**Unity3D镜面反射脚本**

Posted on 2013年02月26日 by U3d / [Unity3D脚本/插件](http://www.unitymanual.com/category/script)/被围观 307 次

首先，新建一个Shader：

Shader "FX/Mirror Reflection"{  
Properties {  
\_MainTex (“Base (RGB)”, 2D) = “white” {}  
\_ReflectionTex (“Reflection”, 2D) = “white” { TexGen ObjectLinear }  
}  
// two texture cards: full thing  
Subshader {  
Pass {  
SetTexture[\_MainTex] { combine texture }  
SetTexture[\_ReflectionTex] { matrix [\_ProjMatrix] combine texture \* previous }  
}  
}  
// fallback: just main texture  
Subshader {  
Pass {  
SetTexture [\_MainTex] { combine texture }  
}  
}  
}

第二，新建一个脚本MirrorReflection.cs:

注意：被赋予脚本的物体不能缩放也就是物体的Rransform.scale的值均为1。

using UnityEngine;  
using System.Collections;

// This is in fact just the Water script from Pro Standard Assets,  
// just with refraction stuff removed.

[ExecuteInEditMode] // Make mirror live-update even when not in play mode  
public class MirrorReflection : MonoBehaviour  
{  
public bool m\_DisablePixelLights = true;  
public int m\_TextureSize = 256;  
public float m\_ClipPlaneOffset = 0.07f;

public LayerMask m\_ReflectLayers = -1;

private Hashtable m\_ReflectionCameras = new Hashtable(); // Camera -> Camera table

private RenderTexture m\_ReflectionTexture = null;  
private int m\_OldReflectionTextureSize = 0;

private static bool s\_InsideRendering = false;

// This is called when it’s known that the object will be rendered by some  
// camera. We render reflections and do other updates here.  
// Because the script executes in edit mode, reflections for the scene view  
// camera will just work!  
public void OnWillRenderObject()  
{  
if (!enabled || !renderer || !renderer.sharedMaterial || !renderer.enabled)  
return;

Camera cam = Camera.current;  
if (!cam)  
return;

// Safeguard from recursive reflections.  
if (s\_InsideRendering)  
return;  
s\_InsideRendering = true;

Camera reflectionCamera;  
CreateMirrorObjects(cam, out reflectionCamera);

// find out the reflection plane: position and normal in world space  
Vector3 pos = transform.position;  
Vector3 normal = transform.up;

// Optionally disable pixel lights for reflection  
int oldPixelLightCount = QualitySettings.pixelLightCount;  
if (m\_DisablePixelLights)  
QualitySettings.pixelLightCount = 0;

UpdateCameraModes(cam, reflectionCamera);

// Render reflection  
// Reflect camera around reflection plane  
float d = -Vector3.Dot(normal, pos) – m\_ClipPlaneOffset;  
Vector4 reflectionPlane = new Vector4(normal.x, normal.y, normal.z, d);

Matrix4x4 reflection = Matrix4x4.zero;  
CalculateReflectionMatrix(ref reflection, reflectionPlane);  
Vector3 oldpos = cam.transform.position;  
Vector3 newpos = reflection.MultiplyPoint(oldpos);  
reflectionCamera.worldToCameraMatrix = cam.worldToCameraMatrix \* reflection;

// Setup oblique projection matrix so that near plane is our reflection  
// plane. This way we clip everything below/above it for free.  
Vector4 clipPlane = CameraSpacePlane(reflectionCamera, pos, normal, 1.0f);  
Matrix4x4 projection = cam.projectionMatrix;  
CalculateObliqueMatrix(ref projection, clipPlane);  
reflectionCamera.projectionMatrix = projection;

reflectionCamera.cullingMask = ~(1 << 4) & m\_ReflectLayers.value; // never render water layer  
reflectionCamera.targetTexture = m\_ReflectionTexture;  
GL.SetRevertBackfacing(true);  
reflectionCamera.transform.position = newpos;  
Vector3 euler = cam.transform.eulerAngles;  
reflectionCamera.transform.eulerAngles = new Vector3(0, euler.y, euler.z);  
reflectionCamera.Render();  
reflectionCamera.transform.position = oldpos;  
GL.SetRevertBackfacing(false);  
Material[] materials = renderer.sharedMaterials;  
foreach (Material mat in materials)  
{  
if (mat.HasProperty("\_ReflectionTex"))  
mat.SetTexture("\_ReflectionTex", m\_ReflectionTexture);  
}

// Set matrix on the shader that transforms UVs from object space into screen  
// space. We want to just project reflection texture on screen.  
Matrix4x4 scaleOffset = Matrix4x4.TRS(  
new Vector3(0.5f, 0.5f, 0.5f), Quaternion.identity, new Vector3(0.5f, 0.5f, 0.5f));  
Vector3 scale = transform.lossyScale;  
Matrix4x4 mtx = transform.localToWorldMatrix \* Matrix4x4.Scale(new Vector3(1.0f / scale.x, 1.0f / scale.y, 1.0f / scale.z));  
mtx = scaleOffset \* cam.projectionMatrix \* cam.worldToCameraMatrix \* mtx;  
foreach (Material mat in materials)  
{  
mat.SetMatrix("\_ProjMatrix", mtx);  
}

// Restore pixel light count  
if (m\_DisablePixelLights)  
QualitySettings.pixelLightCount = oldPixelLightCount;

s\_InsideRendering = false;  
}

// Cleanup all the objects we possibly have created  
void OnDisable()  
{  
if (m\_ReflectionTexture)  
{  
DestroyImmediate(m\_ReflectionTexture);  
m\_ReflectionTexture = null;  
}  
foreach (DictionaryEntry kvp in m\_ReflectionCameras)  
DestroyImmediate(((Camera)kvp.Value).gameObject);  
m\_ReflectionCameras.Clear();  
}

private void UpdateCameraModes(Camera src, Camera dest)  
{  
if (dest == null)  
return;  
// set camera to clear the same way as current camera  
dest.clearFlags = src.clearFlags;  
dest.backgroundColor = src.backgroundColor;  
if (src.clearFlags == CameraClearFlags.Skybox)  
{  
Skybox sky = src.GetComponent(typeof(Skybox)) as Skybox;  
Skybox mysky = dest.GetComponent(typeof(Skybox)) as Skybox;  
if (!sky || !sky.material)  
{  
mysky.enabled = false;  
}  
else  
{  
mysky.enabled = true;  
mysky.material = sky.material;  
}  
}  
// update other values to match current camera.  
// even if we are supplying custom camera&projection matrices,  
// some of values are used elsewhere (e.g. skybox uses far plane)  
dest.farClipPlane = src.farClipPlane;  
dest.nearClipPlane = src.nearClipPlane;  
dest.orthographic = src.orthographic;  
dest.fieldOfView = src.fieldOfView;  
dest.aspect = src.aspect;  
dest.orthographicSize = src.orthographicSize;  
}

// On-demand create any objects we need  
private void CreateMirrorObjects(Camera currentCamera, out Camera reflectionCamera)  
{  
reflectionCamera = null;

// Reflection render texture  
if (!m\_ReflectionTexture || m\_OldReflectionTextureSize != m\_TextureSize)  
{  
if (m\_ReflectionTexture)  
DestroyImmediate(m\_ReflectionTexture);  
m\_ReflectionTexture = new RenderTexture(m\_TextureSize, m\_TextureSize, 16);  
m\_ReflectionTexture.name = "\_\_MirrorReflection" + GetInstanceID();  
m\_ReflectionTexture.isPowerOfTwo = true;  
m\_ReflectionTexture.hideFlags = HideFlags.DontSave;  
m\_OldReflectionTextureSize = m\_TextureSize;  
}

// Camera for reflection  
reflectionCamera = m\_ReflectionCameras[currentCamera] as Camera;  
if (!reflectionCamera) // catch both not-in-dictionary and in-dictionary-but-deleted-GO  
{  
GameObject go = new GameObject("Mirror Refl Camera id" + GetInstanceID() + " for " + currentCamera.GetInstanceID(), typeof(Camera), typeof(Skybox));  
reflectionCamera = go.camera;  
reflectionCamera.enabled = false;  
reflectionCamera.transform.position = transform.position;  
reflectionCamera.transform.rotation = transform.rotation;  
reflectionCamera.gameObject.AddComponent("FlareLayer");  
go.hideFlags = HideFlags.HideAndDontSave;  
m\_ReflectionCameras[currentCamera] = reflectionCamera;  
}  
}

// Extended sign: returns -1, 0 or 1 based on sign of a  
private static float sgn(float a)  
{  
if (a > 0.0f) return 1.0f;  
if (a < 0.0f) return -1.0f;  
return 0.0f;  
}

// Given position/normal of the plane, calculates plane in camera space.  
private Vector4 CameraSpacePlane(Camera cam, Vector3 pos, Vector3 normal, float sideSign)  
{  
Vector3 offsetPos = pos + normal \* m\_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));  
}

// Adjusts the given projection matrix so that near plane is the given clipPlane  
// clipPlane is given in camera space. See article in Game Programming Gems 5.  
private static void CalculateObliqueMatrix(ref 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];  
}

// Calculates reflection matrix around the given plane  
private static void CalculateReflectionMatrix(ref Matrix4x4 reflectionMat, Vector4 plane)  
{  
reflectionMat.m00 = (1F – 2F \* plane[0] \* plane[0]);  
reflectionMat.m01 = (-2F \* plane[0] \* plane[1]);  
reflectionMat.m02 = (-2F \* plane[0] \* plane[2]);  
reflectionMat.m03 = (-2F \* plane[3] \* plane[0]);

reflectionMat.m10 = (-2F \* plane[1] \* plane[0]);  
reflectionMat.m11 = (1F – 2F \* plane[1] \* plane[1]);  
reflectionMat.m12 = (-2F \* plane[1] \* plane[2]);  
reflectionMat.m13 = (-2F \* plane[3] \* plane[1]);

reflectionMat.m20 = (-2F \* plane[2] \* plane[0]);  
reflectionMat.m21 = (-2F \* plane[2] \* plane[1]);  
reflectionMat.m22 = (1F – 2F \* plane[2] \* plane[2]);  
reflectionMat.m23 = (-2F \* plane[3] \* plane[2]);

reflectionMat.m30 = 0F;  
reflectionMat.m31 = 0F;  
reflectionMat.m32 = 0F;  
reflectionMat.m33 = 1F;  
}  
}