**镜面反射代码**

Posted on 2013年06月05日 by U3d / [Unity3D脚本/插件](http://www.unitymanual.com/category/script)/被围观 80 次

文件名MirrorReflection.cs

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 001 | **using** UnityEngine; |
| 002 | **using** System.Collections; |
| 003 |  |
| 004 | *// This is in fact just the Water script from Pro Standard Assets,* |
| 005 | *// just with refraction stuff removed.* |
| 006 |  |
| 007 | [ExecuteInEditMode] *// Make mirror live-update even when not in play mode* |
| 008 | **public** **class** MirrorReflection : MonoBehaviour |
| 009 | { |
| 010 | **public** **bool** m\_DisablePixelLights = **true**; |
| 011 | **public** **int** m\_TextureSize = 256; |
| 012 | **public** **float** m\_ClipPlaneOffset = 0.07f; |
| 013 |  |
| 014 | **public** LayerMask m\_ReflectLayers = -1; |
| 015 |  |
| 016 | **private** Hashtable m\_ReflectionCameras = new Hashtable(); *// Camera -> Camera table* |
| 017 |  |
| 018 | **private** RenderTexture m\_ReflectionTexture = **null**; |
| 019 | **private** **int** m\_OldReflectionTextureSize = 0; |
| 020 |  |
| 021 | **private** **static** **bool** s\_InsideRendering = **false**; |
| 022 |  |
| 023 | *// This is called when it's known that the object will be rendered by some* |
| 024 | *// camera. We render reflections and do other updates here.* |
| 025 | *// Because the script executes in edit mode, reflections for the scene view* |
| 026 | *// camera will just work!* |
| 027 | **public** **void** OnWillRenderObject() |
| 028 | { |
| 029 | **if**( !enabled || !renderer || !renderer.sharedMaterial || !renderer.enabled ) |
| 030 | **return**; |
| 031 |  |
| 032 | Camera cam = Camera.current; |
| 033 | **if**( !cam ) |
| 034 | **return**; |
| 035 |  |
| 036 | *// Safeguard from recursive reflections.* |
| 037 | **if**( s\_InsideRendering ) |
| 038 | **return**; |
| 039 | s\_InsideRendering = **true**; |
| 040 |  |
| 041 | Camera reflectionCamera; |
| 042 | CreateMirrorObjects( cam, **out** reflectionCamera ); |
| 043 |  |
| 044 | *// find out the reflection plane: position and normal in world space* |
| 045 | Vector3 pos = transform.position; |
| 046 | Vector3 normal = transform.up; |
| 047 |  |
| 048 | *// Optionally disable pixel lights for reflection* |
| 049 | **int** oldPixelLightCount = QualitySettings.pixelLightCount; |
| 050 |  |
| 051 | **if**( m\_DisablePixelLights ) |
| 052 | QualitySettings.pixelLightCount = 0; |
| 053 |  |
| 054 | UpdateCameraModes( cam, reflectionCamera ); |
| 055 |  |
| 056 | *// Render reflection* |
| 057 | *// Reflect camera around reflection plane* |
| 058 | **float** d = -Vector3.Dot (normal, pos) - m\_ClipPlaneOffset; |
| 059 | Vector4 reflectionPlane = new Vector4 (normal.x, normal.y, normal.z, d); |
| 060 |  |
| 061 | Matrix4x4 reflection = Matrix4x4.zero; |
| 062 | CalculateReflectionMatrix (**ref** reflection, reflectionPlane); |
| 063 | Vector3 oldpos = cam.transform.position; |
| 064 | Vector3 newpos = reflection.MultiplyPoint( oldpos ); |
| 065 | reflectionCamera.worldToCameraMatrix = cam.worldToCameraMatrix \* reflection; |
| 066 |  |
| 067 | *// Setup oblique projection matrix so that near plane is our reflection* |
| 068 | *// plane. This way we clip everything below/above it for free.* |
| 069 | Vector4 clipPlane = CameraSpacePlane( reflectionCamera, pos, normal, 1.0f ); |
| 070 | Matrix4x4 projection = cam.projectionMatrix; |
| 071 | CalculateObliqueMatrix (**ref** projection, clipPlane); |
| 072 | reflectionCamera.projectionMatrix = projection; |
| 073 |  |
| 074 | reflectionCamera.cullingMask = ~(1<<4) & m\_ReflectLayers.**value**; *// never render water layer* |
| 075 | reflectionCamera.targetTexture = m\_ReflectionTexture; |
| 076 | GL.SetRevertBackfacing (**true**); |
| 077 | reflectionCamera.transform.position = newpos; |
| 078 | Vector3 euler = cam.transform.eulerAngles; |
| 079 | reflectionCamera.transform.eulerAngles = new Vector3(0, euler.y, euler.z); |
| 080 | reflectionCamera.Render(); |
| 081 | reflectionCamera.transform.position = oldpos; |
| 082 | GL.SetRevertBackfacing (**false**); |
| 083 | Material[] materials = renderer.sharedMaterials; |
| 084 | **foreach**( Material mat **in** materials ) { |
| 085 | **if**( mat.HasProperty("\_ReflectionTex") ) |
| 086 | mat.SetTexture( "\_ReflectionTex", m\_ReflectionTexture ); |
| 087 | } |
| 088 |  |
| 089 | *// Set matrix on the shader that transforms UVs from object space into screen* |
| 090 | *// space. We want to just project reflection texture on screen.* |
| 091 | Matrix4x4 scaleOffset = Matrix4x4.TRS( |
| 092 | new Vector3(0.5f,0.5f,0.5f), Quaternion.identity, new Vector3(0.5f,0.5f,0.5f) ); |
| 093 | Vector3 scale = transform.lossyScale; |
| 094 | Matrix4x4 mtx = transform.localToWorldMatrix \* Matrix4x4.Scale( new Vector3(1.0f/scale.x, 1.0f/scale.y, 1.0f/scale.z) ); |
| 095 | mtx = scaleOffset \* cam.projectionMatrix \* cam.worldToCameraMatrix \* mtx; |
| 096 | **foreach**( Material mat **in** materials ) { |
| 097 | mat.SetMatrix( "\_ProjMatrix", mtx ); |
| 098 | } |
| 099 |  |
| 100 | *// Restore pixel light count* |
| 101 | **if**( m\_DisablePixelLights ) |
| 102 | QualitySettings.pixelLightCount = oldPixelLightCount; |
| 103 |  |
| 104 | s\_InsideRendering = **false**; |
| 105 | } |
| 106 |  |
| 107 |  |
| 108 | *// Cleanup all the objects we possibly have created* |
| 109 | **void** OnDisable(){ |
| 110 | **if**( m\_ReflectionTexture ) { |
| 111 | DestroyImmediate( m\_ReflectionTexture ); |
| 112 | m\_ReflectionTexture = **null**; |
| 113 | } |
| 114 | **foreach**( DictionaryEntry kvp **in** m\_ReflectionCameras ) |
| 115 | DestroyImmediate( ((Camera)kvp.**Value**).gameObject ); |
| 116 | m\_ReflectionCameras.Clear(); |
| 117 | } |
| 118 |  |
| 119 |  |
| 120 | **private** **void** UpdateCameraModes( Camera src, Camera dest ) |
| 121 | { |
| 122 | **if**( dest == **null** ) |
| 123 | **return**; |
| 124 | *// set camera to clear the same way as current camera* |
| 125 | dest.clearFlags = src.clearFlags; |
| 126 | dest.backgroundColor = src.backgroundColor; |
| 127 | **if**( src.clearFlags == CameraClearFlags.Skybox ) |
| 128 | { |
| 129 | Skybox sky = src.GetComponent(typeof(Skybox)) **as** Skybox; |
| 130 | Skybox mysky = dest.GetComponent(typeof(Skybox)) **as** Skybox; |
| 131 | **if**( !sky || !sky.material ) |
| 132 | { |
| 133 | mysky.enabled = **false**; |
| 134 | } |
| 135 | **else** |
| 136 | { |
| 137 | mysky.enabled = **true**; |
| 138 | mysky.material = sky.material; |
| 139 | } |
| 140 | } |
| 141 | *// update other values to match current camera.* |
| 142 | *// even if we are supplying custom camera&projection matrices,* |
| 143 | *// some of values are used elsewhere (e.g. skybox uses far plane)* |
| 144 | dest.farClipPlane = src.farClipPlane; |
| 145 | dest.nearClipPlane = src.nearClipPlane; |
| 146 | dest.orthographic = src.orthographic; |
| 147 | dest.fieldOfView = src.fieldOfView; |
| 148 | dest.aspect = src.aspect; |
| 149 | dest.orthographicSize = src.orthographicSize; |
| 150 | } |
| 151 |  |
| 152 | *// On-demand create any objects we need* |
| 153 | **private** **void** CreateMirrorObjects( Camera currentCamera, **out** Camera reflectionCamera ) |
| 154 | { |
| 155 | reflectionCamera = **null**; |
| 156 |  |
| 157 | *// Reflection render texture* |
| 158 | **if**( !m\_ReflectionTexture || m\_OldReflectionTextureSize != m\_TextureSize ) |
| 159 | { |
| 160 | **if**( m\_ReflectionTexture ) |
| 161 | DestroyImmediate( m\_ReflectionTexture ); |
| 162 | m\_ReflectionTexture = new RenderTexture( m\_TextureSize, m\_TextureSize, 16 ); |
| 163 | m\_ReflectionTexture.name = "\_\_MirrorReflection" + GetInstanceID(); |
| 164 | m\_ReflectionTexture.isPowerOfTwo = **true**; |
| 165 | m\_ReflectionTexture.hideFlags = HideFlags.DontSave; |
| 166 | m\_OldReflectionTextureSize = m\_TextureSize; |
| 167 | } |
| 168 |  |
| 169 | *// Camera for reflection* |
| 170 | reflectionCamera = m\_ReflectionCameras[currentCamera] **as** Camera; |
| 171 | **if**( !reflectionCamera ) *// catch both not-in-dictionary and in-dictionary-but-deleted-GO* |
| 172 | { |
| 173 | GameObject go = new GameObject( "Mirror Refl Camera id" + GetInstanceID() + " for " + currentCamera.GetInstanceID(), typeof(Camera), typeof(Skybox) ); |
| 174 | reflectionCamera = go.camera; |
| 175 | reflectionCamera.enabled = **false**; |
| 176 | reflectionCamera.transform.position = transform.position; |
| 177 | reflectionCamera.transform.rotation = transform.rotation; |
| 178 | reflectionCamera.gameObject.AddComponent("FlareLayer"); |
| 179 | go.hideFlags = HideFlags.HideAndDontSave; |
| 180 | m\_ReflectionCameras[currentCamera] = reflectionCamera; |
| 181 | } |
| 182 | } |
| 183 |  |
| 184 | *// Extended sign: returns -1, 0 or 1 based on sign of a* |
| 185 | **private** **static** **float** sgn(**float** a) |
| 186 | { |
| 187 | **if** (a > 0.0f) **return** 1.0f; |
| 188 | **if** (a < 0.0f) **return** -1.0f; |
| 189 | **return** 0.0f; |
| 190 | } |
| 191 |  |
| 192 | *// Given position/normal of the plane, calculates plane in camera space.* |
| 193 | **private** Vector4 CameraSpacePlane (Camera cam, Vector3 pos, Vector3 normal, **float** sideSign) |
| 194 | { |
| 195 | Vector3 offsetPos = pos + normal \* m\_ClipPlaneOffset; |
| 196 | Matrix4x4 m = cam.worldToCameraMatrix; |
| 197 | Vector3 cpos = m.MultiplyPoint( offsetPos ); |
| 198 | Vector3 cnormal = m.MultiplyVector( normal ).normalized \* sideSign; |
| 199 | **return** new Vector4( cnormal.x, cnormal.y, cnormal.z, -Vector3.Dot(cpos,cnormal) ); |
| 200 | } |
| 201 |  |
| 202 | *// Adjusts the given projection matrix so that near plane is the given clipPlane* |
| 203 | *// clipPlane is given in camera space. See article in Game Programming Gems 5.* |
| 204 | **private** **static** **void** CalculateObliqueMatrix (**ref** Matrix4x4 projection, Vector4 clipPlane) |
| 205 | { |
| 206 | Vector4 q = projection.inverse \* new Vector4(sgn(clipPlane.x),sgn(clipPlane.y),1.0f,1.0f |
| 207 | ); |
| 208 | Vector4 c = clipPlane \* (2.0F / (Vector4.Dot (clipPlane, q))); |
| 209 | *// third row = clip plane - fourth row* |
| 210 | projection[2] = c.x - projection[3]; |
| 211 | projection[6] = c.y - projection[7]; |
| 212 | projection[10] = c.z - projection[11]; |
| 213 | projection[14] = c.w - projection[15]; |
| 214 | } Unity3D教程手册 |
| 215 |  |
| 216 | *// Calculates reflection matrix around the given plane* |
| 217 | **private** **static** **void** CalculateReflectionMatrix (**ref** Matrix4x4 reflectionMat, Vector4 plane) |
| 218 | { |
| 219 | reflectionMat.m00 = (1F - 2F\*plane[0]\*plane[0]); |
| 220 | reflectionMat.m01 = ( - 2F\*plane[0]\*plane[1]); |
| 221 | reflectionMat.m02 = ( - 2F\*plane[0]\*plane[2]); |
| 222 | reflectionMat.m03 = ( - 2F\*plane[3]\*plane[0]); |
| 223 |  |
| 224 | reflectionMat.m10 = ( - 2F\*plane[1]\*plane[0]); |
| 225 | reflectionMat.m11 = (1F - 2F\*plane[1]\*plane[1]); |
| 226 | reflectionMat.m12 = ( - 2F\*plane[1]\*plane[2]); |
| 227 | reflectionMat.m13 = ( - 2F\*plane[3]\*plane[1]); |
| 228 |  |
| 229 | reflectionMat.m20 = ( - 2F\*plane[2]\*plane[0]); |
| 230 | reflectionMat.m21 = ( - 2F\*plane[2]\*plane[1]); |
| 231 | reflectionMat.m22 = (1F - 2F\*plane[2]\*plane[2]); |
| 232 | reflectionMat.m23 = ( - 2F\*plane[3]\*plane[2]); |
| 233 |  |
| 234 | reflectionMat.m30 = 0F; |
| 235 | reflectionMat.m31 = 0F; |
| 236 | reflectionMat.m32 = 0F; |
| 237 | reflectionMat.m33 = 1F; |
| 238 | } |
| 239 | } |
| 240 |  |

文件名MirrorReflection.shader

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Shader "FX/Mirror Reflection" { |
| 02 | Properties { |
| 03 | \_MainTex ("Base (RGB)", 2D) = "white" {} |
| 04 | \_ReflectionTex ("Reflection", 2D) = "white" { TexGen ObjectLinear } |
| 05 | } |
| 06 |  |
| 07 | *// two texture cards: full thing* |
| 08 | Subshader { |
| 09 | Pass { |
| 10 | SetTexture[\_MainTex] { combine texture } |
| 11 | SetTexture[\_ReflectionTex] { matrix [\_ProjMatrix] combine texture \* previous } |
| 12 | } |
| 13 | } |
| 14 |  |
| 15 | *// fallback: just main texture* |
| 16 | Subshader { |
| 17 | Pass { |
| 18 | SetTexture [\_MainTex] { combine texture } |
| 19 | } |
| 20 | } |
| 21 | } |