学习笔记_立方体贴图

笔记本： DirectX 12
创建时间： 2022/9/14 11:16 **更新时间：** 2022/9/20 10:54
作者： handsome小赞

- **HLSL 实现**

```
TextureCube gCubeMap;
SamplerState gsamLinearWrap    : register(s2);

...

// 在像素着色器中
float3 v = float3(x,y,z); // 某查找向量
float4 = color= gCube Map.Sample(gsamLinear Wrap,v);
```

- 环境贴图

> - 使视场角为 90°的摄像机位于场景中某物体的中心点O处
> - 为立方体图纹理资源创建SRV时，应将其**维度**指定为
>   D3D12_SRV_DIMENSION_TEXTURECUBE
>   且使用 TextureCube属性

- 绘制天空纹理

> - 为实现天空球距离摄像机无限远，可以简单的把天空球的中心置于摄像机上，使它总是以摄像机为中心，随摄像机移动而移动。

- 将立方体图映射到球面

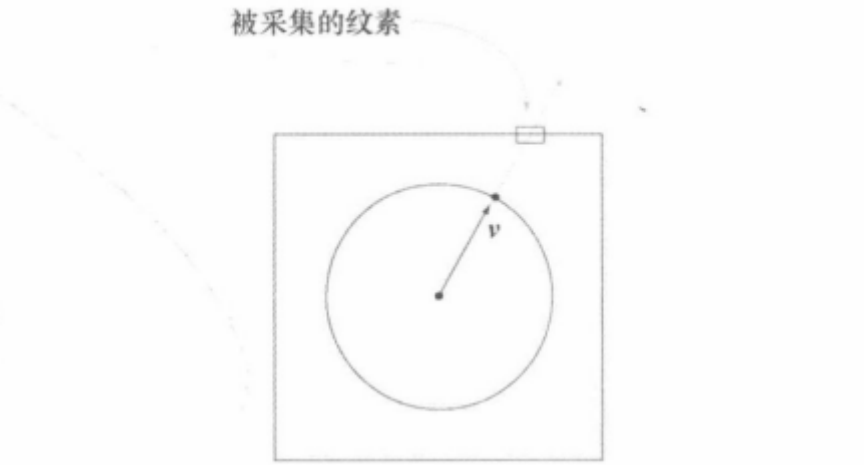

图 18.4　为了简单起见，这里展示的是 2D 示意图；因此，图中的正方形即为 3D 空间中的立方体（图），而圆形即为 3D 空间中的（天空）球体。假设天空球与环境图都以同一个空间中的原点为中心。接下来，为了向球面上的点投影纹理，我们把端点为原点的向量作为查找向量射向球面。以此将立方体图投影到球面上

- **注意** 应对 渲染模板以及深度/模板缓冲区进行手动清理，并把绘制天空的工作放在最后

- 模拟反射

  - 运用环境图去模拟来自周围环境的镜面反射。

    计算每个像素的反射向量并用它来对环境图进行采样：

    ```
    const float shininess = 1.0f - roughness;

    // 加入镜面反射数据
    float3 r = reflect(-toEyeW, pin.NormalW);
    float4 reflectionColor = gCubeMap.Sample(gsamLinearWrap, r);
    float3 fresnelFactor = SchlickFresnel(fresnelR0, pin.NormalW, r);
    litColor.rgb += shininess * fresnelFactor * reflectionColor.rgb;
    ```

  - 方案一：环境图的映射方式是以其立方体中心的端点，向立方体发出的射线采集纹素。

    > 缺点：同一入射角观察不同点本应看到不同的图像，但是由于同一入射角所用的查找向量是相同的。

  - 方案二：给环境图关联一个代理几何体，用射线与包围盒的交点 v = p + tr 来代替反射向量的查找向量 r

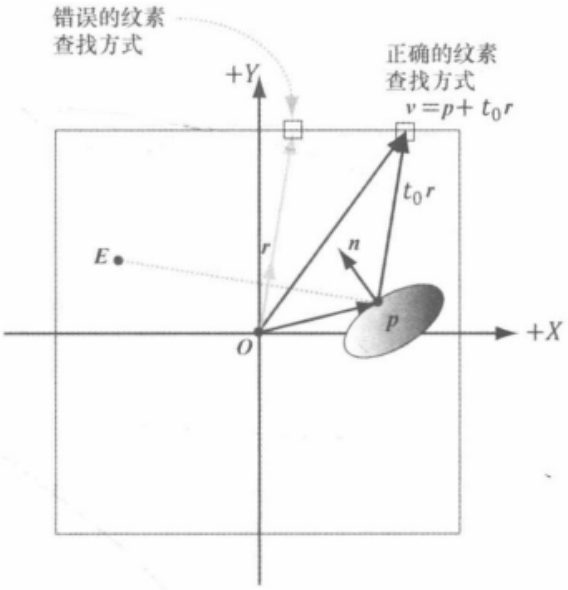

图 18.7　在这种情况下，我们不再用反射向量 $\boldsymbol{r}$ 来作为立方体图的查找向量，而是用射线与包围盒的交点 $\boldsymbol{v} = \boldsymbol{p} + t_0\boldsymbol{r}$ 来代替。注意，由于点 $\boldsymbol{p}$ 与包围盒代理几何体的中心位置有关，所以上述交点可作为此立方体图的查找向量

p = RayOrign - BoxCenter

- 动态立方体图

  - 每一帧都要将摄像机置于场景内，以它作为立方体图的原点，沿坐标轴共6个方向将场景分六次逐个渲染到立方体图的对应面上。
  - **注意** 因为动态地渲染立方体图开销会比较大，因为每一帧都需要将场景绘制到6个渲染目标中。因此，我们要试着将场景中需要用到的动态立方体图的地方降到最少。比如我们可以只为突出场景中关键物品时才用动态反射。而其他次要物品采用静态反射。一般来讲，动态立方体图常用的是256x256像素的低分辨率立方体图。
  - **动态立方体图辅助类**

    > CubeRenderTarget 类

- **构建立方体图资源**

```
void CubeRenderTarget::BuildResource()
{
  D3D12_RESOURCE_DESC texDesc;
  ZeroMemory(&texDesc, sizeof(D3D12_RESOURCE_DESC));
  texDesc.Dimension = D3D12_RESOURCE_DIMENSION_TEXTURE2D;
  texDesc.Alignment = 0;
  texDesc.Width = mWidth;
  texDesc.Height = mHeight;
  texDesc.DepthOrArraySize = 6;
  texDesc.MipLevels = 1;
  texDesc.Format = mFormat;
  texDesc.SampleDesc.Count = 1;
  texDesc.SampleDesc.Quality = 0;
  texDesc.Layout = D3D12_TEXTURE_LAYOUT_UNKNOWN;
  texDesc.Flags = D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET;

  ThrowIfFailed(md3dDevice->CreateCommittedResource(
    &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_DEFAULT),
    D3D12_HEAP_FLAG_NONE,
    &texDesc,
    D3D12_RESOURCE_STATE_GENERIC_READ,
    nullptr,
    IID_PPV_ARGS(&mCubeMap)));
}
```

- **分配额外的描述符空间**
  - 为渲染立方体图，还需要添加6个渲染目标视图，另外，还要附加一个深度/模板缓冲区，因此我们必须重写 D3DApp::CreateRTVAndDsvDescriptorHeaps 方法来为这些额外的描述符分配描述符堆
  - 此外，还需要新增一个 SRV，以便在生产立方体图之后将它绑定为着色器的输入数据
- **构建描述符**
  - 为整个立方体图资源创建 SRV，
    `D3D12_SHADER_RESOURCE_VIEW_DESC::ViewDimension = D3D12_SRV_DIMENSION_TEXTURECUBE`
  - 为每个立方体面创建 RTV，
    `D3D12_RENDER_TARGET_VIEW_DESC::ViewDimension = D3D12_RTV_DIMENSION_TEXTURE2DARRAY`
- **构建深度缓冲区**
  - 立方体图的各面与主后台缓冲区的分辨率是不同的。因此，要渲染立方体图的诸面，我们就需要采用与立方体图面分辨率大小相匹配的深度缓冲区。
    考虑到每次只渲染一个面，所以立方体图的渲染工作仅需 1 个深度缓冲区即可。
  - D3D12_RESOURCE_DESC::Width/Height = 立方体图尺寸
- **立方体图的视口与裁剪矩形**
  - 在CubeRenderTarget 类的构造函数内就指定 视口和剪裁矩形的属性。
    mViewport、mScissorRect
- **设置立方体图摄像机**
  - 视场角90°
    6台摄像机
- **用几何着色器绘制动态立方体图**

> 仅需一次绘制调用，即可渲染好整个立方体图
>
> - 参考 Direct3D 10的例程“CubeMapGS”
> - **注意** NVIDIA公司堆Maxwell 够ji进行了最新的优化，使几何着色器向多个渲染目标复制几何体的操作不会造成过大的性能损失。