

# Shaders and VFX

**Introduction to Game Development in Unity**  
**Spring 2026, 98-127, Lecture 5**

Instructors: Jingxuan Chen, Dario Quintero, Shangyi Zhu, Jeffrey Wang

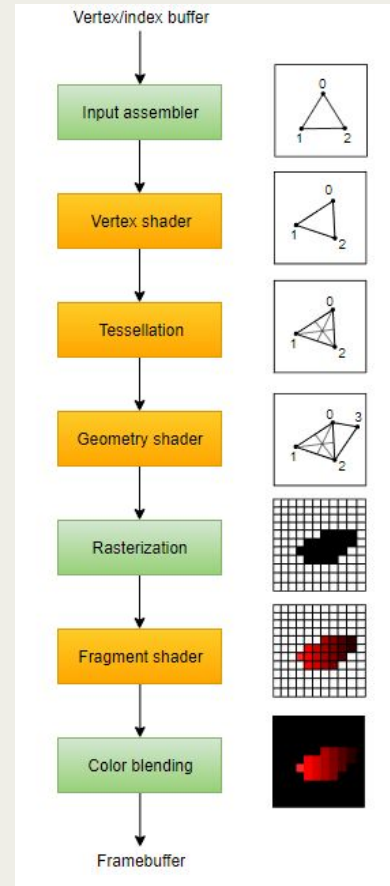
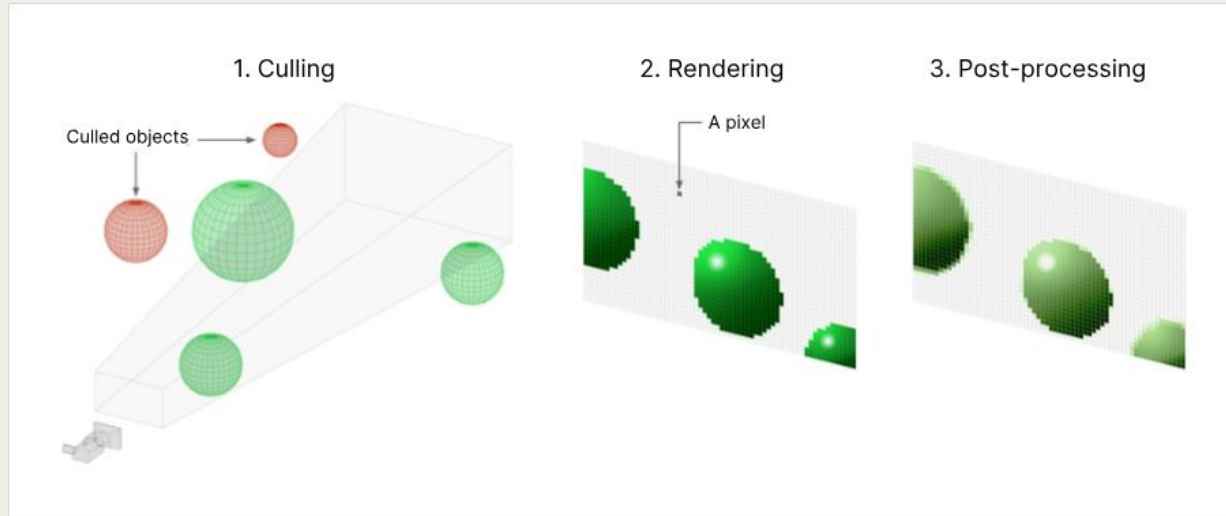
# Table of contents

1. Overview of the rendering pipeline
2. What are shaders
3. Why make shaders
4. Learning Shader Graph
5. Learning Unity Particle System



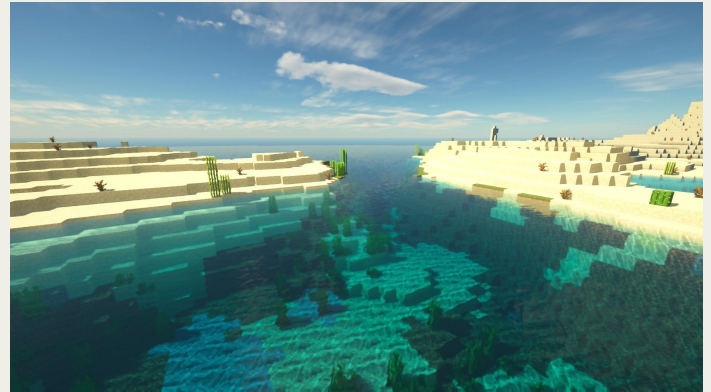
# What is the Rendering Pipeline

- The ordered set of steps Unity uses to turn scene data into a final image.



# What are Shaders

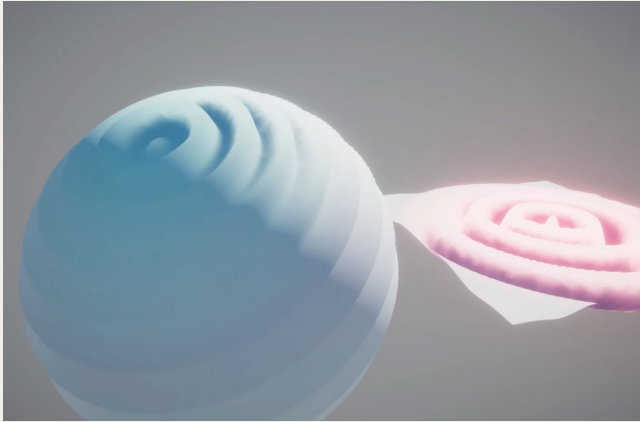
Any programmable operation to manipulate data in the rendering pipeline, specialize on render function on the GPU



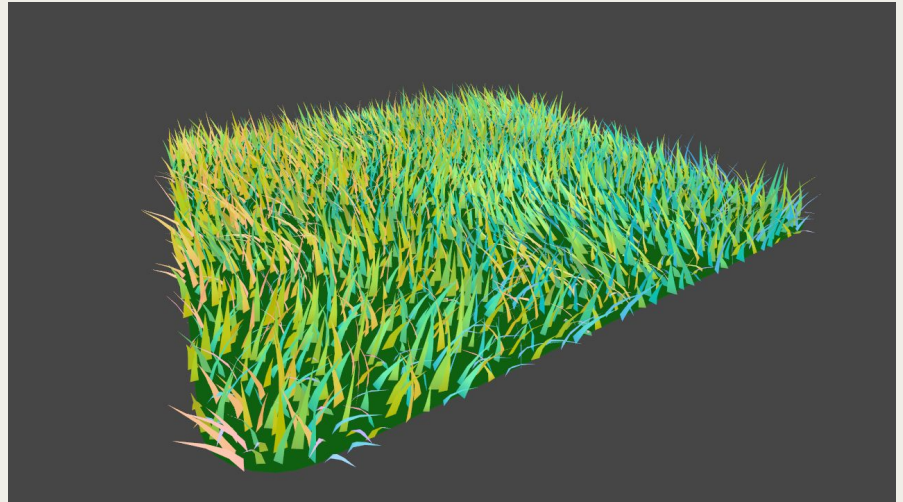
# Vertex Shaders: Edit vertices

Transform vertex of a 3D model -> 3D position in virtual space to a 2D coordinate on the screen.

- manipulate properties such as **vertex animation and position deformation**



Ripple texture waves



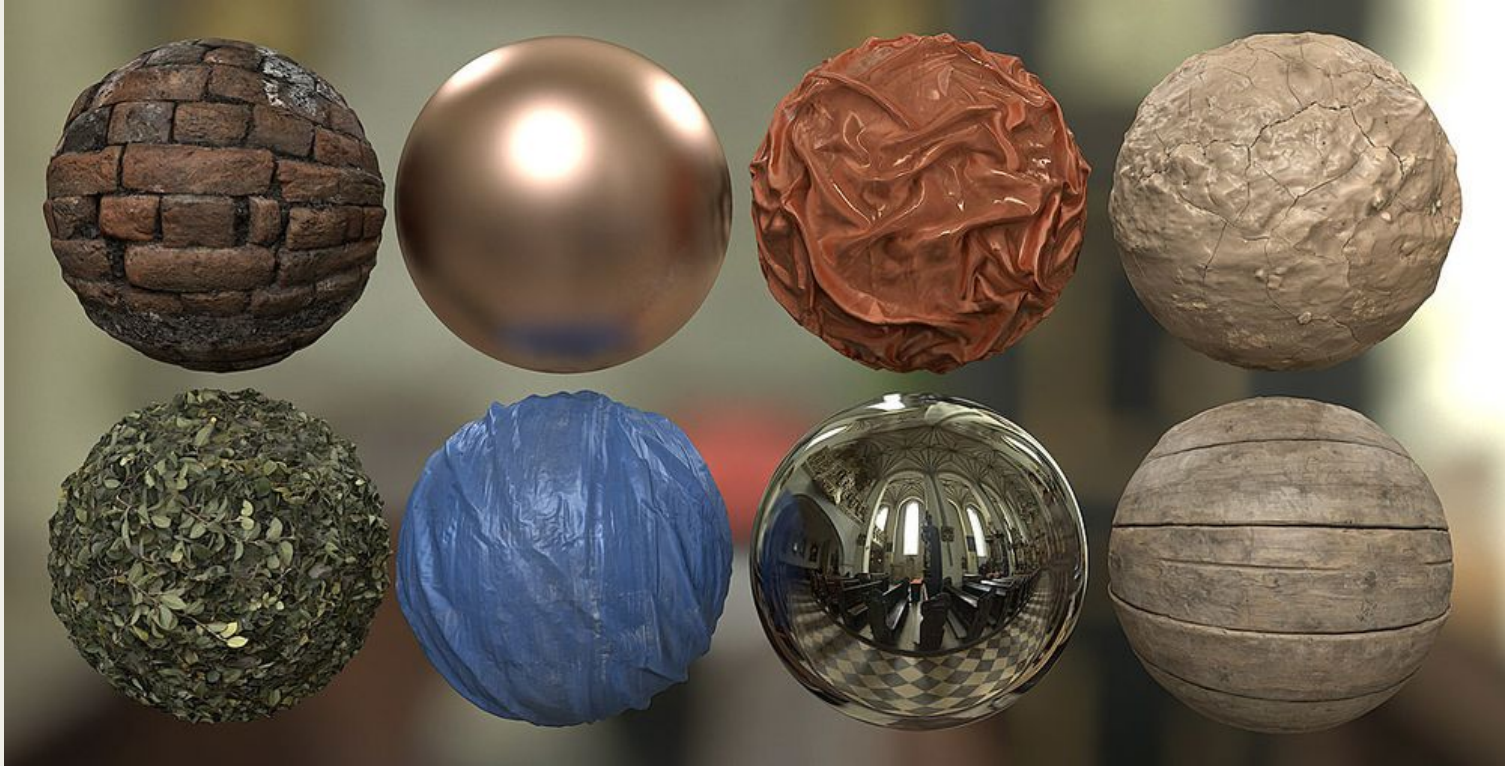
Procedural Grass

# Fragment Shaders: Edit Pixels

Changing color, textures, and lighting

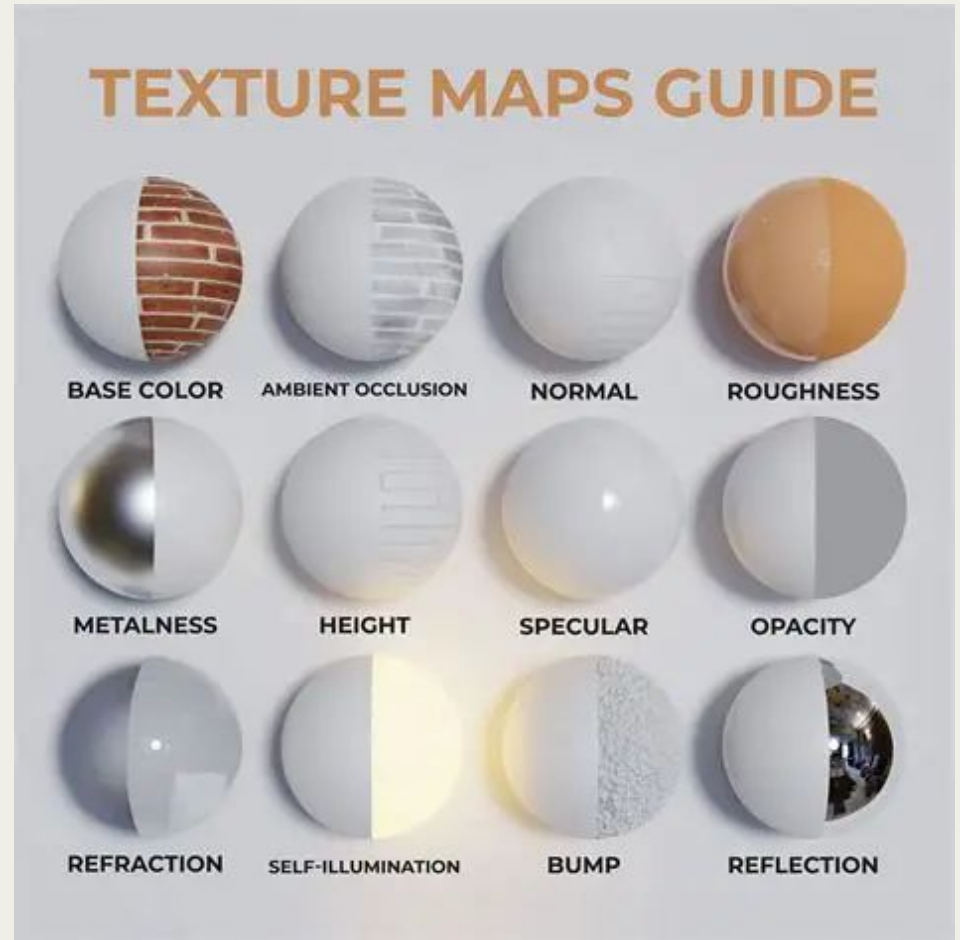


# Why do shader when you have materials?

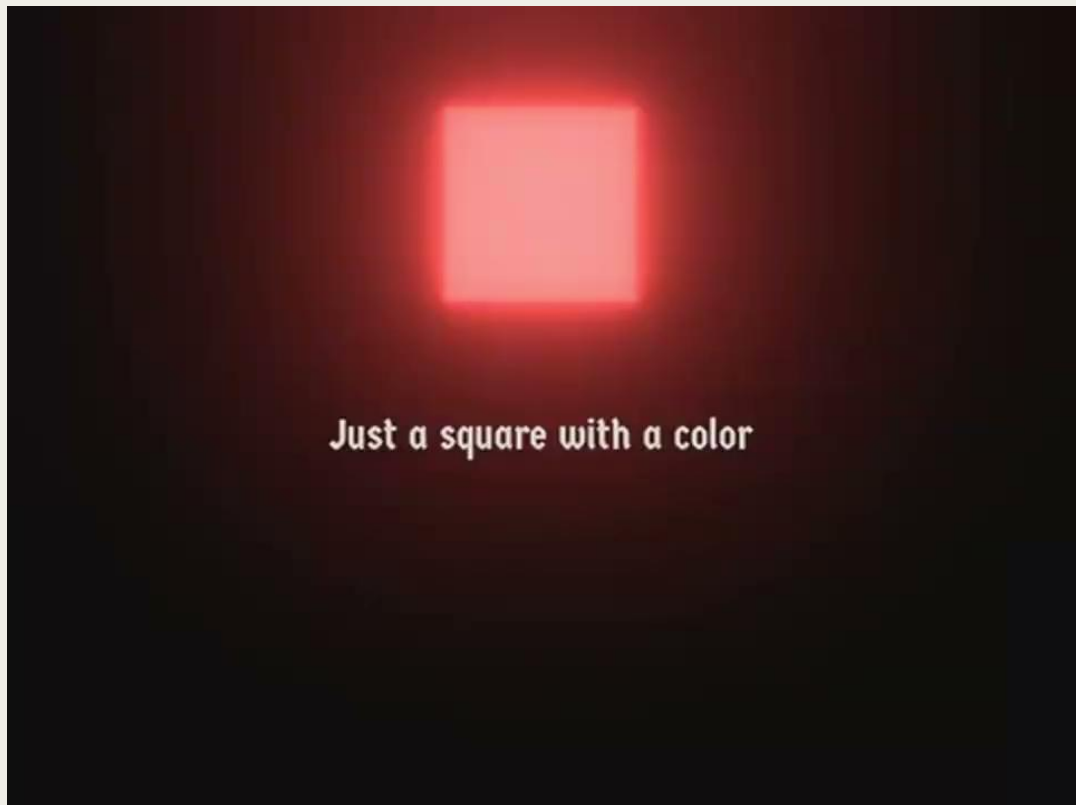


# What are Materials

**Materials** define how the surface of a looks by combining shaders and properties



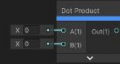
# Unity Defaults not enough?



# Why write your own custom shader?



# Shader Graph:

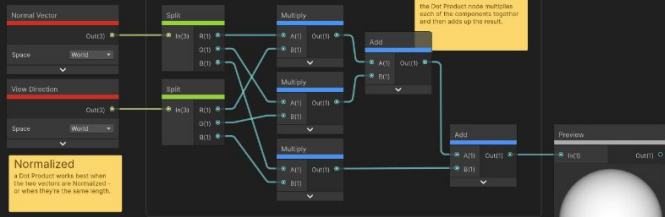


## Dot Product Node

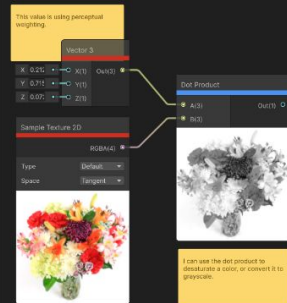
The Dot Product node returns the dot product, or scalar product, of the two input vectors A and B.

The dot product is a value equal to the magnitudes of the two vectors multiplied together and then multiplied by the cosine of the angle between them.

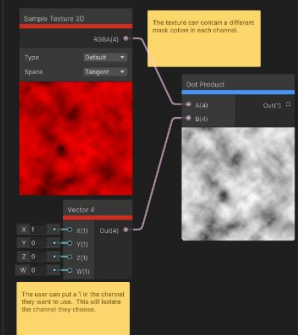
## Under The Hood



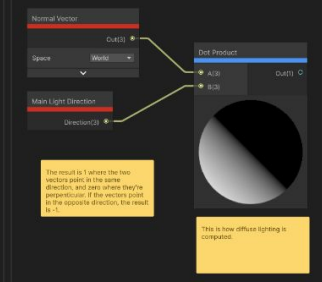
## Dot Product used for desaturation



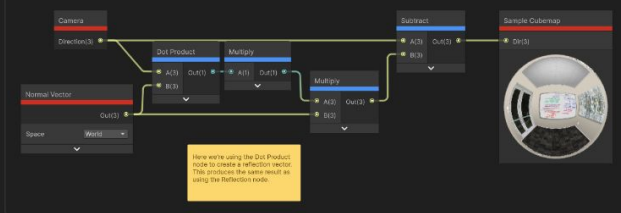
## Dot Product used for channel selection



## Dot Product used to compare two vectors.



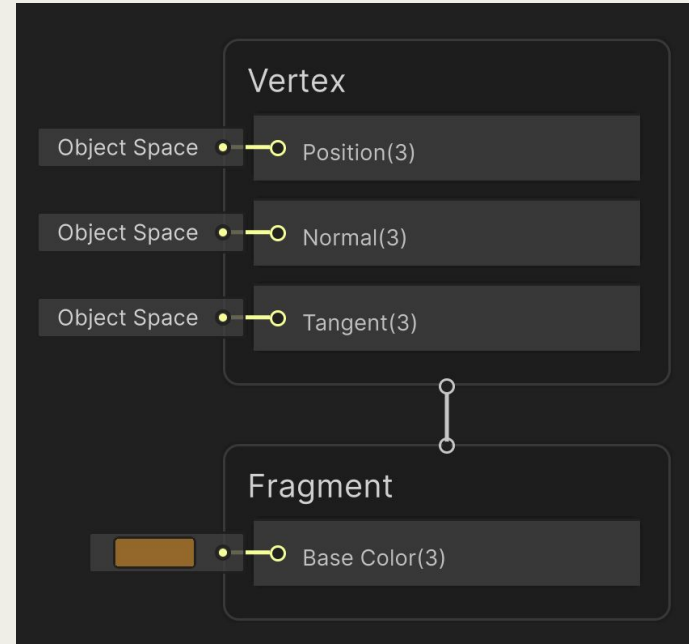
## Using a Dot Product to Create a Reflection Vector



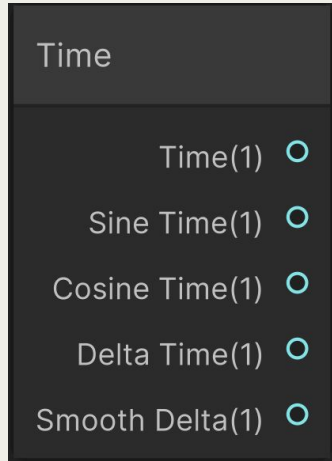
# Advantages of Shader Graph

\* aside: some evidence suggest shader graph does not work with webgl

- Little to no programming required
- Visually intuitive
- Real-time preview
- High reusability
- Combines + abstracts linking between vertex and fragment shader

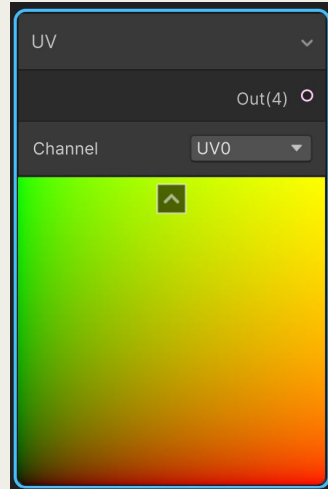


# Input Nodes: Gets input data for your shader



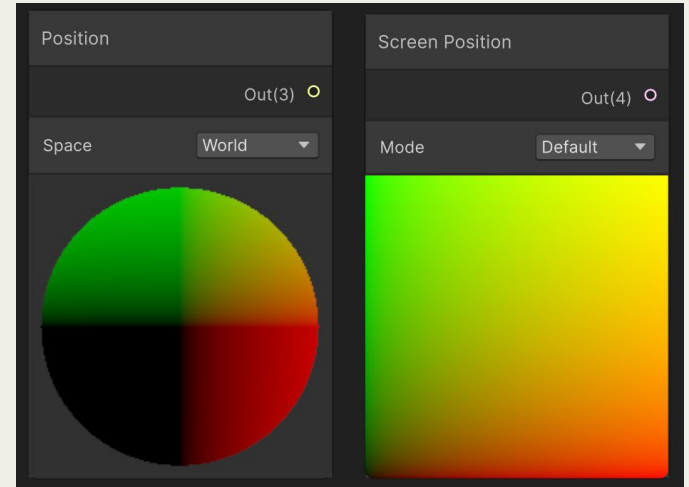
## Time

Gets time



## UV

Gets mesh UV coordinates

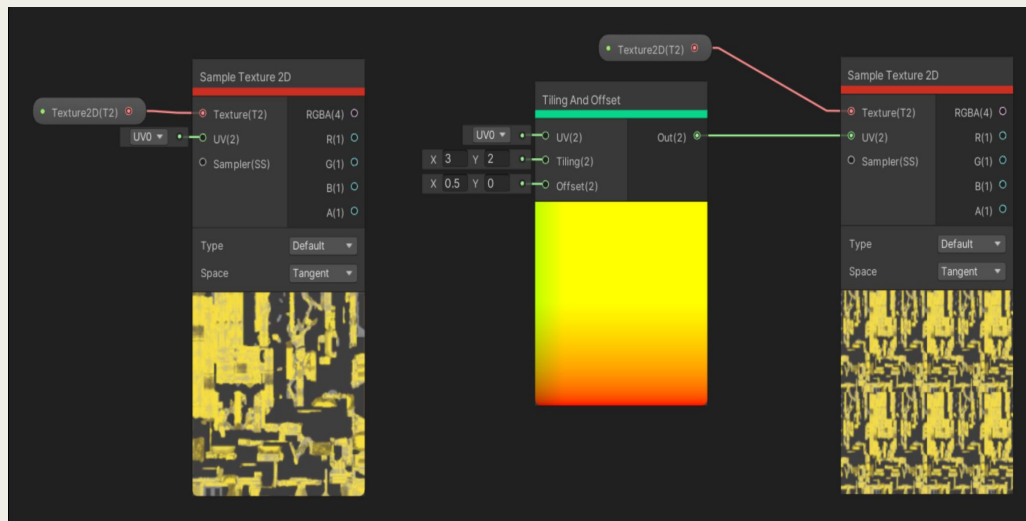
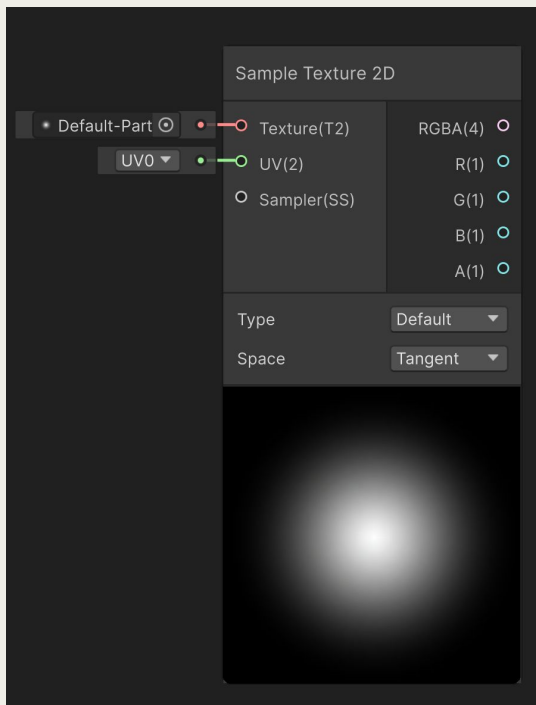


## Position

Gets vertex Position

# Texture Nodes

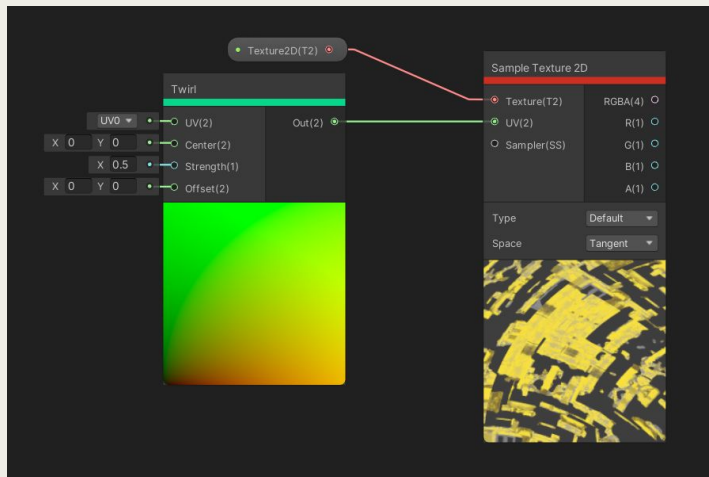
## Sample Texture 2D



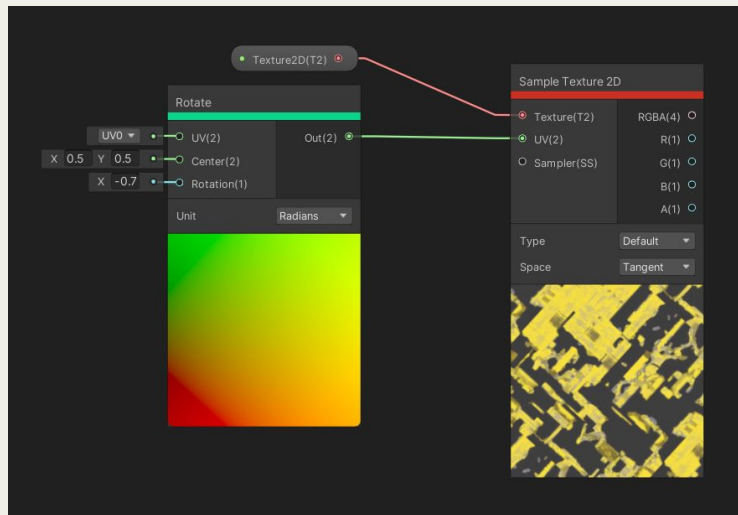
## Tiling and Offsets

Create a tilting and offset to the UV coordinates (greater for creating scrolling animations via textures)

# Texture Nodes

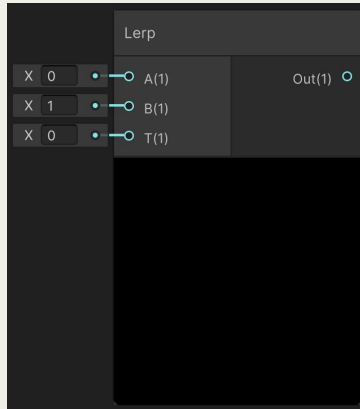


Twirl



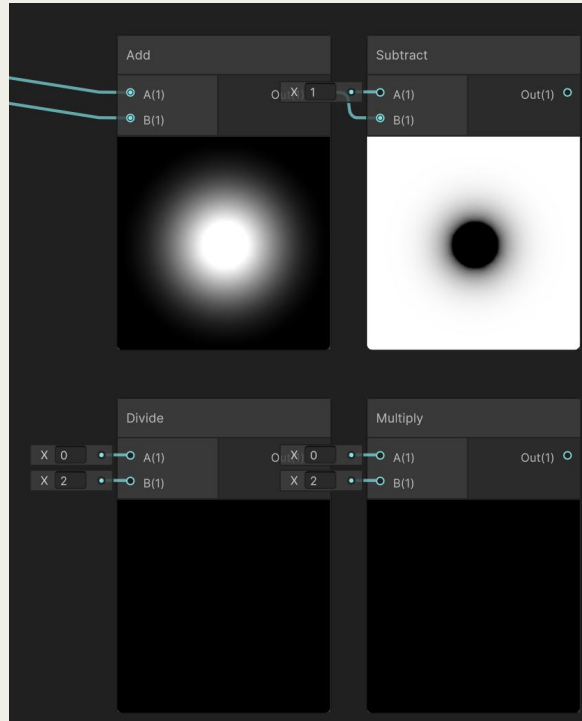
Rotate

# Math Nodes



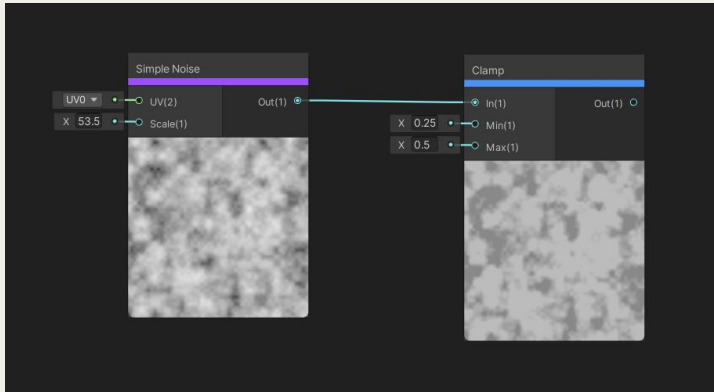
## Lerp

Linear interpolation  
between two object



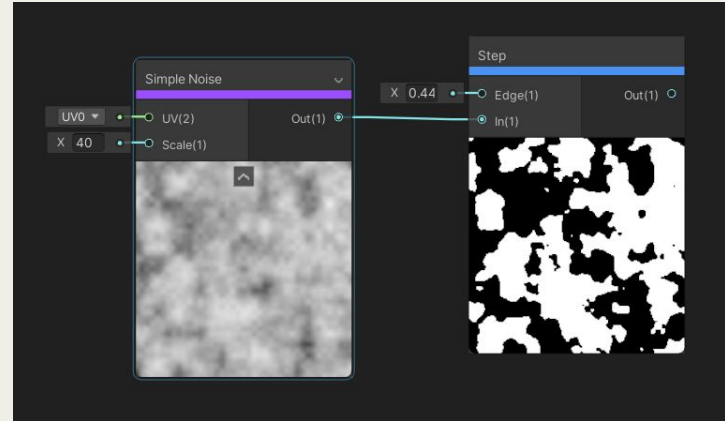
## Arithmetic operations

# Math Nodes



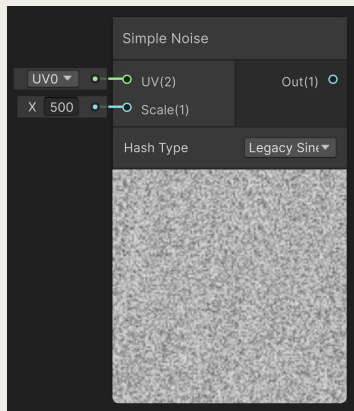
## Clamp

Clamp a value in range

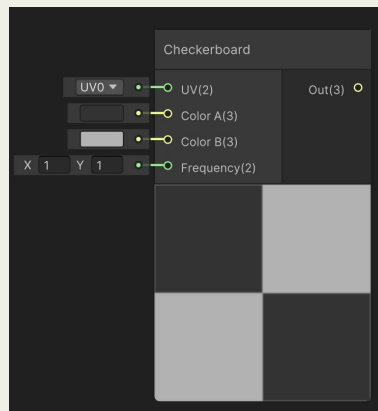


## Step (thresholding)

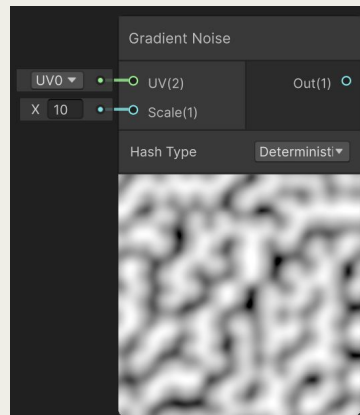
# Noise



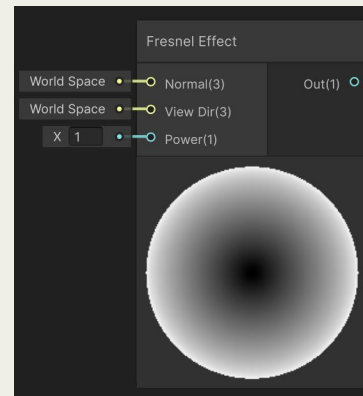
Simple Noise



Checkerboard



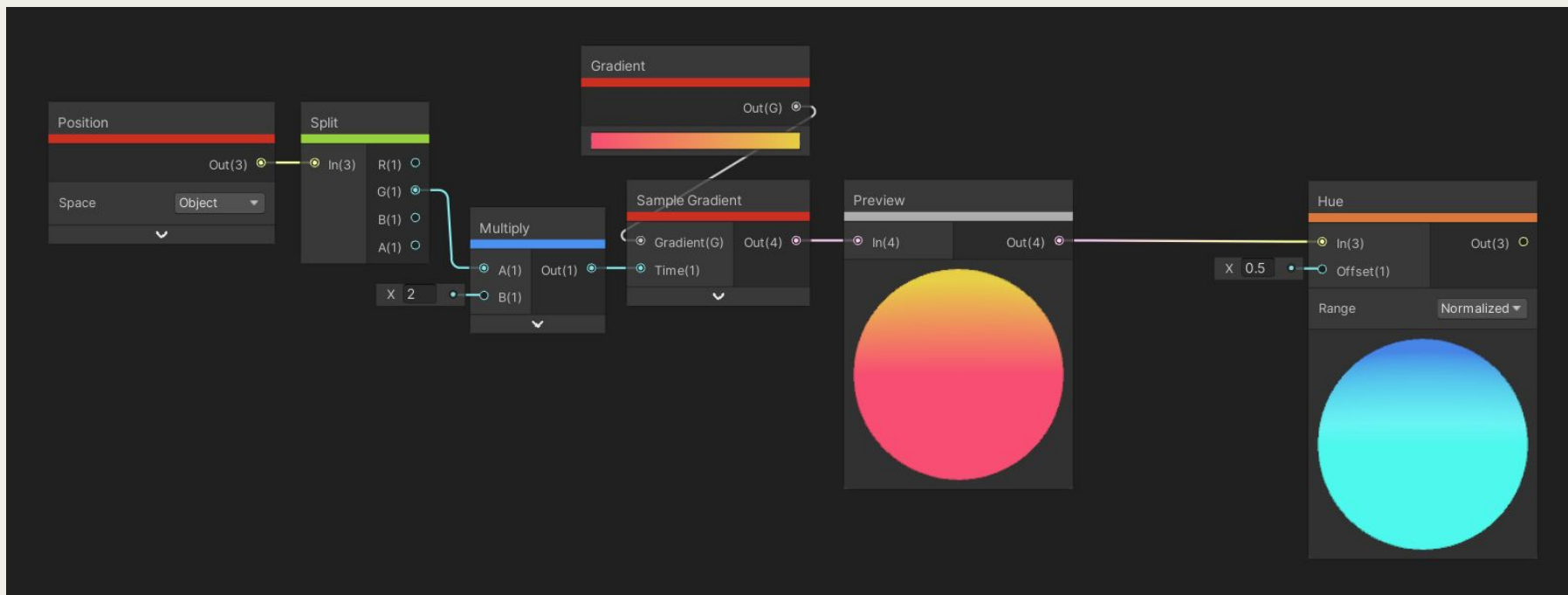
Gradient



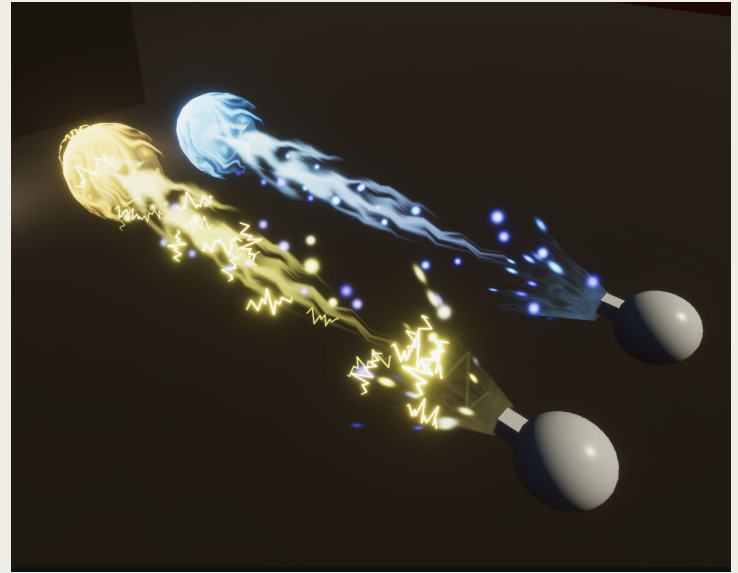
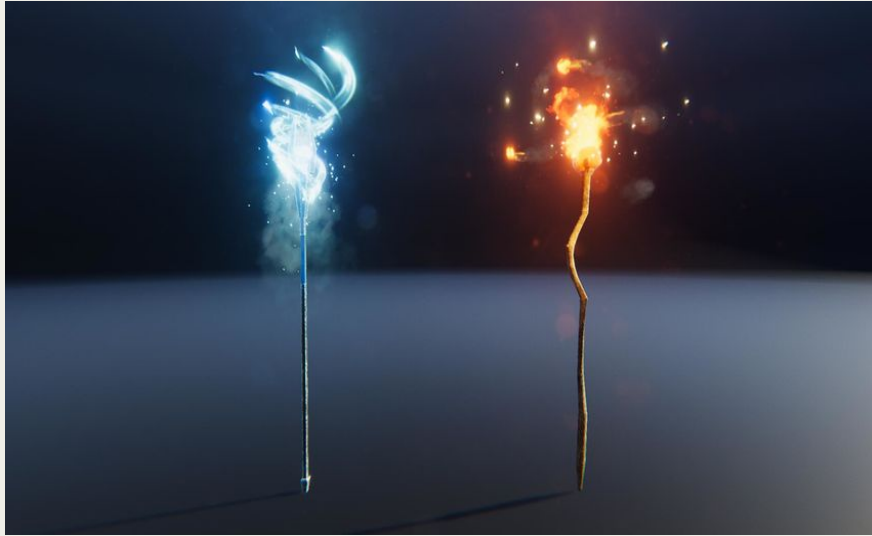
Fresnel Effect

# Artistic Nodes (photoshop editing)

- Contrast, hue, saturation, blend modes, etc.

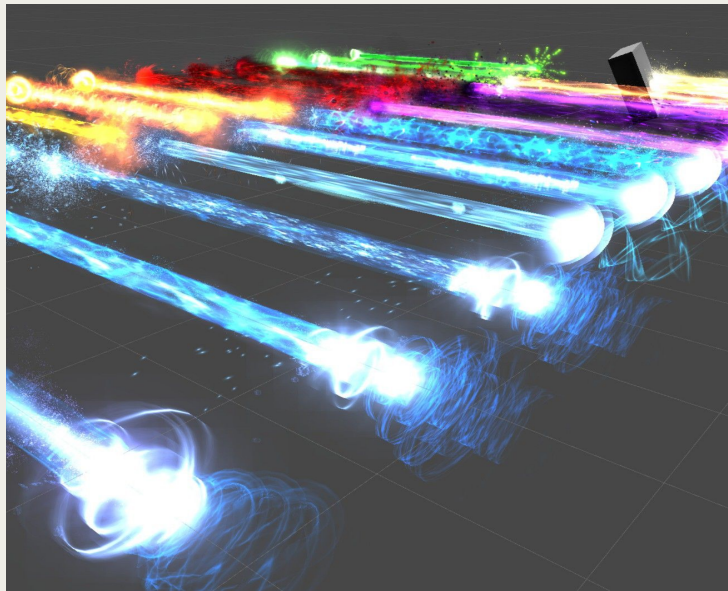


Now that we are done with shader, we have particles

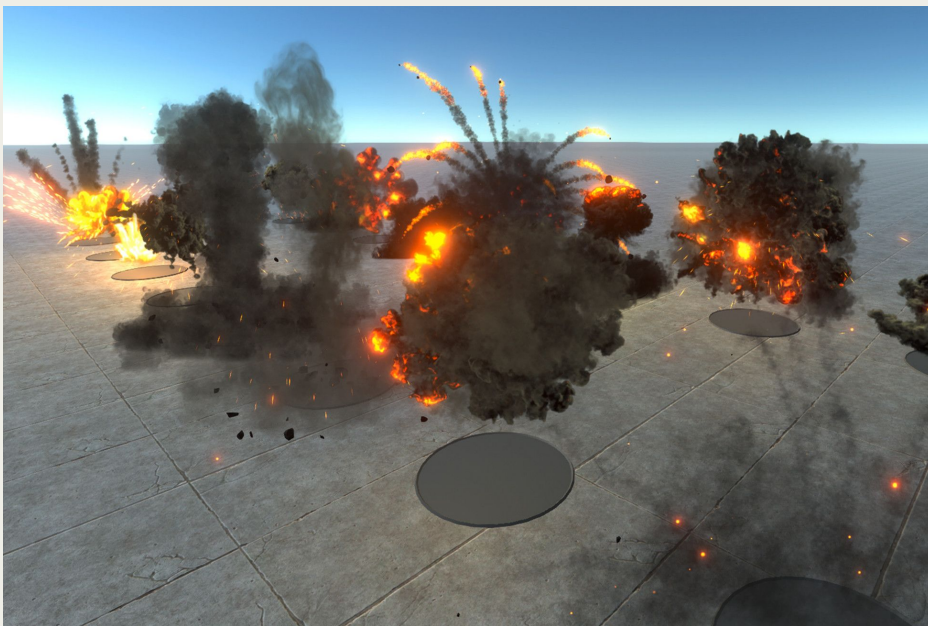
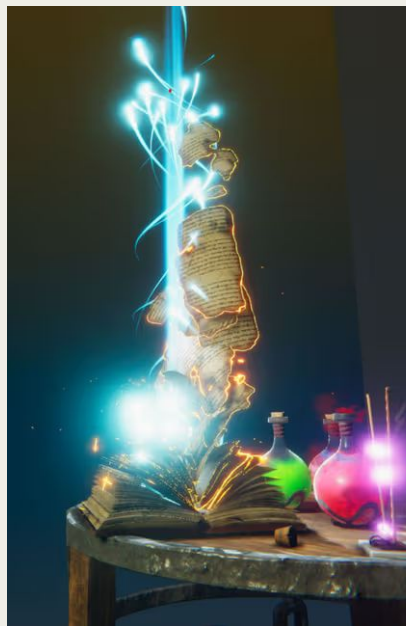


# Visual Effects

basically just particle system + shaders

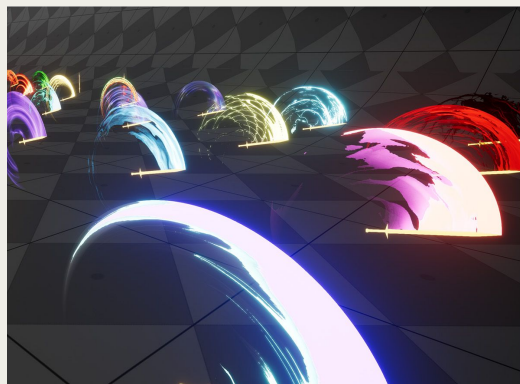
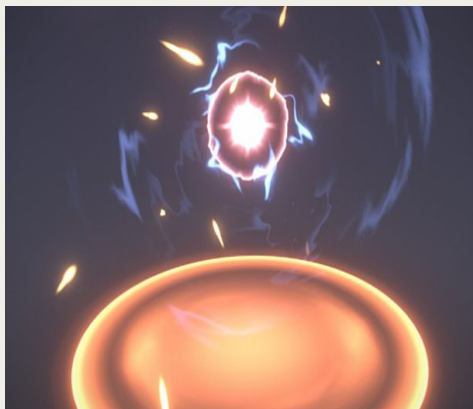


# More Particles



# More shaders but still using a particle system

lasers/ trail effects, special things



Explosion on Enemies



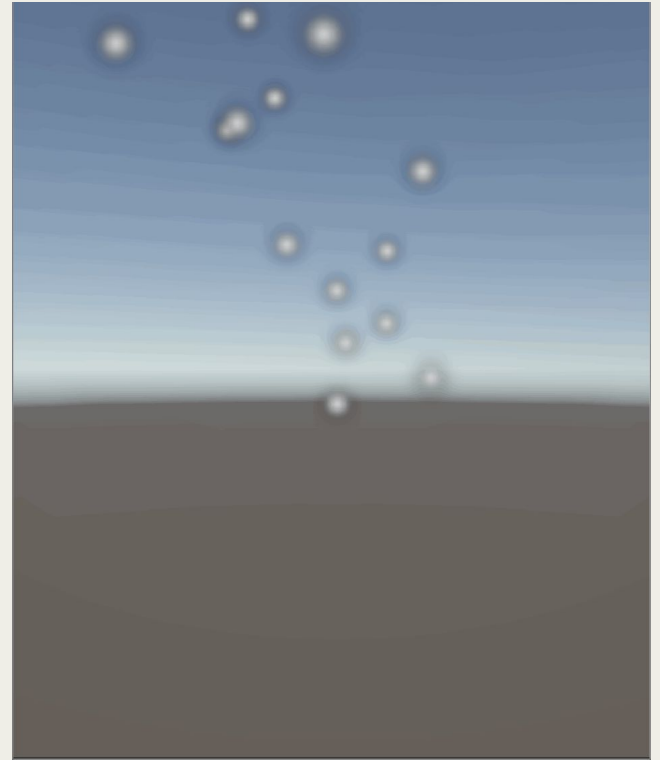
# What is a particle system?

## Emitters

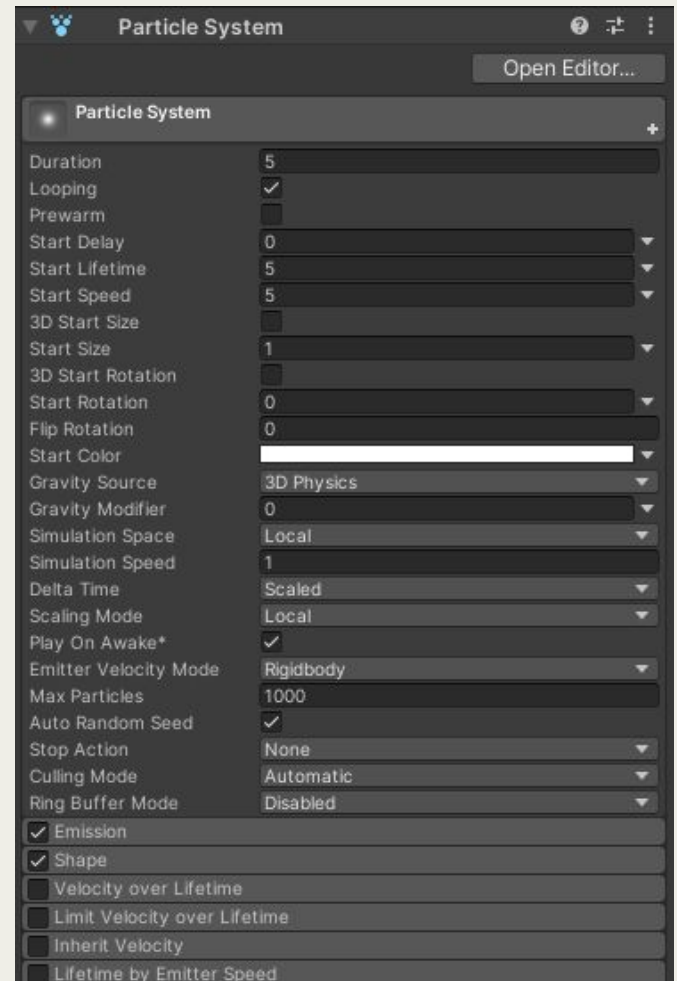
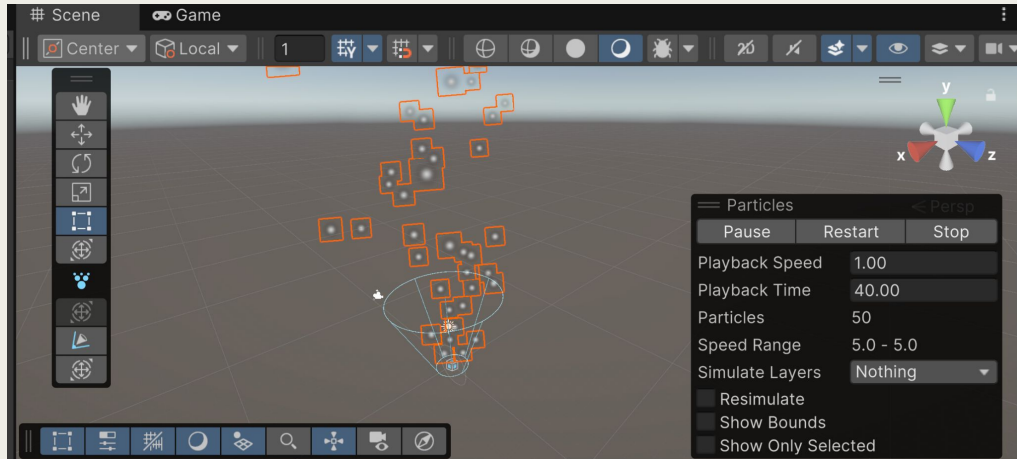
- **Constant Spawn Rate** - Issues Spawn events at a constant rate
- **Burst** - Generates a large number of particles at once
- **Capacity** - Maximum number of particles that can exist simultaneously

## Particles - mesh that you instantiate

- **Set LifeTime** - Particle lifetime
- **Set Position Shape** - Initial particle position
- **Set Velocity** - Initial particle velocity



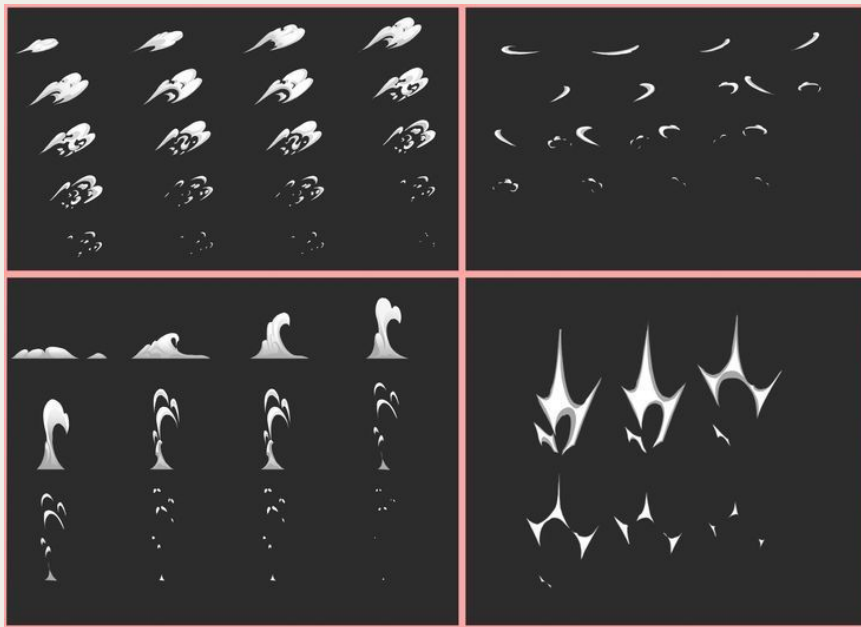
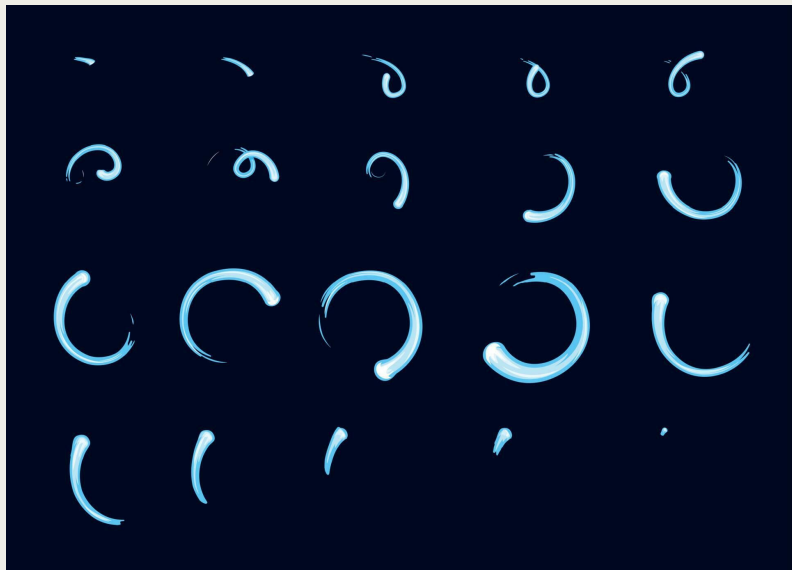
# Properties + Modules



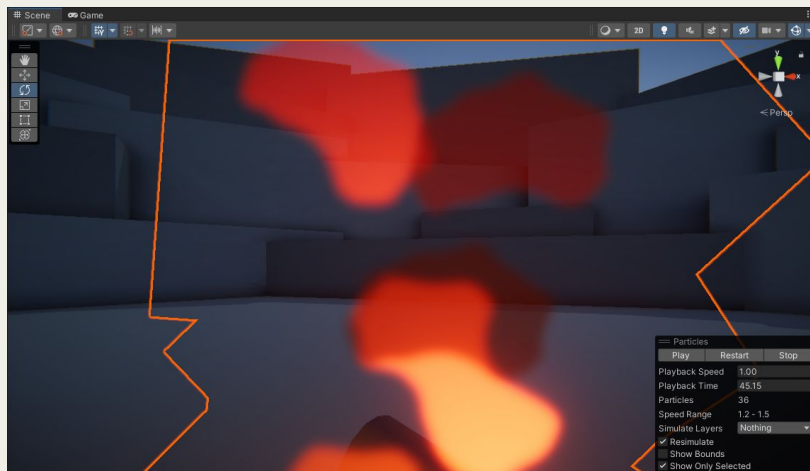
# Flip books for 2D

<https://cgheven.com/assets/flipbooks>

Free VFX image sequences and flipbooks



# Layering Textures



# Examples



# Scripting for particle system

```
using UnityEngine;
using UnityEngine.InputSystem;

@ Unity Script | 0 references
public class hello : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    private ParticleSystem ps;

    @ Unity Message | 0 references
    void Start()
    {
        ps = GetComponent<ParticleSystem>();
    }

    @ Unity Message | 0 references
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            ToggleEmission();
        }
    }

    1 reference
    void ToggleEmission()
    {
        // Get the module
        var emission = ps.emission;
        // Modify it
        emission.enabled = !emission.enabled;
    }
}
```

# Helpful Resources

- [Inigo Quilez :: computer graphics, maths, shaders, fractals, demoscene](#)
- <https://www.shadertoy.com/>
- <https://thebookofshaders.com/>