唐老狮系列教程

利用深度纹理实现全局雾效的基本原理

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

主要讲解内容

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

主要讲解内容

1. 知识回顾 全局雾效的三种计算方式
2. 为什么要实现屏幕后处理效果的全局雾效
3. 利用深度纹理实现 全局雾效的基本原理

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

知识回顾 全局雾效的三种计算方式

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

全局雾效的三种计算方式

Linear（线性）： $f = (end - |d|) / (end - start)$

Exponential（指数）： $f = 1 - e^{-density*|d|}$

Exponential Squared（指数的平方）： $f = 1 - e^{-(density-|d|)^2}$

它们都是用来计算雾的混合因子 **f** 的，

有了混合因子，会用雾的颜色和物体本来的颜色进行混合计算

最终的颜色 = （1-f）* 物体的颜色 + f * 雾的颜色

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

为什么要实现屏幕后处理效果的全局雾效

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

为什么要实现屏幕后处理效果的全局雾效

既然Unity中已经提供了全局雾效，**那为什么我们还要自己来实现呢?**

主要是因为**Unity自带的全局雾效有以下几个缺点:**

1. **需要为每个自定义Shader按规则书写雾效处理代码**
2. **自带的全局雾效无法实现一些自定义效果**

   比如:

   基于高度的雾效 - 可以用来做出悬浮的水雾效果

   不规则的雾效（结合噪声图实现） - 可以为雾增加随机性和不规则形

   动态变化的雾、基于纹理的雾 等等

总体而言，就是Unity自带的全局雾效只能满足最基础的效果，较为局限。



唐老狮系列教程-深度纹理实现全局雾效基本原理

为什么要实现屏幕后处理效果的全局雾效

因此我们通过

结合深度纹理来制作屏幕后处理的全局雾效，来让大家感受同一种效果的不同实现思路。

而基于深度纹理的全局雾效，它相对于Unity自带的全局雾效的好处是：

1. **一次屏幕后处理便可以得到雾的效果，不用为每个自定义Shader添加雾效代码**
2. **可以基于该全局雾效拓展出多种雾效，可以方便的模拟出线性、指数、指数平方雾效，甚至实现一些基于高度的雾效、使用噪声图的雾效、动态变化的雾效等等**



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

Linear（线性）： $f = (end - |d|) / (end - start)$

Exponential（指数）： $f = 1 - e^{-density*|d|}$

Exponential Squared（指数的平方）： $f = 1 - e^{-(density-|d|)^2}$

通过这三个公式，我们发现，想要计算全局雾效

关键点是得到离摄像机的距离，因为对于 雾开始的位置start、雾最浓的位置end、雾的浓度density、自然对数的底e，它们都是已知的或自定义的。



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

因此我们想要实现基于深度纹理的屏幕后处理的全局雾效的

关键点就是**如何利用深度纹理来获得每个像素在世界空间下的位置?**

这样才能计算出离摄像机的距离，才能利用雾的计算公式计算混合因子来实现雾效!

回忆之前学习的基于**深度纹理实现的运动模糊相关知识中**，我们也利用了深度纹理获取了像素点在世界空间下的位置

```
//1.得到裁剪空间下的两个点
//获取深度值
float depth = SAMPLE_DEPTH_TEXTURE(_CameraDepthTexture, i.uv_depth);
//裁剪空间下的一个组合坐标 把0~1范围变换到-1~1范围
//第一个点
float4 nowClipPos = float4(i.uv.x * 2 - 1, i.uv.y * 2 - 1, depth * 2 - 1, 1);
//用裁剪空间到世界空间的变换矩阵 得到 世界空间下的带你
float4 worldPos = mul(_ClipToWorldMatrix, nowClipPos);
//透视除法
worldPos /= worldPos.w;
```



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

```
//1.得到裁剪空间下的两个点
//获取深度值
float depth = SAMPLE_DEPTH_TEXTURE(_CameraDepthTexture, i.uv_depth);
//裁剪空间下的一个组合坐标 把0~1范围变换到-1~1范围
//第一个点
float4 nowClipPos = float4(i.uv.x * 2 - 1, i.uv.y * 2 - 1, depth * 2 - 1, 1);
//用裁剪空间到世界空间的变换矩阵 得到 世界空间下的带你
float4 worldPos = mul(_ClipToWorldMatrix, nowClipPos);
//透视除法
worldPos /= worldPos.w;
```

但是这种做法有一个**很大的缺点**，那就是**性能消耗较高**，原因主要有以下两点：

1. **在片元着色器函数中进行计算，计算次数较大**
2. **每次都进行了矩阵变换计算，计算量较大**

因此在**实现全局雾效时**，我们将**不会使用这种方式**，而是使用一种性能更好的计算方式！

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

这种性能更好的新方法的主要思路还是**利用深度纹理来获得每个像素在世界空间下的位置**

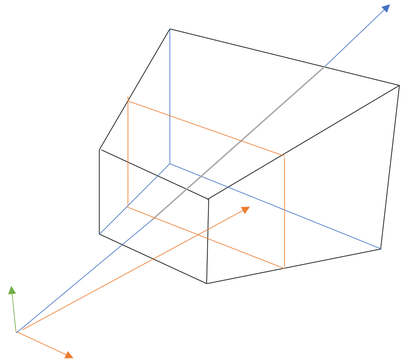
除此以外我们**还需要获得摄像机指向像素对应世界坐标的方向向量**

并**利用坐标偏移的方式得到像素的世界坐标！**

像素的世界坐标 = 摄像机位置 + 观察空间线性深度值 * 摄像机指向像素世界坐标的方向向量

而其中摄像机位置已知，观察空间线性深度值已知（从深度纹理中采样后计算）

那么关键点就是讲解如何计算出 摄像机指向像素世界坐标的方向向量



WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

关键思路：

顶点着色器中：

1. 屏幕后处理中处理的内容是一张抓取的屏幕图像，相当于是一个面片，它具有4个顶点（四个角）

3. 通过C#代码计算四个顶点在世界坐标系下的射线方向后传递给顶点着色器

片元着色器中：

当数据传递到片元着色器要处理每个像素时，像素对应的射线方向是基于4个顶点的射线插值计算而来（无需我们自己计算）

3. 利用 像素世界坐标 = 摄像机位置 + 深度值 * 世界空间下射线方向 得到对应像素在世界空间下位置

4. 利用得到的世界空间下位置利用雾的公式计算出对应雾效颜色

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

顶点着色器中：

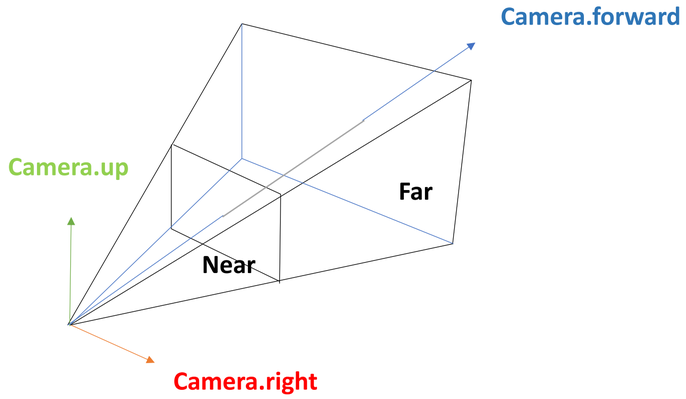
1. 屏幕后处理中处理的内容是一张抓取的屏幕图像，相当于是一个面片，它具有4个顶点（四个角）

我们其实可以认为屏幕后处理中处理的**屏幕图像的四个顶点**，就是摄像机视锥体中**近裁剪面的四个角**

因为在裁剪空间变换中，我们是将观察空间变换到了裁剪空间再到NDC空间中，

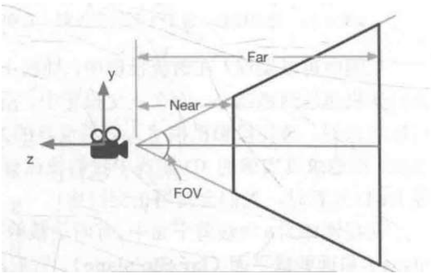
最终又变换到了屏幕空间中，可以理解相当于把近裁剪面变换到了屏幕空间中。

因此近裁剪面的四个角相当于是屏幕图像四个顶点在世界空间下的位置



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理



顶点着色器中：

2. 通过C#代码计算四个顶点在世界坐标系下的射线方向后传递给顶点着色器

这一步我们可以在C#中计算好，然后将结果作为参数传递到Shader的变量中，在顶点着色器中使用即可

halfH = Near * tan(FOV/2)　halfW = halfH * aspect `print(Camera.main.aspect);` 摄像机参数得到Game窗口的宽高比 Aspect = 宽 : 高 = 宽 / 高

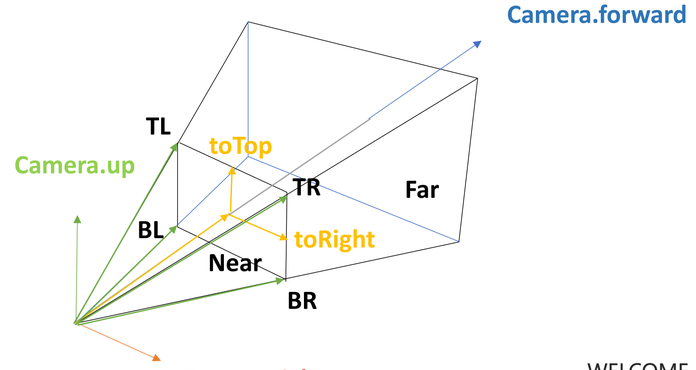
toTop = Camera.up * halfH　　toRight = Camera.Right * halfW

TL = Camera.forward * Near + toTop – toRight

TR = Camera.forward * Near + toTop + toRight

BL = Camera.forward * Near - toTop – toRight

BR = Camera.forward * Near - toTop + toRight



WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

halfH = Near * tan(FOV/2) halfW = halfH * aspect

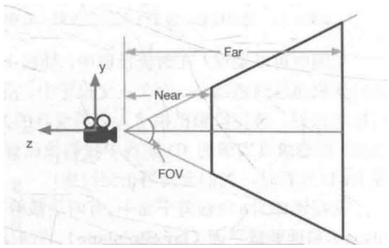
toTop = Camera.up * halfH toRight = Camera.Right * halfW

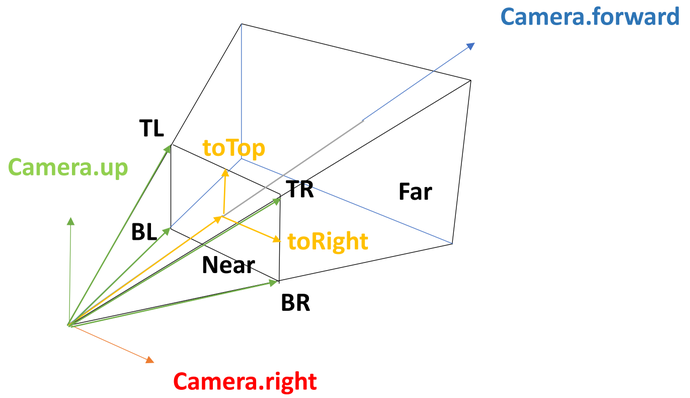
TL = Camera.forward * Near + toTop – toRight

TR = Camera.forward * Near + toTop + toRight

BL = Camera.forward * Near - toTop – toRight

BR = Camera.forward * Near - toTop + toRight





推导出来了四个顶点的方向向量，我们是不是就可以利用它们得到四个顶点的世界空间下坐标了呢？

比如得到左上角的方向向量的单位向量，然后乘以左上角顶点对应像素点的深度值

左上角像素点对应世界坐标 = 摄像机位置 + TL.Normalized * Depth；

注意，如果这样去计算，那么得到的结果是错误的！！！！

因为深度值Depth即使我们将其转换为观察空间下的线性值，它表示的也是离摄像机在Z轴方向的距离，并不是两点之间的距离（欧式距离），因此我们还需要对该向量进行处理！

唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

halfH = Near * tan(FOV/2) halfW = halfH * aspect

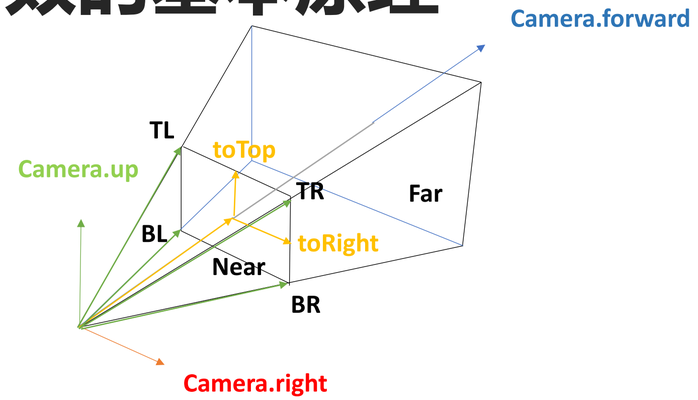
toTop = Camera.up * halfH toRight = Camera.Right * halfW

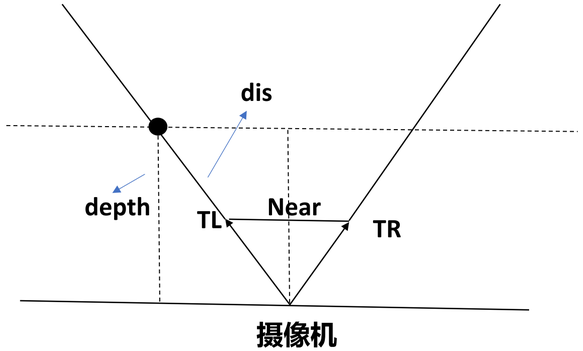
TL = Camera.forward * Near + toTop – toRight

TR = Camera.forward * Near + toTop + toRight

BL = Camera.forward * Near - toTop – toRight

BR = Camera.forward * Near - toTop + toRight





我们可以利用相似三角形的原理，推导出深度值和两点之间距离（欧式距离）的关系。

$$\frac{Depth}{Near} = \frac{dis}{|TL|} \quad => \quad dis = \frac{|TL|}{Near} * Depth$$

而 左上角像素点对应世界坐标 = 摄像机位置 + TL.Normalized * Depth 就变为了

左上角像素点对应世界坐标 = 摄像机位置 + TL.Normalized * |TL|/Near * Depth

那也就意味着，真正最终和深度一起计算的确定世界坐标位置的方向向量其实就是 TL.Normalized * |TL|/Near

由于近裁剪面4个点是对称的， |TL|/Near 可以通用 ， 只需要变换前面的单位向量即可

唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

halfH = Near * tan(FOV/2) halfW = halfH * aspect

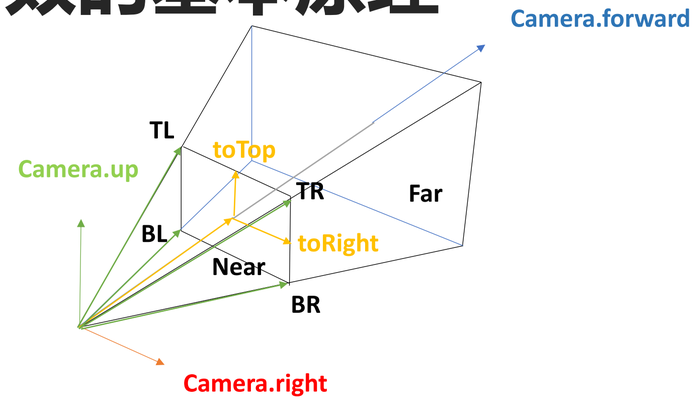
toTop = Camera.up * halfH toRight = Camera.Right * halfW

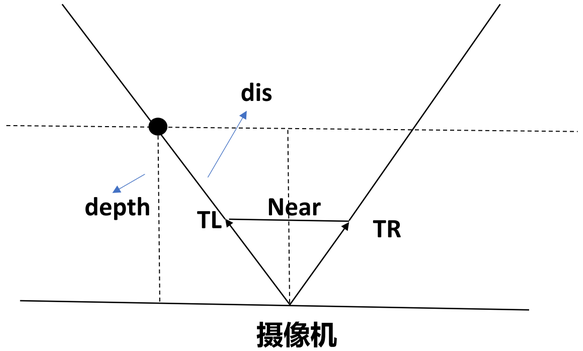
TL = Camera.forward * Near + toTop – toRight

TR = Camera.forward * Near + toTop + toRight

BL = Camera.forward * Near - toTop – toRight

BR = Camera.forward * Near - toTop + toRight





通过推导，我们已经得到了四个顶点对应的方向向量信息了

Scale = |TL|/Near

RayTL = TL.Normalized * Scale

RayTR = TR.Normalized * Scale

RayBL = BL.Normalized * Scale

RayBR = BR.Normalized * Scale

我们只需要在顶点着色器中根据顶点的位置设置对应的向量即可！

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

片元着色器中：

当数据传递到片元着色器要处理每个像素时，像素对应的射线方向是基于4个顶点的射线插值计算而来（无需我们自己计算）

3. 利用 像素世界坐标 = 摄像机位置 + 深度值 * 世界空间下射线方向 得到对应像素在世界空间下位置

我们已经有了对应的射线，直接从深度纹理中采样获取深度值

并利用 LinearEyeDepth 内置函数得到像素到摄像机的实际距离 便可以利用上面的公式进行计算

得到每个像素在世界空间下的位置了



唐老狮系列教程-深度纹理实现全局雾效基本原理

利用深度纹理实现全局雾效的基本原理

片元着色器中：

4. 利用得到的世界空间下位置利用雾的公式计算出对应雾效颜色

有了世界空间下的位置，我们就可以利用雾的计算公式进行雾效的混合因子计算

利用算出的雾效混合因子参与雾颜色和像素颜色的混合运算即可

注意：在之后的具体实现中，我们将基于线性公式实现基于高度的全局雾效



唐老狮系列教程-深度纹理实现全局雾效基本原理

总结

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度纹理实现全局雾效基本原理

主要讲解内容

1. 知识回顾 全局雾效的三种计算方式

   线性、指数、指数的平方，他们是用来计算混合因子的

2. 为什么要实现屏幕后处理效果的全局雾效

   一次屏幕后处理便可以得到雾的效果，不用为每个自定义Shader添加雾效代码

   可以基于该全局雾效拓展出多种雾效，可以方便的模拟出线性、指数、指数平方雾效，

   甚至实现一些基于高度的雾效、使用噪声图的雾效、动态变化的雾效等等

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度纹理实现全局雾效基本原理

主要讲解内容

3. 利用深度纹理实现 全局雾效的基本原理

像素的世界坐标 = 摄像机位置 + 观察空间线性深度值 * 摄像机指向像素世界坐标的方向向量

利用深度纹理实现全局雾效的基本原理

halfH = Near * tan(FOV/2) halfW = halfH * aspect

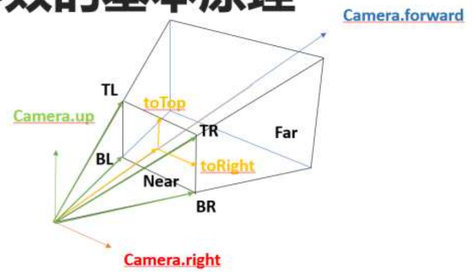
toTop = Camera.up * halfH toRight = Camera.Right * halfW

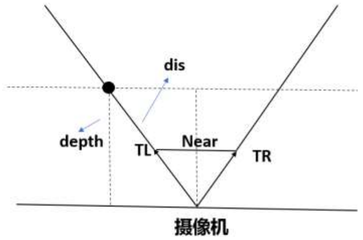
TL = Camera.forward * Near + toTop – toRight

TR = Camera.forward * Near + toTop + toRight

BL = Camera.forward * Near - toTop – toRight

BR = Camera.forward * Near - toTop + toRight





通过推导，我们已经得到了四个顶点对应的方向向量信息了

Scale = |TL|/Near

RayTL = TL.Normalized * Scale

RayTR = TR.Normalized * Scale

RayBL = BL.Normalized * Scale

RayBR = BR.Normalized * Scale



唐老狮系列教程

Thank

谢谢您的聆听

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY