using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu ("Image Effects/Rendering/Screen Space Ambient Obscurance")]

class ScreenSpaceAmbientObscurance : PostEffectsBase {

[Range (0,3)]

public float intensity = 0.5f;

[Range (0.1f,3)]

public float radius = 0.2f;

[Range (0,3)]

public int blurIterations = 1;

[Range (0,5)]

public float blurFilterDistance = 1.25f;

[Range (0,1)]

public int downsample = 0;

public Texture2D rand = null;

public Shader aoShader= null;

private Material aoMaterial = null;

public override bool CheckResources () {

CheckSupport (true);

aoMaterial = CheckShaderAndCreateMaterial (aoShader, aoMaterial);

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

void OnDisable () {

if (aoMaterial)

DestroyImmediate (aoMaterial);

aoMaterial = null;

}

[ImageEffectOpaque]

void OnRenderImage (RenderTexture source, RenderTexture destination) {

if (CheckResources () == false) {

Graphics.Blit (source, destination);

return;

}

Matrix4x4 P = GetComponent<Camera>().projectionMatrix;

var invP= P.inverse;

Vector4 projInfo = new Vector4

((-2.0f / (Screen.width \* P[0])),

(-2.0f / (Screen.height \* P[5])),

((1.0f - P[2]) / P[0]),

((1.0f + P[6]) / P[5]));

aoMaterial.SetVector ("\_ProjInfo", projInfo); // used for unprojection

aoMaterial.SetMatrix ("\_ProjectionInv", invP); // only used for reference

aoMaterial.SetTexture ("\_Rand", rand); // not needed for DX11 :)

aoMaterial.SetFloat ("\_Radius", radius);

aoMaterial.SetFloat ("\_Radius2", radius\*radius);

aoMaterial.SetFloat ("\_Intensity", intensity);

aoMaterial.SetFloat ("\_BlurFilterDistance", blurFilterDistance);

int rtW = source.width;

int rtH = source.height;

RenderTexture tmpRt = RenderTexture.GetTemporary (rtW>>downsample, rtH>>downsample);

RenderTexture tmpRt2;

Graphics.Blit (source, tmpRt, aoMaterial, 0);

if (downsample > 0) {

tmpRt2 = RenderTexture.GetTemporary (rtW, rtH);

Graphics.Blit(tmpRt, tmpRt2, aoMaterial, 4);

RenderTexture.ReleaseTemporary (tmpRt);

tmpRt = tmpRt2;

// @NOTE: it's probably worth a shot to blur in low resolution

// instead with a bilat-upsample afterwards ...

}

for (int i = 0; i < blurIterations; i++) {

aoMaterial.SetVector("\_Axis", new Vector2(1.0f,0.0f));

tmpRt2 = RenderTexture.GetTemporary (rtW, rtH);

Graphics.Blit (tmpRt, tmpRt2, aoMaterial, 1);

RenderTexture.ReleaseTemporary (tmpRt);

aoMaterial.SetVector("\_Axis", new Vector2(0.0f,1.0f));

tmpRt = RenderTexture.GetTemporary (rtW, rtH);

Graphics.Blit (tmpRt2, tmpRt, aoMaterial, 1);

RenderTexture.ReleaseTemporary (tmpRt2);

}

aoMaterial.SetTexture ("\_AOTex", tmpRt);

Graphics.Blit (source, destination, aoMaterial, 2);

RenderTexture.ReleaseTemporary (tmpRt);

}

}

}