using System;

using UnityEngine;

using Random = UnityEngine.Random;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu("Image Effects/Noise/Noise and Scratches")]

public class NoiseAndScratches : MonoBehaviour

{

/// Monochrome noise just adds grain. Non-monochrome noise

/// more resembles VCR as it adds noise in YUV color space,

/// thus introducing magenta/green colors.

public bool monochrome = true;

private bool rgbFallback = false;

// Noise grain takes random intensity from Min to Max.

public float grainIntensityMin = 0.1f;

public float grainIntensityMax = 0.2f;

/// The size of the noise grains (1 = one pixel).

public float grainSize = 2.0f;

// Scratches take random intensity from Min to Max.

public float scratchIntensityMin = 0.05f;

public float scratchIntensityMax = 0.25f;

/// Scratches jump to another locations at this times per second.

public float scratchFPS = 10.0f;

/// While scratches are in the same location, they jitter a bit.

public float scratchJitter = 0.01f;

public Texture grainTexture;

public Texture scratchTexture;

public Shader shaderRGB;

public Shader shaderYUV;

private Material m\_MaterialRGB;

private Material m\_MaterialYUV;

private float scratchTimeLeft = 0.0f;

private float scratchX, scratchY;

protected void Start ()

{

// Disable if we don't support image effects

if (!SystemInfo.supportsImageEffects) {

enabled = false;

return;

}

if ( shaderRGB == null || shaderYUV == null )

{

Debug.Log( "Noise shaders are not set up! Disabling noise effect." );

enabled = false;

}

else

{

if ( !shaderRGB.isSupported ) // disable effect if RGB shader is not supported

enabled = false;

else if ( !shaderYUV.isSupported ) // fallback to RGB if YUV is not supported

rgbFallback = true;

}

}

protected Material material {

get {

if ( m\_MaterialRGB == null ) {

m\_MaterialRGB = new Material( shaderRGB );

m\_MaterialRGB.hideFlags = HideFlags.HideAndDontSave;

}

if ( m\_MaterialYUV == null && !rgbFallback ) {

m\_MaterialYUV = new Material( shaderYUV );

m\_MaterialYUV.hideFlags = HideFlags.HideAndDontSave;

}

return (!rgbFallback && !monochrome) ? m\_MaterialYUV : m\_MaterialRGB;

}

}

protected void OnDisable() {

if ( m\_MaterialRGB )

DestroyImmediate( m\_MaterialRGB );

if ( m\_MaterialYUV )

DestroyImmediate( m\_MaterialYUV );

}

private void SanitizeParameters()

{

grainIntensityMin = Mathf.Clamp( grainIntensityMin, 0.0f, 5.0f );

grainIntensityMax = Mathf.Clamp( grainIntensityMax, 0.0f, 5.0f );

scratchIntensityMin = Mathf.Clamp( scratchIntensityMin, 0.0f, 5.0f );

scratchIntensityMax = Mathf.Clamp( scratchIntensityMax, 0.0f, 5.0f );

scratchFPS = Mathf.Clamp( scratchFPS, 1, 30 );

scratchJitter = Mathf.Clamp( scratchJitter, 0.0f, 1.0f );

grainSize = Mathf.Clamp( grainSize, 0.1f, 50.0f );

}

// Called by the camera to apply the image effect

void OnRenderImage (RenderTexture source, RenderTexture destination)

{

SanitizeParameters();

if ( scratchTimeLeft <= 0.0f )

{

scratchTimeLeft = Random.value \* 2 / scratchFPS; // we have sanitized it earlier, won't be zero

scratchX = Random.value;

scratchY = Random.value;

}

scratchTimeLeft -= Time.deltaTime;

Material mat = material;

mat.SetTexture("\_GrainTex", grainTexture);

mat.SetTexture("\_ScratchTex", scratchTexture);

float grainScale = 1.0f / grainSize; // we have sanitized it earlier, won't be zero

mat.SetVector("\_GrainOffsetScale", new Vector4(

Random.value,

Random.value,

(float)Screen.width / (float)grainTexture.width \* grainScale,

(float)Screen.height / (float)grainTexture.height \* grainScale

));

mat.SetVector("\_ScratchOffsetScale", new Vector4(

scratchX + Random.value\*scratchJitter,

scratchY + Random.value\*scratchJitter,

(float)Screen.width / (float) scratchTexture.width,

(float)Screen.height / (float) scratchTexture.height

));

mat.SetVector("\_Intensity", new Vector4(

Random.Range(grainIntensityMin, grainIntensityMax),

Random.Range(scratchIntensityMin, scratchIntensityMax),

0, 0 ));

Graphics.Blit (source, destination, mat);

}

}

}