using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent(typeof (Camera))]

[AddComponentMenu("Image Effects/Color Adjustments/Tonemapping")]

public class Tonemapping : PostEffectsBase

{

public enum TonemapperType

{

SimpleReinhard,

UserCurve,

Hable,

Photographic,

OptimizedHejiDawson,

AdaptiveReinhard,

AdaptiveReinhardAutoWhite,

};

public enum AdaptiveTexSize

{

Square16 = 16,

Square32 = 32,

Square64 = 64,

Square128 = 128,

Square256 = 256,

Square512 = 512,

Square1024 = 1024,

};

public TonemapperType type = TonemapperType.Photographic;

public AdaptiveTexSize adaptiveTextureSize = AdaptiveTexSize.Square256;

// CURVE parameter

public AnimationCurve remapCurve;

private Texture2D curveTex = null;

// UNCHARTED parameter

public float exposureAdjustment = 1.5f;

// REINHARD parameter

public float middleGrey = 0.4f;

public float white = 2.0f;

public float adaptionSpeed = 1.5f;

// usual & internal stuff

public Shader tonemapper = null;

public bool validRenderTextureFormat = true;

private Material tonemapMaterial = null;

private RenderTexture rt = null;

private RenderTextureFormat rtFormat = RenderTextureFormat.ARGBHalf;

public override bool CheckResources()

{

CheckSupport(false, true);

tonemapMaterial = CheckShaderAndCreateMaterial(tonemapper, tonemapMaterial);

if (!curveTex && type == TonemapperType.UserCurve)

{

curveTex = new Texture2D(256, 1, TextureFormat.ARGB32, false, true);

curveTex.filterMode = FilterMode.Bilinear;

curveTex.wrapMode = TextureWrapMode.Clamp;

curveTex.hideFlags = HideFlags.DontSave;

}

if (!isSupported)

ReportAutoDisable();

return isSupported;

}

public float UpdateCurve()

{

float range = 1.0f;

if (remapCurve.keys.Length < 1)

remapCurve = new AnimationCurve(new Keyframe(0, 0), new Keyframe(2, 1));

if (remapCurve != null)

{

if (remapCurve.length > 0)

range = remapCurve[remapCurve.length - 1].time;

for (float i = 0.0f; i <= 1.0f; i += 1.0f/255.0f)

{

float c = remapCurve.Evaluate(i\*1.0f\*range);

curveTex.SetPixel((int) Mathf.Floor(i\*255.0f), 0, new Color(c, c, c));

}

curveTex.Apply();

}

return 1.0f/range;

}

private void OnDisable()

{

if (rt)

{

DestroyImmediate(rt);

rt = null;

}

if (tonemapMaterial)

{

DestroyImmediate(tonemapMaterial);

tonemapMaterial = null;

}

if (curveTex)

{

DestroyImmediate(curveTex);

curveTex = null;

}

}

private bool CreateInternalRenderTexture()

{

if (rt)

{

return false;

}

rtFormat = SystemInfo.SupportsRenderTextureFormat(RenderTextureFormat.RGHalf) ? RenderTextureFormat.RGHalf : RenderTextureFormat.ARGBHalf;

rt = new RenderTexture(1, 1, 0, rtFormat);

rt.hideFlags = HideFlags.DontSave;

return true;

}

// attribute indicates that the image filter chain will continue in LDR

[ImageEffectTransformsToLDR]

private void OnRenderImage(RenderTexture source, RenderTexture destination)

{

if (CheckResources() == false)

{

Graphics.Blit(source, destination);

return;

}

#if UNITY\_EDITOR

validRenderTextureFormat = true;

if (source.format != RenderTextureFormat.ARGBHalf)

{

validRenderTextureFormat = false;

}

#endif

// clamp some values to not go out of a valid range

exposureAdjustment = exposureAdjustment < 0.001f ? 0.001f : exposureAdjustment;

// SimpleReinhard tonemappers (local, non adaptive)

if (type == TonemapperType.UserCurve)

{

float rangeScale = UpdateCurve();

tonemapMaterial.SetFloat("\_RangeScale", rangeScale);

tonemapMaterial.SetTexture("\_Curve", curveTex);

Graphics.Blit(source, destination, tonemapMaterial, 4);

return;

}

if (type == TonemapperType.SimpleReinhard)

{

tonemapMaterial.SetFloat("\_ExposureAdjustment", exposureAdjustment);

Graphics.Blit(source, destination, tonemapMaterial, 6);

return;

}

if (type == TonemapperType.Hable)

{

tonemapMaterial.SetFloat("\_ExposureAdjustment", exposureAdjustment);

Graphics.Blit(source, destination, tonemapMaterial, 5);

return;

}

if (type == TonemapperType.Photographic)

{

tonemapMaterial.SetFloat("\_ExposureAdjustment", exposureAdjustment);

Graphics.Blit(source, destination, tonemapMaterial, 8);

return;

}

if (type == TonemapperType.OptimizedHejiDawson)

{

tonemapMaterial.SetFloat("\_ExposureAdjustment", 0.5f\*exposureAdjustment);

Graphics.Blit(source, destination, tonemapMaterial, 7);

return;

}

// still here?

// => adaptive tone mapping:

// builds an average log luminance, tonemaps according to

// middle grey and white values (user controlled)

// AdaptiveReinhardAutoWhite will calculate white value automagically

bool freshlyBrewedInternalRt = CreateInternalRenderTexture(); // this retrieves rtFormat, so should happen before rt allocations

RenderTexture rtSquared = RenderTexture.GetTemporary((int) adaptiveTextureSize, (int) adaptiveTextureSize, 0, rtFormat);

Graphics.Blit(source, rtSquared);

int downsample = (int) Mathf.Log(rtSquared.width\*1.0f, 2);

int div = 2;

var rts = new RenderTexture[downsample];

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

{

rts[i] = RenderTexture.GetTemporary(rtSquared.width/div, rtSquared.width/div, 0, rtFormat);

div \*= 2;

}

// downsample pyramid

var lumRt = rts[downsample - 1];

Graphics.Blit(rtSquared, rts[0], tonemapMaterial, 1);

if (type == TonemapperType.AdaptiveReinhardAutoWhite)

{

for (int i = 0; i < downsample - 1; i++)

{

Graphics.Blit(rts[i], rts[i + 1], tonemapMaterial, 9);

lumRt = rts[i + 1];

}

}

else if (type == TonemapperType.AdaptiveReinhard)

{

for (int i = 0; i < downsample - 1; i++)

{

Graphics.Blit(rts[i], rts[i + 1]);

lumRt = rts[i + 1];

}

}

// we have the needed values, let's apply adaptive tonemapping

adaptionSpeed = adaptionSpeed < 0.001f ? 0.001f : adaptionSpeed;

tonemapMaterial.SetFloat("\_AdaptionSpeed", adaptionSpeed);

rt.MarkRestoreExpected(); // keeping luminance values between frames, RT restore expected

#if UNITY\_EDITOR

if (Application.isPlaying && !freshlyBrewedInternalRt)

Graphics.Blit(lumRt, rt, tonemapMaterial, 2);

else

Graphics.Blit(lumRt, rt, tonemapMaterial, 3);

#else

Graphics.Blit (lumRt, rt, tonemapMaterial, freshlyBrewedInternalRt ? 3 : 2);

#endif

middleGrey = middleGrey < 0.001f ? 0.001f : middleGrey;

tonemapMaterial.SetVector("\_HdrParams", new Vector4(middleGrey, middleGrey, middleGrey, white\*white));

tonemapMaterial.SetTexture("\_SmallTex", rt);

if (type == TonemapperType.AdaptiveReinhard)

{

Graphics.Blit(source, destination, tonemapMaterial, 0);

}

else if (type == TonemapperType.AdaptiveReinhardAutoWhite)

{

Graphics.Blit(source, destination, tonemapMaterial, 10);

}

else

{

Debug.LogError("No valid adaptive tonemapper type found!");

Graphics.Blit(source, destination); // at least we get the TransformToLDR effect

}

// cleanup for adaptive

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

{

RenderTexture.ReleaseTemporary(rts[i]);

}

RenderTexture.ReleaseTemporary(rtSquared);

}

}

}