**方式1：通过ndc坐标重建**

float d = SAMPLE\_DEPTH\_TEXTURE(\_CameraDepthTexture, i.uv\_depth);

float4 H = float4(i.uv.x \* 2 - 1, i.uv.y \* 2 - 1, d \* 2 - 1, 1);

float4 D = mul(\_CurrentViewProjectionInverseMatrix, H);

float4 worldPos = D / D.w;

证明：

按正常来说

WorldPos = (vp)^-1 \* clip.w \* ndc.xyzw = clip.w \* (vp)^-1 \* ndc.xyzw = (wx, wy, wz, 1)

Clip.w \* [(vp)^-1 \* ndc.xyzw].w = 1

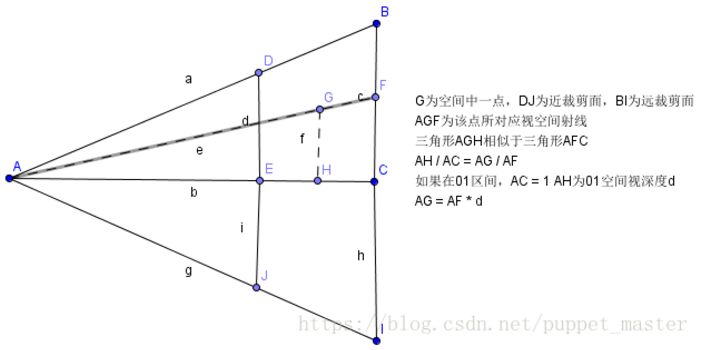
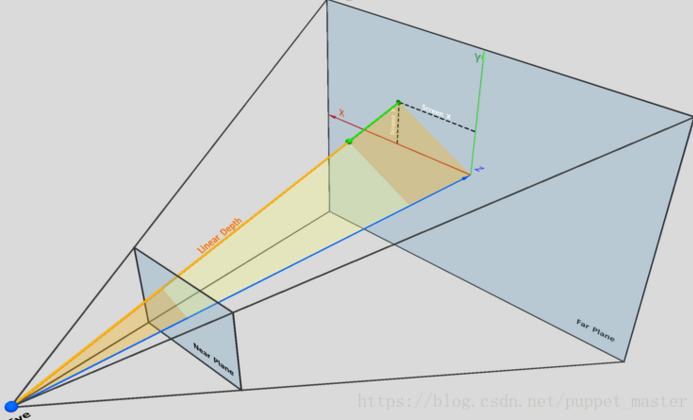
得到：clip.w = 1 / [(vp)^-1 \* ndc.xyzw].w

worldPos = (vp)^-1 \* ndc.xyzw / [(vp)^-1 \* ndc.xyzw].w

**方式2：屏幕射线插值方法重建**

上面的方法虽然可以得到世界坐标，但是需要逐像素进行矩阵计算，性能比较差。

这种射线方式的重建，是Cryket公司2011年创建的



也就是通过插值得到每个片元的射线，这个插值是通过四边形的四个顶点射线得到的

由上图可知，最终世界坐标位置

worldPos = OA + AG(世界坐标系上的向量)

其中OA就是摄像机在世界坐标系的位置，AG是世界坐标系上的向量

看看在C#如何实现：

void OnRenderImage(RenderTexture src, RenderTexture dest)

{

Matrix4x4 frustumCorners = Matrix4x4.identity;

float fov = camera.fieldOfView;

float far = camera.farClipPlane;

float near = camera.nearClipPlane;

float aspect = camera.aspect;

float halfHeight = near \* Mathf.Tan(fov \* 0.5f \* Mathf.Deg2Rad);

// toRight与toTop是世界空间坐标系下的向量（该向量的起点为近平面的中心点）

Vector3 toRight = cameraTransform.right \* halfHeight \* aspect;

Vector3 toTop = cameraTransform.up \* halfHeight;

// topLeft为向量（起点为世界坐标摄像机位置，终点为世界坐标四边形左上角）

Vector3 topLeft = cameraTransform.forward \* near + toTop - toRight;

/\* 现在已知摄像机到topLeft的向量，我们现在需要得到一个深度为depth像素的世界坐标

\* 也就是只需要用topLeft的单位向量\*摄像机到该像素的长度即可

\* 像素长度 / |topLeft| = depth / near

\* 得到像素长度 = depth \* |topLeft| / near

\*/

float scale = topLeft.magnitude / near;

topLeft.Normalize();

topLeft \*= scale;

Vector3 topRight = cameraTransform.forward \* near + toTop + toRight;

topRight.Normalize();

topRight \*= scale;

Vector3 bottomLeft = cameraTransform.forward \* near - toTop - toRight;

bottomLeft.Normalize();

bottomLeft \*= scale;

Vector3 bottomRight = cameraTransform.forward \* near - toTop + toRight;

bottomRight.Normalize();

bottomRight \*= scale;

frustumCorners.SetRow(0, bottomLeft);

frustumCorners.SetRow(1, bottomRight);

frustumCorners.SetRow(2, topRight);

frustumCorners.SetRow(3, topLeft);

material.SetMatrix("\_FrustumCornerRay", frustumCorners);

material.SetMatrix("\_ViewProjectionInverseMatrix", (camera.projectionMatrix \* camera.worldToCameraMatrix).inverse);

material.SetFloat("\_FogDensity", fogDesity);

material.SetColor("\_FogColor", fogColor);

material.SetFloat("\_FogStart", fogStart);

material.SetFloat("\_FogEnd", fogEnd);

if (material)

{

Graphics.Blit(src, dest, material);

}

else

{

Graphics.Blit(src, dest);

}

}

Shader "Unlit/FogWithDepthTexture"

{

Properties

{

\_MainTex ("Texture", 2D) = "white" {}

\_FogDensity ("Fog Density", Float) = 1.0

\_FogColor ("Fog Color", Color) = (1.0, 1.0, 1.0, 1.0)

\_FogStart ("Fog Start", Float) = 0.0

\_FogEnd ("Fog End", Float) = 1.0

}

CGINCLUDE

#include "UnityCG.cginc"

sampler2D \_CameraDepthTexture;

sampler2D \_MainTex;

float4 \_MainTex\_ST;

float4 \_MainTex\_TexelSize;

float \_FogDensity;

fixed4 \_FogColor;

float \_FogStart;

float \_FogEnd;

float4x4 \_FrustumCornerRay;

struct v2f

{

float4 pos : SV\_POSITION;

half2 uv : TEXCOORD0;

half2 uv\_depth : TEXCOORD1;

float4 interpolatedRay : TEXCOORD2;

};

v2f vert(appdata\_img v)

{

v2f o;

o.pos = UnityObjectToClipPos(v.vertex);

o.uv = v.texcoord;

o.uv\_depth = v.texcoord;

int index = 0;

if (v.texcoord.x < 0.5 && v.texcoord.y < 0.5)

{

index = 0;

}

else if (v.texcoord.x >= 0.5 && v.texcoord.y < 0.5)

{

index = 1;

}

else if (v.texcoord.x >= 0.5 && v.texcoord.y >= 0.5)

{

index = 2;

}

else

{

index = 3;

}

#if UNITY\_UV\_STARTS\_AT\_TOP

if (\_MainTex\_TexelSize.y < 0) // 如果没有开启抗锯齿则unity会自动翻转，此时y>0

{

o.uv\_depth = 1 - o.uv\_depth.y;

index = 3 - index;

}

#endif

o.interpolatedRay = \_FrustumCornerRay[index];

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

float linearDepth = LinearEyeDepth(SAMPLE\_DEPTH\_TEXTURE(\_CameraDepthTexture, i.uv\_depth));

// 通过深度计算世界坐标

float3 worldPos = \_WorldSpaceCameraPos + linearDepth \* i.interpolatedRay.xyz;

// 计算雾

float fogDensity = (\_FogEnd - worldPos.y) / (\_FogEnd - \_FogStart);

fogDensity = saturate(fogDensity \* \_FogDensity);

fixed4 finalColor = tex2D(\_MainTex, i.uv);

finalColor.rgb = lerp(finalColor.rgb, \_FogColor.rgb, fogDensity);

return finalColor;

}

ENDCG

SubShader

{

Tags { "RenderType"="Opaque" }

Pass

{

ZWrite Off

ZTest Off

Cull Off

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

ENDCG

}

}

}

