Introduction to Compute Shaders

Francis Joseph Serina

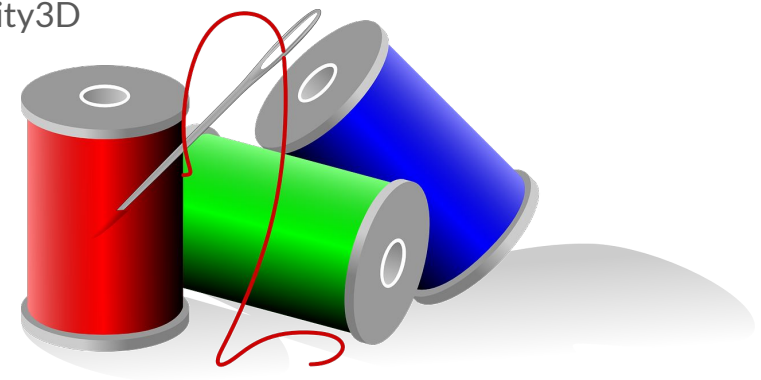
About me

Former high school computer science teacher

Unity3D Developer

Goals

- Difference between CPU and GPU Threads
- Graphics Pipeline and Shaders
- When to use Compute Shaders
- Overview on how to use Compute Shaders in Unity3D



Imagine this...

Imagine this...

Imagine this...

Imagine this...

Programmers call this **Parallel Processing**

Parallel Processing

Concurrency is when 2 or more activities are happening simultaneously

Parallel Processing - Purpose

- Separation of Concerns

Parallel Processing - Purpose

- Separation of Concerns
- Performance

Parallel Processing - Purpose

- Separation of Concerns
- Performance
  - Task Parallelism

Parallel Processing - Purpose

- Separation of Concerns
- Performance
  - Task Parallelism
  - Data Parallelism

Parallel Processing - Purpose

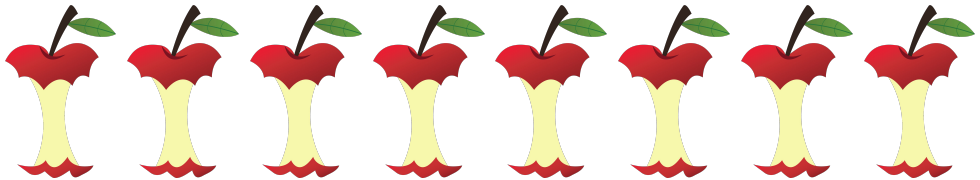
- Separation of Concerns
- Performance *<- this is what Shaders are meant for*
  - Task Parallelism
  - Data Parallelism

What are Shaders?

Programs running on the GPU instead of the CPU

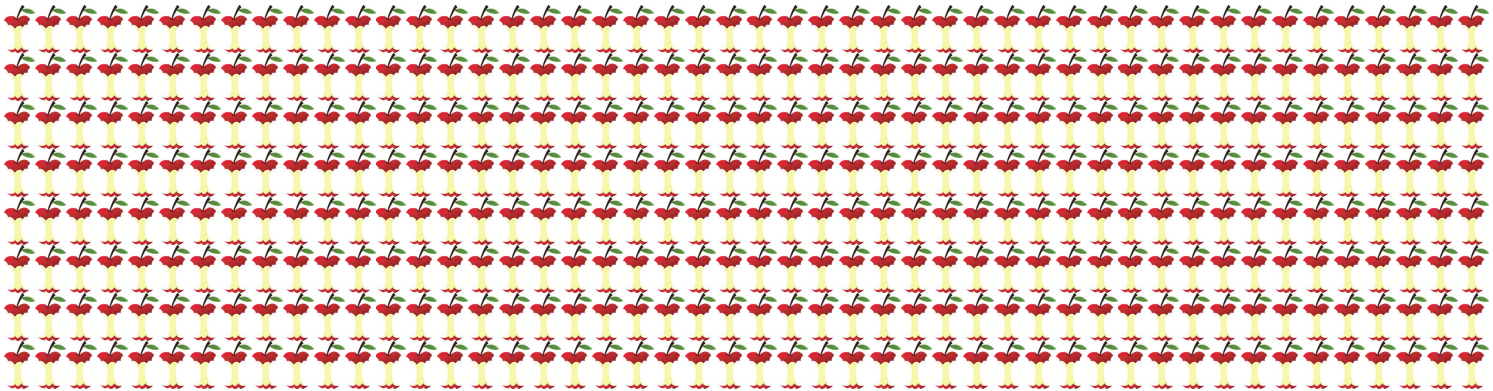
What are Shaders?

Programs running on the GPU instead of the CPU



What are Shaders?

Programs running on the GPU instead of the CPU



What are Shaders?

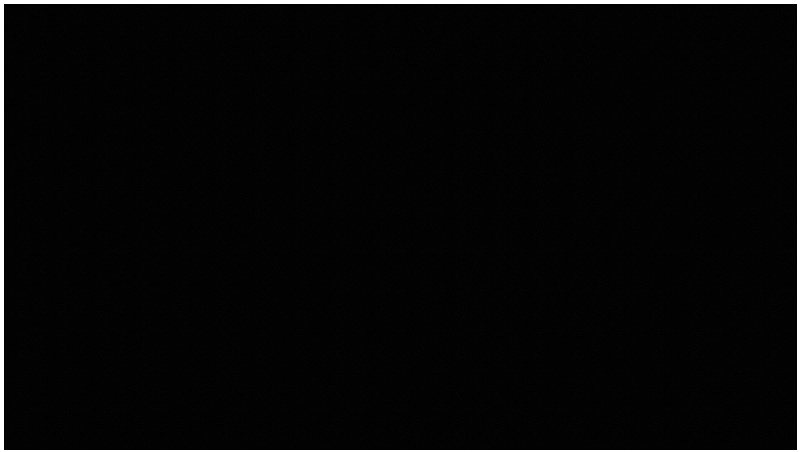
Originally intended for Graphics

What are Compute Shaders?

Programs run on the GPU but not necessarily for Graphics

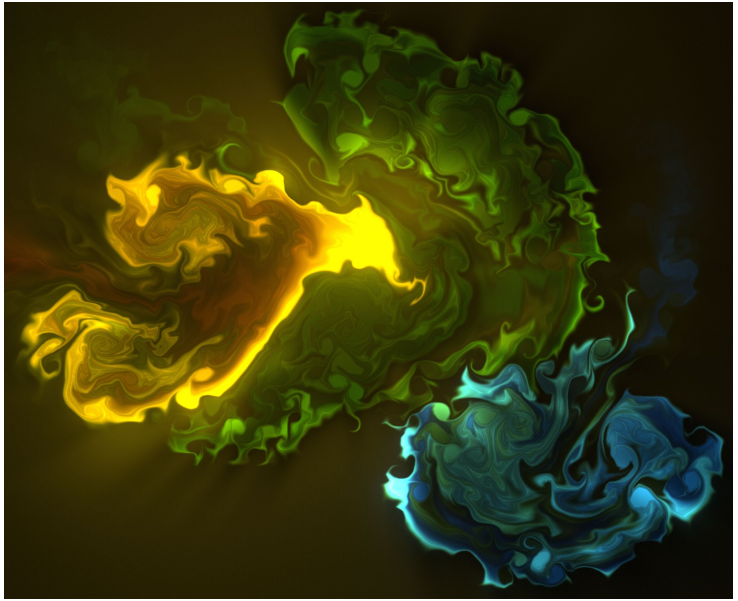
Sample Applications

WebGL Fluid Simulation(github.io)



Sample Applications

WebGL Fluid Simulation(github.io)



Sample Applications

Conway's Game of Life

Sample Applications

Computer Vision

- Convolutions
- Fast Fourier Transform
- Feature Detection
- Post-processing Image Effects

Simulations

- Weathering and Erosion (Terrain Generation)
- Particle Systems
- AI Pathfinding
- Diffusion and Propagation

Sample Applications

Embarrassingly Parallel or Pleasantly Parallel or Conveniently Concurrent

- Little to no effort to divide the problem into parallel tasks
- Highly scalable algorithms

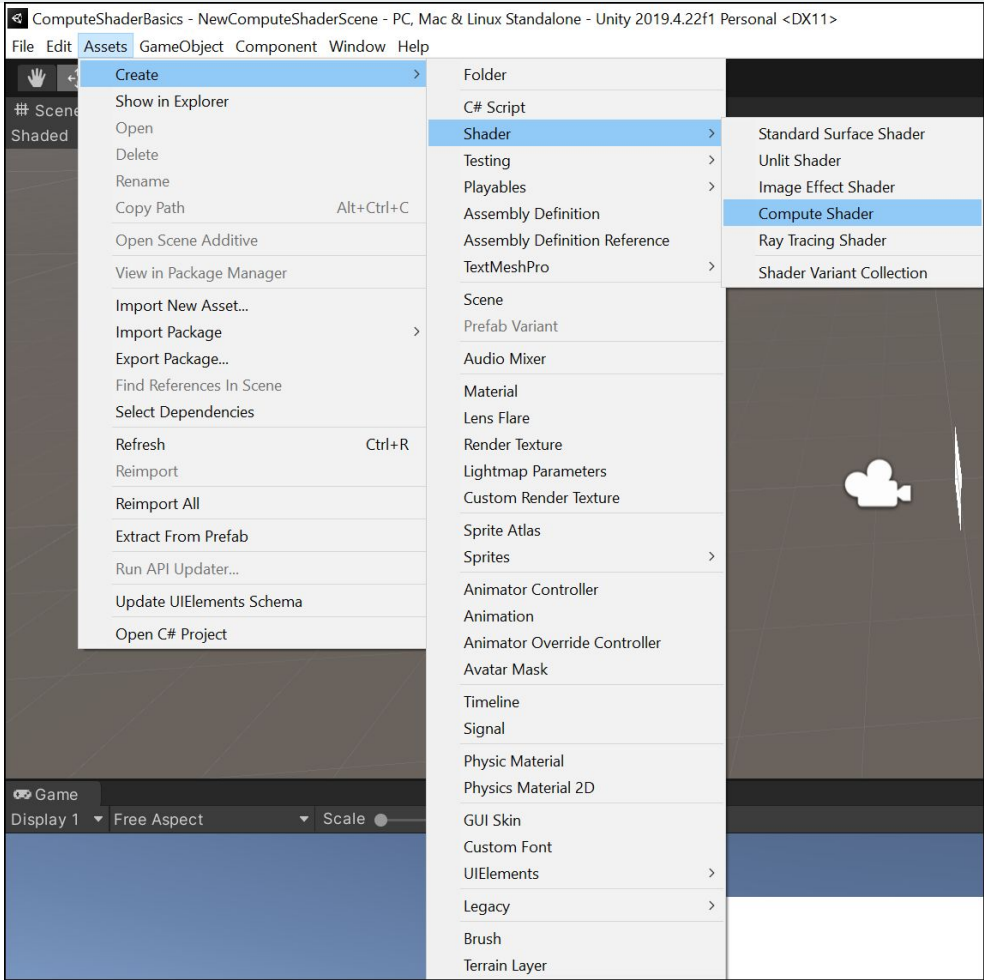
Quick Guide:
Compute Shaders in Unity3D

Overview

- Create a compute shader
- Create a MonoBehaviour to use the compute shader
- Set up the scene to show the output

Creating a new Compute Shader

- Assets Menu
- Create
- Shader
- Compute Shader



NewComputeShader.compute

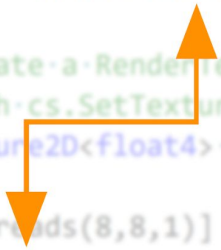
```
1   //·Each·#kernel·tells·which·function·to·compile;·you·can·have·many·kernels
2   #pragma·kernel·CSMain
3
4   //·Create·a·RenderTexture·with·enableRandomWrite·flag·and·set·it
5   //·with·cs.SetTexture
6   RWTexture2D<float4>·Result;
7
8   [numthreads(8,8,1)]
9   void·CSMain·(uint3·id·:·SV_DispatchThreadID)
10  {
11  ····//·TODO:·insert·actual·code·here!
12
13  ····Result[id.xy]·=·float4(id.x·&·id.y,·(id.x·&·15)/15.0,·(id.y·&·15)/15.0,·0.0);
14  }
```

NewComputeShader.compute

```
1   // Each #kernel tells which function to compile; you can have many kernels
2   #pragma kernel CSMain
3
4   // Create a RenderTexture with enableRandomWrite flag and set it
5   // with cs.SetTexture
6   RWTexture2D<float4> Result;
7
8   [numthreads(8,8,1)]
9   void CSMain (uint3 id : SV_DispatchThreadID)
10  {
11      // TODO: insert actual code here!
12
13      Result[id.xy] = float4(id.x & id.y, (id.x & 15)/15.0, (id.y & 15)/15.0, 0.0);
14  }
```

NewComputeShader.compute

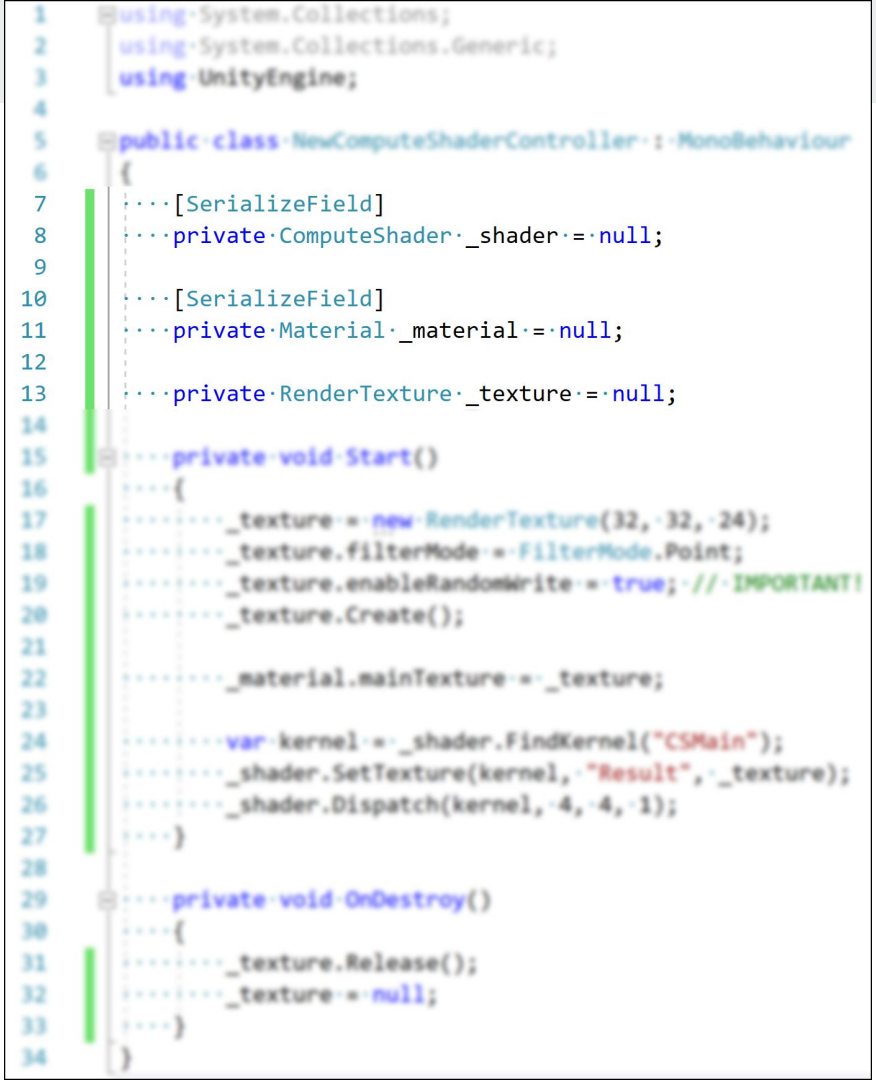
```
1   //·Each·#kernel·tells·which·function·to·compile;·you·can·have·many·kernels
2   #pragma·kernel·CSMain
3
4   //·Create·a·RenderTexture·with·enableRandomWrite·flag·and·set·it
5   //·with·cs.SetTexture
6   RWTexture2D<float4>·Result;
7
8   [numthreads(8,8,1)]
9   void·CSMain·(uint3·id·:·SV_DispatchThreadID)
10  {
11  ····//·TODO:·insert·actual·code·here!
12
13  ····Result[id.xy]·=·float4(id.x·&·id.y,·(id.x·&·15)/15.0,·(id.y·&·15)/15.0,·0.0);
14  }
```



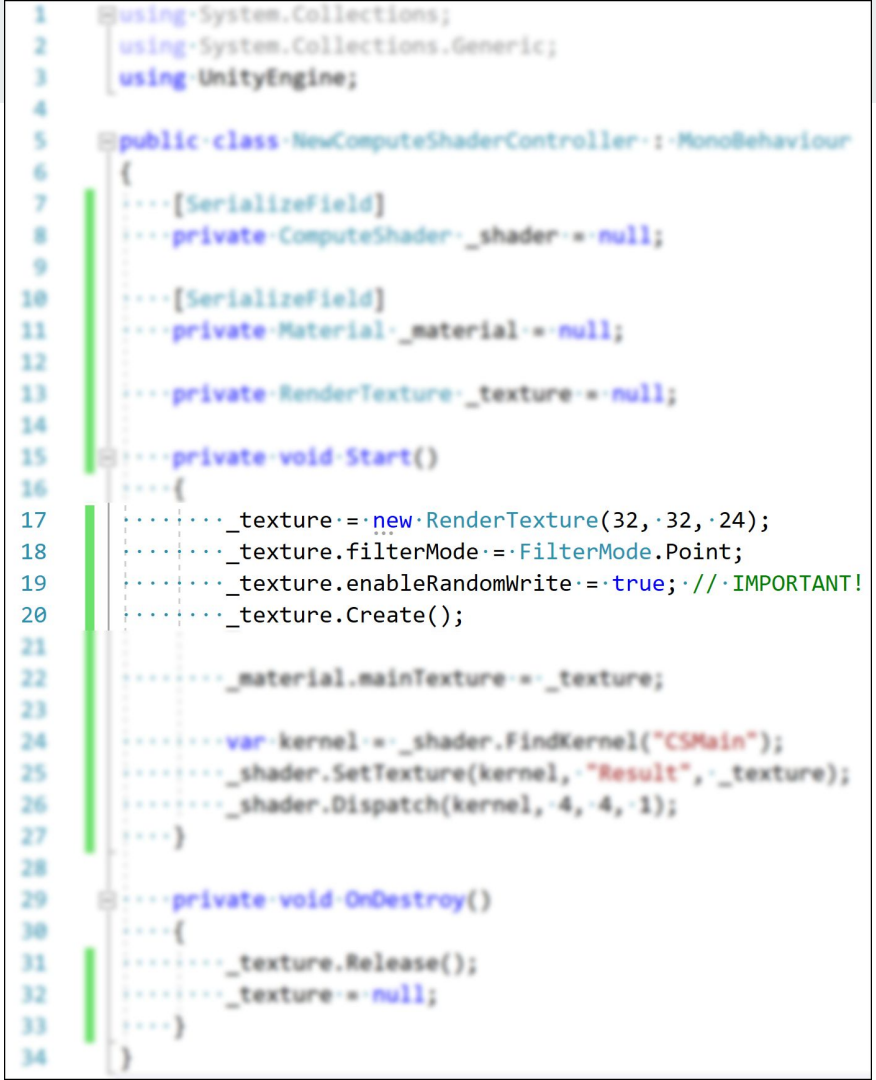
NewComputeShader.compute

```
1   // Each #kernel tells which function to compile; you can have many kernels
2   #pragma kernel CSMain
3
4   // Create a RenderTexture with enableRandomWrite flag and set it
5   // with cs.SetTexture
6   RWTexture2D<float4> Result;
7
8   [numthreads(8,8,1)]
9   void CSMain (uint3 id : SV_DispatchThreadID)
10  {
11      // TODO: insert actual code here!
12
13      Result[id.xy] = float4(id.x & id.y, (id.x & 15)/15.0, (id.y & 15)/15.0, 0.0);
14  }
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewComputeShaderController : MonoBehaviour
{
    [SerializeField]
    private ComputeShader _shader = null;

    [SerializeField]
    private Material _material = null;

    private RenderTexture _texture = null;

    private void Start()
    {
        _texture = new RenderTexture(32, 32, 24);
        _texture.filterMode = FilterMode.Point;
        _texture.enableRandomWrite = true; // IMPORTANT!
        _texture.Create();

        _material.mainTexture = _texture;

        var kernel = _shader.FindKernel("CSMain");
        _shader.SetTexture(kernel, "Result", _texture);
        _shader.Dispatch(kernel, 4, 4, 1);
    }

    private void OnDestroy()
    {
        _texture.Release();
        _texture = null;
    }
}
```

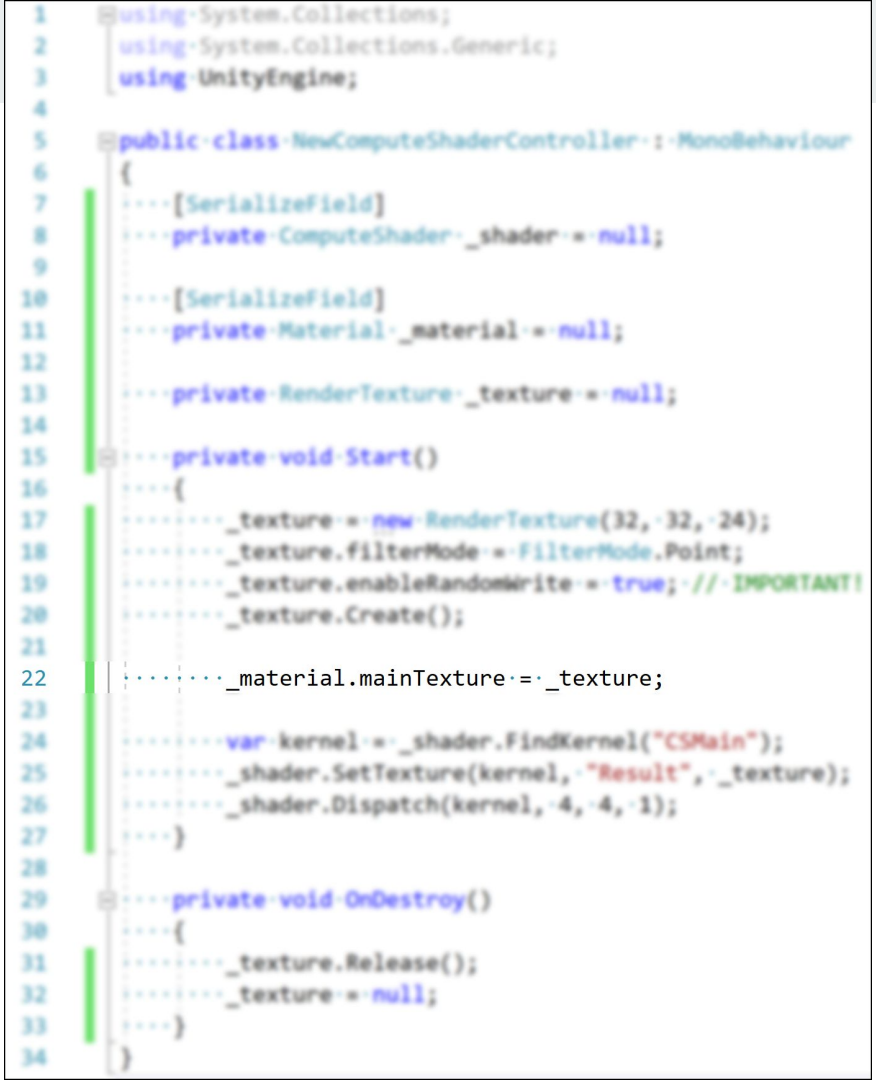


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewComputeShaderController : MonoBehaviour
{
    [SerializeField]
    private ComputeShader _shader = null;

    [SerializeField]
    private Material _material = null;

    private RenderTexture _texture = null;

    private void Start()
    {
        _texture = new RenderTexture(32, 32, 24);
        _texture.filterMode = FilterMode.Point;
        _texture.enableRandomWrite = true; // IMPORTANT!
        _texture.Create();

        _material.mainTexture = _texture;

        var kernel = _shader.FindKernel("CSMain");
        _shader.SetTexture(kernel, "Result", _texture);
        _shader.Dispatch(kernel, 4, 4, 1);
    }

    private void OnDestroy()
    {
        _texture.Release();
        _texture = null;
    }
}
```

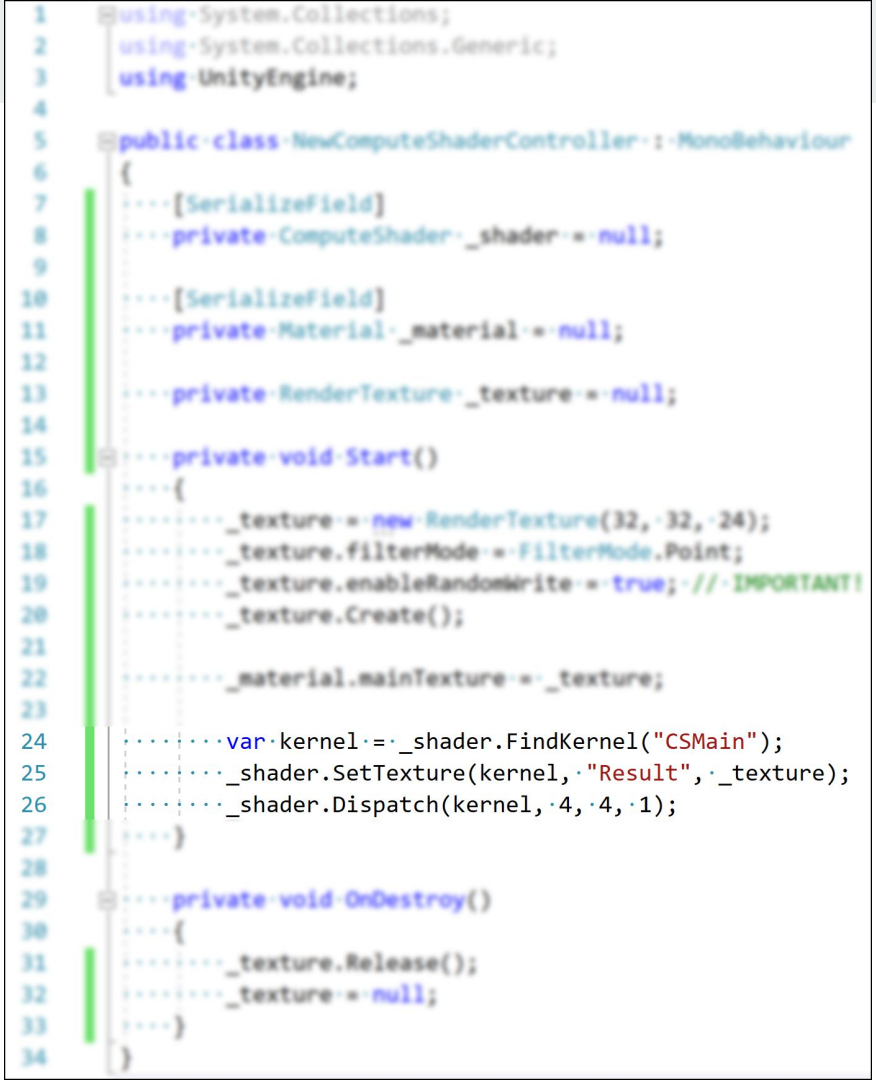


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewComputeShaderController : MonoBehaviour
{
    [SerializeField]
    private ComputeShader _shader = null;

    [SerializeField]
    private Material _material = null;

    private RenderTexture _texture = null;

    private void Start()
    {
        _texture = new RenderTexture(32, 32, 24);
        _texture.filterMode = FilterMode.Point;
        _texture.enableRandomWrite = true; // IMPORTANT!
        _texture.Create();

        _material.mainTexture = _texture;

        var kernel = _shader.FindKernel("CSMain");
        _shader.SetTexture(kernel, "Result", _texture);
        _shader.Dispatch(kernel, 4, 4, 1);
    }

    private void OnDestroy()
    {
        _texture.Release();
        _texture = null;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewComputeShaderController : MonoBehaviour
{
    [SerializeField]
    private ComputeShader _shader = null;

    [SerializeField]
    private Material _material = null;

    private RenderTexture _texture = null;

    private void Start()
    {
        _texture = new RenderTexture(32, 32, 24);
        _texture.filterMode = FilterMode.Point;
        _texture.enableRandomWrite = true; // IMPORTANT!
        _texture.Create();

        _material.mainTexture = _texture;

        var kernel = _shader.FindKernel("CSMain");
        _shader.SetTexture(kernel, "Result", _texture);
        _shader.Dispatch(kernel, 4, 4, 1);
    }

    private void OnDestroy()
    {
        _texture.Release();
        _texture = null;
    }
}
```



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewComputeShaderController : MonoBehaviour
{
    [SerializeField]
    private ComputeShader _shader = null;

    [SerializeField]
    private Material _material = null;

    private RenderTexture _texture = null;

    private void Start()
    {
        _texture = new RenderTexture(32, 32, 24);
        _texture.filterMode = FilterMode.Point;
        _texture.enableRandomWrite = true; // IMPORTANT!
        _texture.Create();

        _material.mainTexture = _texture;

        var kernel = _shader.FindKernel("CSMain");
        _shader.SetTexture(kernel, "Result", _texture);
        _shader.Dispatch(kernel, 4, 4, 1);
    }

    private void OnDestroy()
    {
        _texture.Release();
        _texture = null;
    }
}
```



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewComputeShaderController : MonoBehaviour
{
    [SerializeField]
    private ComputeShader _shader = null;

    [SerializeField]
    private Material _material = null;

    private RenderTexture _texture = null;

    private void Start()
    {
        _texture = new RenderTexture(32, 32, 24);
        _texture.filterMode = FilterMode.Point;
        _texture.enableRandomWrite = true; // IMPORTANT!
        _texture.Create();

        _material.mainTexture = _texture;

        var kernel = _shader.FindKernel("CSMain");
        _shader.SetTexture(kernel, "Result", _texture);
        _shader.Dispatch(kernel, 4, 4, 1);
    }

    private void OnDestroy()
    {
        _texture.Release();
        _texture = null;
    }
}
```

NewComputeShaderController.cs

```
var kernel = _shader.FindKernel("CSMain");
```

- Remember the `#pragma kernel` before?

NewComputeShaderController.cs

```
var kernel = _shader.FindKernel("CSMain");
```

- Remember the `#pragma kernel` before?

```
_shader.SetTexture(kernel, "Result", _texture);
```

- Set the texture to the buffer

NewComputeShaderController.cs

```
var kernel = _shader.FindKernel("CSMain");
```

- Remember the `#pragma kernel` before?

```
_shader.SetTexture(kernel, "Result", _texture);
```

- Set the texture to the buffer

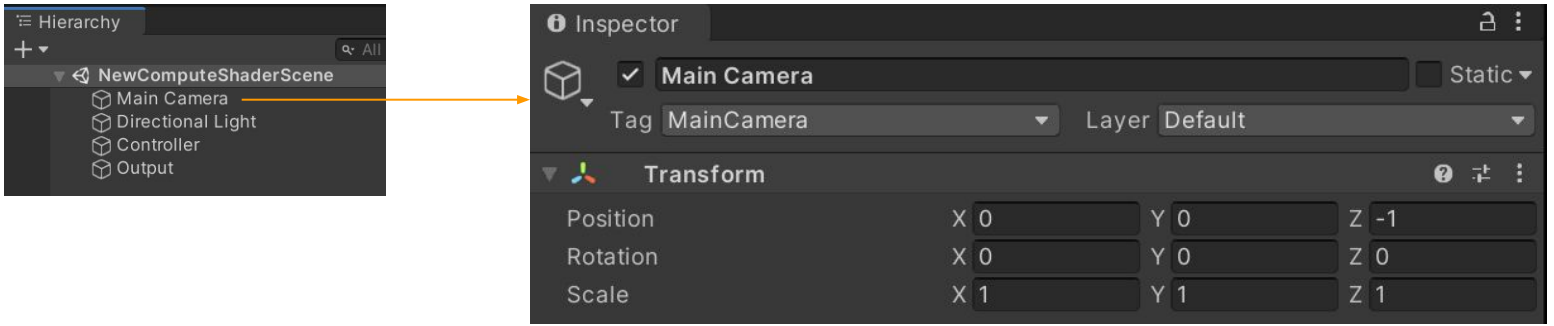
```
_shader.Dispatch(kernel, 4, 4, 1);
```

- Execute the kernel with the number of **thread groups**
- Each **thread group has 64 threads** from `[numthreads(8,8,1)]`

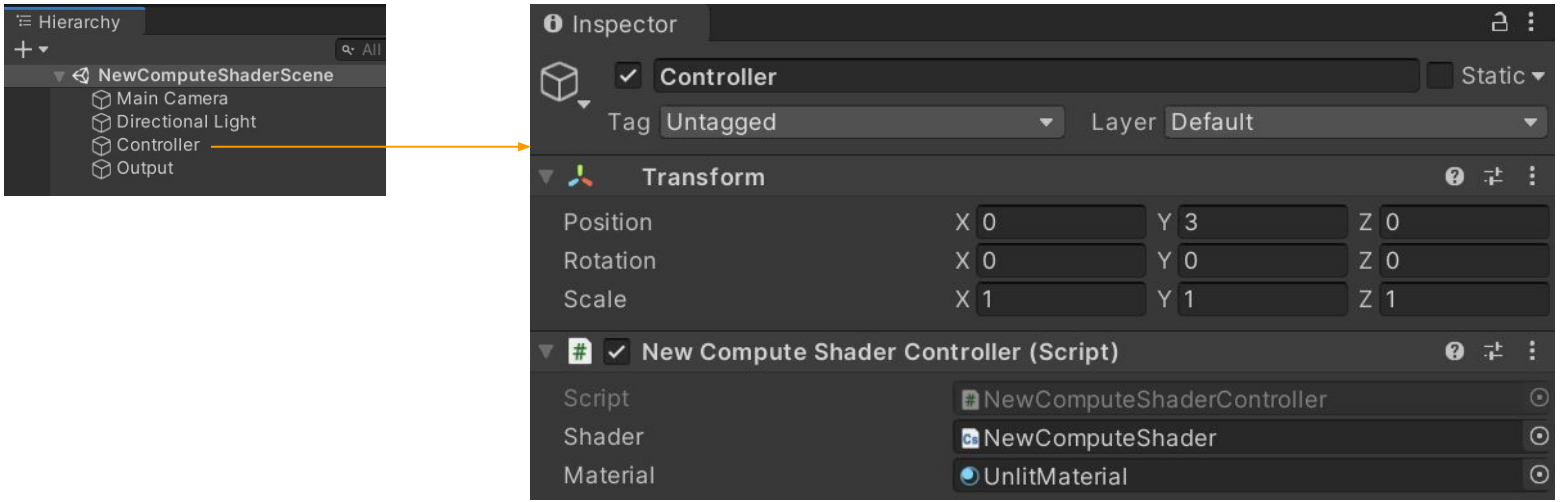
Scene Setup


Inspector
UnlitMaterial
Shader
Unlit/Texture
Base (RGB)
None
(Texture)
Tiling
X 1
Y 1
Offset
X 0
Y 0
Select
Render Queue
From Shader
2000
Double Sided Global Illumination

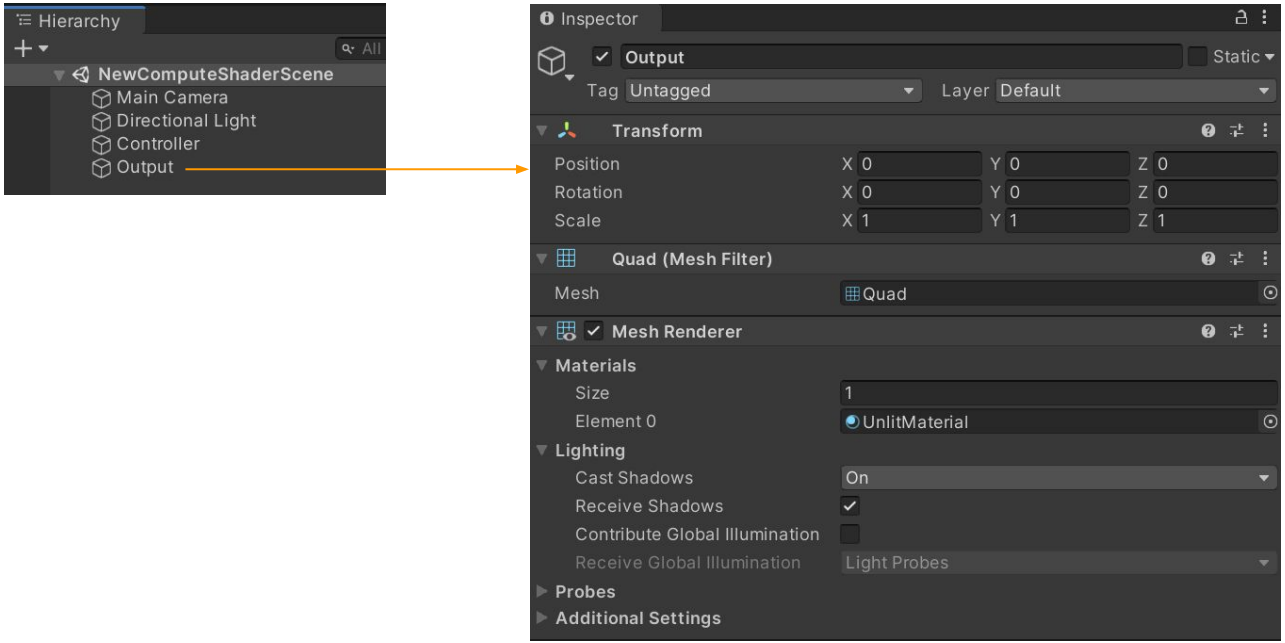
Scene Setup


Hierarchy
All
NewComputeShaderScene
Main Camera
Directional Light
Controller
Output
Inspector
Main Camera
Static
Tag
MainCamera
Layer
Default
Transform
Position
X 0
Y 0
Z -1
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

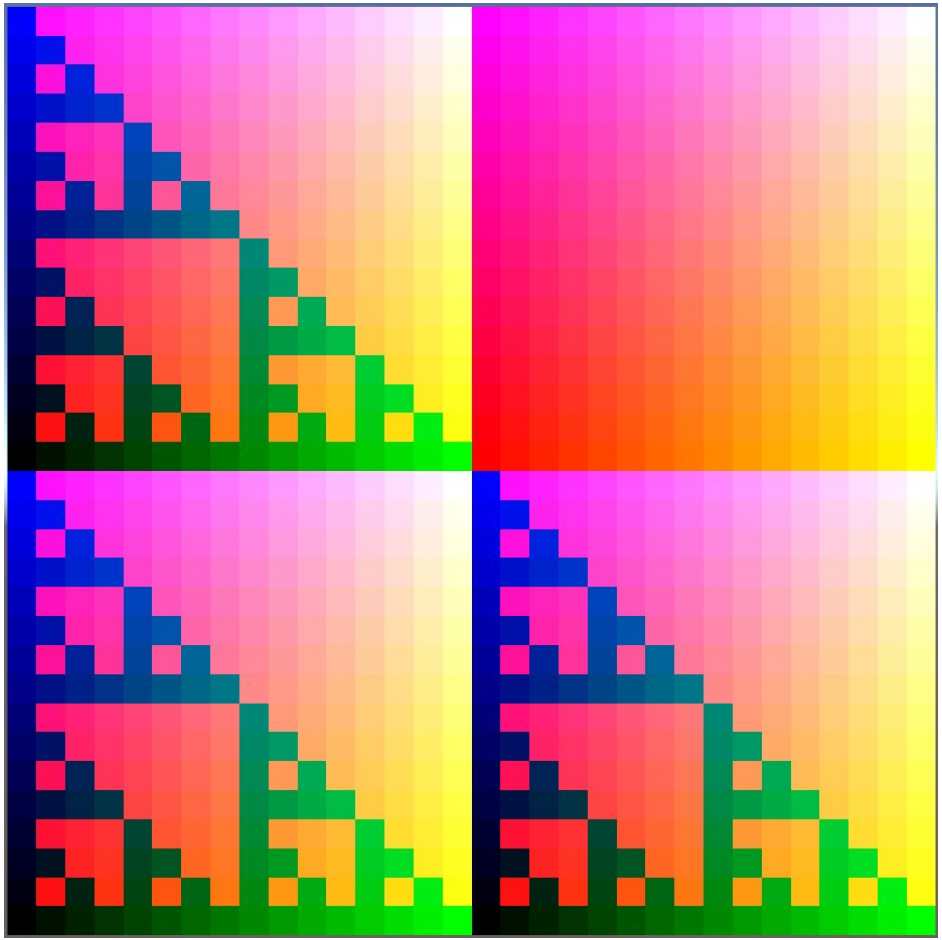
Scene Setup


Hierarchy
All
NewComputeShaderScene
Main Camera
Directional Light
Controller
Output
Inspector
Controller
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 3
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
New Compute Shader Controller (Script)
Script
NewComputeShaderController
Shader
NewComputeShader
Material
UnlitMaterial

Scene Setup


Hierarchy
All
NewComputeShaderScene
Main Camera
Directional Light
Controller
Output
Inspector
Output
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Quad (Mesh Filter)
Mesh Quad
Mesh Renderer
Materials
Size 1
Element 0 UnlitMaterial
Lighting
Cast Shadows On
Receive Shadows
Contribute Global Illumination
Receive Global Illumination Light Probes
Probes
Additional Settings

Output



Source Code

- Source Code is at https://github.com/xeratol/ComputeShaderBasics
  - 01-NewComputeShader
    - How to use default compute shader boilerplate code
  - 02-Color
    - give a texture a solid color
  - 03-GrayScale
    - post-processing effect using Compute Shaders
    - Cg Programming/Unity/Computing Image Effects
  - 04-ThreadIDDemo
    - thread ID and other parameters
  - 05-Life
    - Conway's Game of Life

Q&A

https://github.com/xeratol/ComputeShaderBasics

Image Licenses

- Meeting Business Brainstorming - Free vector graphic on Pixabay Pixabay License
- Thread Rolls Needle - Free vector graphic on Pixabay Pixabay License
- Child Joke Childhood Pixabay License
- Steps Running Kids - Free photo on Pixabay Pixabay License
- Free Images : play, color, paint, drawing, mural, finger painting, child art, kid painting, kids art 2352x1568 - - 958981 - Free stock photos CC0 Public Domain
- Clip Art Of A Bitten Apple Bite Clipart Clipartxtras - Snow White Apple Clip Art PNG Image | Transparent PNG Free Download on SeekPNG Personal Use
- File:The OpenGL - DirectX graphics pipeline.png CC BY-SA 3.0
- Conway's Game of Life - Wikipedia CC BY-SA 3.0
- Man Jogging Cartoon.svg from Wikimedia Commons by Videoplasty.com, CC-BY-SA 4.0

References

CPU vs GPU? What's the Difference? Which Is Better?