using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

public enum LensflareStyle34

{

Ghosting = 0,

Anamorphic = 1,

Combined = 2,

}

public enum TweakMode34

{

Basic = 0,

Complex = 1,

}

public enum HDRBloomMode

{

Auto = 0,

On = 1,

Off = 2,

}

public enum BloomScreenBlendMode

{

Screen = 0,

Add = 1,

}

[ExecuteInEditMode]

[RequireComponent(typeof(Camera))]

[AddComponentMenu("Image Effects/Bloom and Glow/BloomAndFlares (3.5, Deprecated)")]

public class BloomAndFlares : PostEffectsBase

{

public TweakMode34 tweakMode = 0;

public BloomScreenBlendMode screenBlendMode = BloomScreenBlendMode.Add;

public HDRBloomMode hdr = HDRBloomMode.Auto;

private bool doHdr = false;

public float sepBlurSpread = 1.5f;

public float useSrcAlphaAsMask = 0.5f;

public float bloomIntensity = 1.0f;

public float bloomThreshold = 0.5f;

public int bloomBlurIterations = 2;

public bool lensflares = false;

public int hollywoodFlareBlurIterations = 2;

public LensflareStyle34 lensflareMode = (LensflareStyle34)1;

public float hollyStretchWidth = 3.5f;

public float lensflareIntensity = 1.0f;

public float lensflareThreshold = 0.3f;

public Color flareColorA = new Color(0.4f, 0.4f, 0.8f, 0.75f);

public Color flareColorB = new Color(0.4f, 0.8f, 0.8f, 0.75f);

public Color flareColorC = new Color(0.8f, 0.4f, 0.8f, 0.75f);

public Color flareColorD = new Color(0.8f, 0.4f, 0.0f, 0.75f);

public Texture2D lensFlareVignetteMask;

public Shader lensFlareShader;

private Material lensFlareMaterial;

public Shader vignetteShader;

private Material vignetteMaterial;

public Shader separableBlurShader;

private Material separableBlurMaterial;

public Shader addBrightStuffOneOneShader;

private Material addBrightStuffBlendOneOneMaterial;

public Shader screenBlendShader;

private Material screenBlend;

public Shader hollywoodFlaresShader;

private Material hollywoodFlaresMaterial;

public Shader brightPassFilterShader;

private Material brightPassFilterMaterial;

public override bool CheckResources()

{

CheckSupport(false);

screenBlend = CheckShaderAndCreateMaterial(screenBlendShader, screenBlend);

lensFlareMaterial = CheckShaderAndCreateMaterial(lensFlareShader, lensFlareMaterial);

vignetteMaterial = CheckShaderAndCreateMaterial(vignetteShader, vignetteMaterial);

separableBlurMaterial = CheckShaderAndCreateMaterial(separableBlurShader, separableBlurMaterial);

addBrightStuffBlendOneOneMaterial = CheckShaderAndCreateMaterial(addBrightStuffOneOneShader, addBrightStuffBlendOneOneMaterial);

hollywoodFlaresMaterial = CheckShaderAndCreateMaterial(hollywoodFlaresShader, hollywoodFlaresMaterial);

brightPassFilterMaterial = CheckShaderAndCreateMaterial(brightPassFilterShader, brightPassFilterMaterial);

if (!isSupported)

ReportAutoDisable();

return isSupported;

}

void OnRenderImage(RenderTexture source, RenderTexture destination)

{

if (CheckResources() == false)

{

Graphics.Blit(source, destination);

return;

}

// screen blend is not supported when HDR is enabled (will cap values)

doHdr = false;

if (hdr == HDRBloomMode.Auto)

doHdr = source.format == RenderTextureFormat.ARGBHalf && GetComponent<Camera>().hdr;

else

{

doHdr = hdr == HDRBloomMode.On;

}

doHdr = doHdr && supportHDRTextures;

BloomScreenBlendMode realBlendMode = screenBlendMode;

if (doHdr)

realBlendMode = BloomScreenBlendMode.Add;

var rtFormat = (doHdr) ? RenderTextureFormat.ARGBHalf : RenderTextureFormat.Default;

RenderTexture halfRezColor = RenderTexture.GetTemporary(source.width / 2, source.height / 2, 0, rtFormat);

RenderTexture quarterRezColor = RenderTexture.GetTemporary(source.width / 4, source.height / 4, 0, rtFormat);

RenderTexture secondQuarterRezColor = RenderTexture.GetTemporary(source.width / 4, source.height / 4, 0, rtFormat);

RenderTexture thirdQuarterRezColor = RenderTexture.GetTemporary(source.width / 4, source.height / 4, 0, rtFormat);

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

float oneOverBaseSize = 1.0f / 512.0f;

// downsample

Graphics.Blit(source, halfRezColor, screenBlend, 2); // <- 2 is stable downsample

Graphics.Blit(halfRezColor, quarterRezColor, screenBlend, 2); // <- 2 is stable downsample

RenderTexture.ReleaseTemporary(halfRezColor);

// cut colors (thresholding)

BrightFilter(bloomThreshold, useSrcAlphaAsMask, quarterRezColor, secondQuarterRezColor);

quarterRezColor.DiscardContents();

// blurring

if (bloomBlurIterations < 1) bloomBlurIterations = 1;

for (int iter = 0; iter < bloomBlurIterations; iter++)

{

float spreadForPass = (1.0f + (iter \* 0.5f)) \* sepBlurSpread;

separableBlurMaterial.SetVector("offsets", new Vector4(0.0f, spreadForPass \* oneOverBaseSize, 0.0f, 0.0f));

RenderTexture src = iter == 0 ? secondQuarterRezColor : quarterRezColor;

Graphics.Blit(src, thirdQuarterRezColor, separableBlurMaterial);

src.DiscardContents();

separableBlurMaterial.SetVector("offsets", new Vector4((spreadForPass / widthOverHeight) \* oneOverBaseSize, 0.0f, 0.0f, 0.0f));

Graphics.Blit(thirdQuarterRezColor, quarterRezColor, separableBlurMaterial);

thirdQuarterRezColor.DiscardContents();

}

// lens flares: ghosting, anamorphic or a combination

if (lensflares)

{

if (lensflareMode == 0)

{

BrightFilter(lensflareThreshold, 0.0f, quarterRezColor, thirdQuarterRezColor);

quarterRezColor.DiscardContents();

// smooth a little, this needs to be resolution dependent

/\*

separableBlurMaterial.SetVector ("offsets", Vector4 (0.0ff, (2.0ff) / (1.0ff \* quarterRezColor.height), 0.0ff, 0.0ff));

Graphics.Blit (thirdQuarterRezColor, secondQuarterRezColor, separableBlurMaterial);

separableBlurMaterial.SetVector ("offsets", Vector4 ((2.0ff) / (1.0ff \* quarterRezColor.width), 0.0ff, 0.0ff, 0.0ff));

Graphics.Blit (secondQuarterRezColor, thirdQuarterRezColor, separableBlurMaterial);

\*/

// no ugly edges!

Vignette(0.975f, thirdQuarterRezColor, secondQuarterRezColor);

thirdQuarterRezColor.DiscardContents();

BlendFlares(secondQuarterRezColor, quarterRezColor);

secondQuarterRezColor.DiscardContents();

}

// (b) hollywood/anamorphic flares?

else

{

// thirdQuarter has the brightcut unblurred colors

// quarterRezColor is the blurred, brightcut buffer that will end up as bloom

hollywoodFlaresMaterial.SetVector("\_threshold", new Vector4(lensflareThreshold, 1.0f / (1.0f - lensflareThreshold), 0.0f, 0.0f));

hollywoodFlaresMaterial.SetVector("tintColor", new Vector4(flareColorA.r, flareColorA.g, flareColorA.b, flareColorA.a) \* flareColorA.a \* lensflareIntensity);

Graphics.Blit(thirdQuarterRezColor, secondQuarterRezColor, hollywoodFlaresMaterial, 2);

thirdQuarterRezColor.DiscardContents();

Graphics.Blit(secondQuarterRezColor, thirdQuarterRezColor, hollywoodFlaresMaterial, 3);

secondQuarterRezColor.DiscardContents();

hollywoodFlaresMaterial.SetVector("offsets", new Vector4((sepBlurSpread \* 1.0f / widthOverHeight) \* oneOverBaseSize, 0.0f, 0.0f, 0.0f));

hollywoodFlaresMaterial.SetFloat("stretchWidth", hollyStretchWidth);

Graphics.Blit(thirdQuarterRezColor, secondQuarterRezColor, hollywoodFlaresMaterial, 1);

thirdQuarterRezColor.DiscardContents();

hollywoodFlaresMaterial.SetFloat("stretchWidth", hollyStretchWidth \* 2.0f);

Graphics.Blit(secondQuarterRezColor, thirdQuarterRezColor, hollywoodFlaresMaterial, 1);

secondQuarterRezColor.DiscardContents();

hollywoodFlaresMaterial.SetFloat("stretchWidth", hollyStretchWidth \* 4.0f);

Graphics.Blit(thirdQuarterRezColor, secondQuarterRezColor, hollywoodFlaresMaterial, 1);

thirdQuarterRezColor.DiscardContents();

if (lensflareMode == (LensflareStyle34)1)

{

for (int itera = 0; itera < hollywoodFlareBlurIterations; itera++)

{

separableBlurMaterial.SetVector("offsets", new Vector4((hollyStretchWidth \* 2.0f / widthOverHeight) \* oneOverBaseSize, 0.0f, 0.0f, 0.0f));

Graphics.Blit(secondQuarterRezColor, thirdQuarterRezColor, separableBlurMaterial);

secondQuarterRezColor.DiscardContents();

separableBlurMaterial.SetVector("offsets", new Vector4((hollyStretchWidth \* 2.0f / widthOverHeight) \* oneOverBaseSize, 0.0f, 0.0f, 0.0f));

Graphics.Blit(thirdQuarterRezColor, secondQuarterRezColor, separableBlurMaterial);

thirdQuarterRezColor.DiscardContents();

}

AddTo(1.0f, secondQuarterRezColor, quarterRezColor);

secondQuarterRezColor.DiscardContents();

}

else

{

// (c) combined

for (int ix = 0; ix < hollywoodFlareBlurIterations; ix++)

{

separableBlurMaterial.SetVector("offsets", new Vector4((hollyStretchWidth \* 2.0f / widthOverHeight) \* oneOverBaseSize, 0.0f, 0.0f, 0.0f));

Graphics.Blit(secondQuarterRezColor, thirdQuarterRezColor, separableBlurMaterial);

secondQuarterRezColor.DiscardContents();

separableBlurMaterial.SetVector("offsets", new Vector4((hollyStretchWidth \* 2.0f / widthOverHeight) \* oneOverBaseSize, 0.0f, 0.0f, 0.0f));

Graphics.Blit(thirdQuarterRezColor, secondQuarterRezColor, separableBlurMaterial);

thirdQuarterRezColor.DiscardContents();

}

Vignette(1.0f, secondQuarterRezColor, thirdQuarterRezColor);

secondQuarterRezColor.DiscardContents();

BlendFlares(thirdQuarterRezColor, secondQuarterRezColor);

thirdQuarterRezColor.DiscardContents();

AddTo(1.0f, secondQuarterRezColor, quarterRezColor);

secondQuarterRezColor.DiscardContents();

}

}

}

// screen blend bloom results to color buffer

screenBlend.SetFloat("\_Intensity", bloomIntensity);

screenBlend.SetTexture("\_ColorBuffer", source);

Graphics.Blit(quarterRezColor, destination, screenBlend, (int)realBlendMode);

RenderTexture.ReleaseTemporary(quarterRezColor);

RenderTexture.ReleaseTemporary(secondQuarterRezColor);

RenderTexture.ReleaseTemporary(thirdQuarterRezColor);

}

private void AddTo(float intensity\_, RenderTexture from, RenderTexture to)

{

addBrightStuffBlendOneOneMaterial.SetFloat("\_Intensity", intensity\_);

Graphics.Blit(from, to, addBrightStuffBlendOneOneMaterial);

}

private void BlendFlares(RenderTexture from, RenderTexture to)

{

lensFlareMaterial.SetVector("colorA", new Vector4(flareColorA.r, flareColorA.g, flareColorA.b, flareColorA.a) \* lensflareIntensity);

lensFlareMaterial.SetVector("colorB", new Vector4(flareColorB.r, flareColorB.g, flareColorB.b, flareColorB.a) \* lensflareIntensity);

lensFlareMaterial.SetVector("colorC", new Vector4(flareColorC.r, flareColorC.g, flareColorC.b, flareColorC.a) \* lensflareIntensity);

lensFlareMaterial.SetVector("colorD", new Vector4(flareColorD.r, flareColorD.g, flareColorD.b, flareColorD.a) \* lensflareIntensity);

Graphics.Blit(from, to, lensFlareMaterial);

}

private void BrightFilter(float thresh, float useAlphaAsMask, RenderTexture from, RenderTexture to)

{

if (doHdr)

brightPassFilterMaterial.SetVector("threshold", new Vector4(thresh, 1.0f, 0.0f, 0.0f));

else

brightPassFilterMaterial.SetVector("threshold", new Vector4(thresh, 1.0f / (1.0f - thresh), 0.0f, 0.0f));

brightPassFilterMaterial.SetFloat("useSrcAlphaAsMask", useAlphaAsMask);

Graphics.Blit(from, to, brightPassFilterMaterial);

}

private void Vignette(float amount, RenderTexture from, RenderTexture to)

{

if (lensFlareVignetteMask)

{

screenBlend.SetTexture("\_ColorBuffer", lensFlareVignetteMask);

Graphics.Blit(from, to, screenBlend, 3);

}

else

{

vignetteMaterial.SetFloat("vignetteIntensity", amount);

Graphics.Blit(from, to, vignetteMaterial);

}

}

}

}