立方体纹理

立方体纹理是环境映射的一种实现方法。立方体纹理一共包含了6张图像，这些图像对应了一个立方体的6个面，立方体纹理的名称也由此而来。立方体的每个面表示沿着世界空间下的轴向（上、下、左、右、前、后）观察所得的图像。对立方体纹理采样，要提供一个三维的纹理坐标，这个三维纹理坐标表示了我们在世界空间下的一个3D方向。这个方向矢量从立方体的中心出发，当它向外部延伸时就会和立方体的6个纹理之一发生相交，而采样得到的结果就是由该交点计算而来的。

示意图：

形状, 矩形

描述已自动生成

Ps：什么是环境映射？环境映射是一种用于模拟物体周围环境的光照和反射效果的渲染技术。环境映射通常在着色器中实现，因此可以称为环境映射着色器。

使用立方体纹理的优缺点：

优点：

实现简单快速，得到的效果也比较好。

缺点：

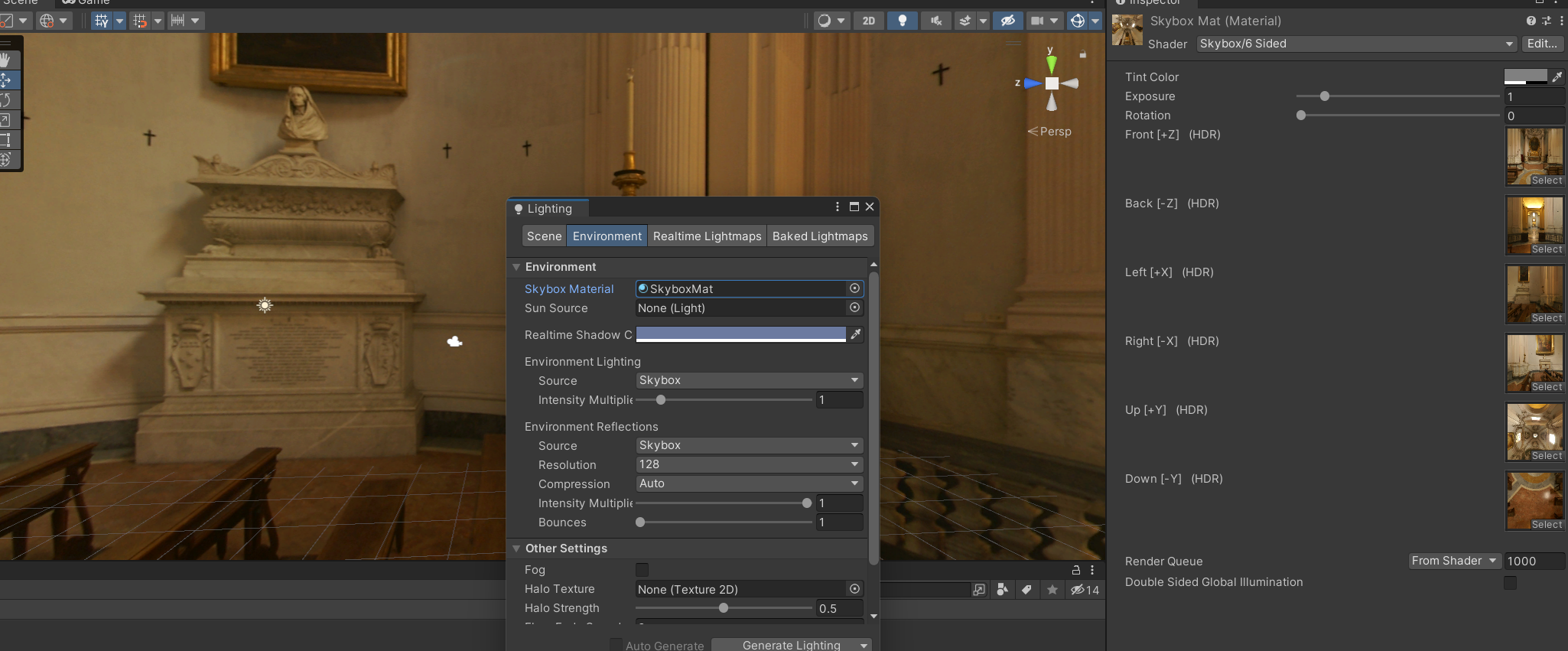
当场景中引入新的物体、光源，或者物体发生移动时就需要重新生成立方体纹理。并且立方体纹理只能反射环境，但不能反射物体本身（立方体纹理不能模拟多次反射效果），也就是说立方体纹理适用于凸面而不适用于凹面，因为凹面会反射自身。

1. 天空盒

天空盒时游戏中用于模拟背景的一种方法，当在场景中使用天空盒时，场景就被包围在一个立方体内，立方体每个面使用的技术就是立方体纹理映射技术。

如何使用天空盒：

1. 新建一个材质并把Shader设置为Skybox/6 Sided
2. 给材质的六面赋纹理，并把纹理的Warp Mode设置为Clamp
3. 给Window/Rendering/Lighting/Environment的Skybox选项赋这个立方体纹理



Ps：1.此处设置的Skybox是全局设置，将应用于该场景中的所有摄像机，若想某些摄像机使用不同的天空盒，可向Camera添加Skybox组件覆盖掉设置。2.Unity中天空盒在所有不透明物体被渲染之后渲染，其背后使用的网格是一个立方体或一个细分后的球体。

如何创建用于环境映射的立方体纹理

1. 直接由一些特殊布局的纹理创建。我们需要提供一张具有特殊布局的纹理，例如类似立方体展开图的交叉布局、全景布局等。然后，我们只需要把该纹理的Texture Shape 设置为 Cube 即可，然后在 Mapping 选项中设置纹理的布局属性，Unity会为我们做好剩下的事情。（上面的例子就是这个方法）

2. 手动创建一个Cubemap资源，再把6张图赋给它。通过Assets > Create > Legacy > Cubemap创建一个Cubemap，然后把6张纹理拖曳到它的面板中。（Unity推荐使用第一种方法创建立方体纹理，这是因为第一种方法可以对纹理数据进行压缩，而且可以支持边缘修正、光滑反射和 HDR 等功能）

3. 脚本生成。常用于希望根据物体所在的场景中的位置不同，从而生成不同的立方体纹理。利用Unity提供的Camera.RenderToCubemap 函数可以把从任意位置观察到的场景图像存储到6张图像中，从而创建出该位置上对应的立方体纹理。

代码：

文本

描述已自动生成

代码含义：

renderFromPosition：一个 Transform 类型的变量，用于指定渲染立方体贴图的位置。

cubemap：一个 Cubemap 类型的变量，用于存储渲染结果的立方体贴图。

在菜单项中，通过 [MenuItem("Cubemap/RenderCubeMap")] 将 RenderCubeMap 方法绑定到 "Cubemap" 菜单下的 "RenderCubeMap" 选项。

在 OnWizardCreate 方法中，首先创建一个名为 "CubemapCamera" 的游戏对象，并添加一个相机组件。然后将相机的位置设置为 renderFromPosition 的位置。接下来，使用相机的 RenderToCubemap 方法将渲染结果存储到 cubemap 变量中。最后，使用 DestroyImmediate 方法销毁创建的游戏对象。

将上述代码放入项目后，我们依据下面的步骤来创建 Cubemap:

1.创建一个空的GameObject对象。我们会使用该 GameObject 的位置信息来渲染立方体纹理

2.通过Assets > Create > Legacy > Cubemap创建一个 Cubemap， 并勾选其Readable选项

3.通过 Cubemap > RenderCubeMap 打开窗口

4.把第1步中创建的 GameObject 和第2步中创建的 Cubemap 分别拖曳到窗口中的 Render From Position 和 Cubemap 选项，单击 Render 按钮，即可生成 Cubemap。

Ps：需要为 Cubemap 设置 Face size，Face size值越大，渲染出来的立方体纹理分辨率越大，效果可能更好，但需要占用的内存也越大，这可以由面板最下方显示的内存大小得到。

反射

原理：根据反射到摄像机中的光线方向（即视角方向），由光路可逆的原则再根据法线，反向求得入射光线的方向，再根据入射光线的方向对立方体纹理采样即可。

文本

描述已自动生成

\_ReflectionColor控制反射颜色，\_ReflectionAmount控制材质的反射程度，\_Cubemap用于模拟反射的环境映射纹理。

文本

描述已自动生成

文本

描述已自动生成

因为光路可逆，所以可以通过反射光线求出入射光线方向，reflect 函数是一个用于计算反射向量的内置函数，把反射光线反过来求反射，就可以得到入射光线。它接受两个参数：入射向量和法线向量，并返回一个反射向量。

文本

描述已自动生成

对立方体纹理采样并不使用tex2D函数，使用Cg提供的texCUBE函数，texCUBE函数接收一个立方体纹理的变量，然后接收入射光方向。



1. 折射

原理：根据折射到摄像机中的光线方向（即视角方向），由光路可逆的原则和法线方向、折射率参数求得入射光线的方向，再根据入射光线的方向对立方体纹理采样即可。

文本

描述已自动生成

新添加的三个属性里前两个都可以类比反射代码里定义的属性，\_RefractRatio用于调整折射的比率，以此可以计算折射方向。

手机屏幕的截图

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

顶点着色器中使用了refract函数来计算折射方向，refract函数接收两个参数，第一个是归一化后的入射光线矢量，第二个参数是归一化后的表面法线矢量，第三个参数是折射率的比。返回值是折射方向，这个向量的模等于入射光线的模。

文本

描述已自动生成

片元着色器中对折射方向的立方体纹理进行采样。



3．菲涅尔反射

菲涅耳反射描述了一种光学现象，即当光线照射到物体表面上时，一部分发生反射，一部分进入物体内部，发生折射或散射。被反射的光和入射光之间存在一定的比率关系，这个比率关系可以通过菲涅耳等式进行计算。

在实时渲染中通常使用菲涅尔等式的近似公式进行计算，根据视角方向控制反射程度。

Schlick近似等式：



其中F0是一个反射系数，用于控制菲涅尔反射的强度，v是视角方向，n是表面法线

Empricial近似等式：

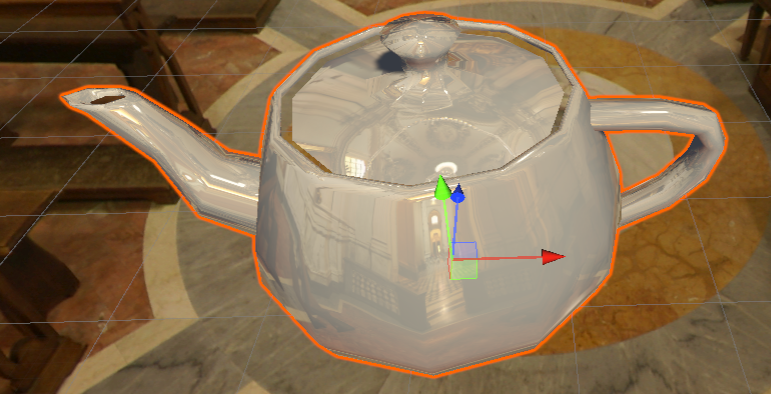


bias是一个基础的菲涅尔值，表示在较浅的入射角度下的反射强度；scale是一个缩放因子，用于调整菲涅尔效应的强度，可以控制反射强度随着入射角度的变化而增加或减少；power用于调整菲涅尔效应的强度曲线，它可以控制反射强度在不同入射角度下的变化速率。

文本

描述已自动生成

和之前的反射纹理映射Shader相比，只更改了片元着色器，在片元着色器中计算菲涅尔反射的变量，并把它混合漫反射和反射光照。着色器中，lerp 函数用于执行线性插值操作。它接受三个参数：a、b 和 t。a表示起始值或向量；b表示结束值或向量；t表示插值因子，它通常是一个介于 0 和 1 之间的值。



渲染纹理

GPU允许我们把整个三维场景渲染到一个中间缓冲中，即渲染目标纹理，而不是传统的帧缓冲或后备缓冲Unity为渲染目标纹理定义了一种专门的纹理类型——渲染纹理。

在Unity中使用渲染纹理的方式：

1.在 Project 目录下创建一个渲染纹理，然后把某个摄像机的渲染目标设置成该渲染纹理，这样一来该摄像机的渲染结果就会实时更新到渲染纹理中，而不会显示在屏幕上。

2.另一种方式是在屏幕后处理时使用 GrabPass 命令或 OnRenderImage 函数来获取当前屏幕图像，Unity 会把这个屏幕图像放到一张和屏幕分辨率等同的渲染纹理中，下面我们可以在自定义的 Pass 中把它们当成普通的纹理来处理，从而实现各种屏幕特效

1. 镜子效果

文本

描述已自动生成

文本

描述已自动生成

在顶点着色器中翻转了x分量的纹理坐标，因为镜子里的图像左右翻转/

2.玻璃效果

可以在 Unity Shader 中使用一种特殊的 Pass 来完成获取屏幕图像的目的，这就是GrabPass。当我们在Shader中定义了一个GrabPass后，Unity 会把当前屏幕的图像绘制在一张纹理中，以便我们在后续的 Pass 中访问它。可以用于实现玻璃的效果。

Ps：使用GrabPass时要注意设置渲染队列，代码中不包含混合指令也需要设置渲染队列为Transparent

文本

描述已自动生成

\_Distortion用于控制模拟折射时图像的扭曲程度，\_RefractionAmount用于控制折射程度，为0时只包含反射效果，为1时只包括折射效果。



在SubShader的Tag把渲染队列设置为透明，又把渲染类型设置为不透明，因为把Queue设置为Transparent可以保证该物体渲染时，其他所有不透明物体已经被渲染到屏幕上了，而设置RenderType是为了在使用着色器替换时该物体时被正确渲染，而这一过程发生在得到摄像机的深度和法线纹理时，所以这么设置Tag并不冲突。然后使用GrabPass定义了一个抓取屏幕图像的Pass，Pass内部定义了一个字符串，该字符串内部的名称决定抓取得到的屏幕图像将会被存入哪个纹理中（也可以省略声明该字符串，但直接声明会有更好的性能）。

图片包含 图形用户界面

描述已自动生成

定义了一个\_RefractionTex对应GrabPass时声明的纹理名称，\_RefractionTex\_TexlSize可以得到纹素的大小

文本

描述已自动生成

图片包含 文本

描述已自动生成

把顶点坐标转换到裁剪空间后，使用ComputeGrabScreenPos函数得到对应被抓取的屏幕图像的采样坐标

文本

描述已自动生成

求出顶点的世界坐标，然后求出顶点到摄像机的向量，对法线纹理进行采样，通过UnpackNormal函数得到切线空间下的法线方向，然后和折射时图像的扭曲程度与纹理大小相乘对屏幕图像采样坐标点进行偏移，以此模拟折射效果。对scrPos进行透视除法得到真实的屏幕坐标，再使用该坐标对抓取的屏幕图像\_RefractionTex进行采样，得到模拟的折射颜色。之后把法线方向从切线空间变换到世界空间下，得到视角方向相对于法线方向的反射方向，随后使用反射方向对Cubemap采样，把结果和主纹理颜色相乘得到反射颜色。最后使用\_RefractAmount属性对反射和折射颜色进行混合，作为最后的输出颜色。

GrabPass和渲染纹理的比较：

GrabPass的好处在于实现简单，我们只需要在Shader中写几行代码就可以实现抓取屏幕的目的。但从效率上来讲，使用渲染纹理的效率往往要好于GrabPass，尤其在移动设备上。使用渲染纹理我们可以自定义渲染纹理的大小，尽管这种方法需要把部分场景再次渲染一遍，但我们可以通过调整摄像机的渲染层来减少二次渲染时的场景大小，或使用其他方法来控制摄像机是否需要开启。而使用GrabPass获取到的图像分辨率和显示屏幕是一致的，这意味着在一些高分辨率的设备上可能会造成严重的带宽影响。而且在移动设备上，GrabPass虽然不会重新渲染场景，但它往往需要CPU直接读取后备缓冲（back buffer）中的数据，破坏了CPU和GPU之间的并行性，这是比较耗时的，甚至在一些移动设备上这是不支持的。

程序纹理

程序纹理指的是那些由计算机生成的图像，通常使用一些特定的算法来创建个性化图案或非常真实的自然元素，使用程序纹理的好处在于可以使用各种参数来控制纹理的外观，而这些属性不仅仅是那些颜色属性，甚至可以是完全不同类型的图案属性，这使得我们可以得到更加丰富的动画和视觉效果。

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

图片包含 形状

描述已自动生成