using System;

using System.Collections.Generic;

using UnityEngine;

namespace UnityStandardAssets.Water

{

[ExecuteInEditMode]

[RequireComponent(typeof(WaterBase))]

public class PlanarReflection : MonoBehaviour

{

public LayerMask reflectionMask;

public bool reflectSkybox = false;

public Color clearColor = Color.grey;

public String reflectionSampler = "\_ReflectionTex";

public float clipPlaneOffset = 0.07F;

Vector3 m\_Oldpos;

Camera m\_ReflectionCamera;

Material m\_SharedMaterial;

Dictionary<Camera, bool> m\_HelperCameras;

public void Start()

{

m\_SharedMaterial = ((WaterBase)gameObject.GetComponent(typeof(WaterBase))).sharedMaterial;

}

Camera CreateReflectionCameraFor(Camera cam)

{

String reflName = gameObject.name + "Reflection" + cam.name;

GameObject go = GameObject.Find(reflName);

if (!go)

{

go = new GameObject(reflName, typeof(Camera));

}

if (!go.GetComponent(typeof(Camera)))

{

go.AddComponent(typeof(Camera));

}

Camera reflectCamera = go.GetComponent<Camera>();

reflectCamera.backgroundColor = clearColor;

reflectCamera.clearFlags = reflectSkybox ? CameraClearFlags.Skybox : CameraClearFlags.SolidColor;

SetStandardCameraParameter(reflectCamera, reflectionMask);

if (!reflectCamera.targetTexture)

{

reflectCamera.targetTexture = CreateTextureFor(cam);

}

return reflectCamera;

}

void SetStandardCameraParameter(Camera cam, LayerMask mask)

{

cam.cullingMask = mask & ~(1 << LayerMask.NameToLayer("Water"));

cam.backgroundColor = Color.black;

cam.enabled = false;

}

RenderTexture CreateTextureFor(Camera cam)

{

RenderTexture rt = new RenderTexture(Mathf.FloorToInt(cam.pixelWidth \* 0.5F),

Mathf.FloorToInt(cam.pixelHeight \* 0.5F), 24);

rt.hideFlags = HideFlags.DontSave;

return rt;

}

public void RenderHelpCameras(Camera currentCam)

{

if (null == m\_HelperCameras)

{

m\_HelperCameras = new Dictionary<Camera, bool>();

}

if (!m\_HelperCameras.ContainsKey(currentCam))

{

m\_HelperCameras.Add(currentCam, false);

}

if (m\_HelperCameras[currentCam])

{

return;

}

if (!m\_ReflectionCamera)

{

m\_ReflectionCamera = CreateReflectionCameraFor(currentCam);

}

RenderReflectionFor(currentCam, m\_ReflectionCamera);

m\_HelperCameras[currentCam] = true;

}

public void LateUpdate()

{

if (null != m\_HelperCameras)

{

m\_HelperCameras.Clear();

}

}

public void WaterTileBeingRendered(Transform tr, Camera currentCam)

{

RenderHelpCameras(currentCam);

if (m\_ReflectionCamera && m\_SharedMaterial)

{

m\_SharedMaterial.SetTexture(reflectionSampler, m\_ReflectionCamera.targetTexture);

}

}

public void OnEnable()

{

Shader.EnableKeyword("WATER\_REFLECTIVE");

Shader.DisableKeyword("WATER\_SIMPLE");

}

public void OnDisable()

{

Shader.EnableKeyword("WATER\_SIMPLE");

Shader.DisableKeyword("WATER\_REFLECTIVE");

}

void RenderReflectionFor(Camera cam, Camera reflectCamera)

{

if (!reflectCamera)

{

return;

}

if (m\_SharedMaterial && !m\_SharedMaterial.HasProperty(reflectionSampler))

{

return;

}

reflectCamera.cullingMask = reflectionMask & ~(1 << LayerMask.NameToLayer("Water"));

SaneCameraSettings(reflectCamera);

reflectCamera.backgroundColor = clearColor;

reflectCamera.clearFlags = reflectSkybox ? CameraClearFlags.Skybox : CameraClearFlags.SolidColor;

if (reflectSkybox)

{

if (cam.gameObject.GetComponent(typeof(Skybox)))

{

Skybox sb = (Skybox)reflectCamera.gameObject.GetComponent(typeof(Skybox));

if (!sb)

{

sb = (Skybox)reflectCamera.gameObject.AddComponent(typeof(Skybox));

}

sb.material = ((Skybox)cam.GetComponent(typeof(Skybox))).material;

}

}

GL.invertCulling = true;

Transform reflectiveSurface = transform; //waterHeight;

Vector3 eulerA = cam.transform.eulerAngles;

reflectCamera.transform.eulerAngles = new Vector3(-eulerA.x, eulerA.y, eulerA.z);

reflectCamera.transform.position = cam.transform.position;

Vector3 pos = reflectiveSurface.transform.position;

pos.y = reflectiveSurface.position.y;

Vector3 normal = reflectiveSurface.transform.up;

float d = -Vector3.Dot(normal, pos) - clipPlaneOffset;

Vector4 reflectionPlane = new Vector4(normal.x, normal.y, normal.z, d);

Matrix4x4 reflection = Matrix4x4.zero;

reflection = CalculateReflectionMatrix(reflection, reflectionPlane);

m\_Oldpos = cam.transform.position;

Vector3 newpos = reflection.MultiplyPoint(m\_Oldpos);

reflectCamera.worldToCameraMatrix = cam.worldToCameraMatrix \* reflection;

Vector4 clipPlane = CameraSpacePlane(reflectCamera, pos, normal, 1.0f);

Matrix4x4 projection = cam.projectionMatrix;

projection = CalculateObliqueMatrix(projection, clipPlane);

reflectCamera.projectionMatrix = projection;

reflectCamera.transform.position = newpos;

Vector3 euler = cam.transform.eulerAngles;

reflectCamera.transform.eulerAngles = new Vector3(-euler.x, euler.y, euler.z);

reflectCamera.Render();

GL.invertCulling = false;

}

void SaneCameraSettings(Camera helperCam)

{

helperCam.depthTextureMode = DepthTextureMode.None;

helperCam.backgroundColor = Color.black;

helperCam.clearFlags = CameraClearFlags.SolidColor;

helperCam.renderingPath = RenderingPath.Forward;

}

static Matrix4x4 CalculateObliqueMatrix(Matrix4x4 projection, Vector4 clipPlane)

{

Vector4 q = projection.inverse \* new Vector4(

Sgn(clipPlane.x),

Sgn(clipPlane.y),

1.0F,

1.0F

);

Vector4 c = clipPlane \* (2.0F / (Vector4.Dot(clipPlane, q)));

// third row = clip plane - fourth row

projection[2] = c.x - projection[3];

projection[6] = c.y - projection[7];

projection[10] = c.z - projection[11];

projection[14] = c.w - projection[15];

return projection;

}

static Matrix4x4 CalculateReflectionMatrix(Matrix4x4 reflectionMat, Vector4 plane)

{

reflectionMat.m00 = (1.0F - 2.0F \* plane[0] \* plane[0]);

reflectionMat.m01 = (- 2.0F \* plane[0] \* plane[1]);

reflectionMat.m02 = (- 2.0F \* plane[0] \* plane[2]);

reflectionMat.m03 = (- 2.0F \* plane[3] \* plane[0]);

reflectionMat.m10 = (- 2.0F \* plane[1] \* plane[0]);

reflectionMat.m11 = (1.0F - 2.0F \* plane[1] \* plane[1]);

reflectionMat.m12 = (- 2.0F \* plane[1] \* plane[2]);

reflectionMat.m13 = (- 2.0F \* plane[3] \* plane[1]);

reflectionMat.m20 = (- 2.0F \* plane[2] \* plane[0]);

reflectionMat.m21 = (- 2.0F \* plane[2] \* plane[1]);

reflectionMat.m22 = (1.0F - 2.0F \* plane[2] \* plane[2]);

reflectionMat.m23 = (- 2.0F \* plane[3] \* plane[2]);

reflectionMat.m30 = 0.0F;

reflectionMat.m31 = 0.0F;

reflectionMat.m32 = 0.0F;

reflectionMat.m33 = 1.0F;

return reflectionMat;

}

static float Sgn(float a)

{

if (a > 0.0F)

{

return 1.0F;

}

if (a < 0.0F)

{

return -1.0F;

}

return 0.0F;

}

Vector4 CameraSpacePlane(Camera cam, Vector3 pos, Vector3 normal, float sideSign)

{

Vector3 offsetPos = pos + normal \* clipPlaneOffset;

Matrix4x4 m = cam.worldToCameraMatrix;

Vector3 cpos = m.MultiplyPoint(offsetPos);

Vector3 cnormal = m.MultiplyVector(normal).normalized \* sideSign;

return new Vector4(cnormal.x, cnormal.y, cnormal.z, -Vector3.Dot(cpos, cnormal));

}

}

}