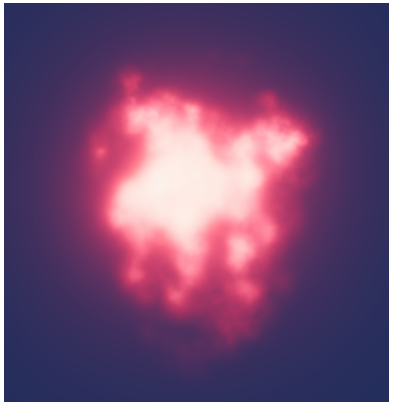
ImpactLab - Game Dev
Lecture 8: VI and More Shader Graph

Summer 2024

School of Computing and Data Science

Wentworth Institute of Technology

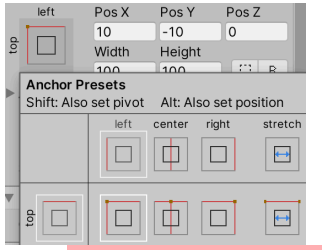


Wentworth
Computing & Data Science

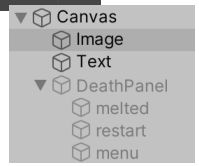
Health UI

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

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)

Wentworth
Computing & Data Science

VFX of Diablo 3

I've based this part of the workshop on creating the two-texture shader from Julian Love's GDC presentation:

https://www.youtube.com/watch?v=YPy2hytwDLM

The overall shader will be very general, which makes it a bit more complicated.

For example, we'll be using two textures in our shader, but each textures has a RGB component and Alpha component. Each Texture will be able to scroll at different speeds and tile at different scales.

This variation allows us to create very different effects with slight changes to the base RGBA textures.

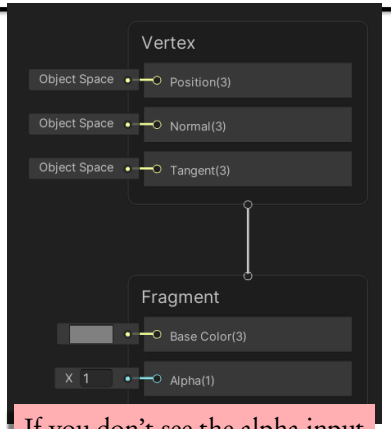
Initial Shader Graph

We'll use a basic unlit shader

Again, our focus will be on the fragment shader.

Reminder: 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.

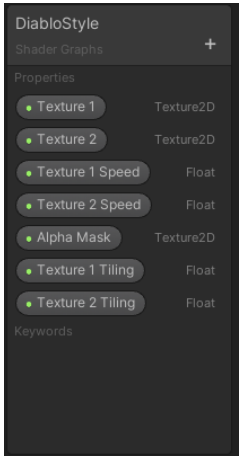
Start by creating two Sample Texture 2D nodes (and add a noise texture to each one).



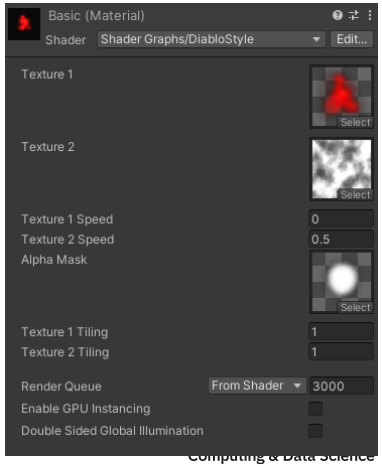
If you don't see the alpha input in the fragment shader, you'll need to set the shader to "Transparent" in the global properties.

Shader Graph: Properties

We're starting to get a larger number of parameters that can be adjusted and it's not ideal to open the shader to change them.



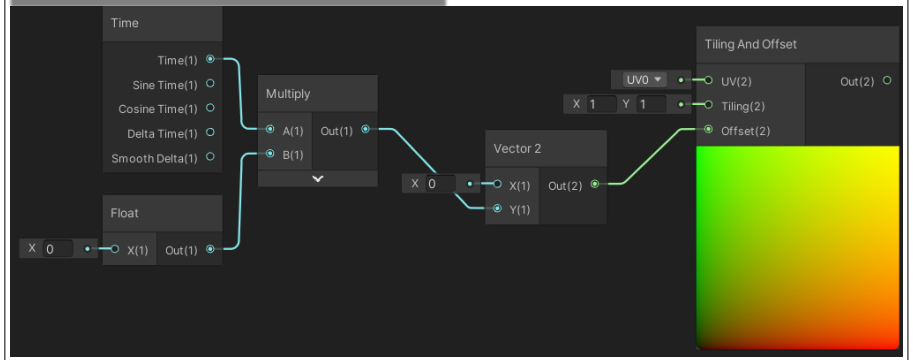
Creating a property adds the value to the blackboard (left) so that the value can be changed in the material (right)



Scrolling Two Textures

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.

We're only going to scrolling up (negative Y), so the float must be a negative value.

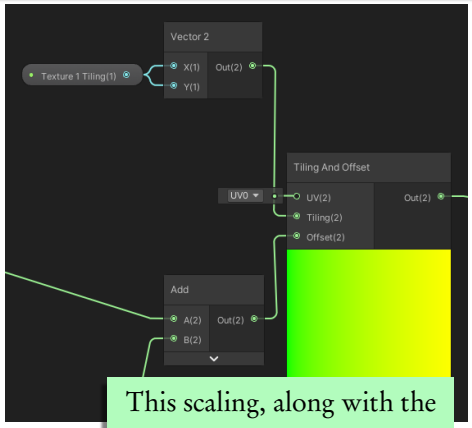


Scaling Two Textures

The Tiling and Offset node also includes a Tiling input.

If your textures are tillable (like the provided noise textures) you can easily "zoom in or out", changing the scaling of the texture.

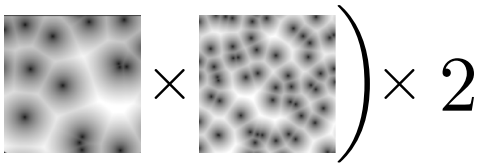
Just like adjusting the speed of the scrolling, create a float or vector2 and plug it into the Tiling input.



This scaling, along with the scrolling creates a lot of variation with only a few textures.

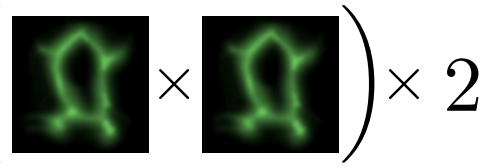
Basic Shader Outline

Alpha Channel



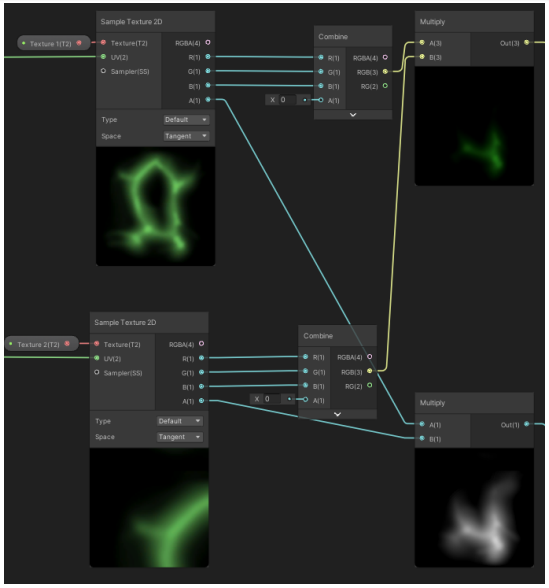
Alpha 1 Alpha 2

Color Channel



RGB 1 RGB 2

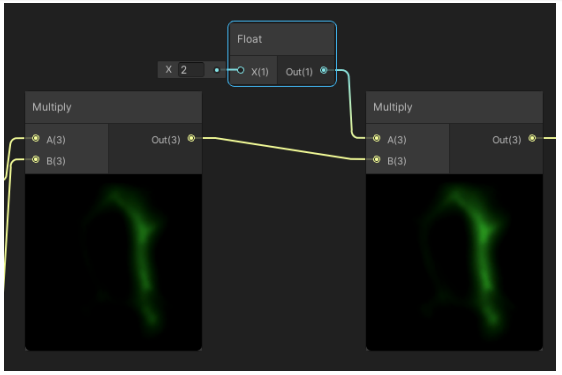
RGB and Alpha Setup



Note that each texture uses the individual components. The RGB are rejoined and multiplied (at the top) and the alpha channels are multiplied separately at the bottom.

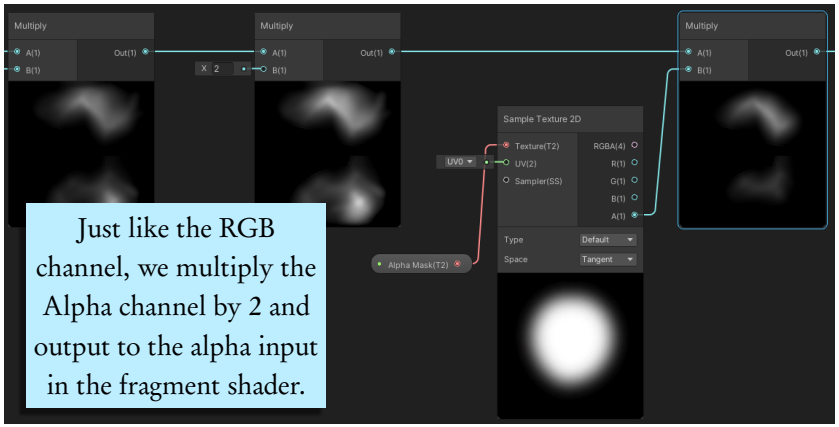
You're free to use whatever textures you want here. Creating four textures, 2 alpha and 2 RGB, can allow for even more variation.

RGB Multiply



Now we simply multiply the RGB channel by 2 and output to the color input of the fragment shader.

Alpha Multiply and Mask



Just like the RGB channel, we multiply the Alpha channel by 2 and output to the alpha input in the fragment shader.

I've included another texture, a mask. This softly fades out the edges to transparent so we don't see any edges of the texture. Masks like this can be very important for this kind of shader.

Particle Systems?

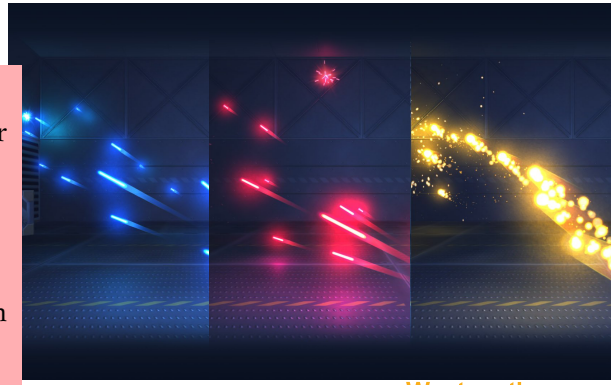
Our Diablo 3 style shader is best used in a particle system.

However, there are some major issues if we apply what we've created to our particle material:

- Color over lifetime doesn't do anything
- All scrolling starts in the same place for every particle (no randomness per particle)

Particle Systems: Brief Introduction

- Particle systems are typically used as visual effects representing everything from fire and smoke to sparks to weapon fire.



There are many tutorials out there for creating specific particle systems.

Today, we'll create a basic system and then reproduce a partial effect from a game.

Particle Systems: Brief Introduction

Particle systems are just a component that is added to a GameObject.

In each particle system there is an initial area for settings. Below that are other modules that can be activated to provide different options for how the particles behave.

You'll usually always use "Emission", "Shape" and "Renderer". The others will depend on the situation.

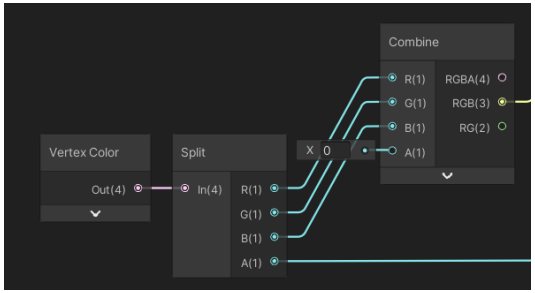
The Renderer module is where we will apply are material to the particles.

Let's play around with a particle system first before we jump back to our shaders

Vertex Color

When using the color over lifetime property of particle systems, the color (and alpha) is applied to the vertices of the particle quad.

Shader Graph can take that information, using the *Vertex Color* node, and use it in the shader.



Note that I split the vertex color into its RGBA components.

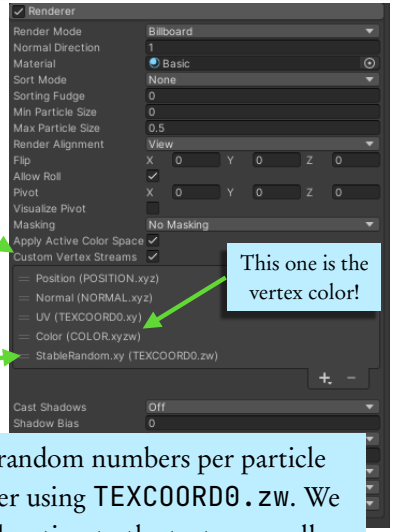
I recombine the RGB to multiply with the color of the texture.

You should do the same with the separate alpha channel, but multiplied with the alpha from the textures.

Better Randomness: Custom Vertex Streams

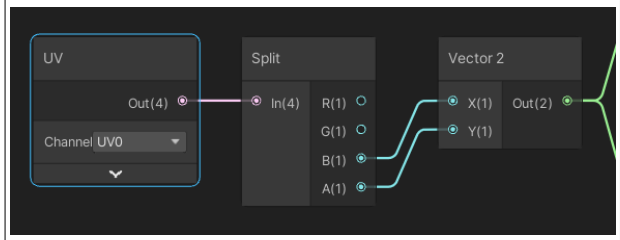
By default, custom vertex streams are not enabled in you particle systems. You must click the check box in the Renderer component:

From here, you can add additional information that can be sent to the shader, click the + icon and select `StableRandom.xy` so that it appears in the list.



`StableRandom.xy` generates two random numbers per particle (`0.0-1.0`) that are sent into the shader using `TEXCOORD0.zw`. We will use these to give a random start location to the texture scroll.

Better Randomness: Shader Input



`TEXCOORD0` is the UV0 channel in the UV node.

Since the two random numbers are `zw`, they correspond to Blue and Alpha if we do a split.

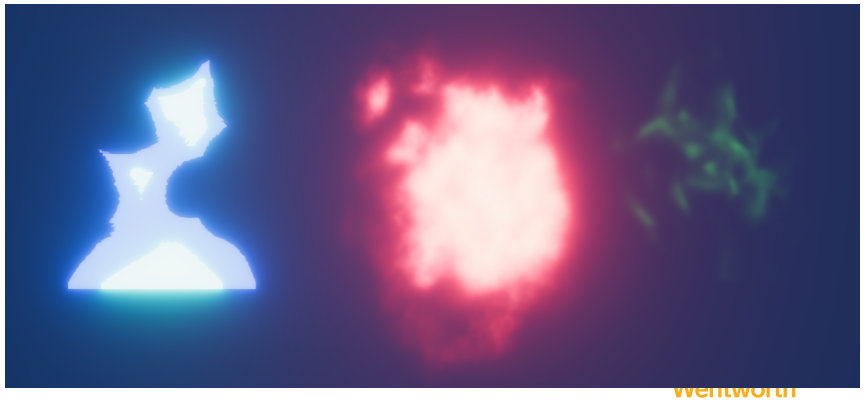
RG are the `xy` UV coords on the particle.

Here, I just put them into a Vector2 and add them to the offset value before it gets input into the Tiling and Offset node.

Now, each particle will have a slightly different starting place for the offset of the RGB and Alpha textures.

Final Scene

In my shader graph workshop repo, I've included a Main scene that includes three effects. The RIME fire and two effects using the Diablo style shader.



Cinemachine

How the camera reacts to the player is very important for any game.

You can certainly code your own camera controller to get exactly what you want.

However, I'm going to show you a built in solution:

Cinemachine

Window->Package Manager

Find Cinemachine and install it.

Add the "CinemachineBrain" component to your Main Camera

Create an empty gameobject and add the "CinemachineVirtualCamera" component

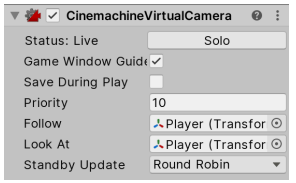
Cinemachine

Now the fun part:

Setting up the camera:

Click on your virtual camera

The virtual camera is now your main camera.



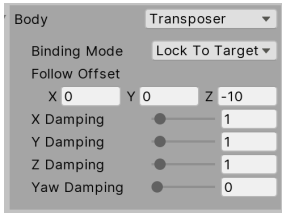
Set the Follow and Look At to the player gameobject.

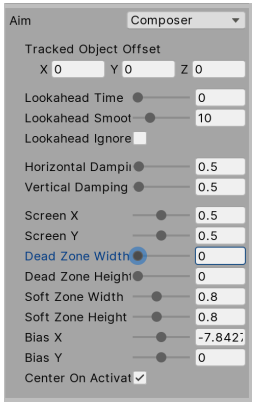
The virtual camera has many settings, most of which we won't mess with.

I certainly don't know every setting!

Cinemachine: Basic Settings

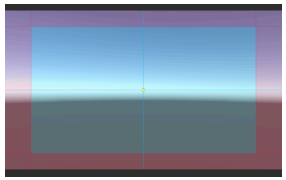
Many settings in the VCam can just be played with until you get the behavior you want.





The "Aim" controls where the camera looks.

Play around with the Damping, Dead Zone, and Soft Zone sliders.



Set the body to "Transposer" and the Blending Mode to "Simple Follow with World Up".

These setting deal with how the "Camera Body" moves.

Coming Up:

- That's All Folks!