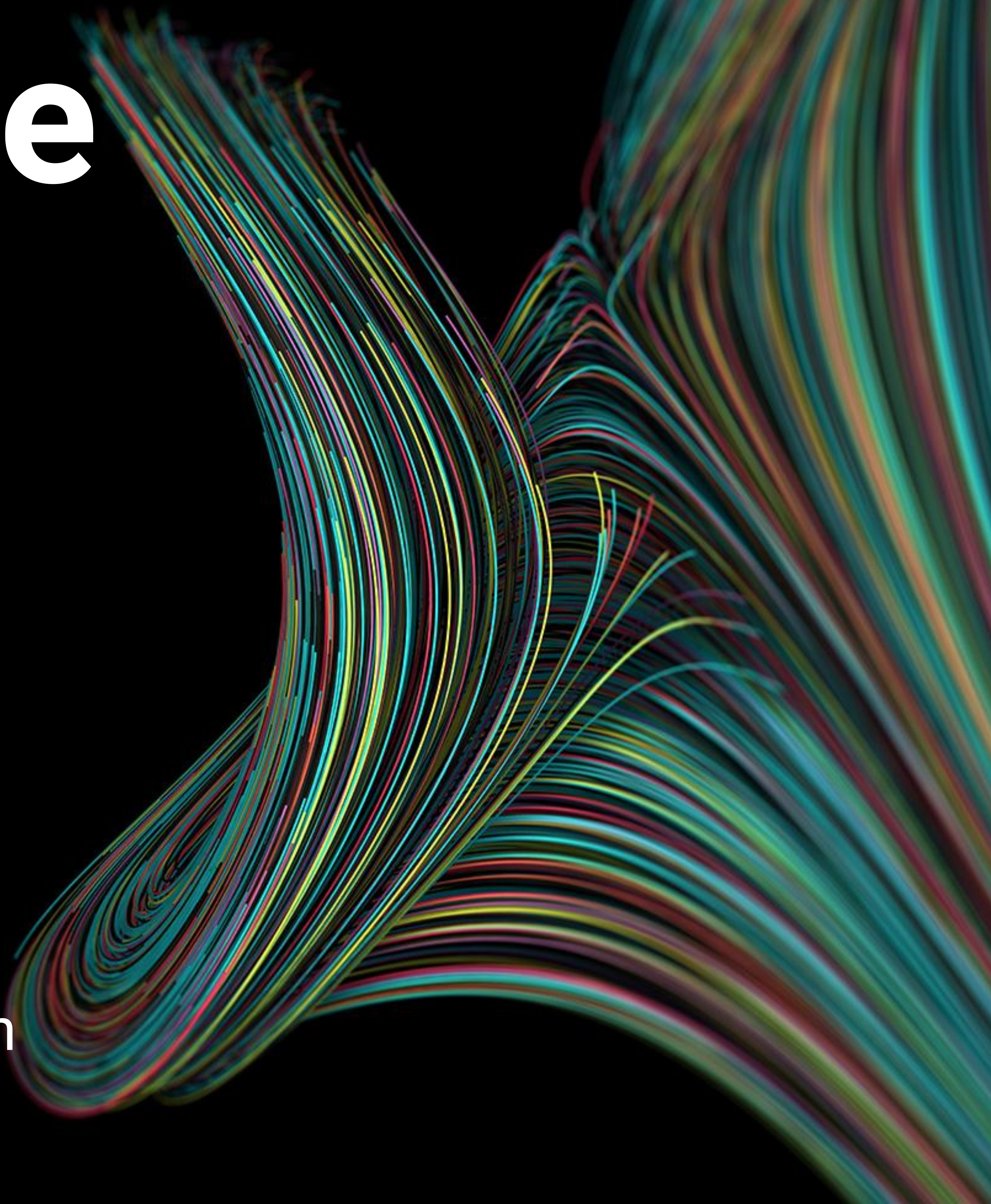
Render Pipeline From Scratch

Peter Bay Bastian

Software Developer **Unity Technologies**

http://github.com/pbbastian/SRPFromScratch



Generative Art – Made with Unity

Before we get started

- Assumptions
  - You have a basic idea of how Unity works
  - You can write C#
  - You've written a basic shader in Unity before
- Goals
  - Create a basic render pipeline
  - Get familiar with the SRP API and the new shader library
- Non-goals
  - Advanced rendering techniques
  - Math

Workshop Format

- Learn
  - I give an overview of what we’re going to do in this part of the workshop

- Watch
  - I code and explain what I’m doing

- Create
  - You write the code
  - Heavily commented solutions available per-part in provided Unity project

Workshop Overview

- Up and running
  - Create the pipeline, asset & clear the screen
- Unlit rendering
  - Render opaque and transparent objects without lighting
- Lit Rendering
  - Modify shaders to support lighting
- Skybox & custom render texture
  - Draw into a custom render texture, copy depth, and draw the sky

Up and running

What even is a Scriptable Render Pipeline?

- Render Pipeline
  - Performs the actual rendering

- Asset
  - Stores settings for your pipeline
  - Stores references to other assets

- Shaders!
  - You’re in control
  - Built-in shaders won’t work anymore
  - Opportunity to use the new shader library

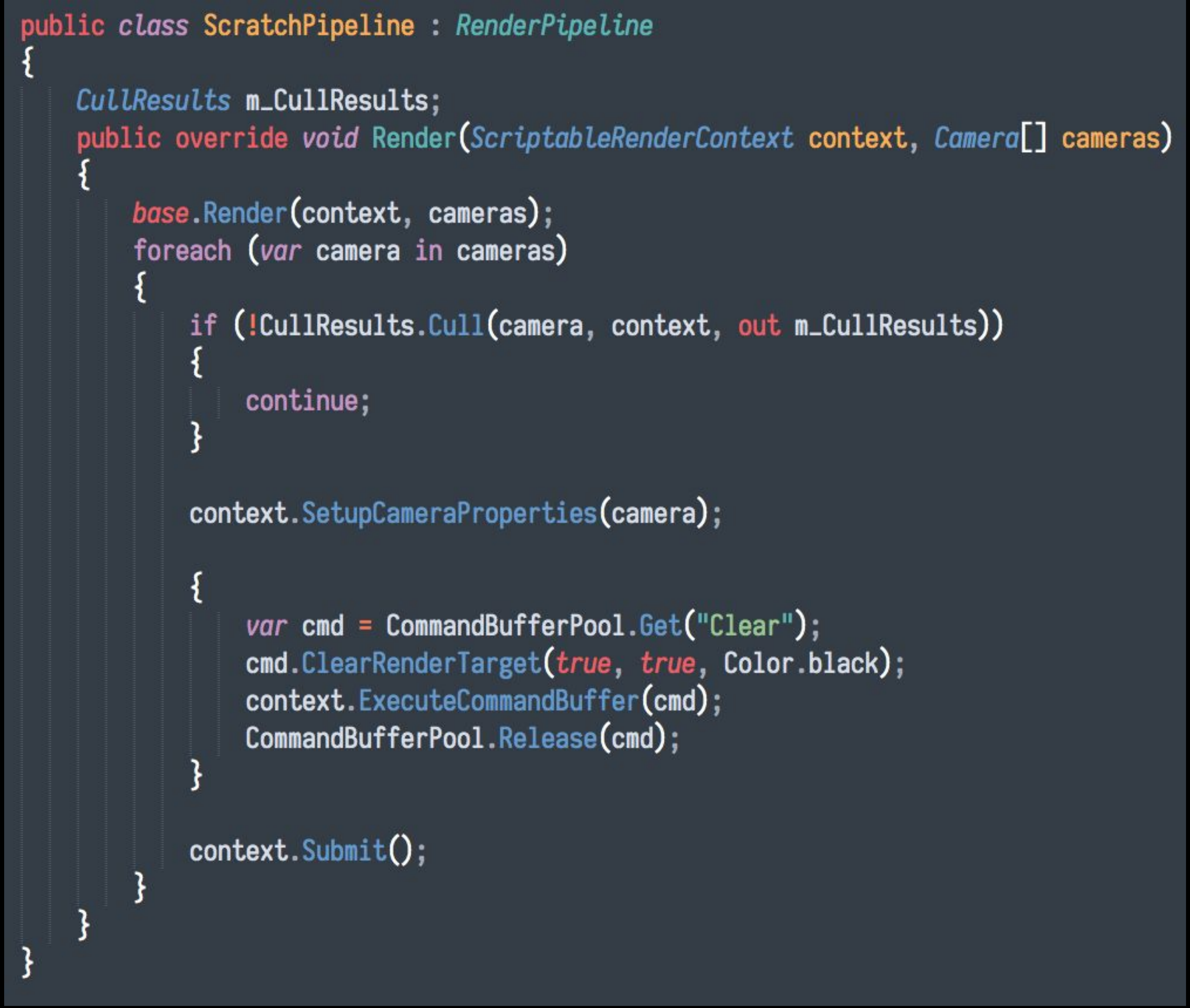
unity

Reminders

Create the render pipeline class

Implement the Render method

Store the CullResults in a field



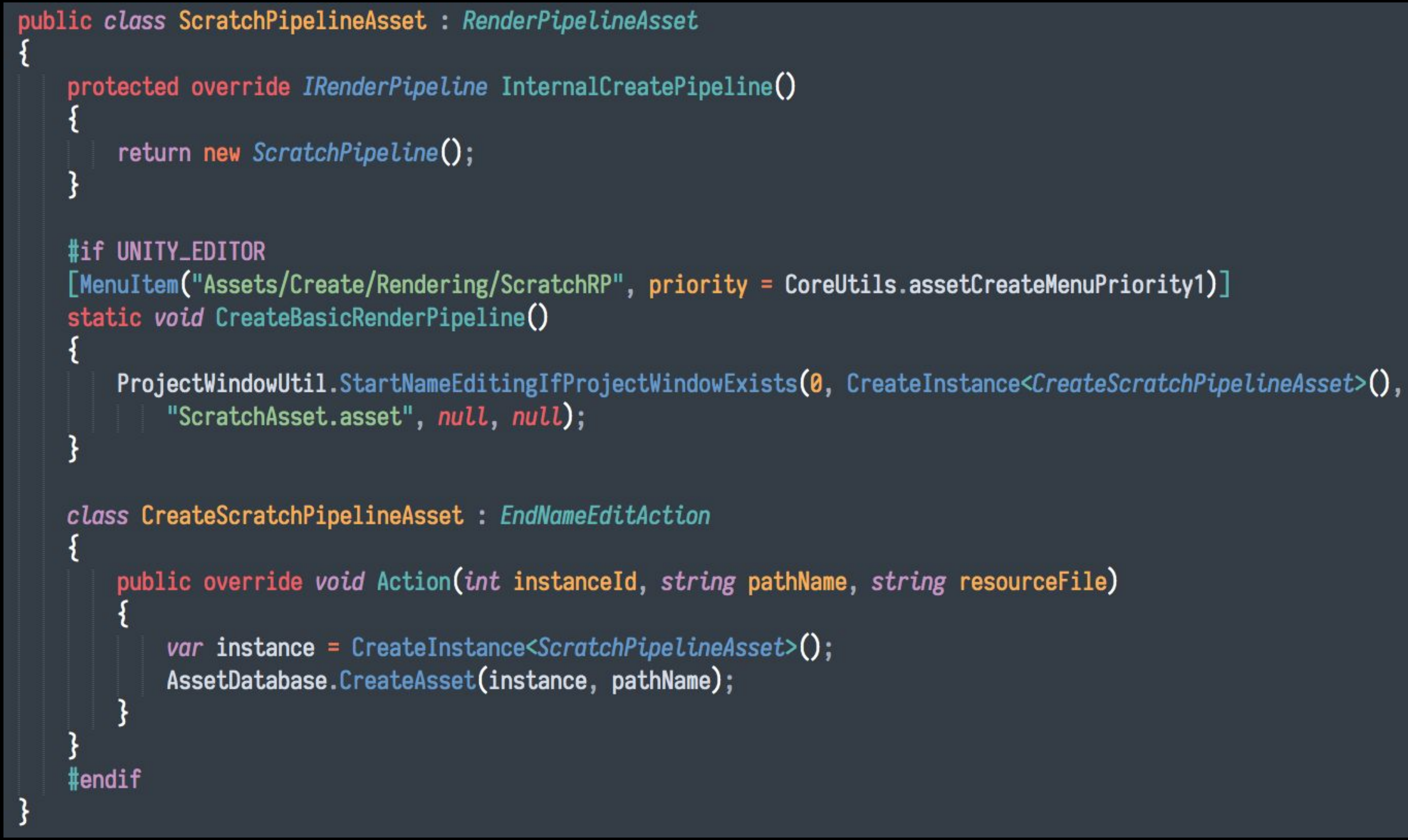
```
public class ScratchPipeline : RenderPipeline
{
    CullResults m_CullResults;
    public override void Render(ScriptableRenderContext context, Camera[] cameras)
    {
        base.Render(context, cameras);
        foreach (var camera in cameras)
        {
            if (!CullResults.Cull(camera, context, out m_CullResults))
            {
                continue;
            }

            context.SetupCameraProperties(camera);

            {
                var cmd = CommandBufferPool.Get("Clear");
                cmd.ClearRenderTarget(true, true, Color.black);
                context.ExecuteCommandBuffer(cmd);
                CommandBufferPool.Release(cmd);
            }

            context.Submit();
        }
    }
}
```

unity

Reminders

Create the pipeline asset class

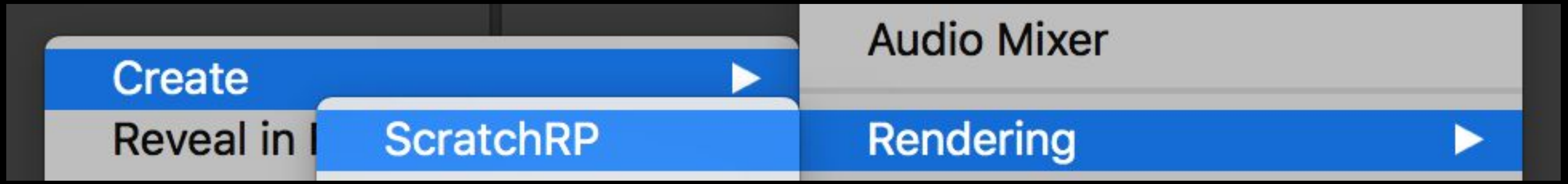
Make sure the class names match up with your own

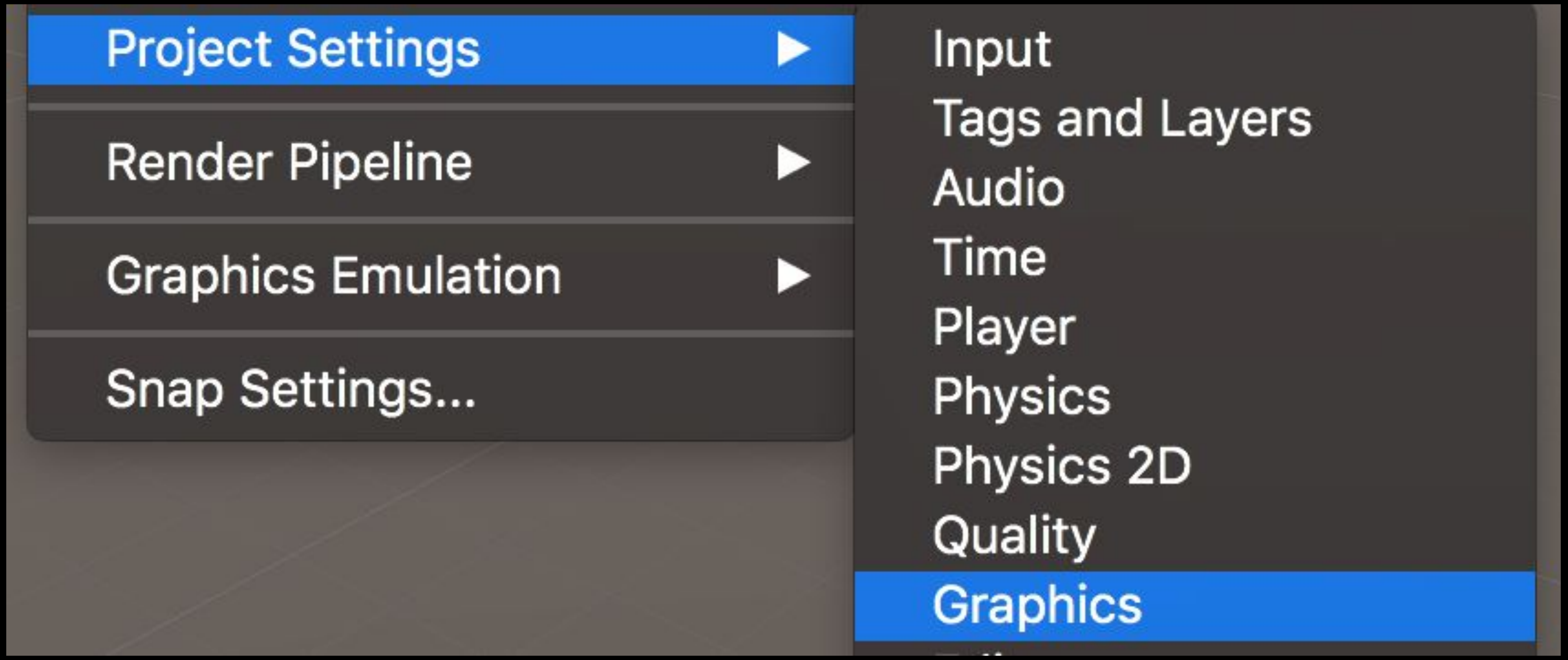


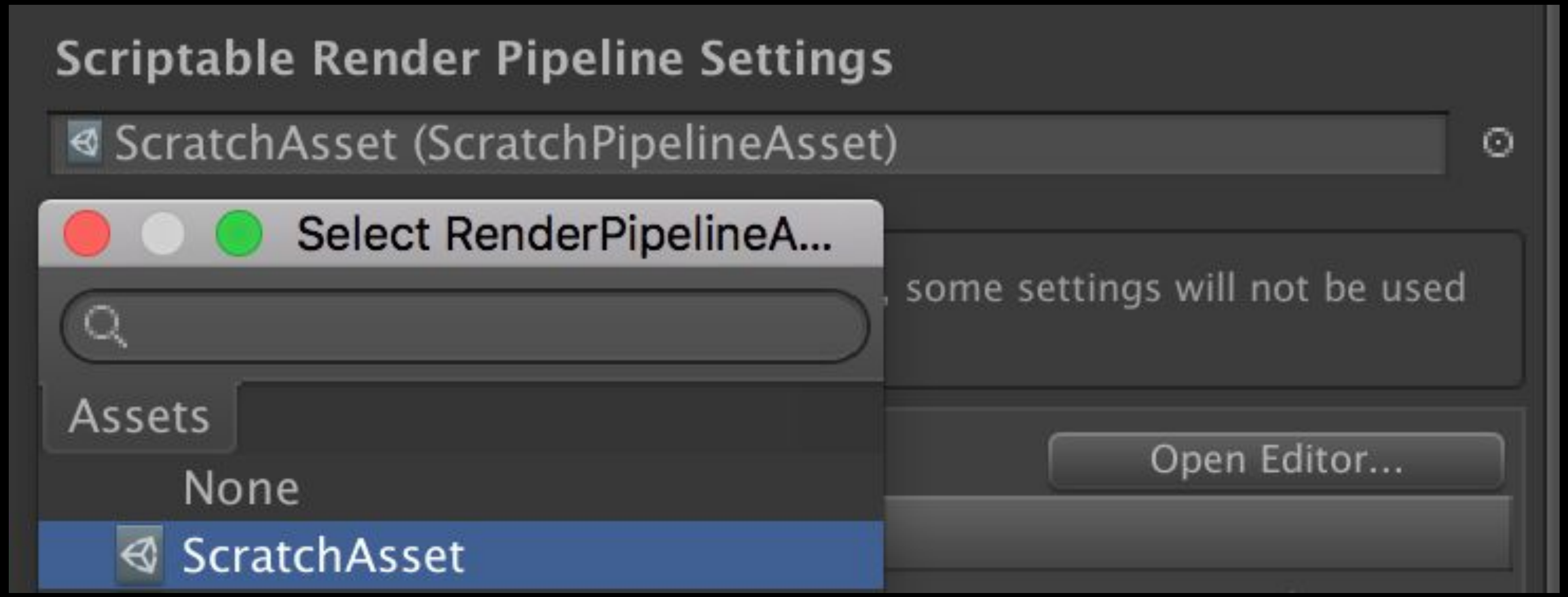
```
public class ScratchPipelineAsset : RenderPipelineAsset
{
    protected override IRenderPipeline InternalCreatePipeline()
    {
        return new ScratchPipeline();
    }

    #if UNITY_EDITOR
    [MenuItem("Assets/Create/Rendering/ScratchRP", priority = CoreUtils.assetCreateMenuPriority1)]
    static void CreateBasicRenderPipeline()
    {
        ProjectWindowUtil.StartNameEditingIfProjectWindowExists(0, CreateInstance<CreateScratchPipelineAsset>(),
            "ScratchAsset.asset", null, null);
    }

    class CreateScratchPipelineAsset : EndNameEditAction
    {
        public override void Action(int instanceId, string pathName, string resourceFile)
        {
            var instance = CreateInstance<ScratchPipelineAsset>();
            AssetDatabase.CreateAsset(instance, pathName);
        }
    }
    #endif
}
```

unity

Reminders






Scriptable Render Pipeline Settings

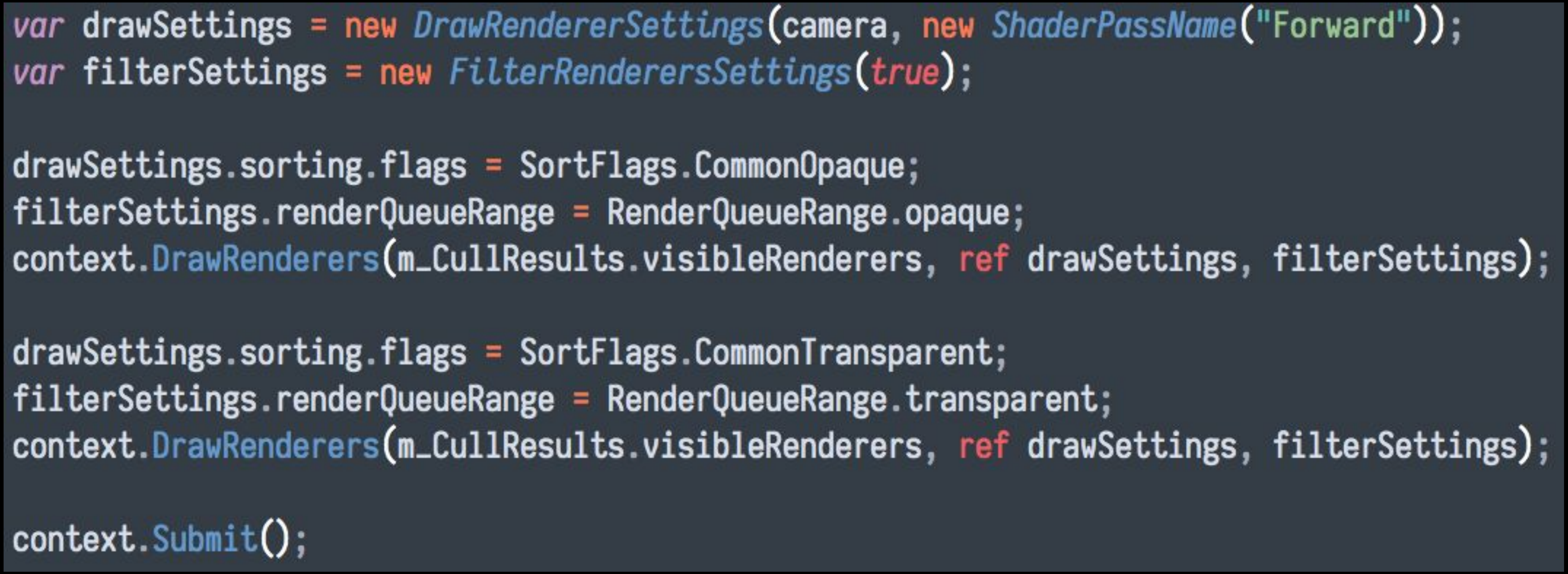
ScratchAsset (ScratchPipelineAsset)

, some settings will not be used

Unlit rendering

unity

Reminders



```
var drawSettings = new DrawRendererSettings(camera, new ShaderPassName("Forward"));
var filterSettings = new FilterRenderersSettings(true);

drawSettings.sorting.flags = SortFlags.CommonOpaque;
filterSettings.renderQueueRange = RenderQueueRange.opaque;
context.DrawRenderers(m_CullResults.visibleRenderers, ref drawSettings, filterSettings);

drawSettings.sorting.flags = SortFlags.CommonTransparent;
filterSettings.renderQueueRange = RenderQueueRange.transparent;
context.DrawRenderers(m_CullResults.visibleRenderers, ref drawSettings, filterSettings);

context.Submit();
```

Shaders

- New shader library
  - Developed in tandem with HDRP and LWRP
  - Complete replacement for the built-in library
  - Currently requires some set-up wrt. transformation matrices
    - Already set up in the provided project

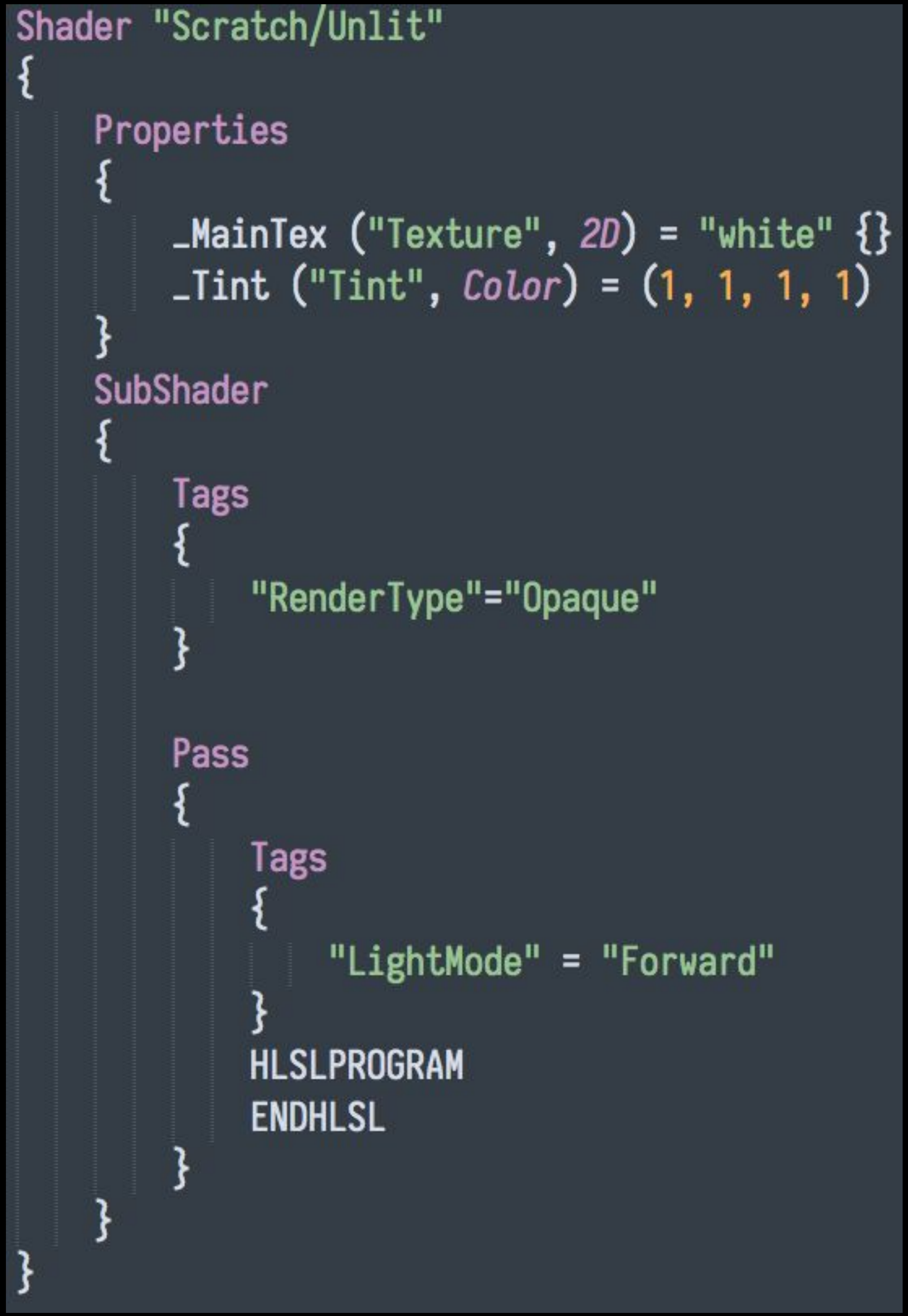
unity

Reminders

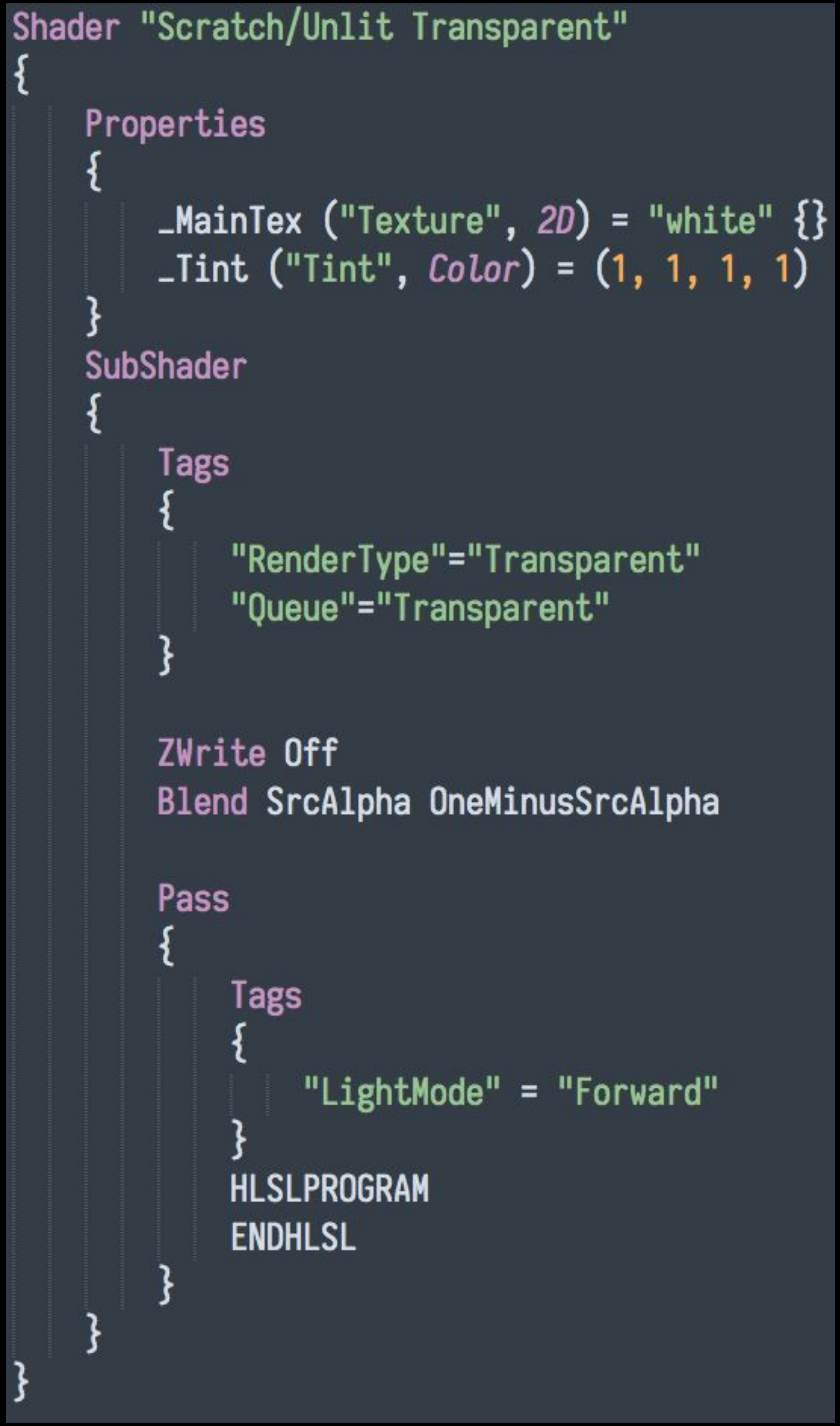
Create the Unlit shader

Create the Unlit Transparent shader

We’ll fill them in the HLSL part next up



```
Shader "Scratch/Unlit"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
        _Tint ("Tint", Color) = (1, 1, 1, 1)
    }
    SubShader
    {
        Tags
        {
            "RenderType"="Opaque"
        }

        Pass
        {
            Tags
            {
                "LightMode" = "Forward"
            }
            HLSLPROGRAM
            ENDHLSL
        }
    }
}
```



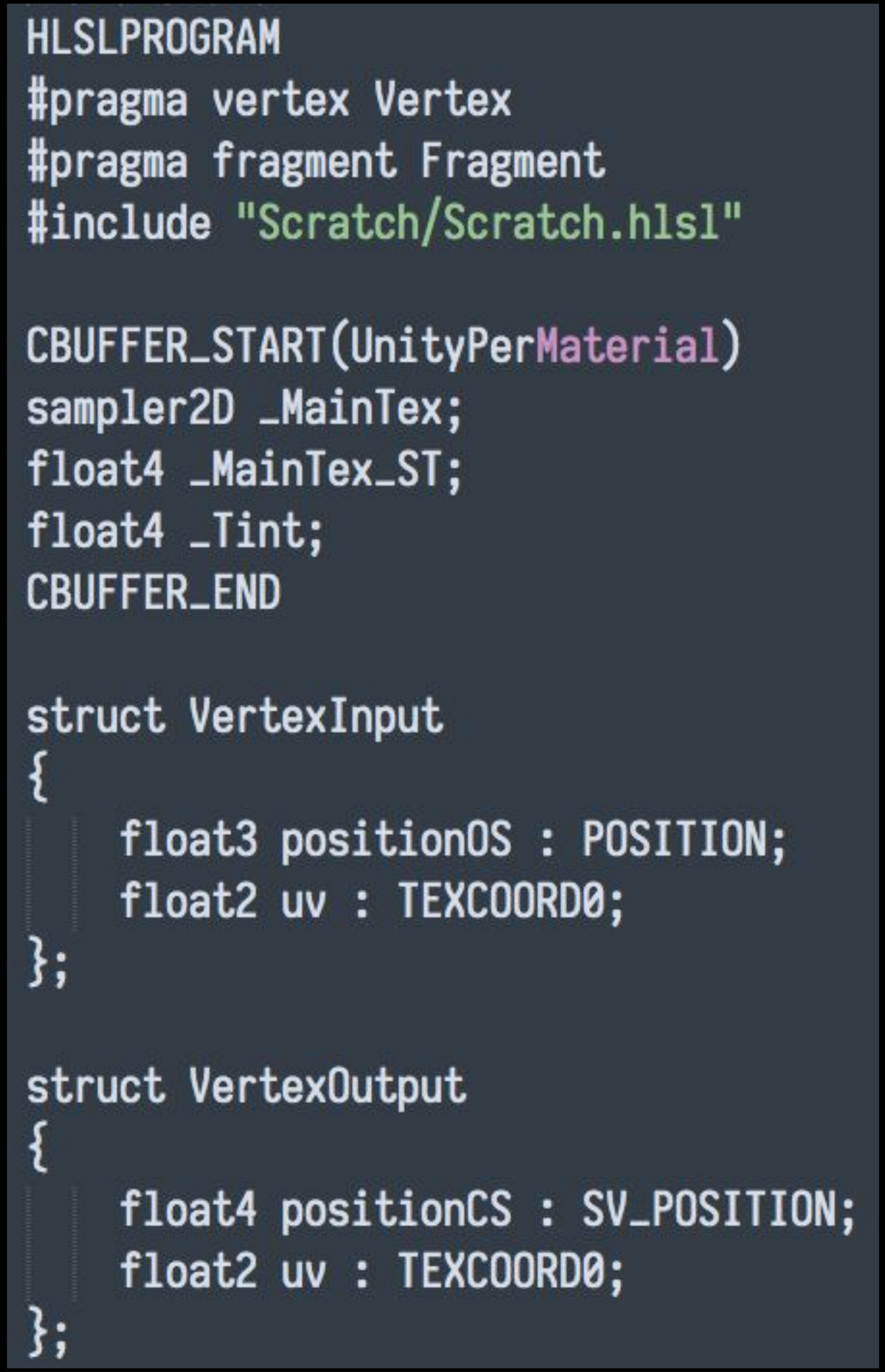
```
Shader "Scratch/Unlit Transparent"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
        _Tint ("Tint", Color) = (1, 1, 1, 1)
    }
    SubShader
    {
        Tags
        {
            "RenderType"="Transparent"
            "Queue"="Transparent"
        }

        ZWrite Off
        Blend SrcAlpha OneMinusSrcAlpha

        Pass
        {
            Tags
            {
                "LightMode" = "Forward"
            }
            HLSLPROGRAM
            ENDHLSL
        }
    }
}
```

unity

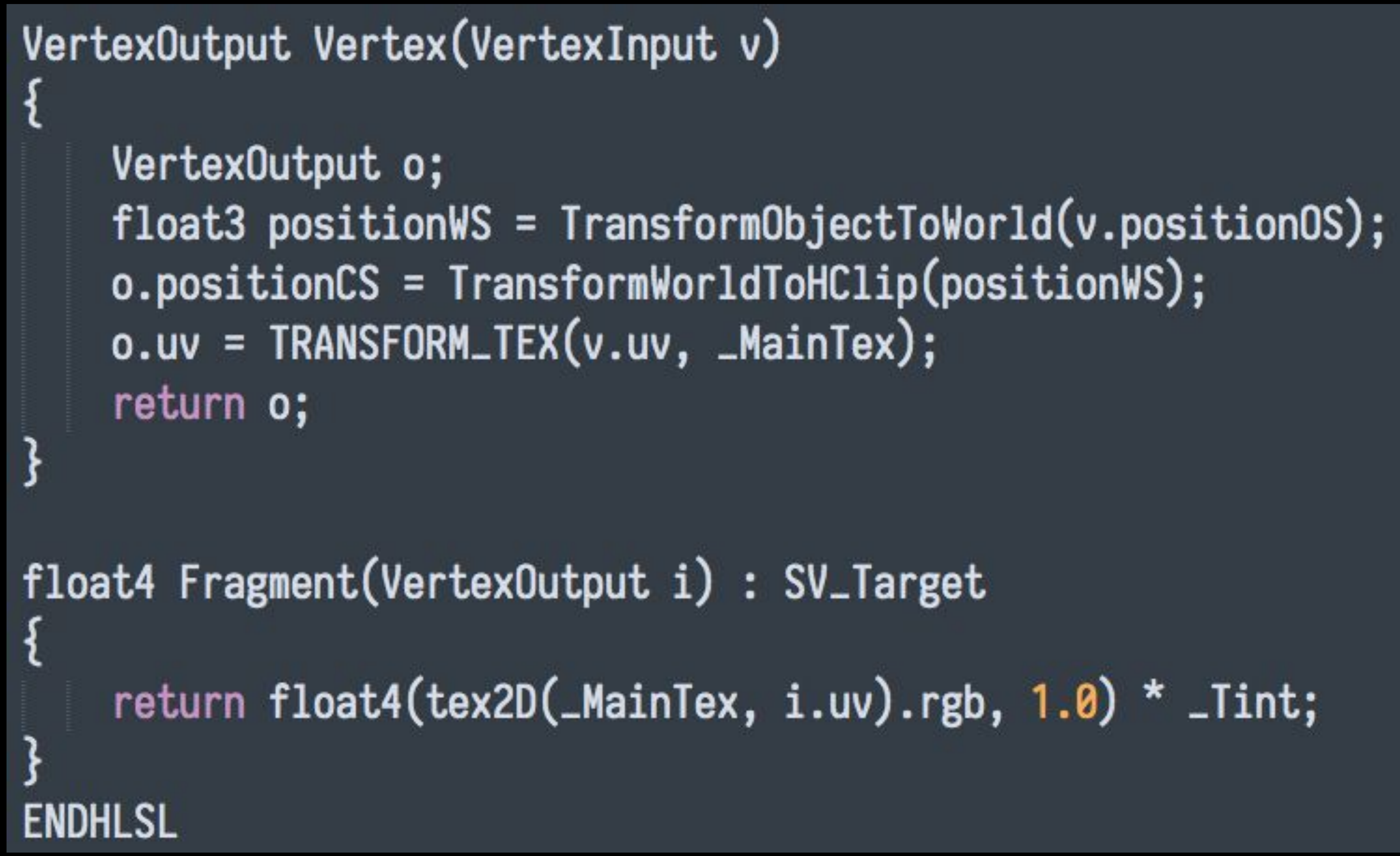
Reminders

Fill out the HLSL parts of the Unlit and Unlit Transparent shaders

The same code will work in both



```
HLSLPROGRAM
#pragma vertex Vertex
#pragma fragment Fragment
#include "Scratch/Scratch.hlsl"

CBUFFER_START(UnityPerMaterial)
sampler2D _MainTex;
float4 _MainTex_ST;
float4 _Tint;
CBUFFER_END

struct VertexInput
{
    float3 positionOS : POSITION;
    float2 uv : TEXCOORD0;
};

struct VertexOutput
{
    float4 positionCS : SV_POSITION;
    float2 uv : TEXCOORD0;
};
```

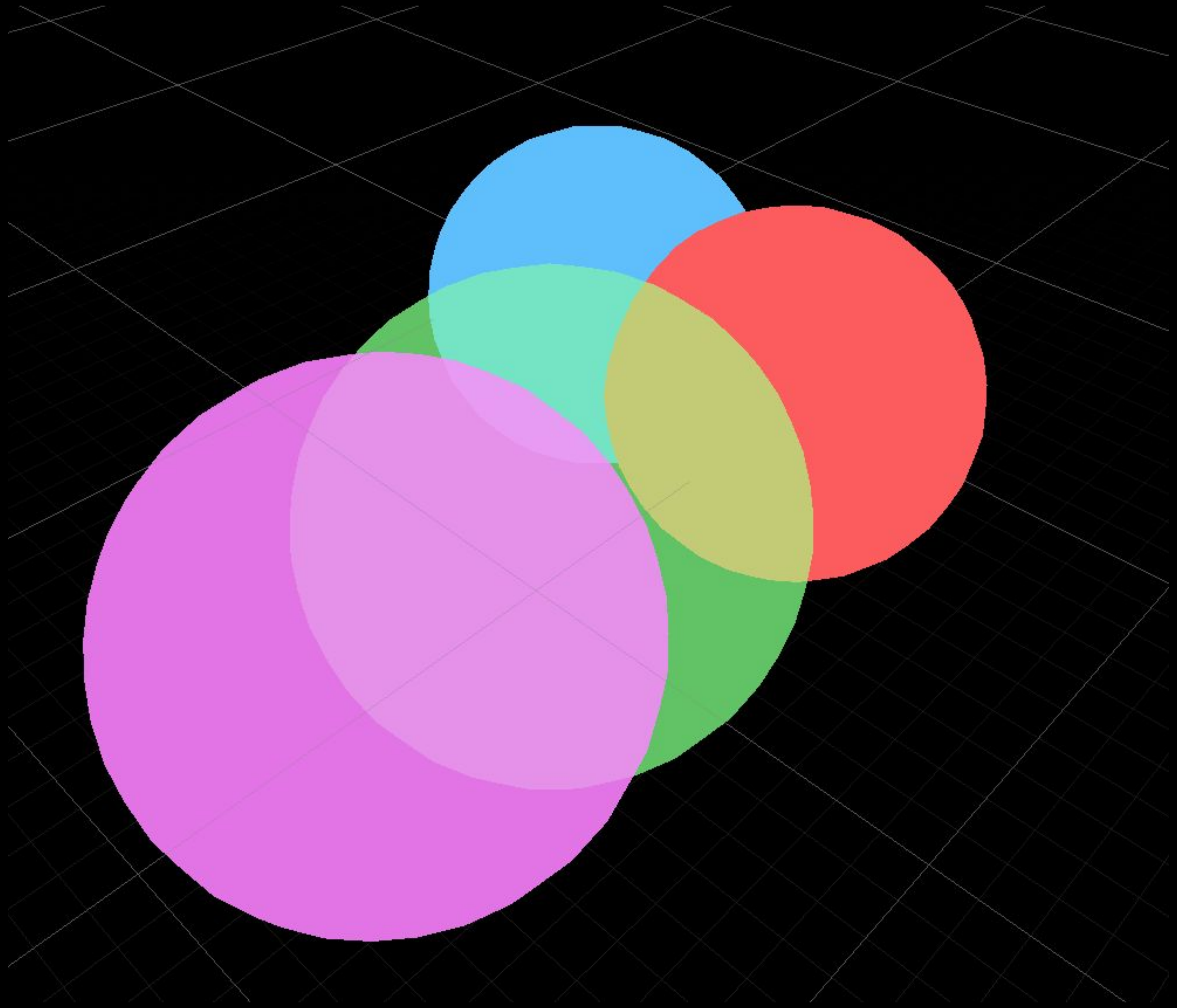


```
VertexOutput Vertex(VertexInput v)
{
    VertexOutput o;
    float3 positionWS = TransformObjectToWorld(v.positionOS);
    o.positionCS = TransformWorldToHClip(positionWS);
    o.uv = TRANSFORM_TEX(v.uv, _MainTex);
    return o;
}

float4 Fragment(VertexOutput i) : SV_Target
{
    return float4(tex2D(_MainTex, i.uv).rgb, 1.0) * _Tint;
}
ENDHLSL
```

unity

Reminders

- Time to use the shaders
  - Create materials
    - Tip: Right click the shader and create the material via that menu – that will hook up the shader to the new material automatically
  - Apply it to some objects in the scene



Lit rendering

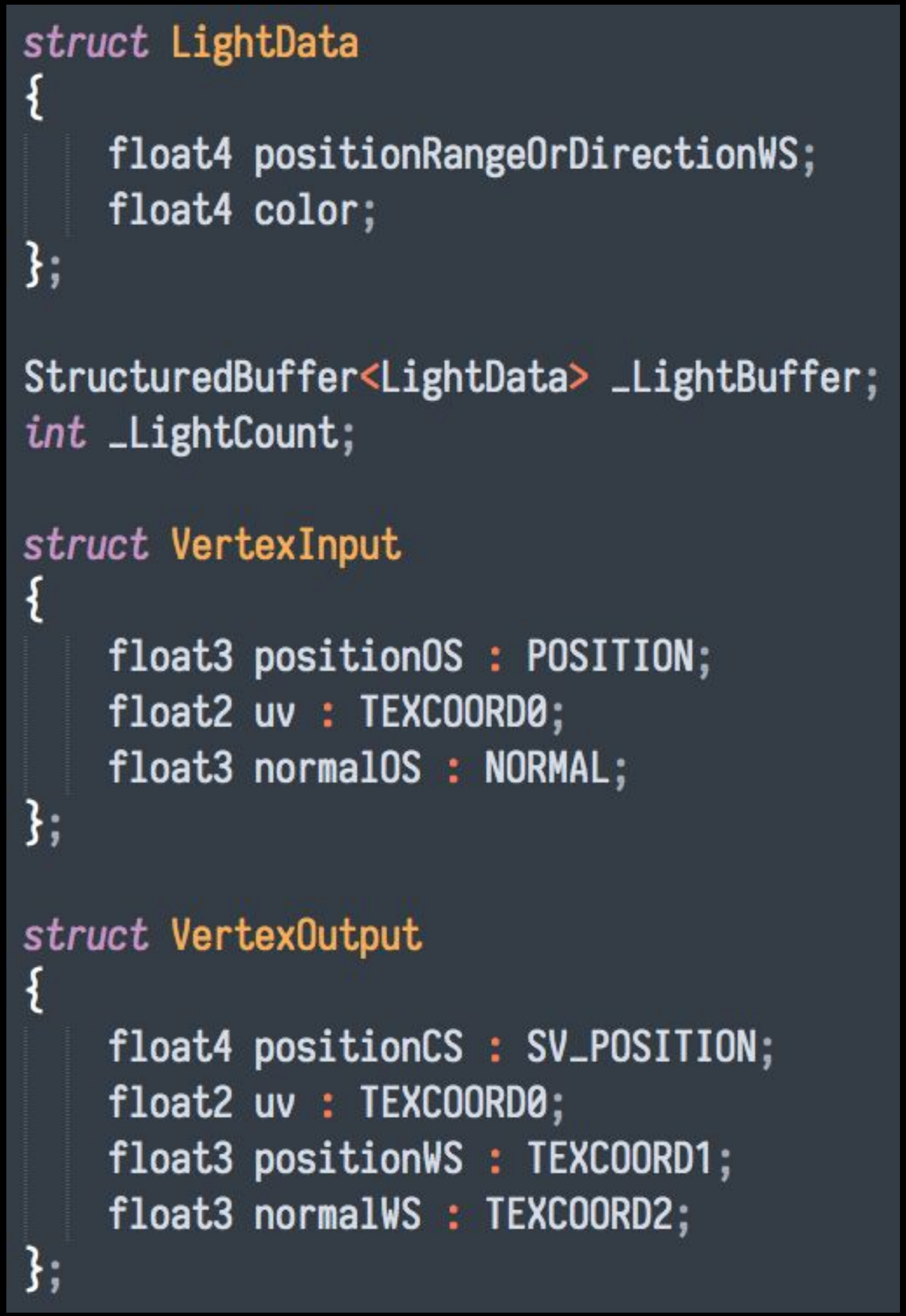
Time to introduce lighting

- Store lights
  - Use a StructuredBuffer
    - C# equivalent is called ComputeBuffer in Unity
  - SRP provides us a list of visible lights in the cull results

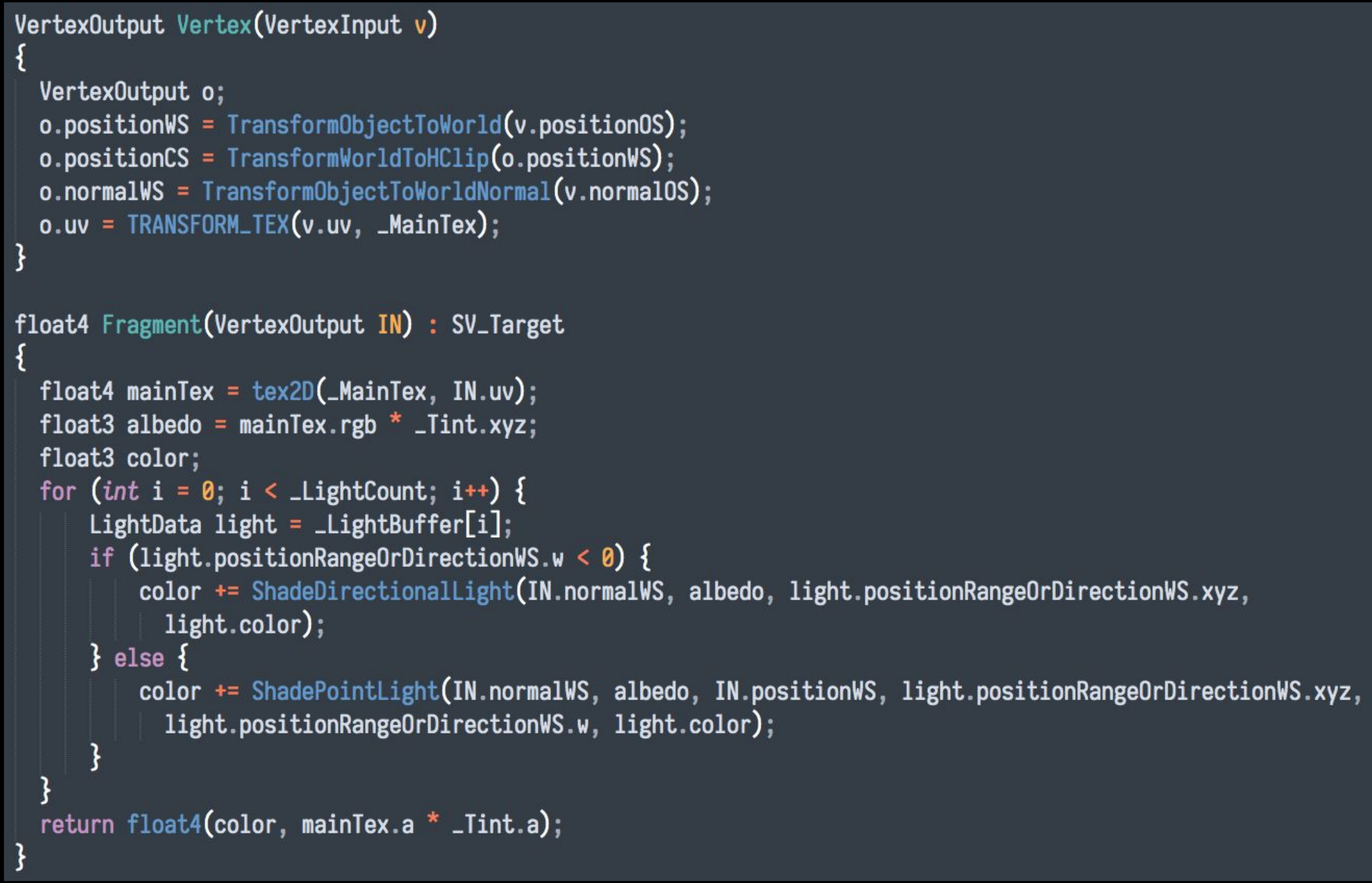
- Create a Lit shader
  - Loop over lights from structured buffer
  - Use shading function from workshop shader library
  - Transparent Lit will be very similar
  - No changes to Unlit shaders

unity

Reminders



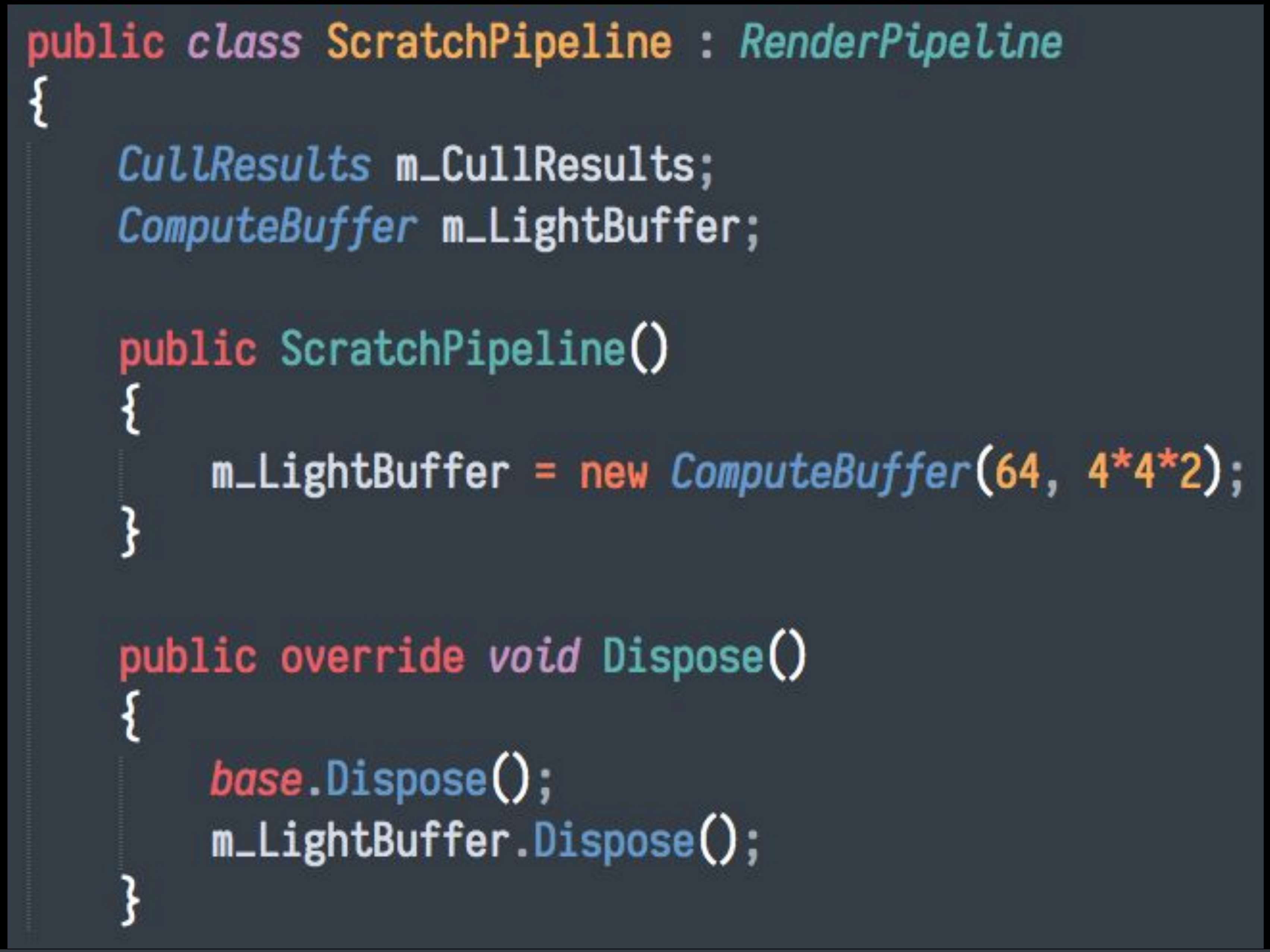
```
struct LightData
{
    float4 positionRangeOrDirectionWS;
    float4 color;
};

StructuredBuffer<LightData> _LightBuffer;
int _LightCount;

struct VertexInput
{
    float3 positionOS : POSITION;
    float2 uv : TEXCOORD0;
    float3 normalOS : NORMAL;
};

struct VertexOutput
{
    float4 positionCS : SV_POSITION;
    float2 uv : TEXCOORD0;
    float3 positionWS : TEXCOORD1;
    float3 normalWS : TEXCOORD2;
};
```



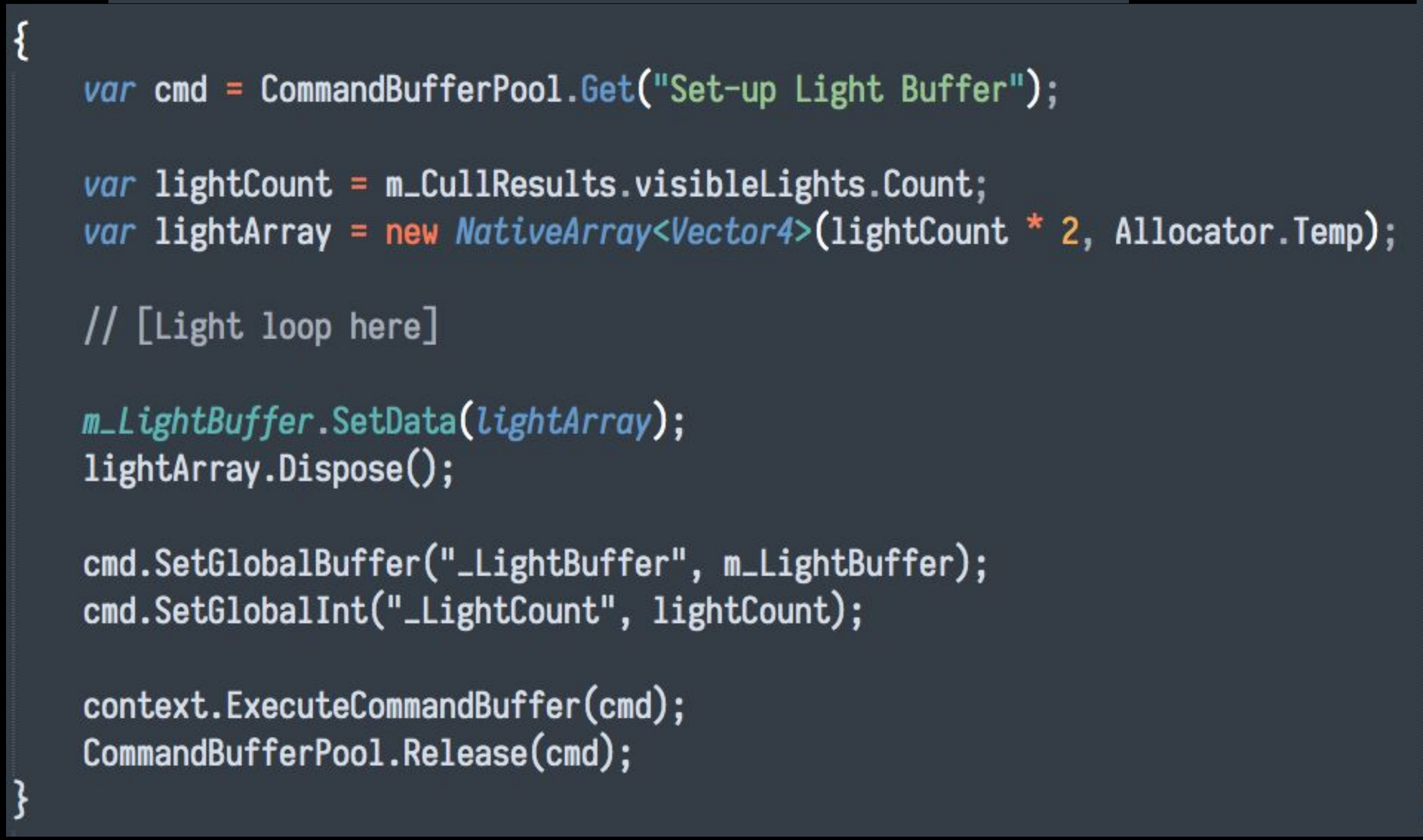
```
VertexOutput Vertex(VertexInput v)
{
  VertexOutput o;
  o.positionWS = TransformObjectToWorld(v.positionOS);
  o.positionCS = TransformWorldToHClip(o.positionWS);
  o.normalWS = TransformObjectToWorldNormal(v.normalOS);
  o.uv = TRANSFORM_TEX(v.uv, _MainTex);
}

float4 Fragment(VertexOutput IN) : SV_Target
{
  float4 mainTex = tex2D(_MainTex, IN.uv);
  float3 albedo = mainTex.rgb * _Tint.xyz;
  float3 color;
  for (int i = 0; i < _LightCount; i++) {
    LightData light = _LightBuffer[i];
    if (light.positionRangeOrDirectionWS.w < 0) {
      color += ShadeDirectionalLight(IN.normalWS, albedo, light.positionRangeOrDirectionWS.xyz,
        light.color);
    } else {
      color += ShadePointLight(IN.normalWS, albedo, IN.positionWS, light.positionRangeOrDirectionWS.xyz,
        light.positionRangeOrDirectionWS.w, light.color);
    }
  }
  return float4(color, mainTex.a * _Tint.a);
}
```

unity

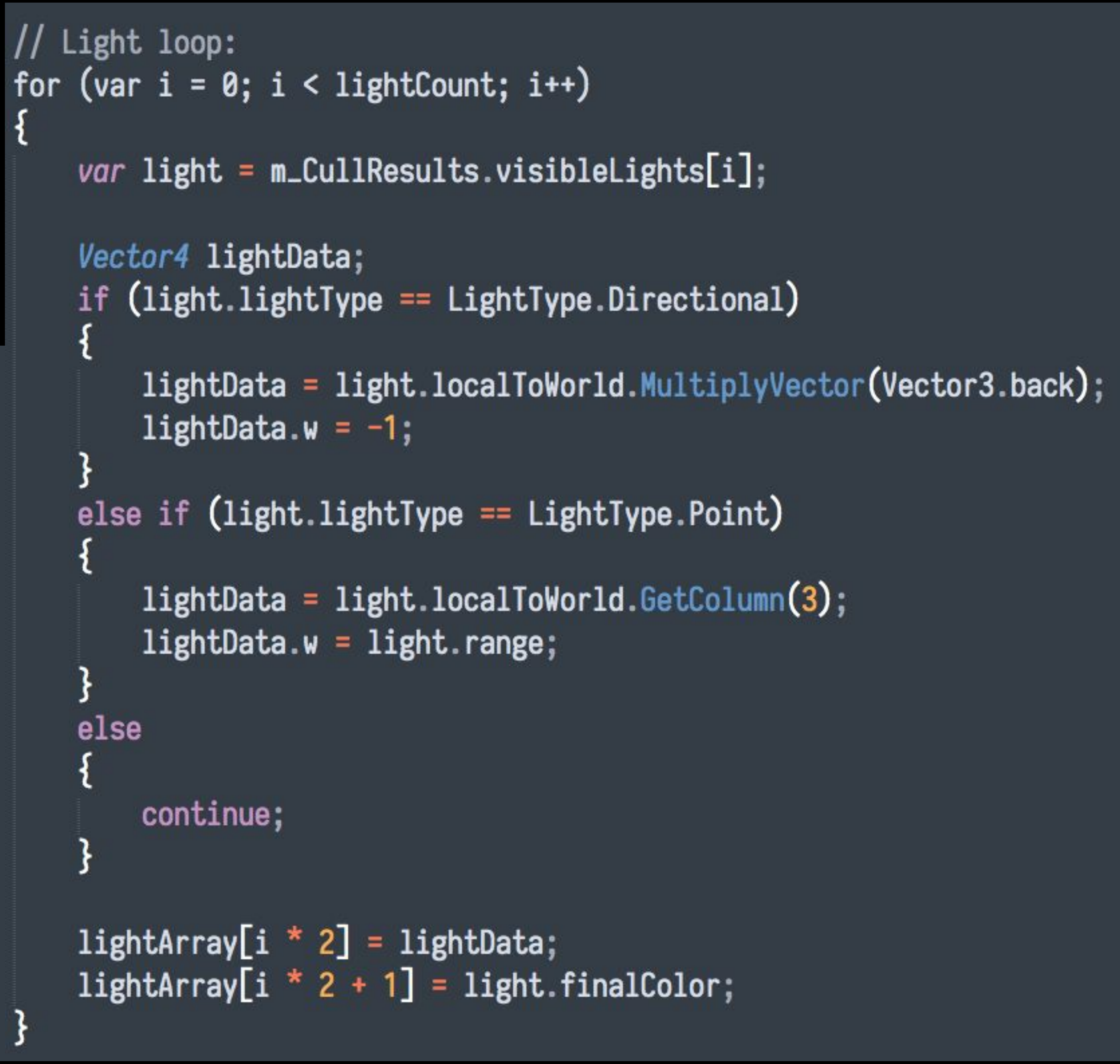
Reminders



```
public class ScratchPipeline : RenderPipeline
{
    CullResults m_CullResults;
    ComputeBuffer m_LightBuffer;

    public ScratchPipeline()
    {
        m_LightBuffer = new ComputeBuffer(64, 4*4*2);
    }

    public override void Dispose()
    {
        base.Dispose();
        m_LightBuffer.Dispose();
    }
```



```
{
    var cmd = CommandBufferPool.Get("Set-up Light Buffer");

    var lightCount = m_CullResults.visibleLights.Count;
    var lightArray = new NativeArray<Vector4>(lightCount * 2, Allocator.Temp);

    // [Light loop here]

    m_LightBuffer.SetData(lightArray);
    lightArray.Dispose();

    cmd.SetGlobalBuffer("_LightBuffer", m_LightBuffer);
    cmd.SetGlobalInt("_LightCount", lightCount);

    context.ExecuteCommandBuffer(cmd);
    CommandBufferPool.Release(cmd);
}
```

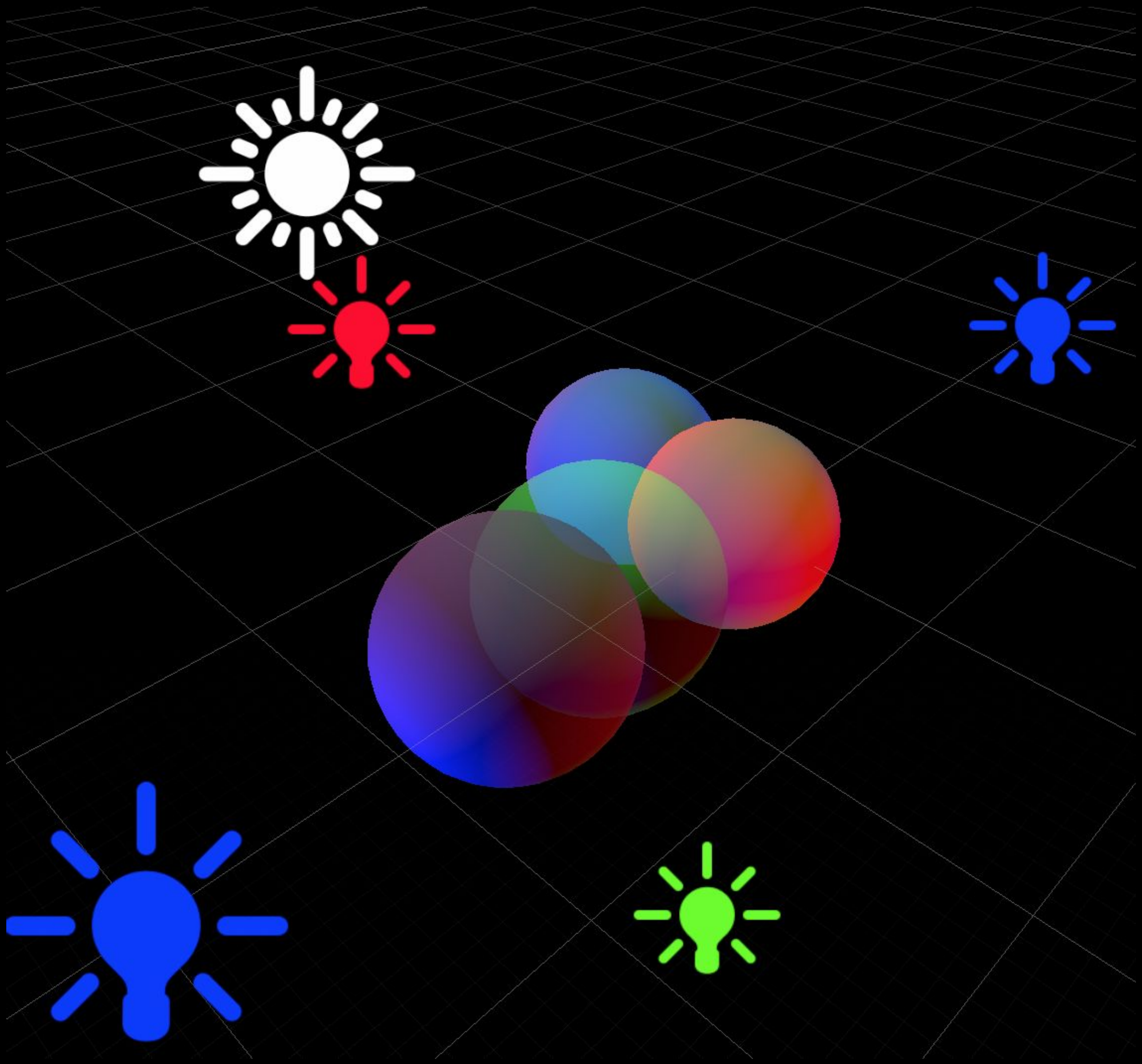


```
// Light loop:
for (var i = 0; i < lightCount; i++)
{
    var light = m_CullResults.visibleLights[i];

    Vector4 lightData;
    if (light.lightType == LightType.Directional)
    {
        lightData = light.localToWorld.MultiplyVector(Vector3.back);
        lightData.w = -1;
    }
    else if (light.lightType == LightType.Point)
    {
        lightData = light.localToWorld.GetColumn(3);
        lightData.w = light.range;
    }
    else
    {
        continue;
    }

    lightArray[i * 2] = lightData;
    lightArray[i * 2 + 1] = light.finalColor;
}
```

unity

Reminders

- Time to use the shaders
  - Create materials
    - Tip: Right click the shader and create the material via that menu – that will hook up the shader to the new material automatically
  - Apply it to some objects in the scene
  - Remember to add some lights!
    - Try to add a directional and a point light



Skybox & custom render texture

Camera provided render targets

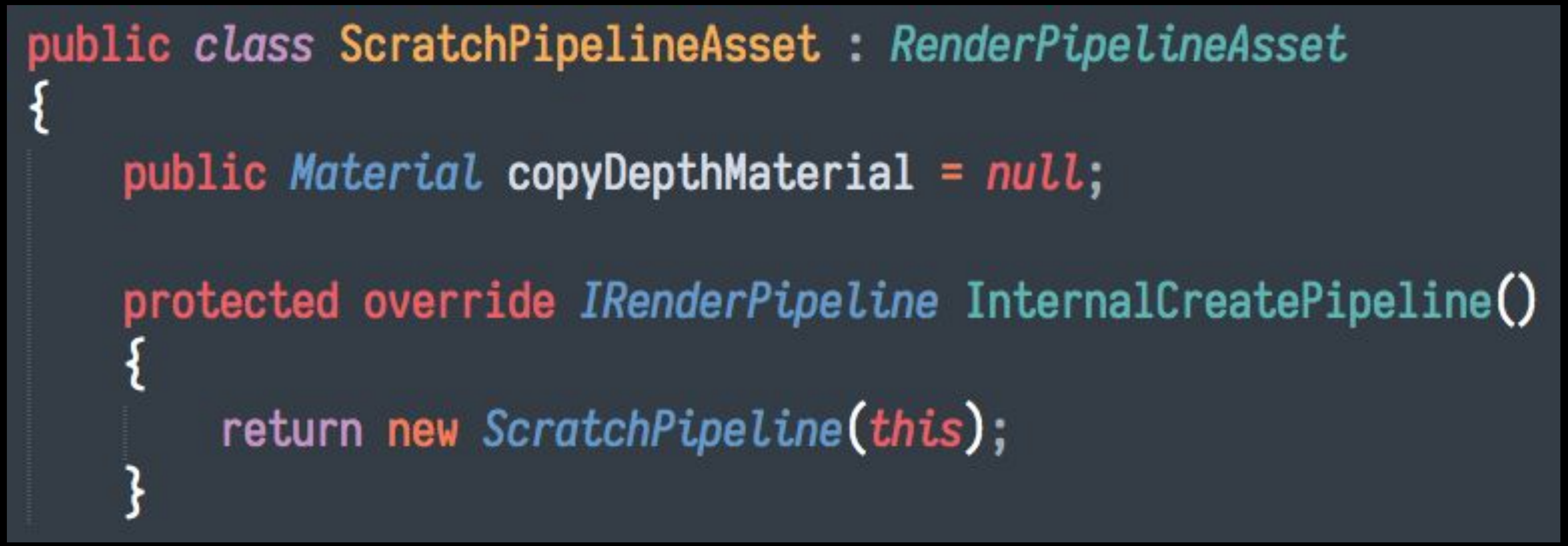
- Issues
  - Platform differences
  - Can’t refer to depth buffer of Camera-provided RT

- Solution
  - Create our own color and depth RTs
  - Will need to copy depth to Camera depth buffer and blit color RT
    - Recommend just grabbing the CopyDepth from the part 4 solution

unity

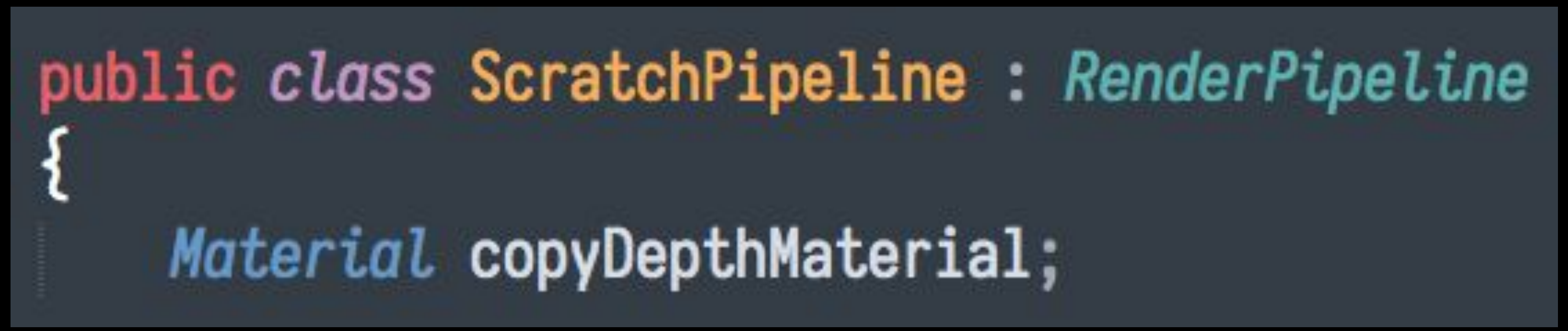
Reminders

Add a public Material field to the asset class



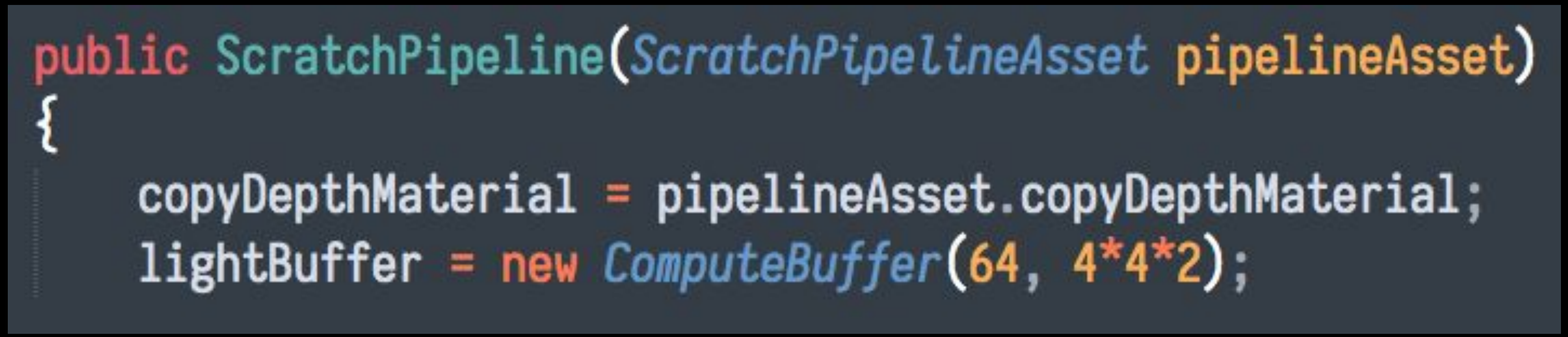
```
public class ScratchPipelineAsset : RenderPipelineAsset
{
    public Material copyDepthMaterial = null;

    protected override IRenderPipeline InternalCreatePipeline()
    {
        return new ScratchPipeline(this);
    }
```

Add a Material field to the render pipeline class



```
public class ScratchPipeline : RenderPipeline
{
    Material copyDepthMaterial;
```

Modify the constructor to take a reference to the asset, and then grab the material

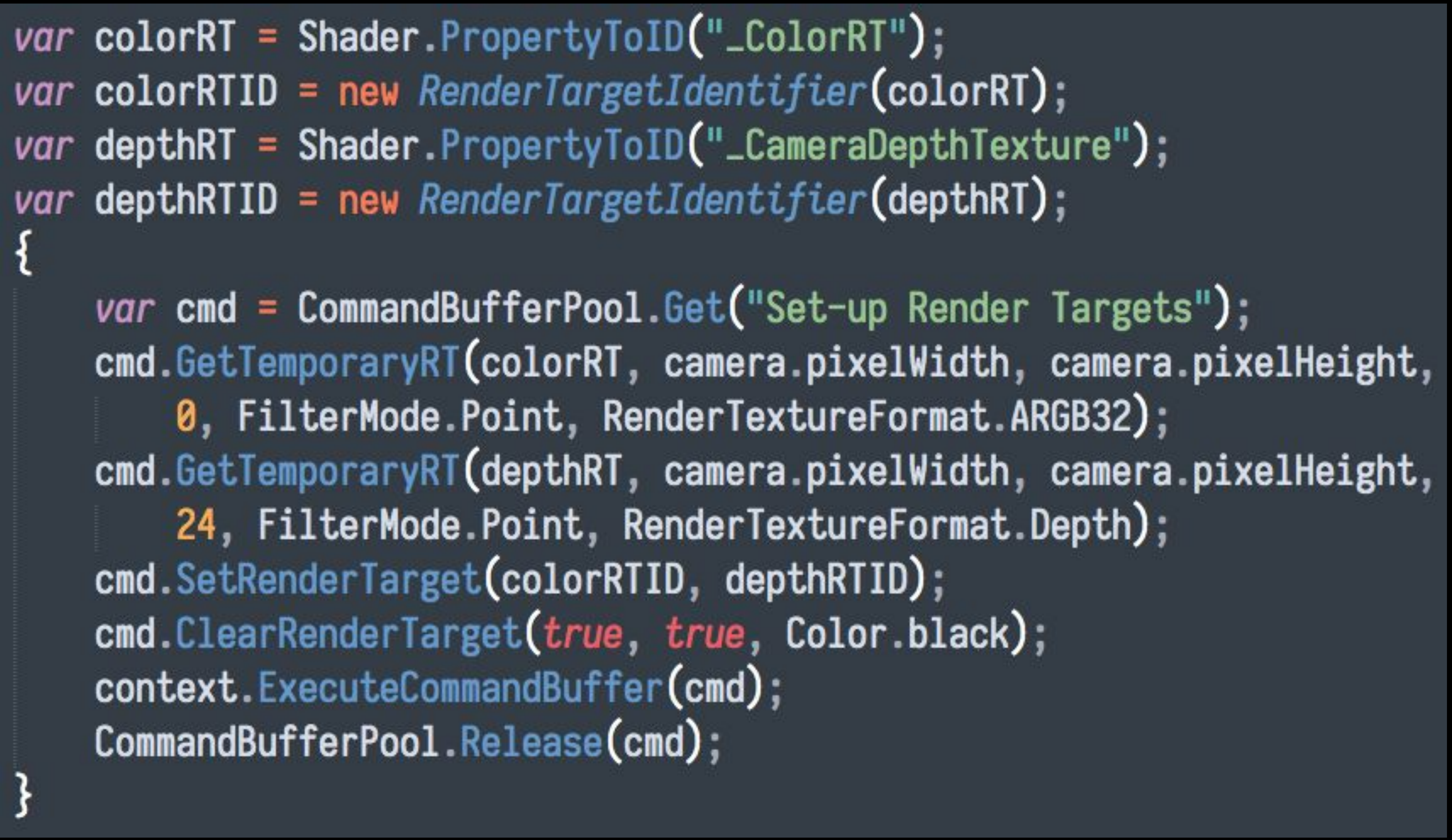


```
public ScratchPipeline(ScratchPipelineAsset pipelineAsset)
{
    copyDepthMaterial = pipelineAsset.copyDepthMaterial;
    lightBuffer = new ComputeBuffer(64, 4*4*2);
```

unity

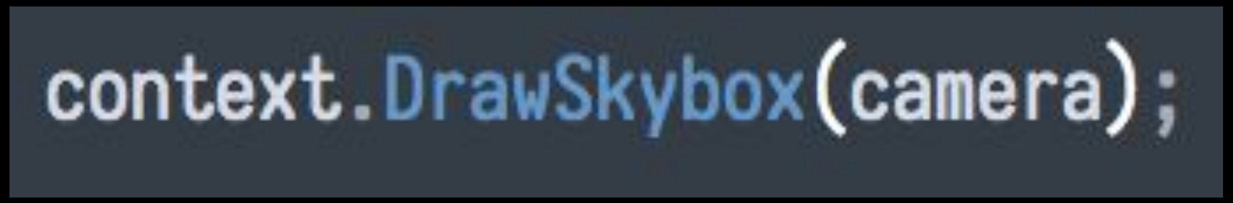
Reminders

Augment the “Clear” block with RT creation



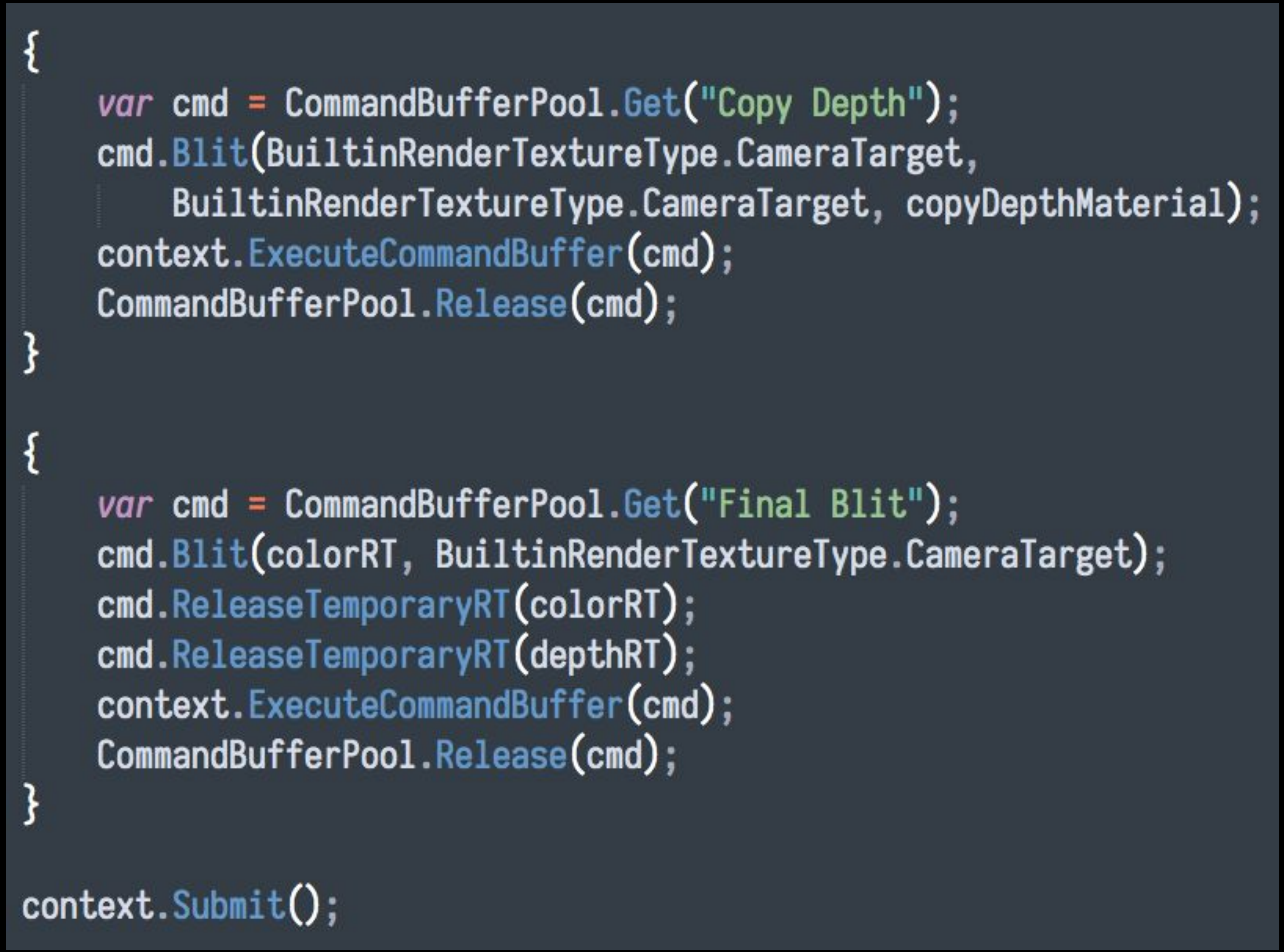
```
var colorRT = Shader.PropertyToID("_ColorRT");
var colorRTID = new RenderTargetIdentifier(colorRT);
var depthRT = Shader.PropertyToID("_CameraDepthTexture");
var depthRTID = new RenderTargetIdentifier(depthRT);
{
    var cmd = CommandBufferPool.Get("Set-up Render Targets");
    cmd.GetTemporaryRT(colorRT, camera.pixelWidth, camera.pixelHeight,
        0, FilterMode.Point, RenderTextureFormat.ARGB32);
    cmd.GetTemporaryRT(depthRT, camera.pixelWidth, camera.pixelHeight,
        24, FilterMode.Point, RenderTextureFormat.Depth);
    cmd.SetRenderTarget(colorRTID, depthRTID);
    cmd.ClearRenderTarget(true, true, Color.black);
    context.ExecuteCommandBuffer(cmd);
    CommandBufferPool.Release(cmd);
}
```

Draw the skybox in between opaque and transparent



```
context.DrawSkybox(camera);
```

Do the depth copy and blit at the very end



```
{
    var cmd = CommandBufferPool.Get("Copy Depth");
    cmd.Blit(BuiltinRenderTextureType.CameraTarget,
        BuiltinRenderTextureType.CameraTarget, copyDepthMaterial);
    context.ExecuteCommandBuffer(cmd);
    CommandBufferPool.Release(cmd);
}

{
    var cmd = CommandBufferPool.Get("Final Blit");
    cmd.Blit(colorRT, BuiltinRenderTextureType.CameraTarget);
    cmd.ReleaseTemporaryRT(colorRT);
    cmd.ReleaseTemporaryRT(depthRT);
    context.ExecuteCommandBuffer(cmd);
    CommandBufferPool.Release(cmd);
}

context.Submit();
```

unity