using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu ("Image Effects/Bloom and Glow/Bloom")]

public class Bloom : PostEffectsBase

{

public enum LensFlareStyle

{

Ghosting = 0,

Anamorphic = 1,

Combined = 2,

}

public enum TweakMode

{

Basic = 0,

Complex = 1,

}

public enum HDRBloomMode

{

Auto = 0,

On = 1,

Off = 2,

}

public enum BloomScreenBlendMode

{

Screen = 0,

Add = 1,

}

public enum BloomQuality

{

Cheap = 0,

High = 1,

}

public TweakMode tweakMode = 0;

public BloomScreenBlendMode screenBlendMode = BloomScreenBlendMode.Add;

public HDRBloomMode hdr = HDRBloomMode.Auto;

private bool doHdr = false;

public float sepBlurSpread = 2.5f;

public BloomQuality quality = BloomQuality.High;

public float bloomIntensity = 0.5f;

public float bloomThreshold = 0.5f;

public Color bloomThresholdColor = Color.white;

public int bloomBlurIterations = 2;

public int hollywoodFlareBlurIterations = 2;

public float flareRotation = 0.0f;

public LensFlareStyle lensflareMode = (LensFlareStyle) 1;

public float hollyStretchWidth = 2.5f;

public float lensflareIntensity = 0.0f;

public float lensflareThreshold = 0.3f;

public float lensFlareSaturation = 0.75f;

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 screenBlendShader;

private Material screenBlend;

public Shader blurAndFlaresShader;

private Material blurAndFlaresMaterial;

public Shader brightPassFilterShader;

private Material brightPassFilterMaterial;

public override bool CheckResources ()

{

CheckSupport (false);

screenBlend = CheckShaderAndCreateMaterial (screenBlendShader, screenBlend);

lensFlareMaterial = CheckShaderAndCreateMaterial(lensFlareShader,lensFlareMaterial);

blurAndFlaresMaterial = CheckShaderAndCreateMaterial (blurAndFlaresShader, blurAndFlaresMaterial);

brightPassFilterMaterial = CheckShaderAndCreateMaterial(brightPassFilterShader, brightPassFilterMaterial);

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

public 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;

var rtW2= source.width/2;

var rtH2= source.height/2;

var rtW4= source.width/4;

var rtH4= source.height/4;

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

float oneOverBaseSize = 1.0f / 512.0f;

// downsample

RenderTexture quarterRezColor = RenderTexture.GetTemporary (rtW4, rtH4, 0, rtFormat);

RenderTexture halfRezColorDown = RenderTexture.GetTemporary (rtW2, rtH2, 0, rtFormat);

if (quality > BloomQuality.Cheap) {

Graphics.Blit (source, halfRezColorDown, screenBlend, 2);

RenderTexture rtDown4 = RenderTexture.GetTemporary (rtW4, rtH4, 0, rtFormat);

Graphics.Blit (halfRezColorDown, rtDown4, screenBlend, 2);

Graphics.Blit (rtDown4, quarterRezColor, screenBlend, 6);

RenderTexture.ReleaseTemporary(rtDown4);

}

else {

Graphics.Blit (source, halfRezColorDown);

Graphics.Blit (halfRezColorDown, quarterRezColor, screenBlend, 6);

}

RenderTexture.ReleaseTemporary (halfRezColorDown);

// cut colors (thresholding)

RenderTexture secondQuarterRezColor = RenderTexture.GetTemporary (rtW4, rtH4, 0, rtFormat);

BrightFilter (bloomThreshold \* bloomThresholdColor, quarterRezColor, secondQuarterRezColor);

// blurring

if (bloomBlurIterations < 1) bloomBlurIterations = 1;

else if (bloomBlurIterations > 10) bloomBlurIterations = 10;

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

{

float spreadForPass = (1.0f + (iter \* 0.25f)) \* sepBlurSpread;

// vertical blur

RenderTexture blur4 = RenderTexture.GetTemporary (rtW4, rtH4, 0, rtFormat);

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, spreadForPass \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (secondQuarterRezColor, blur4, blurAndFlaresMaterial, 4);

RenderTexture.ReleaseTemporary(secondQuarterRezColor);

secondQuarterRezColor = blur4;

// horizontal blur

blur4 = RenderTexture.GetTemporary (rtW4, rtH4, 0, rtFormat);

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

Graphics.Blit (secondQuarterRezColor, blur4, blurAndFlaresMaterial, 4);

RenderTexture.ReleaseTemporary (secondQuarterRezColor);

secondQuarterRezColor = blur4;

if (quality > BloomQuality.Cheap)

{

if (iter == 0)

{

Graphics.SetRenderTarget(quarterRezColor);

GL.Clear(false, true, Color.black); // Clear to avoid RT restore

Graphics.Blit (secondQuarterRezColor, quarterRezColor);

}

else

{

quarterRezColor.MarkRestoreExpected(); // using max blending, RT restore expected

Graphics.Blit (secondQuarterRezColor, quarterRezColor, screenBlend, 10);

}

}

}

if (quality > BloomQuality.Cheap)

{

Graphics.SetRenderTarget(secondQuarterRezColor);

GL.Clear(false, true, Color.black); // Clear to avoid RT restore

Graphics.Blit (quarterRezColor, secondQuarterRezColor, screenBlend, 6);

}

// lens flares: ghosting, anamorphic or both (ghosted anamorphic flares)

if (lensflareIntensity > Mathf.Epsilon)

{

RenderTexture rtFlares4 = RenderTexture.GetTemporary (rtW4, rtH4, 0, rtFormat);

if (lensflareMode == 0)

{

// ghosting only

BrightFilter (lensflareThreshold, secondQuarterRezColor, rtFlares4);

if (quality > BloomQuality.Cheap)

{

// smooth a little

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, (1.5f) / (1.0f \* quarterRezColor.height), 0.0f, 0.0f));

Graphics.SetRenderTarget(quarterRezColor);

GL.Clear(false, true, Color.black); // Clear to avoid RT restore

Graphics.Blit (rtFlares4, quarterRezColor, blurAndFlaresMaterial, 4);

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 ((1.5f) / (1.0f \* quarterRezColor.width), 0.0f, 0.0f, 0.0f));

Graphics.SetRenderTarget(rtFlares4);

GL.Clear(false, true, Color.black); // Clear to avoid RT restore

Graphics.Blit (quarterRezColor, rtFlares4, blurAndFlaresMaterial, 4);

}

// no ugly edges!

Vignette (0.975f, rtFlares4, rtFlares4);

BlendFlares (rtFlares4, secondQuarterRezColor);

}

else

{

//Vignette (0.975ff, rtFlares4, rtFlares4);

//DrawBorder(rtFlares4, screenBlend, 8);

float flareXRot = 1.0f \* Mathf.Cos(flareRotation);

float flareyRot = 1.0f \* Mathf.Sin(flareRotation);

float stretchWidth = (hollyStretchWidth \* 1.0f / widthOverHeight) \* oneOverBaseSize;

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 (flareXRot, flareyRot, 0.0f, 0.0f));

blurAndFlaresMaterial.SetVector ("\_Threshhold", new Vector4 (lensflareThreshold, 1.0f, 0.0f, 0.0f));

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

blurAndFlaresMaterial.SetFloat ("\_Saturation", lensFlareSaturation);

// "pre and cut"

quarterRezColor.DiscardContents();

Graphics.Blit (rtFlares4, quarterRezColor, blurAndFlaresMaterial, 2);

// "post"

rtFlares4.DiscardContents();

Graphics.Blit (quarterRezColor, rtFlares4, blurAndFlaresMaterial, 3);

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 (flareXRot \* stretchWidth, flareyRot \* stretchWidth, 0.0f, 0.0f));

// stretch 1st

blurAndFlaresMaterial.SetFloat ("\_StretchWidth", hollyStretchWidth);

quarterRezColor.DiscardContents();

Graphics.Blit (rtFlares4, quarterRezColor, blurAndFlaresMaterial, 1);

// stretch 2nd

blurAndFlaresMaterial.SetFloat ("\_StretchWidth", hollyStretchWidth \* 2.0f);

rtFlares4.DiscardContents();

Graphics.Blit (quarterRezColor, rtFlares4, blurAndFlaresMaterial, 1);

// stretch 3rd

blurAndFlaresMaterial.SetFloat ("\_StretchWidth", hollyStretchWidth \* 4.0f);

quarterRezColor.DiscardContents();

Graphics.Blit (rtFlares4, quarterRezColor, blurAndFlaresMaterial, 1);

// additional blur passes

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

{

stretchWidth = (hollyStretchWidth \* 2.0f / widthOverHeight) \* oneOverBaseSize;

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 (stretchWidth \* flareXRot, stretchWidth \* flareyRot, 0.0f, 0.0f));

rtFlares4.DiscardContents();

Graphics.Blit (quarterRezColor, rtFlares4, blurAndFlaresMaterial, 4);

blurAndFlaresMaterial.SetVector ("\_Offsets", new Vector4 (stretchWidth \* flareXRot, stretchWidth \* flareyRot, 0.0f, 0.0f));

quarterRezColor.DiscardContents();

Graphics.Blit (rtFlares4, quarterRezColor, blurAndFlaresMaterial, 4);

}

if (lensflareMode == (LensFlareStyle) 1)

// anamorphic lens flares

AddTo (1.0f, quarterRezColor, secondQuarterRezColor);

else

{

// "combined" lens flares

Vignette (1.0f, quarterRezColor, rtFlares4);

BlendFlares (rtFlares4, quarterRezColor);

AddTo (1.0f, quarterRezColor, secondQuarterRezColor);

}

}

RenderTexture.ReleaseTemporary (rtFlares4);

}

int blendPass = (int) realBlendMode;

//if (Mathf.Abs(chromaticBloom) < Mathf.Epsilon)

// blendPass += 4;

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

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

if (quality > BloomQuality.Cheap)

{

RenderTexture halfRezColorUp = RenderTexture.GetTemporary (rtW2, rtH2, 0, rtFormat);

Graphics.Blit (secondQuarterRezColor, halfRezColorUp);

Graphics.Blit (halfRezColorUp, destination, screenBlend, blendPass);

RenderTexture.ReleaseTemporary (halfRezColorUp);

}

else

Graphics.Blit (secondQuarterRezColor, destination, screenBlend, blendPass);

RenderTexture.ReleaseTemporary (quarterRezColor);

RenderTexture.ReleaseTemporary (secondQuarterRezColor);

}

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

{

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

to.MarkRestoreExpected(); // additive blending, RT restore expected

Graphics.Blit (from, to, screenBlend, 9);

}

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);

to.MarkRestoreExpected(); // additive blending, RT restore expected

Graphics.Blit (from, to, lensFlareMaterial);

}

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

{

brightPassFilterMaterial.SetVector ("\_Threshhold", new Vector4 (thresh, thresh, thresh, thresh));

Graphics.Blit (from, to, brightPassFilterMaterial, 0);

}

private void BrightFilter (Color threshColor, RenderTexture from, RenderTexture to)

{

brightPassFilterMaterial.SetVector ("\_Threshhold", threshColor);

Graphics.Blit (from, to, brightPassFilterMaterial, 1);

}

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

{

if (lensFlareVignetteMask)

{

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

to.MarkRestoreExpected(); // using blending, RT restore expected

Graphics.Blit (from == to ? null : from, to, screenBlend, from == to ? 7 : 3);

}

else if (from != to)

{

Graphics.SetRenderTarget (to);

GL.Clear(false, true, Color.black); // clear destination to avoid RT restore

Graphics.Blit (from, to);

}

}

}

}