镜面反射是指当一束平行入射的光线射入到一个平面时，能平行地向一个方向反射出来。而本篇文章要给大家讲解的是镜面反射的原理以及在Unity3D中如何实现镜面反射，一起来看看吧。

1、前言

本文章旨在与大家一起探讨学习新知识，如有疏漏或者谬误，请大家不吝指出。

PS：GAD平台导入word后，公式都转成图片了，而且像素很低，看起来模糊，我这边是手动截图替换掉原来的公式，当然，还是不美观，大家看看习不习惯吧。

2、概述

首先我们先明确概念，镜面反射是指当一束平行入射的光线射入到一个平面时，能平行地向一个方向反射出来。我们论述的重点在于实现平面的反射效果，例如其他凹面和不规则面则不适用于此方法（凹面和不规则面的镜面反射可以考虑通过cube map来实现）。。

3、反射矩阵原理

让我们先回顾一下相关的物理知识。如下图所示，视点所接受的光线是通过镜面反射进入视点的，于是在人脑中出现了一个与镜面相对称的虚像。

那么从物理学中我们知道，镜面反射的虚像和实像是与镜面对称的，虚像和实像的顶点连线与镜面垂直，且顶点到镜面的垂直距离是相同的。然后我们考虑使用数学来表述上述规律。即，已知实像的各个顶点坐标与镜面，求实像相对于镜面的各个顶点虚像坐标。这里面其实我们只需要完成一个顶点的变换，其他顶点的变换都是相同的。我们接下来求解这个数学问题。

假设Q(x, y, z)是实像上一个点，三维空间中平面使用等式来表示，其中P是平面上任意一点，N向量是平面的法向量，用(nx, ny, nz)表示，d是原点到平面的距离，P0(x0, y0, z0)是平面上一点。

于是我们可以用下面的等式表示平面的集合：

要求出Q’的坐标，只需要求出Q点到平面的距离D就可以了。联合下列公式求得距离D(θ是指的夹角)：

假设向量n为单位向量，则有：

那么很明显Q’的坐标就等于Q点顺着QP方向移动2D的距离：

其他y’和z’可以依次解出：

由此我们可以得到一个反射矩阵R。

建议大家可以拿个草稿纸自己在纸上算一遍，虽然说不是很难，但是一些高深的知识不就是由基础知识堆积而成的么。

4、实现反射矩阵

考虑基本的渲染管线中的坐标变换，一般我们使用MVP来表示将一个点从物体坐标系转换到裁剪坐标系，其中M(model\_matrix)表示将点从物体坐标系转换到世界坐标系；V(view\_matrix)表示将点从世界坐标系变换到视点坐标系(摄像机坐标系);P(project\_matrix)表示将点从视点坐标系转换到裁剪坐标系。如果我们获取到了在世界坐标系下平面的法线向量和d（d可以通过平面上任意一点求得），那就可以求出在世界坐标系下将顶点转换到反射点的反射矩阵了。于是我们就可以在完成了M矩阵变换后，进行反射矩阵的变换，然后再接着完成V和P矩阵的变换。接着思考，在unity3d中，摄像机有个worldToCameraMatrix变量，这个变量就是V，那我们可以这样做：V=V\*R，这样经过MVP矩阵变换的顶点不就自然而然的经过了反射矩阵的变换了么？所以我们考虑复制一个当前摄像机（这可以通过Camera.CopyFrom来实现），将这个摄像机的worldToCameraMatrix乘以反射矩阵R，那么这个摄像机渲染出来的物体就是虚像啦。我们来看看具体的实现代码：

void RenderRefection()

{

Vector3 normal = Panel.up;

float d = -Vector3.Dot (normal, Panel.position);

Matrix4x4 refMatrix = new Matrix4x4();

refMatrix.m00 = 1-2\*normal.x\*normal.x;

refMatrix.m01 = -2\*normal.x\*normal.y;

refMatrix.m02 = -2\*normal.x\*normal.z;

refMatrix.m03 = -2\*d\*normal.x;

refMatrix.m10 = -2\*normal.x\*normal.y;

refMatrix.m11 = 1-2\*normal.y\*normal.y;

refMatrix.m12 = -2\*normal.y\*normal.z;

refMatrix.m13 = -2\*d\*normal.y;

refMatrix.m20 = -2\*normal.x\*normal.z;

refMatrix.m21 = -2\*normal.y\*normal.z;

refMatrix.m22 = 1-2\*normal.z\*normal.z;

refMatrix.m23 = -2\*d\*normal.z;

refMatrix.m30 = 0;

refMatrix.m31 = 0;

refMatrix.m32 = 0;

refMatrix.m33 = 1;

RefCamera.worldToCameraMatrix = Camera.main.worldToCameraMatrix \* refMatrix;

//在计算漫反射等光照效果时，需要使用顶点的normal和view向量，view跟摄像机位置有关，所以我们也对refcamera做反射变换

RefCamera.transform.position = refMatrix.MultiplyPoint(Camera.main.transform.position);

//以下部分是变换摄像机的方向向量，当然其实这里没有必要，你可以删掉它

Vector3 forward = Camera.main.transform.forward;

//Vector3 up = Camera.main.transform.up;

forward = refMatrix.MultiplyVector(forward);

//up = refMatrix.MultiplyVector(up);

//Quaternion refQ = Quaternion.LookRotation (forward, up);

//RefCamera.transform.rotation = refQ;

RefCamera.transform.forward = forward;

GL.invertCulling = true;

RefCamera.Render();

GL.invertCulling = false;

//将贴图传递给shader

RefCamera.targetTexture.wrapMode = TextureWrapMode.Repeat;

RefMat.SetTexture("\_RefTexture", RefCamera.targetTexture);

}

这是设置反射摄像机的脚本，它负责变换反射摄像机，并设置其渲染到纹理，然后将反射纹理交给镜面的shader来处理。读者需要注意到以下几点：

1、一般来说我们获取平面的方向向量（平面的up朝向）和平面上一点（平面的坐标）会比较容易，那么我们就需要求出方程中的d，而d的求解方式在原理的论述中有提到相关方法，大家可以试着求解，这里直接给出结果。

2、注意顶点经过镜面反射后，需要对背面消隐做反转操作才能正确渲染出来，即GL.invertCulling设置为true。原因是因为我们只对顶点做了反射变换，而法线是没有做反射变换的，那么在进行背面消隐时就会发生剔除错误，所以我们需要对culling做反转操作（OpenGL中顶点绘制顺序是逆时针为正）。

3、这个脚本需要挂在镜面上，当镜面的OnWillRender函数被执行时，我们知道镜面在主摄像机中将要被渲染，这时我们进行反射矩阵的变换，而如果镜面没有被主摄像机渲染到，那么我们就不需要计算反射变换了，这样做可以减少不必要的消耗。

4、镜面需要设置成water层（或者其他自定义层），复制出来的RefCamera在渲染的时候culling mask要设置成忽略water。这样做RefCamera就不会渲染镜面了。如果RefCamera没有这么设置，会发生递归剔除的错误。因为你在镜面的OnWillRender函数里面调用了Camera.Render，而Camera.Render里发现我需要渲染镜面，于是又去调用镜面的OnWillRender。

而shader所要做的事情是计算出镜面在屏幕空间上对应的坐标，然后以此作为UV值去反射纹理取出对应的颜色值，最后使用取出的颜色值与最终颜色做叠加操作就大功告成了，下面是shader的部分代码：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

v2f vert (appdata v)

{

v2f o;

o.vertex = mul(UNITY\_MATRIX\_MVP, v.vertex);

o.uv = TRANSFORM\_TEX(v.uv, \_MainTex);

//ComputeScreenPos是内置函数

o.ScreenPos = ComputeScreenPos(o.vertex);

UNITY\_TRANSFER\_FOG(o,o.vertex);

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

// sample the texture

fixed4 col = tex2D(\_MainTex, i.uv)\*\_Color;

half4 reflectionColor = tex2D(\_RefTexture, i.ScreenPos.xy/i.ScreenPos.w);

col += reflectionColor;

// apply fog

UNITY\_APPLY\_FOG(i.fogCoord, col);

return col;

}

以上只给出了部分关键代码，更完整的请下载附件参考（Unity3D 5.3.4版本）。