using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[AddComponentMenu ("Image Effects/Color Adjustments/Color Correction (Curves, Saturation)")]

public class ColorCorrectionCurves : PostEffectsBase

{

public enum ColorCorrectionMode

{

Simple = 0,

Advanced = 1

}

public AnimationCurve redChannel = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

public AnimationCurve greenChannel = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

public AnimationCurve blueChannel = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

public bool useDepthCorrection = false;

public AnimationCurve zCurve = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

public AnimationCurve depthRedChannel = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

public AnimationCurve depthGreenChannel = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

public AnimationCurve depthBlueChannel = new AnimationCurve(new Keyframe(0f,0f), new Keyframe(1f,1f));

private Material ccMaterial;

private Material ccDepthMaterial;

private Material selectiveCcMaterial;

private Texture2D rgbChannelTex;

private Texture2D rgbDepthChannelTex;

private Texture2D zCurveTex;

public float saturation = 1.0f;

public bool selectiveCc = false;

public Color selectiveFromColor = Color.white;

public Color selectiveToColor = Color.white;

public ColorCorrectionMode mode;

public bool updateTextures = true;

public Shader colorCorrectionCurvesShader = null;

public Shader simpleColorCorrectionCurvesShader = null;

public Shader colorCorrectionSelectiveShader = null;

private bool updateTexturesOnStartup = true;

new void Start ()

{

base.Start ();

updateTexturesOnStartup = true;

}

void Awake () { }

public override bool CheckResources ()

{

CheckSupport (mode == ColorCorrectionMode.Advanced);

ccMaterial = CheckShaderAndCreateMaterial (simpleColorCorrectionCurvesShader, ccMaterial);

ccDepthMaterial = CheckShaderAndCreateMaterial (colorCorrectionCurvesShader, ccDepthMaterial);

selectiveCcMaterial = CheckShaderAndCreateMaterial (colorCorrectionSelectiveShader, selectiveCcMaterial);

if (!rgbChannelTex)

rgbChannelTex = new Texture2D (256, 4, TextureFormat.ARGB32, false, true);

if (!rgbDepthChannelTex)

rgbDepthChannelTex = new Texture2D (256, 4, TextureFormat.ARGB32, false, true);

if (!zCurveTex)

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

rgbChannelTex.hideFlags = HideFlags.DontSave;

rgbDepthChannelTex.hideFlags = HideFlags.DontSave;

zCurveTex.hideFlags = HideFlags.DontSave;

rgbChannelTex.wrapMode = TextureWrapMode.Clamp;

rgbDepthChannelTex.wrapMode = TextureWrapMode.Clamp;

zCurveTex.wrapMode = TextureWrapMode.Clamp;

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

public void UpdateParameters ()

{

CheckResources(); // textures might not be created if we're tweaking UI while disabled

if (redChannel != null && greenChannel != null && blueChannel != null)

{

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

{

float rCh = Mathf.Clamp (redChannel.Evaluate(i), 0.0f, 1.0f);

float gCh = Mathf.Clamp (greenChannel.Evaluate(i), 0.0f, 1.0f);

float bCh = Mathf.Clamp (blueChannel.Evaluate(i), 0.0f, 1.0f);

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

rgbChannelTex.SetPixel ((int) Mathf.Floor(i\*255.0f), 1, new Color(gCh,gCh,gCh) );

rgbChannelTex.SetPixel ((int) Mathf.Floor(i\*255.0f), 2, new Color(bCh,bCh,bCh) );

float zC = Mathf.Clamp (zCurve.Evaluate(i), 0.0f,1.0f);

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

rCh = Mathf.Clamp (depthRedChannel.Evaluate(i), 0.0f,1.0f);

gCh = Mathf.Clamp (depthGreenChannel.Evaluate(i), 0.0f,1.0f);

bCh = Mathf.Clamp (depthBlueChannel.Evaluate(i), 0.0f,1.0f);

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

rgbDepthChannelTex.SetPixel ((int) Mathf.Floor(i\*255.0f), 1, new Color(gCh,gCh,gCh) );

rgbDepthChannelTex.SetPixel ((int) Mathf.Floor(i\*255.0f), 2, new Color(bCh,bCh,bCh) );

}

rgbChannelTex.Apply ();

rgbDepthChannelTex.Apply ();

zCurveTex.Apply ();

}

}

void UpdateTextures ()

{

UpdateParameters ();

}

void OnRenderImage (RenderTexture source, RenderTexture destination)

{

if (CheckResources()==false)

{

Graphics.Blit (source, destination);

return;

}

if (updateTexturesOnStartup)

{

UpdateParameters ();

updateTexturesOnStartup = false;

}

if (useDepthCorrection)

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

RenderTexture renderTarget2Use = destination;

if (selectiveCc)

{

renderTarget2Use = RenderTexture.GetTemporary (source.width, source.height);

}

if (useDepthCorrection)

{

ccDepthMaterial.SetTexture ("\_RgbTex", rgbChannelTex);

ccDepthMaterial.SetTexture ("\_ZCurve", zCurveTex);

ccDepthMaterial.SetTexture ("\_RgbDepthTex", rgbDepthChannelTex);

ccDepthMaterial.SetFloat ("\_Saturation", saturation);

Graphics.Blit (source, renderTarget2Use, ccDepthMaterial);

}

else

{

ccMaterial.SetTexture ("\_RgbTex", rgbChannelTex);

ccMaterial.SetFloat ("\_Saturation", saturation);

Graphics.Blit (source, renderTarget2Use, ccMaterial);

}

if (selectiveCc)

{

selectiveCcMaterial.SetColor ("selColor", selectiveFromColor);

selectiveCcMaterial.SetColor ("targetColor", selectiveToColor);

Graphics.Blit (renderTarget2Use, destination, selectiveCcMaterial);

RenderTexture.ReleaseTemporary (renderTarget2Use);

}

}

}

}