Tutorial: Creating a Film Noir Effect with Red Highlight in Unity

Objective

In this tutorial, you will create a custom post-processing effect for Unity's Universal Render Pipeline (URP) that applies a Film Noir style to your scene, turning everything black and white except for red elements, similar to the visual style of the film *Sin City*.

Prerequisites

- Unity 2022.3 LTS or later with URP installed
- Basic understanding of C# scripting and shader writing

Part 1: Setting Up the Custom Effect

1. Create a new C# script in your project and name it `FilmNoirEffect.cs`
2. Open the script and replace its contents with the following code:

```
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;

[System.Serializable, VolumeComponentMenu("Custom/Film Noir Effect")]
public class FilmNoirEffect : VolumeComponent, IPostProcessComponent
{
    public ColorParameter redThreshold = new ColorParameter(new Color(0.8f, 0.2f,
0.2f));
    public ClampedFloatParameter contrast = new ClampedFloatParameter(0.8f, 0f,
1f);
    public ClampedFloatParameter brightness = new ClampedFloatParameter(0.1f, -1f,
1f);

    public bool IsActive() => true;
    public bool IsTileCompatible() => true;
}
```

ⓘ **Info**

This script defines our custom effect as a Volume Component. Volume Components can be applied as Overrides to Post-Processing Volumes. It includes parameters for adjusting the red threshold, contrast, and brightness.

Part 2: Creating the Shader

1. Create a new shader by right-clicking in the Project window and selecting `Create > Shader > Unlit Shader`. Name it `FilmNoirEffectShader`.

2. Open the shader and replace its contents with the following code:

```
Shader "Custom/FilmNoirEffect"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" "RenderPipeline" = "UniversalPipeline"}
        LOD 0

        HLSLINCLUDE
        #pragma vertex Vert
        #pragma fragment Frag
        #pragma target 2.0

        #include "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/Core.hlsl"
        #include "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/Lighting.hlsl"

        TEXTURE2D(_MainTex);
        SAMPLER(sampler_MainTex);
        float4 _MainTex_TexelSize;
        float3 _RedThreshold;
        float _Contrast;
        float _Brightness;

        struct Attributes
        {
            float4 positionOS : POSITION;
            float2 uv : TEXCOORD0;
        };

        struct Varyings
        {
            float4 positionCS : SV_POSITION;
            float2 uv : TEXCOORD0;
        };

        Varyings Vert(Attributes input)
        {
            Varyings output;
            output.positionCS = TransformObjectToHClip(input.positionOS.xyz);
            output.uv = input.uv;
            return output;
        }

        float4 Frag(Varyings input) : SV_Target
        {
            float4 color = SAMPLE_TEXTURE2D(_MainTex, sampler_MainTex, input.uv);

            // Convert to grayscale
            float luminance = dot(color.rgb, float3(0.299, 0.587, 0.114));
            float3 grayscale = float3(luminance, luminance, luminance);

            // Apply contrast and brightness
            grayscale = saturate(grayscale * (1.0 + _Contrast) + _Brightness);

            // Check if the color is above the red threshold
            float3 redDiff = saturate(color.rgb - _RedThreshold);
            float redMask = dot(redDiff, float3(1,1,1)) > 0 ? 1 : 0;

            // Mix grayscale and original color based on the red mask
            float3 finalColor = lerp(grayscale, color.rgb, redMask);

            return float4(finalColor, color.a);
        }
        ENDHLSL

        Pass
        {
            Name "FilmNoir"

            HLSLPROGRAM
            ENDHLSL
        }
    }
}
```

Tip

This shader converts the image to grayscale but preserves colors that are more red than the specified threshold. Adjust the `_RedThreshold`, `_Contrast`, and `_Brightness` parameters to fine-tune the effect.

Part 3: Implementing the Renderer Feature

1. Create a new C# script and name it `FilmNoirRendererFeature.cs`
2. Open the script and replace its contents with the following code:

```
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;

public class FilmNoirRendererFeature : ScriptableRendererFeature
{
    class FilmNoirRenderPass : ScriptableRenderPass
    {
        private Material material;
        private FilmNoirEffect filmNoirEffect;
        private RenderTargetIdentifier source;
        private RenderTargetHandle tempTexture;
        public FilmNoirRenderPass(Material material)
        {
            this.material = material;
            tempTexture.Init("_TempTexture");
        }

        public override void Execute(ScriptableRenderContext context, ref
RenderingData renderingData)
        {
            if (material == null)
            {
                Debug.LogError("Film Noir Effect material is missing");
                return;
            }

            VolumeStack stack = VolumeManager.instance.stack;
            filmNoirEffect = stack.GetComponent<FilmNoirEffect>();
            if (filmNoirEffect == null) { return; }

            CommandBuffer cmd = CommandBufferPool.Get("Film Noir Effect");

            material.SetVector("_RedThreshold", filmNoirEffect.redThreshold.value);
            material.SetFloat("_Contrast", filmNoirEffect.contrast.value);
            material.SetFloat("_Brightness", filmNoirEffect.brightness.value);

            RenderTextureDescriptor descriptor =
renderingData.cameraData.cameraTargetDescriptor;
            descriptor.depthBufferBits = 0;

            cmd.GetTemporaryRT(tempTexture.id, descriptor);
            Blit(cmd, source, tempTexture.Identifier(), material);
            Blit(cmd, tempTexture.Identifier(), source);

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

        public override void FrameCleanup(CommandBuffer cmd)
        {
            cmd.ReleaseTemporaryRT(tempTexture.id);
        }

        public void Setup(RenderTargetIdentifier source)
        {
            this.source = source;
        }
    }

    FilmNoirRenderPass filmNoirPass;
    public Material filmNoirMaterial;

    public override void Create()
    {
        filmNoirPass = new FilmNoirRenderPass(filmNoirMaterial);
        filmNoirPass.renderPassEvent =
RenderPassEvent.BeforeRenderingPostProcessing;
    }

    public override void AddRenderPasses(ScriptableRenderer renderer, ref
RenderingData renderingData)
    {
        filmNoirPass.Setup(renderer.cameraColorTarget);
        renderer.EnqueuePass(filmNoirPass);
    }
}
```

Important

This script sets up the custom render pass for our Film Noir effect. Make sure to create a material using the `FilmNoirEffectShader` and assign it to the `filmNoirMaterial` field in the inspector.

Part 4: Setting Up the Effect

1. Create a new material in your project and name it `FilmNoirEffectMaterial`
2. Set the shader of this material to `Custom/FilmNoirEffect`
3. In your URP Renderer asset (usually found in `Assets/Settings`):
    - Add a new Renderer Feature
    - Select `FilmNoirRendererFeature`
    - Assign the `FilmNoirEffectMaterial` to the `Film Noir Material` field

Caution

Ensure that the material is correctly assigned in the Renderer Feature. If it's not, the effect won't be visible in your scene.

Part 5: Using the Effect

1. In your scene, add a `GLobal Volume`
    1. `Right-click in the Hierarchy > Volume > Global Volume`
2. Add an override for `Film Noir Effect`
3. Adjust the `Red Threshold`, `Contrast`, and `Brightness` parameters to achieve your desired look

Tip

Play with the `Red Threshold` value to control which shades of red are preserved. Lower values will preserve more reds, while higher values will result in a more stark black and white look with only the most vibrant reds showing through.

Conclusion

✓ **Success**

Congratulations! You now have a striking visual effect that can dramatically alter the mood of your scenes. This technique can be adapted to create various other stylized looks.