using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu ("Image Effects/Camera/Depth of Field (Lens Blur, Scatter, DX11)") ]

public class DepthOfField : PostEffectsBase {

public bool visualizeFocus = false;

public float focalLength = 10.0f;

public float focalSize = 0.05f;

public float aperture = 11.5f;

public Transform focalTransform = null;

public float maxBlurSize = 2.0f;

public bool highResolution = false;

public enum BlurType {

DiscBlur = 0,

DX11 = 1,

}

public enum BlurSampleCount {

Low = 0,

Medium = 1,

High = 2,

}

public BlurType blurType = BlurType.DiscBlur;

public BlurSampleCount blurSampleCount = BlurSampleCount.High;

public bool nearBlur = false;

public float foregroundOverlap = 1.0f;

public Shader dofHdrShader;

private Material dofHdrMaterial = null;

public Shader dx11BokehShader;

private Material dx11bokehMaterial;

public float dx11BokehThreshold = 0.5f;

public float dx11SpawnHeuristic = 0.0875f;

public Texture2D dx11BokehTexture = null;

public float dx11BokehScale = 1.2f;

public float dx11BokehIntensity = 2.5f;

private float focalDistance01 = 10.0f;

private ComputeBuffer cbDrawArgs;

private ComputeBuffer cbPoints;

private float internalBlurWidth = 1.0f;

public override bool CheckResources () {

CheckSupport (true); // only requires depth, not HDR

dofHdrMaterial = CheckShaderAndCreateMaterial (dofHdrShader, dofHdrMaterial);

if (supportDX11 && blurType == BlurType.DX11) {

dx11bokehMaterial = CheckShaderAndCreateMaterial(dx11BokehShader, dx11bokehMaterial);

CreateComputeResources ();

}

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

void OnEnable () {

GetComponent<Camera>().depthTextureMode |= DepthTextureMode.Depth;

}

void OnDisable () {

ReleaseComputeResources ();

if (dofHdrMaterial) DestroyImmediate(dofHdrMaterial);

dofHdrMaterial = null;

if (dx11bokehMaterial) DestroyImmediate(dx11bokehMaterial);

dx11bokehMaterial = null;

}

void ReleaseComputeResources () {

if (cbDrawArgs != null) cbDrawArgs.Release();

cbDrawArgs = null;

if (cbPoints != null) cbPoints.Release();

cbPoints = null;

}

void CreateComputeResources () {

if (cbDrawArgs == null)

{

cbDrawArgs = new ComputeBuffer (1, 16, ComputeBufferType.IndirectArguments);

var args= new int[4];

args[0] = 0; args[1] = 1; args[2] = 0; args[3] = 0;

cbDrawArgs.SetData (args);

}

if (cbPoints == null)

{

cbPoints = new ComputeBuffer (90000, 12+16, ComputeBufferType.Append);

}

}

float FocalDistance01 ( float worldDist) {

return GetComponent<Camera>().WorldToViewportPoint((worldDist-GetComponent<Camera>().nearClipPlane) \* GetComponent<Camera>().transform.forward + GetComponent<Camera>().transform.position).z / (GetComponent<Camera>().farClipPlane-GetComponent<Camera>().nearClipPlane);

}

private void WriteCoc ( RenderTexture fromTo, bool fgDilate) {

dofHdrMaterial.SetTexture("\_FgOverlap", null);

if (nearBlur && fgDilate) {

int rtW = fromTo.width/2;

int rtH = fromTo.height/2;

// capture fg coc

RenderTexture temp2 = RenderTexture.GetTemporary (rtW, rtH, 0, fromTo.format);

Graphics.Blit (fromTo, temp2, dofHdrMaterial, 4);

// special blur

float fgAdjustment = internalBlurWidth \* foregroundOverlap;

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, fgAdjustment , 0.0f, fgAdjustment));

RenderTexture temp1 = RenderTexture.GetTemporary (rtW, rtH, 0, fromTo.format);

Graphics.Blit (temp2, temp1, dofHdrMaterial, 2);

RenderTexture.ReleaseTemporary(temp2);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (fgAdjustment, 0.0f, 0.0f, fgAdjustment));

temp2 = RenderTexture.GetTemporary (rtW, rtH, 0, fromTo.format);

Graphics.Blit (temp1, temp2, dofHdrMaterial, 2);

RenderTexture.ReleaseTemporary(temp1);

// "merge up" with background COC

dofHdrMaterial.SetTexture("\_FgOverlap", temp2);

fromTo.MarkRestoreExpected(); // only touching alpha channel, RT restore expected

Graphics.Blit (fromTo, fromTo, dofHdrMaterial, 13);

RenderTexture.ReleaseTemporary(temp2);

}

else {

// capture full coc in alpha channel (fromTo is not read, but bound to detect screen flip)

fromTo.MarkRestoreExpected(); // only touching alpha channel, RT restore expected

Graphics.Blit (fromTo, fromTo, dofHdrMaterial, 0);

}

}

void OnRenderImage (RenderTexture source, RenderTexture destination) {

if (!CheckResources ()) {

Graphics.Blit (source, destination);

return;

}

// clamp & prepare values so they make sense

if (aperture < 0.0f) aperture = 0.0f;

if (maxBlurSize < 0.1f) maxBlurSize = 0.1f;

focalSize = Mathf.Clamp(focalSize, 0.0f, 2.0f);

internalBlurWidth = Mathf.Max(maxBlurSize, 0.0f);

// focal & coc calculations

focalDistance01 = (focalTransform) ? (GetComponent<Camera>().WorldToViewportPoint (focalTransform.position)).z / (GetComponent<Camera>().farClipPlane) : FocalDistance01 (focalLength);

dofHdrMaterial.SetVector ("\_CurveParams", new Vector4 (1.0f, focalSize, aperture/10.0f, focalDistance01));

// possible render texture helpers

RenderTexture rtLow = null;

RenderTexture rtLow2 = null;

RenderTexture rtSuperLow1 = null;

RenderTexture rtSuperLow2 = null;

float fgBlurDist = internalBlurWidth \* foregroundOverlap;

if (visualizeFocus)

{

//

// 2.

// visualize coc

//

//

WriteCoc (source, true);

Graphics.Blit (source, destination, dofHdrMaterial, 16);

}

else if ((blurType == BlurType.DX11) && dx11bokehMaterial)

{

//

// 1.

// optimized dx11 bokeh scatter

//

//

if (highResolution) {

internalBlurWidth = internalBlurWidth < 0.1f ? 0.1f : internalBlurWidth;

fgBlurDist = internalBlurWidth \* foregroundOverlap;

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

var dest2= RenderTexture.GetTemporary (source.width, source.height, 0, source.format);

// capture COC

WriteCoc (source, false);

// blur a bit so we can do a frequency check

rtSuperLow1 = RenderTexture.GetTemporary(source.width>>1, source.height>>1, 0, source.format);

rtSuperLow2 = RenderTexture.GetTemporary(source.width>>1, source.height>>1, 0, source.format);

Graphics.Blit(source, rtSuperLow1, dofHdrMaterial, 15);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, 1.5f , 0.0f, 1.5f));

Graphics.Blit (rtSuperLow1, rtSuperLow2, dofHdrMaterial, 19);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (1.5f, 0.0f, 0.0f, 1.5f));

Graphics.Blit (rtSuperLow2, rtSuperLow1, dofHdrMaterial, 19);

// capture fg coc

if (nearBlur)

Graphics.Blit (source, rtSuperLow2, dofHdrMaterial, 4);

dx11bokehMaterial.SetTexture ("\_BlurredColor", rtSuperLow1);

dx11bokehMaterial.SetFloat ("\_SpawnHeuristic", dx11SpawnHeuristic);

dx11bokehMaterial.SetVector ("\_BokehParams", new Vector4(dx11BokehScale, dx11BokehIntensity, Mathf.Clamp(dx11BokehThreshold, 0.005f, 4.0f), internalBlurWidth));

dx11bokehMaterial.SetTexture ("\_FgCocMask", nearBlur ? rtSuperLow2 : null);

// collect bokeh candidates and replace with a darker pixel

Graphics.SetRandomWriteTarget (1, cbPoints);

Graphics.Blit (source, rtLow, dx11bokehMaterial, 0);

Graphics.ClearRandomWriteTargets ();

// fg coc blur happens here (after collect!)

if (nearBlur) {

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, fgBlurDist , 0.0f, fgBlurDist));

Graphics.Blit (rtSuperLow2, rtSuperLow1, dofHdrMaterial, 2);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (fgBlurDist, 0.0f, 0.0f, fgBlurDist));

Graphics.Blit (rtSuperLow1, rtSuperLow2, dofHdrMaterial, 2);

// merge fg coc with bg coc

Graphics.Blit (rtSuperLow2, rtLow, dofHdrMaterial, 3);

}

// NEW: LAY OUT ALPHA on destination target so we get nicer outlines for the high rez version

Graphics.Blit (rtLow, dest2, dofHdrMaterial, 20);

// box blur (easier to merge with bokeh buffer)

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (internalBlurWidth, 0.0f , 0.0f, internalBlurWidth));

Graphics.Blit (rtLow, source, dofHdrMaterial, 5);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, internalBlurWidth, 0.0f, internalBlurWidth));

Graphics.Blit (source, dest2, dofHdrMaterial, 21);

// apply bokeh candidates

Graphics.SetRenderTarget (dest2);

ComputeBuffer.CopyCount (cbPoints, cbDrawArgs, 0);

dx11bokehMaterial.SetBuffer ("pointBuffer", cbPoints);

dx11bokehMaterial.SetTexture ("\_MainTex", dx11BokehTexture);

dx11bokehMaterial.SetVector ("\_Screen", new Vector3(1.0f/(1.0f\*source.width), 1.0f/(1.0f\*source.height), internalBlurWidth));

dx11bokehMaterial.SetPass (2);

Graphics.DrawProceduralIndirect (MeshTopology.Points, cbDrawArgs, 0);

Graphics.Blit (dest2, destination); // hackaround for DX11 high resolution flipfun (OPTIMIZEME)

RenderTexture.ReleaseTemporary(dest2);

RenderTexture.ReleaseTemporary(rtSuperLow1);

RenderTexture.ReleaseTemporary(rtSuperLow2);

}

else {

rtLow = RenderTexture.GetTemporary (source.width>>1, source.height>>1, 0, source.format);

rtLow2 = RenderTexture.GetTemporary (source.width>>1, source.height>>1, 0, source.format);

fgBlurDist = internalBlurWidth \* foregroundOverlap;

// capture COC & color in low resolution

WriteCoc (source, false);

source.filterMode = FilterMode.Bilinear;

Graphics.Blit (source, rtLow, dofHdrMaterial, 6);

// blur a bit so we can do a frequency check

rtSuperLow1 = RenderTexture.GetTemporary(rtLow.width>>1, rtLow.height>>1, 0, rtLow.format);

rtSuperLow2 = RenderTexture.GetTemporary(rtLow.width>>1, rtLow.height>>1, 0, rtLow.format);

Graphics.Blit(rtLow, rtSuperLow1, dofHdrMaterial, 15);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, 1.5f , 0.0f, 1.5f));

Graphics.Blit (rtSuperLow1, rtSuperLow2, dofHdrMaterial, 19);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (1.5f, 0.0f, 0.0f, 1.5f));

Graphics.Blit (rtSuperLow2, rtSuperLow1, dofHdrMaterial, 19);

RenderTexture rtLow3 = null;

if (nearBlur) {

// capture fg coc

rtLow3 = RenderTexture.GetTemporary (source.width>>1, source.height>>1, 0, source.format);

Graphics.Blit (source, rtLow3, dofHdrMaterial, 4);

}

dx11bokehMaterial.SetTexture ("\_BlurredColor", rtSuperLow1);

dx11bokehMaterial.SetFloat ("\_SpawnHeuristic", dx11SpawnHeuristic);

dx11bokehMaterial.SetVector ("\_BokehParams", new Vector4(dx11BokehScale, dx11BokehIntensity, Mathf.Clamp(dx11BokehThreshold, 0.005f, 4.0f), internalBlurWidth));

dx11bokehMaterial.SetTexture ("\_FgCocMask", rtLow3);

// collect bokeh candidates and replace with a darker pixel

Graphics.SetRandomWriteTarget (1, cbPoints);

Graphics.Blit (rtLow, rtLow2, dx11bokehMaterial, 0);

Graphics.ClearRandomWriteTargets ();

RenderTexture.ReleaseTemporary(rtSuperLow1);

RenderTexture.ReleaseTemporary(rtSuperLow2);

// fg coc blur happens here (after collect!)

if (nearBlur) {

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, fgBlurDist , 0.0f, fgBlurDist));

Graphics.Blit (rtLow3, rtLow, dofHdrMaterial, 2);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (fgBlurDist, 0.0f, 0.0f, fgBlurDist));

Graphics.Blit (rtLow, rtLow3, dofHdrMaterial, 2);

// merge fg coc with bg coc

Graphics.Blit (rtLow3, rtLow2, dofHdrMaterial, 3);

}

// box blur (easier to merge with bokeh buffer)

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (internalBlurWidth, 0.0f , 0.0f, internalBlurWidth));

Graphics.Blit (rtLow2, rtLow, dofHdrMaterial, 5);

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, internalBlurWidth, 0.0f, internalBlurWidth));

Graphics.Blit (rtLow, rtLow2, dofHdrMaterial, 5);

// apply bokeh candidates

Graphics.SetRenderTarget (rtLow2);

ComputeBuffer.CopyCount (cbPoints, cbDrawArgs, 0);

dx11bokehMaterial.SetBuffer ("pointBuffer", cbPoints);

dx11bokehMaterial.SetTexture ("\_MainTex", dx11BokehTexture);

dx11bokehMaterial.SetVector ("\_Screen", new Vector3(1.0f/(1.0f\*rtLow2.width), 1.0f/(1.0f\*rtLow2.height), internalBlurWidth));

dx11bokehMaterial.SetPass (1);

Graphics.DrawProceduralIndirect (MeshTopology.Points, cbDrawArgs, 0);

// upsample & combine

dofHdrMaterial.SetTexture ("\_LowRez", rtLow2);

dofHdrMaterial.SetTexture ("\_FgOverlap", rtLow3);

dofHdrMaterial.SetVector ("\_Offsets", ((1.0f\*source.width)/(1.0f\*rtLow2.width)) \* internalBlurWidth \* Vector4.one);

Graphics.Blit (source, destination, dofHdrMaterial, 9);

if (rtLow3) RenderTexture.ReleaseTemporary(rtLow3);

}

}

else

{

//

// 2.

// poisson disc style blur in low resolution

//

//

source.filterMode = FilterMode.Bilinear;

if (highResolution) internalBlurWidth \*= 2.0f;

WriteCoc (source, true);

rtLow = RenderTexture.GetTemporary (source.width >> 1, source.height >> 1, 0, source.format);

rtLow2 = RenderTexture.GetTemporary (source.width >> 1, source.height >> 1, 0, source.format);

int blurPass = (blurSampleCount == BlurSampleCount.High || blurSampleCount == BlurSampleCount.Medium) ? 17 : 11;

if (highResolution) {

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, internalBlurWidth, 0.025f, internalBlurWidth));

Graphics.Blit (source, destination, dofHdrMaterial, blurPass);

}

else {

dofHdrMaterial.SetVector ("\_Offsets", new Vector4 (0.0f, internalBlurWidth, 0.1f, internalBlurWidth));

// blur

Graphics.Blit (source, rtLow, dofHdrMaterial, 6);

Graphics.Blit (rtLow, rtLow2, dofHdrMaterial, blurPass);

// cheaper blur in high resolution, upsample and combine

dofHdrMaterial.SetTexture("\_LowRez", rtLow2);

dofHdrMaterial.SetTexture("\_FgOverlap", null);

dofHdrMaterial.SetVector ("\_Offsets", Vector4.one \* ((1.0f\*source.width)/(1.0f\*rtLow2.width)) \* internalBlurWidth);

Graphics.Blit (source, destination, dofHdrMaterial, blurSampleCount == BlurSampleCount.High ? 18 : 12);

}

}

if (rtLow) RenderTexture.ReleaseTemporary(rtLow);

if (rtLow2) RenderTexture.ReleaseTemporary(rtLow2);

}

}

}