using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[AddComponentMenu("Image Effects/Color Adjustments/Contrast Stretch")]

public class ContrastStretch : MonoBehaviour

{

/// Adaptation speed - percents per frame, if playing at 30FPS.

/// Default is 0.02 (2% each 1/30s).

public float adaptationSpeed = 0.02f;

/// If our scene is really dark (or really bright), we might not want to

/// stretch its contrast to the full range.

/// limitMinimum=0, limitMaximum=1 is the same as not applying the effect at all.

/// limitMinimum=1, limitMaximum=0 is always stretching colors to full range.

/// The limit on the minimum luminance (0...1) - we won't go above this.

public float limitMinimum = 0.2f;

/// The limit on the maximum luminance (0...1) - we won't go below this.

public float limitMaximum = 0.6f;

// To maintain adaptation levels over time, we need two 1x1 render textures

// and ping-pong between them.

private RenderTexture[] adaptRenderTex = new RenderTexture[2];

private int curAdaptIndex = 0;

// Computes scene luminance (grayscale) image

public Shader shaderLum;

private Material m\_materialLum;

protected Material materialLum {

get {

if ( m\_materialLum == null ) {

m\_materialLum = new Material(shaderLum);

m\_materialLum.hideFlags = HideFlags.HideAndDontSave;

}

return m\_materialLum;

}

}

// Reduces size of the image by 2x2, while computing maximum/minimum values.

// By repeatedly applying this shader, we reduce the initial luminance image

// to 1x1 image with minimum/maximum luminances found.

public Shader shaderReduce;

private Material m\_materialReduce;

protected Material materialReduce {

get {

if ( m\_materialReduce == null ) {

m\_materialReduce = new Material(shaderReduce);

m\_materialReduce.hideFlags = HideFlags.HideAndDontSave;

}

return m\_materialReduce;

}

}

// Adaptation shader - gradually "adapts" minimum/maximum luminances,

// based on currently adapted 1x1 image and the actual 1x1 image of the current scene.

public Shader shaderAdapt;

private Material m\_materialAdapt;

protected Material materialAdapt {

get {

if ( m\_materialAdapt == null ) {

m\_materialAdapt = new Material(shaderAdapt);

m\_materialAdapt.hideFlags = HideFlags.HideAndDontSave;

}

return m\_materialAdapt;

}

}

// Final pass - stretches the color values of the original scene, based on currently

// adpated minimum/maximum values.

public Shader shaderApply;

private Material m\_materialApply;

protected Material materialApply {

get {

if ( m\_materialApply == null ) {

m\_materialApply = new Material(shaderApply);

m\_materialApply.hideFlags = HideFlags.HideAndDontSave;

}

return m\_materialApply;

}

}

void Start()

{

// Disable if we don't support image effects

if (!SystemInfo.supportsImageEffects) {

enabled = false;

return;

}

if (!shaderAdapt.isSupported || !shaderApply.isSupported || !shaderLum.isSupported || !shaderReduce.isSupported) {

enabled = false;

return;

}

}

void OnEnable()

{

for( int i = 0; i < 2; ++i )

{

if ( !adaptRenderTex[i] ) {

adaptRenderTex[i] = new RenderTexture(1, 1, 0);

adaptRenderTex[i].hideFlags = HideFlags.HideAndDontSave;

}

}

}

void OnDisable()

{

for( int i = 0; i < 2; ++i )

{

DestroyImmediate( adaptRenderTex[i] );

adaptRenderTex[i] = null;

}

if ( m\_materialLum )

DestroyImmediate( m\_materialLum );

if ( m\_materialReduce )

DestroyImmediate( m\_materialReduce );

if ( m\_materialAdapt )

DestroyImmediate( m\_materialAdapt );

if ( m\_materialApply )

DestroyImmediate( m\_materialApply );

}

/// Apply the filter

void OnRenderImage (RenderTexture source, RenderTexture destination)

{

// Blit to smaller RT and convert to luminance on the way

const int TEMP\_RATIO = 1; // 4x4 smaller

RenderTexture rtTempSrc = RenderTexture.GetTemporary(source.width/TEMP\_RATIO, source.height/TEMP\_RATIO);

Graphics.Blit (source, rtTempSrc, materialLum);

// Repeatedly reduce this image in size, computing min/max luminance values

// In the end we'll have 1x1 image with min/max luminances found.

const int FINAL\_SIZE = 1;

//const int FINAL\_SIZE = 1;

while( rtTempSrc.width > FINAL\_SIZE || rtTempSrc.height > FINAL\_SIZE )

{

const int REDUCE\_RATIO = 2; // our shader does 2x2 reduction

int destW = rtTempSrc.width / REDUCE\_RATIO;

if ( destW < FINAL\_SIZE ) destW = FINAL\_SIZE;

int destH = rtTempSrc.height / REDUCE\_RATIO;

if ( destH < FINAL\_SIZE ) destH = FINAL\_SIZE;

RenderTexture rtTempDst = RenderTexture.GetTemporary(destW,destH);

Graphics.Blit (rtTempSrc, rtTempDst, materialReduce);

// Release old src temporary, and make new temporary the source

RenderTexture.ReleaseTemporary( rtTempSrc );

rtTempSrc = rtTempDst;

}

// Update viewer's adaptation level

CalculateAdaptation( rtTempSrc );

// Apply contrast strech to the original scene, using currently adapted parameters

materialApply.SetTexture("\_AdaptTex", adaptRenderTex[curAdaptIndex] );

Graphics.Blit (source, destination, materialApply);

RenderTexture.ReleaseTemporary( rtTempSrc );

}

/// Helper function to do gradual adaptation to min/max luminances

private void CalculateAdaptation( Texture curTexture )

{

int prevAdaptIndex = curAdaptIndex;

curAdaptIndex = (curAdaptIndex+1) % 2;

// Adaptation speed is expressed in percents/frame, based on 30FPS.

// Calculate the adaptation lerp, based on current FPS.

float adaptLerp = 1.0f - Mathf.Pow( 1.0f - adaptationSpeed, 30.0f \* Time.deltaTime );

const float kMinAdaptLerp = 0.01f;

adaptLerp = Mathf.Clamp( adaptLerp, kMinAdaptLerp, 1 );

materialAdapt.SetTexture("\_CurTex", curTexture );

materialAdapt.SetVector("\_AdaptParams", new Vector4(

adaptLerp,

limitMinimum,

limitMaximum,

0.0f

));

// clear destination RT so its contents don't need to be restored

Graphics.SetRenderTarget(adaptRenderTex[curAdaptIndex]);

GL.Clear(false, true, Color.black);

Graphics.Blit (

adaptRenderTex[prevAdaptIndex],

adaptRenderTex[curAdaptIndex],

materialAdapt);

}

}

}