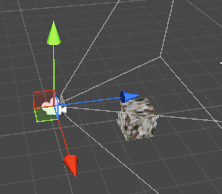
<https://blog.csdn.net/mobilebbki399/article/details/50512059>

<https://blog.csdn.net/mobilebbki399/article/details/50559732>

<https://www.cnblogs.com/zsb517/p/6655546.html>

<https://www.jianshu.com/p/4e8162ed0c8d>

**\_CameraDepthTexture**



在上面的例子中，摄像机的位置：



正方体的位置：



正方体的边长为1

也就是说，最前面的位置为-7.5，那么距离摄像机的z值为2.5个单位

获取深度贴图渲染流程：

在摄像机上添加脚本：

void Start()

{

**gameObject.GetComponent<Camera>().depthTextureMode |= DepthTextureMode.Depth; // 重点**

}

void OnRenderImage(RenderTexture src, RenderTexture dest)

{

if (material)

{

Graphics.Blit(src, dest, material);

}

else

{

Graphics.Blit(src, dest);

}

}

摄像机脚本相应的shader：

Shader "Unlit/CameraDepthNormal"

{

Properties

{

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

**//\_CameraDepthTexture ("Depth Texture", 2D) = "white" {} // 注意这里千万不要添加**

}

SubShader

{

Tags { "RenderType"="Opaque" }

Pass

{

ZWrite Off

ZTest Always

Cull Off

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

sampler2D \_MainTex;

sampler2D \_CameraDepthTexture;

float4 \_MainTex\_ST;

v2f vert (appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

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

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

float depth = SAMPLE\_DEPTH\_TEXTURE(\_**CameraDepthTexture**, i.uv);

float linearDepth = Linear01Depth(depth);

return fixed4(depth, depth, depth, 1.0);

}

ENDCG

}

}

FallBack "Legacy Shaders/Diffuse"

}

注意，物体要能够渲染到深度纹理中，必须要满足几个条件:

1. 物体的材质shader的Queue小于等于2500（已经证实）
2. 必须包括LightMode=ShadowCaster的pass（已经证实）

Tags { "Queue" = "Geometry+501"}这就不能产生阴影了，Tags { "Queue" = "Geometry+500"}这可以产生阴影

根据计算公式我们很容易可以得到，最终的

Depth= -(near+far)/(near-far) – 2\*near\*far/((near-far)\*z)

将near=0.3 far=10 z=-2.5带入得到：depth=0.81443299

最终这个深度要映射到[0,1]

所有最后映射到屏幕上的depth=255\*（0.81443299+1）/2=255\*0.907216=231.34

最终得到的最前面的像素值为(231,231,231)，和我们计算得到的结果一样

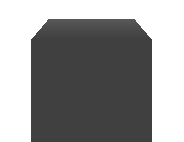
反向推导视觉空间系下的depth

我们通过231，可以得到ndc坐标系下的depth=0.81443299

摄像机坐标系下的depth=-2\*near\*far/((near-far)\*ndc下的depth+near+far)=-2.5和我们得到的结果一样

我们可以通过unity提供的函数得到在摄像机坐标系下归一化的值（实际上就是用当前值除以far）：

2.5/10=0.25\*255=64，和我们显示得到的结果一致



查看Linear01Depth函数：

inline float Linear01Depth( float z )

{

return 1.0 / (\_ZBufferParams.x \* z + \_ZBufferParams.y);

}

float4 \_ZBufferParams //x=1-Far/Near,y=Far/Near,z=x/Far,w=y/Far,该变量用于线性化Z缓存中的深度值

**\_CameraDepthNormalsTexture**

Shader "Unlit/CameraDepthNormal"

{

Properties

{

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

//\_CameraDepthTexture ("Depth Texture", 2D) = "white" {} // 注意这里千万不要添加

}

SubShader

{

Tags { "RenderType"="Opaque" }

Pass

{

ZWrite Off

ZTest Always

Cull Off

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

sampler2D \_MainTex;

sampler2D \_CameraDepthTexture;

sampler2D \_CameraDepthNormalsTexture;

float4 \_MainTex\_ST;

v2f vert (appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

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

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

float depth = 0.0;

float3 normal = float3(0.0, 0.0, 0.0);

DecodeDepthNormal(tex2D(**\_CameraDepthNormalsTexture**, i.uv), depth, normal);

return fixed4(depth, depth, depth, 1.0);

}

ENDCG

}

}

FallBack "Legacy Shaders/Diffuse"

}

**注意：对于像渲染出深度+法线纹理的物体，Tags必须由“RenderType”=“Opaque”标签，否则是不会生成深度+法线纹理图的**