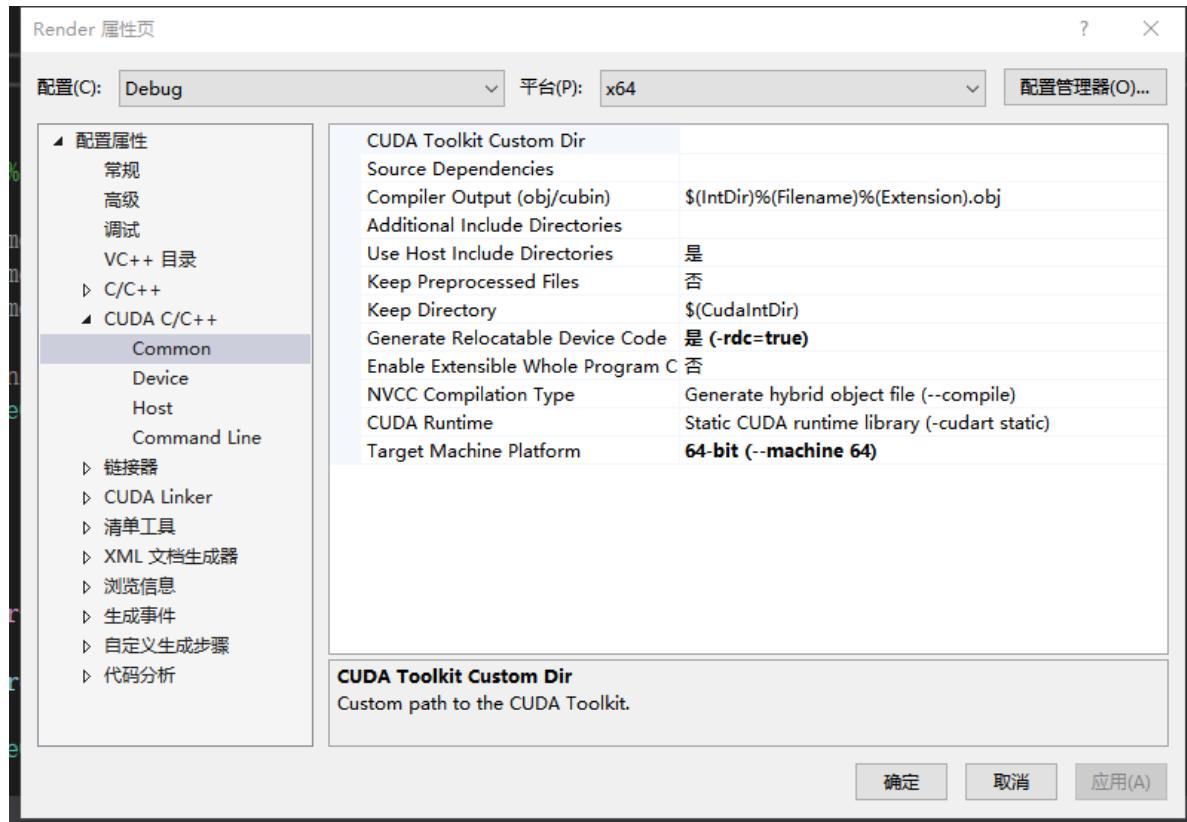


# 运行环境

操作系统：windows10 x86 CUDA 11.0

# 程序开发环境

IDE：Visual Studio 2019，编程语言：C++，CUDA，图形库：SDL2



CUDA这边的设置要可重定位

需要用CUDA C/C++ 来编译的文件：box.h bvh.h camera.h common.h material.h onb.h pdf.h kernal.cu

# 用户使用说明

1.common.h下有两个选项

```
#define CAMERA 2
#define OBJECT 1
```

Camera是相机角度

Object是选择的物体

2. 编译后（一般用release）会产生一个窗口里面是渲染结果

3. 渲染结果也会生成png文件放在Project/result目录下

# 数据结构与算法描述

## 1. 数据结构

所有的obj文件都是用tinyobjloader加载的，加载后有有关material的数组，三角形顶点index的数组，所有顶点及其法向、纹理坐标的数组，这些数据没做处理，直接通过位置推算来使用，纹理是用stb\_image来加载的，通过CUDA写入texture memory。

## 2. 蒙特卡洛路径追踪

采样器和pdf函数是这一块的核心，渲染方程的解是真实解，但是没有强大的算例可以解觉他，所以要用蒙特卡洛的模拟积分法，对范围内采样 $f(x)$ ，然后除以采样的几率，采样越多越接近真实解。对渲染方程一致，采样器是对入射光线的选择，pdf是几率，用来估算积分结果。以下是渲染方程

$$L_r(\vec{x}, \vec{\omega}_r) = \int_{\Omega_i} f_r(\vec{x}, \vec{\omega}_i \rightarrow \vec{\omega}_r) L_i(\vec{x}, \vec{\omega}_i) \cos \theta_i d\omega_i.$$

采样器我打算用的是cosine分布来进行采样，结果取对主轴于射线的cosine值的Ns次方，通过随机两个参数uv来得到椭球面某一个点的位置。对于glossy，我用反射光线作轴，对于漫反射，我用击打点的法向量作主轴。

对于一个材质同时具有Ks和Kd值，则用亮度公式算出它们各自的贡献，除此之外随机一个数0到1的，然后看是否大于 $K_s / (K_s + K_d)$ ，大于就做漫反射操作，小于就做glossy的采样。

pdf的计算，是基于采样器的，假如作漫反射，则pdf为 $\cosine/Pi$ ，对于glossy材质，则pdf为 $(N \cdot R)Ns$ 次方。

我最终把直接光照取消了，因为当ssp次数越大，噪点越少，越接近真实结果。

## 3. GPU版BVH加速结构

利用Morton码来建立层次BVH，首先给每一个三角形标注一个莫顿码，由于有些场景大，三角形密集，所以需要用64位的Morton码来标记，需要把三角形的中心计算出来，然后通过中心求出最大包围盒的相对位置，然后可以从而得到此位置的三维莫顿码，然后对这些三角形排序。

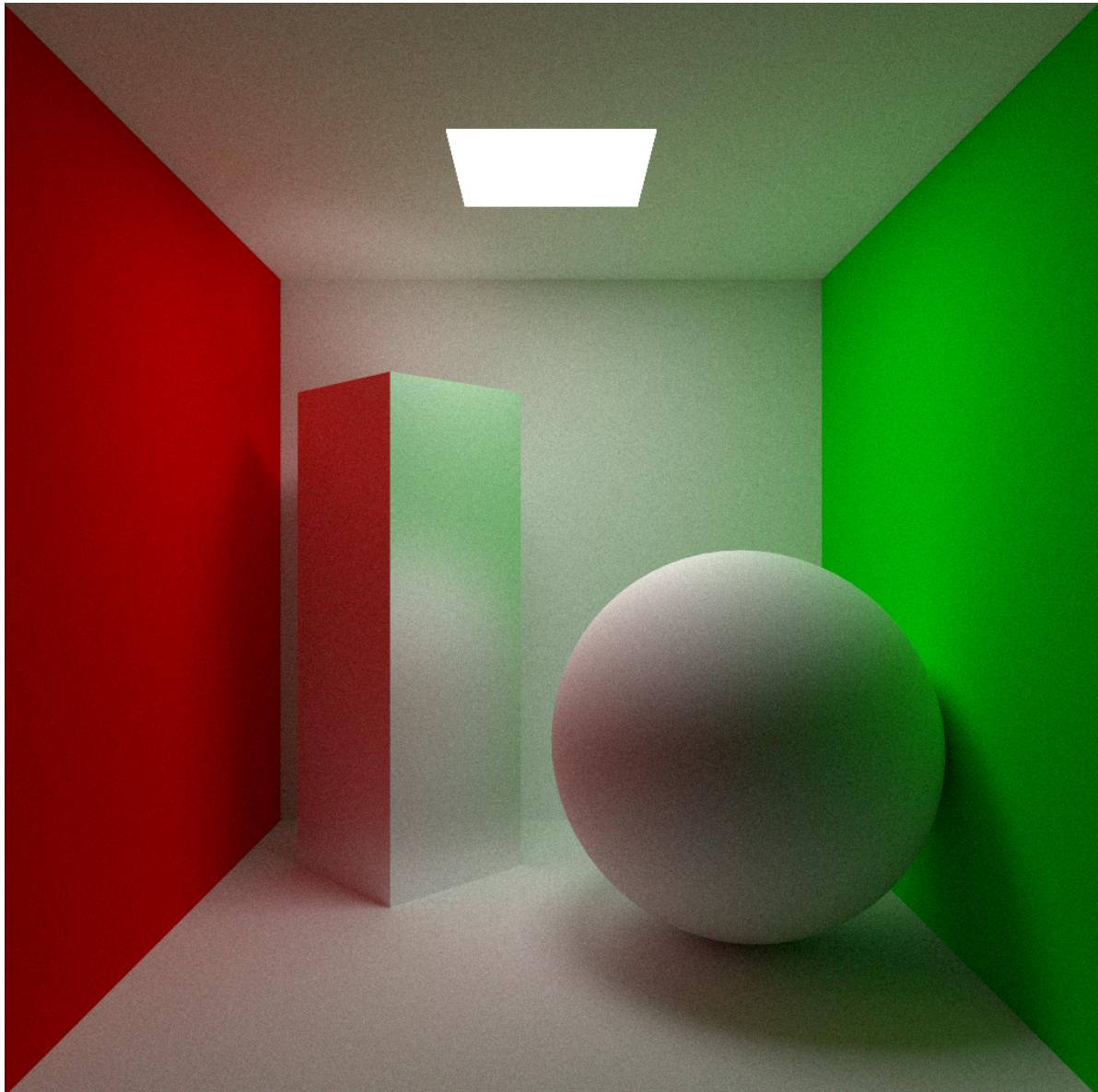
x:	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111	
y:	0 000	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111
0 000	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
1 001	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
2 010	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
3 011	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
4 100	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
5 101	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
6 110	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	
7 111	000000 000001 000100 000101 010000 010001 010100 010101	000110 000111 010010 010011 010110 010111	001100 001101 001110 001111 011000 011001 011100 011101	001110 001111 011010 011011 011110 011111	100000 100001 100100 100101 110000 110001 110100 110101	100010 100011 100110 100111 110010 110011 110110 110111	101000 101001 101100 101101 111000 111001 111100 111101	101010 101011 101110 101111 111010 111011 111110 111111	

二维莫顿码示例

排序后可以把BVH的建立线性化，这样就可以在GPU上建立BVH了，因为每个非叶子节点都可以通过自己的线程id来找到相应的叶子节点范围，从而算出本节点的包围盒，最终得到整个BVH。

# 实验结果

## 1. 原始材质结果



康奈尔盒跑的1024ssp



汽车场景1跑的100ssp



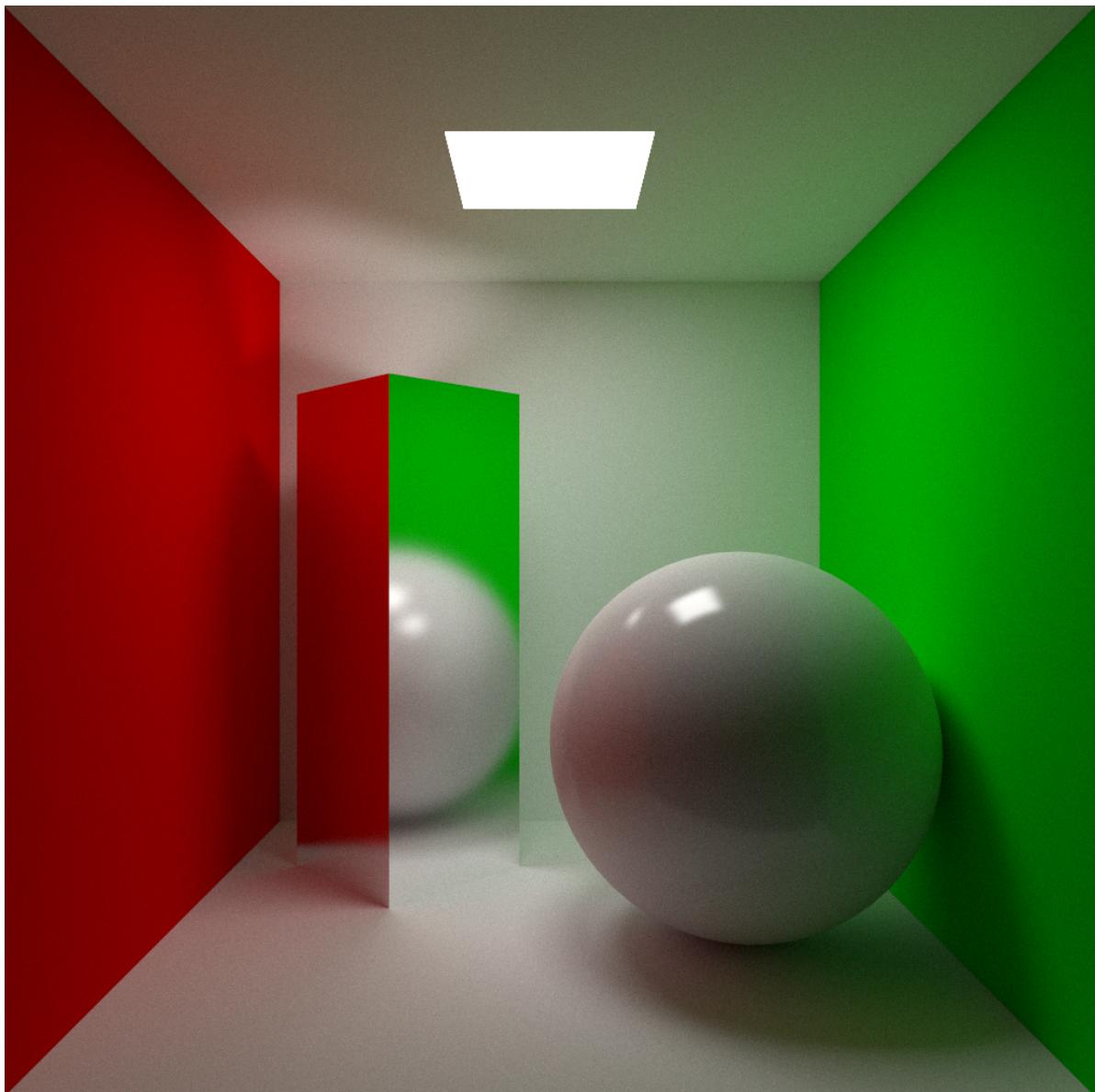
汽车场景2跑的100ssp



dining room 1024ssp结果

显而易见，这些渲染效果都没法达到目标要求，问题也很清楚，有些地方没有反射，所以要微调一下

## 2.微调材质结果



这是微调后cornellbox的结果，10000ssp



这是汽车场景1，1024ssp



汽车场景2, 1024ssp



dining room, 2048ssp



Utah teapot, 2048ssp

另外，CUDA的加速也是有成效，上面的图片用的RTX2060跑出来只需10分钟到60分钟之间