using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[AddComponentMenu("Image Effects/Blur/Blur")]

public class Blur : MonoBehaviour

{

/// Blur iterations - larger number means more blur.

public int iterations = 3;

/// Blur spread for each iteration. Lower values

/// give better looking blur, but require more iterations to

/// get large blurs. Value is usually between 0.5 and 1.0.

public float blurSpread = 0.6f;

// --------------------------------------------------------

// The blur iteration shader.

// Basically it just takes 4 texture samples and averages them.

// By applying it repeatedly and spreading out sample locations

// we get a Gaussian blur approximation.

public Shader blurShader = null;

static Material m\_Material = null;

protected Material material {

get {

if (m\_Material == null) {

m\_Material = new Material(blurShader);

m\_Material.hideFlags = HideFlags.DontSave;

}

return m\_Material;

}

}

protected void OnDisable() {

if ( m\_Material ) {

DestroyImmediate( m\_Material );

}

}

// --------------------------------------------------------

protected void Start()

{

// Disable if we don't support image effects

if (!SystemInfo.supportsImageEffects) {

enabled = false;

return;

}

// Disable if the shader can't run on the users graphics card

if (!blurShader || !material.shader.isSupported) {

enabled = false;

return;

}

}

// Performs one blur iteration.

public void FourTapCone (RenderTexture source, RenderTexture dest, int iteration)

{

float off = 0.5f + iteration\*blurSpread;

Graphics.BlitMultiTap (source, dest, material,

new Vector2(-off, -off),

new Vector2(-off, off),

new Vector2( off, off),

new Vector2( off, -off)

);

}

// Downsamples the texture to a quarter resolution.

private void DownSample4x (RenderTexture source, RenderTexture dest)

{

float off = 1.0f;

Graphics.BlitMultiTap (source, dest, material,

new Vector2(-off, -off),

new Vector2(-off, off),

new Vector2( off, off),

new Vector2( off, -off)

);

}

// Called by the camera to apply the image effect

void OnRenderImage (RenderTexture source, RenderTexture destination) {

int rtW = source.width/4;

int rtH = source.height/4;

RenderTexture buffer = RenderTexture.GetTemporary(rtW, rtH, 0);

// Copy source to the 4x4 smaller texture.

DownSample4x (source, buffer);

// Blur the small texture

for(int i = 0; i < iterations; i++)

{

RenderTexture buffer2 = RenderTexture.GetTemporary(rtW, rtH, 0);

FourTapCone (buffer, buffer2, i);

RenderTexture.ReleaseTemporary(buffer);

buffer = buffer2;

}

Graphics.Blit(buffer, destination);

RenderTexture.ReleaseTemporary(buffer);

}

}

}