# 立方体纹理

立方体纹理 是 环境映射 的一种方法。

若希望 根据物体在场景中位置的不同，生成它们各自不同的立方体纹理。使用的是Unity的Camera.RenderToCubemap 函数，该函数可以把从任意位置观察到的场景图像存储到6张图中，从而创建出该微商对应的立方体纹理。

环境映射最常见的应用是反射和折射：

## 反射

天空盒中的物体，对天空盒进行反射。简单来说，就是通过反射方向，对立方体纹理进行采样。

1 反射方向是什么？

天空盒上的某一点，到物体表面，进入摄像机。我们知道 物体法线，知道摄像机位置，可以得到天空盒上的某一点。

v2f vert(a2v v)

{

v2f o;

o.pos = UnityObjectToClipPos(v.vertex);

o.worldPos = mul(unity\_ObjectToWorld, v.vertex).xyz;

o.worldNormal = UnityObjectToWorldNormal(v.normal);

o.worldViewDir = UnityWorldSpaceViewDir(o.worldPos);

o.worldRefl = reflect(-o.worldViewDir, o.worldNormal);//注意这里使用的是摄像机方向。

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

TRANSFER\_SHADOW(o);

return o;

}

2 采样。

fixed4 frag(v2f i) : SV\_Target

{

float3 worldPos = normalize(i.worldPos);

float3 worldViewDir = normalize(i.worldViewDir);

float3 worldLightDir = UnityWorldSpaceLightDir(i.worldPos);

fixed3 uvColor = tex2D(\_MainTex, i.uv).rgb;

fixed3 albedo = \_Color.rgb \* uvColor;

fixed3 ambient = UNITY\_LIGHTMODEL\_AMBIENT.xyz \* albedo;

// 对Cube纹理采样，并乘以反射主颜色

fixed3 cubeColor = texCUBE(\_ReflectCube, i.worldRefl).rgb \* \_ReflectColor.rgb;

fixed3 diffuse = \_LightColor0.rgb \* albedo \* max(0, dot(i.worldNormal, worldLightDir));

UNITY\_LIGHT\_ATTENUATION(atten, i, i.worldPos);

// 设置最后的颜色，使用Lerp控制diffuse和 反射颜色的过度

fixed3 color = ambient + lerp(diffuse, cubeColor, \_ReflectAmount) \* atten;

return fixed4(color, 1.0);

}

## 折射

v2f vert(a2v v)

{

v2f o;

o.pos = UnityObjectToClipPos(v.vertex);

o.worldPos = mul(unity\_ObjectToWorld, v.vertex).xyz;

o.worldNormal = UnityObjectToWorldNormal(v.normal);

o.worldViewDir = UnityWorldSpaceViewDir(o.worldPos);

// 获取折射方向 refract(摄像机反方向，法线，入射率/出射率)

o.worldRefl = refract(-normalize(o.worldViewDir), normalize(o.worldNormal) , \_RefractRatio);

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

TRANSFER\_SHADOW(o);

return o;

}

## 菲涅尔反射

反射的颜色 与 观察者看的角度有关.

例如 近处的水面可看见水底，远处的水面一片白茫茫

fixed3 cubeColor = texCUBE(\_RefractCube, i.worldRefl).rgb;

//菲涅尔的近似公式。

fixed fresnel = \_FresnelScale + (1 - \_FresnelScale)\*pow(1 - dot(worldViewDir, worldNormal), 5);

UNITY\_LIGHT\_ATTENUATION(atten, i, i.worldPos);

fixed3 color = ambient + lerp(diffuse, cubeColor, saturate(fresnel)) \* atten;

# 渲染纹理

原来渲染：摄像机的渲染结果 -> 颜色缓冲 -> 屏幕

现在: 摄像机的渲染结果 -> 中间缓冲，即渲染目标纹理

多重渲染目标：GPU允许我们把场景同事渲染到多个渲染目标纹理中，而不再需要为每一个渲染目标纹理单独渲染完整的场景。

用途：

1 镜子

2 在屏幕后处理时使用GrabPass命令或OnRenderImage函数获取与屏幕一样大小的渲染纹理，进行后处理。

镜子效果：

主要是利用多重渲染目标，将摄像机的渲染结果保存到一张渲染纹理中，在Shader中进行纹理采样，注意镜子中的x和真实世界的是对称的需要用1-uv.x。将材质赋给对象。

摄像机是新的摄像机，将摄像机的渲染目标设置为 我们创建的渲染纹理。

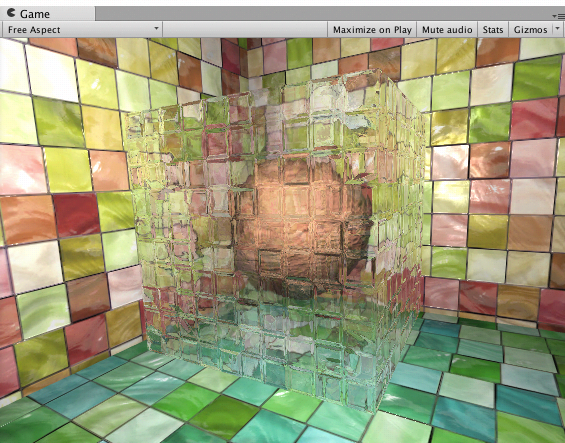
玻璃效果：

GrabPass 会把当前屏幕的图像绘制在一张纹理中，通常使用GrabPass来实现诸如玻璃等透明材质的模拟，使用GrabPass可以让我们对该物体后面的图像进行更复杂的处理。

需要将GrabPass设置为透明通道，才能保证不透明的物体都已经被绘制到屏幕上。

1 首先，使用一张法线纹理来修改模型的法线信息

2 模拟反射： 然后使用一个Cubemap来模拟玻璃的反射

3 模拟折射：使用GrabPass获取玻璃后面的屏幕图像，并在切线空间下的法线对屏幕纹理坐标偏移后，在对屏幕图像进行采样来模拟近似的折射效果

float4 scrPos = ComputeGrabScreenPos(o.pos); // 抓取屏幕图像的采样坐标



总结 ：1 算折射，通过 法线，扭曲度，原屏幕坐标 获得偏移量，折射的颜色就是偏移后的坐标对抓屏的采样。

2 算反射，通过对传入的立方体纹理进行采样

3 两个颜色的模拟混合时最终的颜色

# 程序纹理

用程序创建纹理，并赋值给材质

# 程序材质

通过SubstanceDesigner创造的材质。