using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu ("Image Effects/Camera/Depth of Field (deprecated)") ]

public class DepthOfFieldDeprecated : PostEffectsBase

{

public enum Dof34QualitySetting

{

OnlyBackground = 1,

BackgroundAndForeground = 2,

}

public enum DofResolution

{

High = 2,

Medium = 3,

Low = 4,

}

public enum DofBlurriness

{

Low = 1,

High = 2,

VeryHigh = 4,

}

public enum BokehDestination

{

Background = 0x1,

Foreground = 0x2,

BackgroundAndForeground = 0x3,

}

static private int SMOOTH\_DOWNSAMPLE\_PASS = 6;

static private float BOKEH\_EXTRA\_BLUR = 2.0f;

public Dof34QualitySetting quality = Dof34QualitySetting.OnlyBackground;

public DofResolution resolution = DofResolution.Low;

public bool simpleTweakMode = true;

public float focalPoint = 1.0f;

public float smoothness = 0.5f;

public float focalZDistance = 0.0f;

public float focalZStartCurve = 1.0f;

public float focalZEndCurve = 1.0f;

private float focalStartCurve = 2.0f;

private float focalEndCurve = 2.0f;

private float focalDistance01 = 0.1f;

public Transform objectFocus = null;

public float focalSize = 0.0f;

public DofBlurriness bluriness = DofBlurriness.High;

public float maxBlurSpread = 1.75f;

public float foregroundBlurExtrude = 1.15f;

public Shader dofBlurShader;

private Material dofBlurMaterial = null;

public Shader dofShader;

private Material dofMaterial = null;

public bool visualize = false;

public BokehDestination bokehDestination = BokehDestination.Background;

private float widthOverHeight = 1.25f;

private float oneOverBaseSize = 1.0f / 512.0f;

public bool bokeh = false;

public bool bokehSupport = true;

public Shader bokehShader;

public Texture2D bokehTexture;

public float bokehScale = 2.4f;

public float bokehIntensity = 0.15f;

public float bokehThresholdContrast = 0.1f;

public float bokehThresholdLuminance = 0.55f;

public int bokehDownsample = 1;

private Material bokehMaterial;

private Camera \_camera;

void CreateMaterials () {

dofBlurMaterial = CheckShaderAndCreateMaterial (dofBlurShader, dofBlurMaterial);

dofMaterial = CheckShaderAndCreateMaterial (dofShader,dofMaterial);

bokehSupport = bokehShader.isSupported;

if (bokeh && bokehSupport && bokehShader)

bokehMaterial = CheckShaderAndCreateMaterial (bokehShader, bokehMaterial);

}

public override bool CheckResources () {

CheckSupport (true);

dofBlurMaterial = CheckShaderAndCreateMaterial (dofBlurShader, dofBlurMaterial);

dofMaterial = CheckShaderAndCreateMaterial (dofShader,dofMaterial);

bokehSupport = bokehShader.isSupported;

if (bokeh && bokehSupport && bokehShader)

bokehMaterial = CheckShaderAndCreateMaterial (bokehShader, bokehMaterial);

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

void OnDisable () {

Quads.Cleanup ();

}

void OnEnable () {

\_camera = GetComponent<Camera>();

\_camera.depthTextureMode |= DepthTextureMode.Depth;

}

float FocalDistance01 ( float worldDist) {

return \_camera.WorldToViewportPoint((worldDist-\_camera.nearClipPlane) \* \_camera.transform.forward + \_camera.transform.position).z / (\_camera.farClipPlane-\_camera.nearClipPlane);

}

int GetDividerBasedOnQuality () {

int divider = 1;

if (resolution == DofResolution.Medium)

divider = 2;

else if (resolution == DofResolution.Low)

divider = 2;

return divider;

}

int GetLowResolutionDividerBasedOnQuality ( int baseDivider) {

int lowTexDivider = baseDivider;

if (resolution == DofResolution.High)

lowTexDivider \*= 2;

if (resolution == DofResolution.Low)

lowTexDivider \*= 2;

return lowTexDivider;

}

private RenderTexture foregroundTexture = null;

private RenderTexture mediumRezWorkTexture = null;

private RenderTexture finalDefocus = null;

private RenderTexture lowRezWorkTexture = null;

private RenderTexture bokehSource = null;

private RenderTexture bokehSource2 = null;

void OnRenderImage (RenderTexture source, RenderTexture destination) {

if (CheckResources()==false) {

Graphics.Blit (source, destination);

return;

}

if (smoothness < 0.1f)

smoothness = 0.1f;

// update needed focal & rt size parameter

bokeh = bokeh && bokehSupport;

float bokehBlurAmplifier = bokeh ? BOKEH\_EXTRA\_BLUR : 1.0f;

bool blurForeground = quality > Dof34QualitySetting.OnlyBackground;

float focal01Size = focalSize / (\_camera.farClipPlane - \_camera.nearClipPlane);;

if (simpleTweakMode) {

focalDistance01 = objectFocus ? (\_camera.WorldToViewportPoint (objectFocus.position)).z / (\_camera.farClipPlane) : FocalDistance01 (focalPoint);

focalStartCurve = focalDistance01 \* smoothness;

focalEndCurve = focalStartCurve;

blurForeground = blurForeground && (focalPoint > (\_camera.nearClipPlane + Mathf.Epsilon));

}

else {

if (objectFocus) {

var vpPoint= \_camera.WorldToViewportPoint (objectFocus.position);

vpPoint.z = (vpPoint.z) / (\_camera.farClipPlane);

focalDistance01 = vpPoint.z;

}

else

focalDistance01 = FocalDistance01 (focalZDistance);

focalStartCurve = focalZStartCurve;

focalEndCurve = focalZEndCurve;

blurForeground = blurForeground && (focalPoint > (\_camera.nearClipPlane + Mathf.Epsilon));

}

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

oneOverBaseSize = 1.0f / 512.0f;

dofMaterial.SetFloat ("\_ForegroundBlurExtrude", foregroundBlurExtrude);

dofMaterial.SetVector ("\_CurveParams", new Vector4 (simpleTweakMode ? 1.0f / focalStartCurve : focalStartCurve, simpleTweakMode ? 1.0f / focalEndCurve : focalEndCurve, focal01Size \* 0.5f, focalDistance01));

dofMaterial.SetVector ("\_InvRenderTargetSize", new Vector4 (1.0f / (1.0f \* source.width), 1.0f / (1.0f \* source.height),0.0f,0.0f));

int divider = GetDividerBasedOnQuality ();

int lowTexDivider = GetLowResolutionDividerBasedOnQuality (divider);

AllocateTextures (blurForeground, source, divider, lowTexDivider);

// WRITE COC to alpha channel

// source is only being bound to detect y texcoord flip

Graphics.Blit (source, source, dofMaterial, 3);

// better DOWNSAMPLE (could actually be weighted for higher quality)

Downsample (source, mediumRezWorkTexture);

// BLUR A LITTLE first, which has two purposes

// 1.) reduce jitter, noise, aliasing

// 2.) produce the little-blur buffer used in composition later

Blur (mediumRezWorkTexture, mediumRezWorkTexture, DofBlurriness.Low, 4, maxBlurSpread);

if ((bokeh) && ((BokehDestination.Foreground & bokehDestination) != 0))

{

dofMaterial.SetVector ("\_Threshhold", new Vector4(bokehThresholdContrast, bokehThresholdLuminance, 0.95f, 0.0f));

// add and mark the parts that should end up as bokeh shapes

Graphics.Blit (mediumRezWorkTexture, bokehSource2, dofMaterial, 11);

// remove those parts (maybe even a little tittle bittle more) from the regurlarly blurred buffer

//Graphics.Blit (mediumRezWorkTexture, lowRezWorkTexture, dofMaterial, 10);

Graphics.Blit (mediumRezWorkTexture, lowRezWorkTexture);//, dofMaterial, 10);

// maybe you want to reblur the small blur ... but not really needed.

//Blur (mediumRezWorkTexture, mediumRezWorkTexture, DofBlurriness.Low, 4, maxBlurSpread);

// bigger BLUR

Blur (lowRezWorkTexture, lowRezWorkTexture, bluriness, 0, maxBlurSpread \* bokehBlurAmplifier);

}

else {

// bigger BLUR

Downsample (mediumRezWorkTexture, lowRezWorkTexture);

Blur (lowRezWorkTexture, lowRezWorkTexture, bluriness, 0, maxBlurSpread);

}

dofBlurMaterial.SetTexture ("\_TapLow", lowRezWorkTexture);

dofBlurMaterial.SetTexture ("\_TapMedium", mediumRezWorkTexture);

Graphics.Blit (null, finalDefocus, dofBlurMaterial, 3);

// we are only adding bokeh now if the background is the only part we have to deal with

if ((bokeh) && ((BokehDestination.Foreground & bokehDestination) != 0))

AddBokeh (bokehSource2, bokehSource, finalDefocus);

dofMaterial.SetTexture ("\_TapLowBackground", finalDefocus);

dofMaterial.SetTexture ("\_TapMedium", mediumRezWorkTexture); // needed for debugging/visualization

// FINAL DEFOCUS (background)

Graphics.Blit (source, blurForeground ? foregroundTexture : destination, dofMaterial, visualize ? 2 : 0);

// FINAL DEFOCUS (foreground)

if (blurForeground) {

// WRITE COC to alpha channel

Graphics.Blit (foregroundTexture, source, dofMaterial, 5);

// DOWNSAMPLE (unweighted)

Downsample (source, mediumRezWorkTexture);

// BLUR A LITTLE first, which has two purposes

// 1.) reduce jitter, noise, aliasing

// 2.) produce the little-blur buffer used in composition later

BlurFg (mediumRezWorkTexture, mediumRezWorkTexture, DofBlurriness.Low, 2, maxBlurSpread);

if ((bokeh) && ((BokehDestination.Foreground & bokehDestination) != 0))

{

dofMaterial.SetVector ("\_Threshhold", new Vector4(bokehThresholdContrast \* 0.5f, bokehThresholdLuminance, 0.0f, 0.0f));

// add and mark the parts that should end up as bokeh shapes

Graphics.Blit (mediumRezWorkTexture, bokehSource2, dofMaterial, 11);

// remove the parts (maybe even a little tittle bittle more) that will end up in bokeh space

//Graphics.Blit (mediumRezWorkTexture, lowRezWorkTexture, dofMaterial, 10);

Graphics.Blit (mediumRezWorkTexture, lowRezWorkTexture);//, dofMaterial, 10);

// big BLUR

BlurFg (lowRezWorkTexture, lowRezWorkTexture, bluriness, 1, maxBlurSpread \* bokehBlurAmplifier);

}

else {

// big BLUR

BlurFg (mediumRezWorkTexture, lowRezWorkTexture, bluriness, 1, maxBlurSpread);

}

// simple upsample once

Graphics.Blit (lowRezWorkTexture, finalDefocus);

dofMaterial.SetTexture ("\_TapLowForeground", finalDefocus);

Graphics.Blit (source, destination, dofMaterial, visualize ? 1 : 4);

if ((bokeh) && ((BokehDestination.Foreground & bokehDestination) != 0))

AddBokeh (bokehSource2, bokehSource, destination);

}

ReleaseTextures ();

}

void Blur ( RenderTexture from, RenderTexture to, DofBlurriness iterations, int blurPass, float spread) {

RenderTexture tmp = RenderTexture.GetTemporary (to.width, to.height);

if ((int)iterations > 1) {

BlurHex (from, to, blurPass, spread, tmp);

if ((int)iterations > 2) {

dofBlurMaterial.SetVector ("offsets", new Vector4 (0.0f, spread \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (to, tmp, dofBlurMaterial, blurPass);

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

Graphics.Blit (tmp, to, dofBlurMaterial, blurPass);

}

}

else {

dofBlurMaterial.SetVector ("offsets", new Vector4 (0.0f, spread \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (from, tmp, dofBlurMaterial, blurPass);

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

Graphics.Blit (tmp, to, dofBlurMaterial, blurPass);

}

RenderTexture.ReleaseTemporary (tmp);

}

void BlurFg ( RenderTexture from, RenderTexture to, DofBlurriness iterations, int blurPass, float spread) {

// we want a nice, big coc, hence we need to tap once from this (higher resolution) texture

dofBlurMaterial.SetTexture ("\_TapHigh", from);

RenderTexture tmp = RenderTexture.GetTemporary (to.width, to.height);

if ((int)iterations > 1) {

BlurHex (from, to, blurPass, spread, tmp);

if ((int)iterations > 2) {

dofBlurMaterial.SetVector ("offsets", new Vector4 (0.0f, spread \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (to, tmp, dofBlurMaterial, blurPass);

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

Graphics.Blit (tmp, to, dofBlurMaterial, blurPass);

}

}

else {

dofBlurMaterial.SetVector ("offsets", new Vector4 (0.0f, spread \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (from, tmp, dofBlurMaterial, blurPass);

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

Graphics.Blit (tmp, to, dofBlurMaterial, blurPass);

}

RenderTexture.ReleaseTemporary (tmp);

}

void BlurHex ( RenderTexture from, RenderTexture to, int blurPass, float spread, RenderTexture tmp) {

dofBlurMaterial.SetVector ("offsets", new Vector4 (0.0f, spread \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (from, tmp, dofBlurMaterial, blurPass);

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

Graphics.Blit (tmp, to, dofBlurMaterial, blurPass);

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

Graphics.Blit (to, tmp, dofBlurMaterial, blurPass);

dofBlurMaterial.SetVector ("offsets", new Vector4 (spread / widthOverHeight \* oneOverBaseSize, -spread \* oneOverBaseSize, 0.0f, 0.0f));

Graphics.Blit (tmp, to, dofBlurMaterial, blurPass);

}

void Downsample ( RenderTexture from, RenderTexture to) {

dofMaterial.SetVector ("\_InvRenderTargetSize", new Vector4 (1.0f / (1.0f \* to.width), 1.0f / (1.0f \* to.height), 0.0f, 0.0f));

Graphics.Blit (from, to, dofMaterial, SMOOTH\_DOWNSAMPLE\_PASS);

}

void AddBokeh ( RenderTexture bokehInfo, RenderTexture tempTex, RenderTexture finalTarget) {

if (bokehMaterial) {

var meshes = Quads.GetMeshes (tempTex.width, tempTex.height); // quads: exchanging more triangles with less overdraw

RenderTexture.active = tempTex;

GL.Clear (false, true, new Color (0.0f, 0.0f, 0.0f, 0.0f));

GL.PushMatrix ();

GL.LoadIdentity ();

// point filter mode is important, otherwise we get bokeh shape & size artefacts

bokehInfo.filterMode = FilterMode.Point;

float arW = (bokehInfo.width \* 1.0f) / (bokehInfo.height \* 1.0f);

float sc = 2.0f / (1.0f \* bokehInfo.width);

sc += bokehScale \* maxBlurSpread \* BOKEH\_EXTRA\_BLUR \* oneOverBaseSize;

bokehMaterial.SetTexture ("\_Source", bokehInfo);

bokehMaterial.SetTexture ("\_MainTex", bokehTexture);

bokehMaterial.SetVector ("\_ArScale",new Vector4 (sc, sc \* arW, 0.5f, 0.5f \* arW));

bokehMaterial.SetFloat ("\_Intensity", bokehIntensity);

bokehMaterial.SetPass (0);

foreach(Mesh m in meshes)

if (m) Graphics.DrawMeshNow (m, Matrix4x4.identity);

GL.PopMatrix ();

Graphics.Blit (tempTex, finalTarget, dofMaterial, 8);

// important to set back as we sample from this later on

bokehInfo.filterMode = FilterMode.Bilinear;

}

}

void ReleaseTextures () {

if (foregroundTexture) RenderTexture.ReleaseTemporary (foregroundTexture);

if (finalDefocus) RenderTexture.ReleaseTemporary (finalDefocus);

if (mediumRezWorkTexture) RenderTexture.ReleaseTemporary (mediumRezWorkTexture);

if (lowRezWorkTexture) RenderTexture.ReleaseTemporary (lowRezWorkTexture);

if (bokehSource) RenderTexture.ReleaseTemporary (bokehSource);

if (bokehSource2) RenderTexture.ReleaseTemporary (bokehSource2);

}

void AllocateTextures ( bool blurForeground, RenderTexture source, int divider, int lowTexDivider) {

foregroundTexture = null;

if (blurForeground)

foregroundTexture = RenderTexture.GetTemporary (source.width, source.height, 0);

mediumRezWorkTexture = RenderTexture.GetTemporary (source.width / divider, source.height / divider, 0);

finalDefocus = RenderTexture.GetTemporary (source.width / divider, source.height / divider, 0);

lowRezWorkTexture = RenderTexture.GetTemporary (source.width / lowTexDivider, source.height / lowTexDivider, 0);

bokehSource = null;

bokehSource2 = null;

if (bokeh) {

bokehSource = RenderTexture.GetTemporary (source.width / (lowTexDivider \* bokehDownsample), source.height / (lowTexDivider \* bokehDownsample), 0, RenderTextureFormat.ARGBHalf);

bokehSource2 = RenderTexture.GetTemporary (source.width / (lowTexDivider \* bokehDownsample), source.height / (lowTexDivider \* bokehDownsample), 0, RenderTextureFormat.ARGBHalf);

bokehSource.filterMode = FilterMode.Bilinear;

bokehSource2.filterMode = FilterMode.Bilinear;

RenderTexture.active = bokehSource2;

GL.Clear (false, true, new Color(0.0f, 0.0f, 0.0f, 0.0f));

}

// to make sure: always use bilinear filter setting

source.filterMode = FilterMode.Bilinear;

finalDefocus.filterMode = FilterMode.Bilinear;

mediumRezWorkTexture.filterMode = FilterMode.Bilinear;

lowRezWorkTexture.filterMode = FilterMode.Bilinear;

if (foregroundTexture)

foregroundTexture.filterMode = FilterMode.Bilinear;

}

}

}