using System;

using UnityEngine;

using Random = UnityEngine.Random;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu ("Image Effects/Noise/Noise And Grain (Filmic)")]

public class NoiseAndGrain : PostEffectsBase

{

public float intensityMultiplier = 0.25f;

public float generalIntensity = 0.5f;

public float blackIntensity = 1.0f;

public float whiteIntensity = 1.0f;

public float midGrey = 0.2f;

public bool dx11Grain = false;

public float softness = 0.0f;

public bool monochrome = false;

public Vector3 intensities = new Vector3(1.0f, 1.0f, 1.0f);

public Vector3 tiling = new Vector3(64.0f, 64.0f, 64.0f);

public float monochromeTiling = 64.0f;

public FilterMode filterMode = FilterMode.Bilinear;

public Texture2D noiseTexture;

public Shader noiseShader;

private Material noiseMaterial = null;

public Shader dx11NoiseShader;

private Material dx11NoiseMaterial = null;

private static float TILE\_AMOUNT = 64.0f;

public override bool CheckResources ()

{

CheckSupport (false);

noiseMaterial = CheckShaderAndCreateMaterial (noiseShader, noiseMaterial);

if (dx11Grain && supportDX11)

{

#if UNITY\_EDITOR

dx11NoiseShader = Shader.Find("Hidden/NoiseAndGrainDX11");

#endif

dx11NoiseMaterial = CheckShaderAndCreateMaterial (dx11NoiseShader, dx11NoiseMaterial);

}

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

void OnRenderImage (RenderTexture source, RenderTexture destination)

{

if (CheckResources()==false || (null==noiseTexture))

{

Graphics.Blit (source, destination);

if (null==noiseTexture) {

Debug.LogWarning("Noise & Grain effect failing as noise texture is not assigned. please assign.", transform);

}

return;

}

softness = Mathf.Clamp(softness, 0.0f, 0.99f);

if (dx11Grain && supportDX11)

{

// We have a fancy, procedural noise pattern in this version, so no texture needed

dx11NoiseMaterial.SetFloat("\_DX11NoiseTime", Time.frameCount);

dx11NoiseMaterial.SetTexture ("\_NoiseTex", noiseTexture);

dx11NoiseMaterial.SetVector ("\_NoisePerChannel", monochrome ? Vector3.one : intensities);

dx11NoiseMaterial.SetVector ("\_MidGrey", new Vector3(midGrey, 1.0f/(1.0f-midGrey), -1.0f/midGrey));

dx11NoiseMaterial.SetVector ("\_NoiseAmount", new Vector3(generalIntensity, blackIntensity, whiteIntensity) \* intensityMultiplier);

if (softness > Mathf.Epsilon)

{

RenderTexture rt = RenderTexture.GetTemporary((int) (source.width \* (1.0f-softness)), (int) (source.height \* (1.0f-softness)));

DrawNoiseQuadGrid (source, rt, dx11NoiseMaterial, noiseTexture, monochrome ? 3 : 2);

dx11NoiseMaterial.SetTexture("\_NoiseTex", rt);

Graphics.Blit(source, destination, dx11NoiseMaterial, 4);

RenderTexture.ReleaseTemporary(rt);

}

else

DrawNoiseQuadGrid (source, destination, dx11NoiseMaterial, noiseTexture, (monochrome ? 1 : 0));

}

else

{

// normal noise (DX9 style)

if (noiseTexture) {

noiseTexture.wrapMode = TextureWrapMode.Repeat;

noiseTexture.filterMode = filterMode;

}

noiseMaterial.SetTexture ("\_NoiseTex", noiseTexture);

noiseMaterial.SetVector ("\_NoisePerChannel", monochrome ? Vector3.one : intensities);

noiseMaterial.SetVector ("\_NoiseTilingPerChannel", monochrome ? Vector3.one \* monochromeTiling : tiling);

noiseMaterial.SetVector ("\_MidGrey", new Vector3(midGrey, 1.0f/(1.0f-midGrey), -1.0f/midGrey));

noiseMaterial.SetVector ("\_NoiseAmount", new Vector3(generalIntensity, blackIntensity, whiteIntensity) \* intensityMultiplier);

if (softness > Mathf.Epsilon)

{

RenderTexture rt2 = RenderTexture.GetTemporary((int) (source.width \* (1.0f-softness)), (int) (source.height \* (1.0f-softness)));

DrawNoiseQuadGrid (source, rt2, noiseMaterial, noiseTexture, 2);

noiseMaterial.SetTexture("\_NoiseTex", rt2);

Graphics.Blit(source, destination, noiseMaterial, 1);

RenderTexture.ReleaseTemporary(rt2);

}

else

DrawNoiseQuadGrid (source, destination, noiseMaterial, noiseTexture, 0);

}

}

static void DrawNoiseQuadGrid (RenderTexture source, RenderTexture dest, Material fxMaterial, Texture2D noise, int passNr)

{

RenderTexture.active = dest;

float noiseSize = (noise.width \* 1.0f);

float subDs = (1.0f \* source.width) / TILE\_AMOUNT;

fxMaterial.SetTexture ("\_MainTex", source);

GL.PushMatrix ();

GL.LoadOrtho ();

float aspectCorrection = (1.0f \* source.width) / (1.0f \* source.height);

float stepSizeX = 1.0f / subDs;

float stepSizeY = stepSizeX \* aspectCorrection;

float texTile = noiseSize / (noise.width \* 1.0f);

fxMaterial.SetPass (passNr);

GL.Begin (GL.QUADS);

for (float x1 = 0.0f; x1 < 1.0f; x1 += stepSizeX)

{

for (float y1 = 0.0f; y1 < 1.0f; y1 += stepSizeY)

{

float tcXStart = Random.Range (0.0f, 1.0f);

float tcYStart = Random.Range (0.0f, 1.0f);

//Vector3 v3 = Random.insideUnitSphere;

//Color c = new Color(v3.x, v3.y, v3.z);

tcXStart = Mathf.Floor(tcXStart\*noiseSize) / noiseSize;

tcYStart = Mathf.Floor(tcYStart\*noiseSize) / noiseSize;

float texTileMod = 1.0f / noiseSize;

GL.MultiTexCoord2 (0, tcXStart, tcYStart);

GL.MultiTexCoord2 (1, 0.0f, 0.0f);

//GL.Color( c );

GL.Vertex3 (x1, y1, 0.1f);

GL.MultiTexCoord2 (0, tcXStart + texTile \* texTileMod, tcYStart);

GL.MultiTexCoord2 (1, 1.0f, 0.0f);

//GL.Color( c );

GL.Vertex3 (x1 + stepSizeX, y1, 0.1f);

GL.MultiTexCoord2 (0, tcXStart + texTile \* texTileMod, tcYStart + texTile \* texTileMod);

GL.MultiTexCoord2 (1, 1.0f, 1.0f);

//GL.Color( c );

GL.Vertex3 (x1 + stepSizeX, y1 + stepSizeY, 0.1f);

GL.MultiTexCoord2 (0, tcXStart, tcYStart + texTile \* texTileMod);

GL.MultiTexCoord2 (1, 0.0f, 1.0f);

//GL.Color( c );

GL.Vertex3 (x1, y1 + stepSizeY, 0.1f);

}

}

GL.End ();

GL.PopMatrix ();

}

}

}