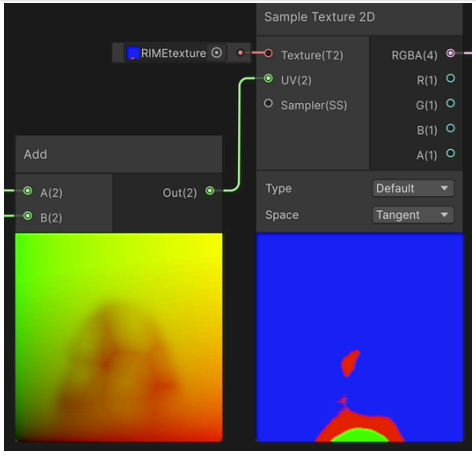
Science

UV Magic: Why does this work?

Our RiME texture is set to **Clamp**, so when Unity attempts to sample the value outside of the bounds of the texture, it sees this:



I've outlined the original image in black



When the UV is a "more" red, it is sampling from "below" the texture, which appears to pull the texture up.

Final Scene

I've created a couple of assets in Blender to use with the stylized fire, feel free to mess around with them: