VCL Final Project: Path Tracing with BVH

2400013085 李瑞祥

January 2026

1 Introduction

本项目基于示例代码[1]，理解原理并成功复现了 Path Tracing 的实验结果，并且引入了 Lab3 中的 BVH 加速结构来加速渲染。整体项目框架基于 Lab3 开发，但是进行了大幅度修改，包括对 GUI 的编辑、原代码的模块化处理等。在终端运行如下指令即可编译运行：

```
xmake run Final
```

编译成功并运行后，项目结果位于 Case 2: Path Tracing 下，可以通过滑块调整 SPP(Samples per Pixel)。

2 Details

2.1 Path Tracing

Path Tracing 部分的代码可以直接参考示例代码，其原理是通过蒙特卡洛积分方法随机采样光线路径（包括在半球或球面上的方向采样），模拟光线在场景中的多次反弹和材质交互，以计算全局光照效果，从而生成更加真实的渲染结果。[2]

对于 3D 渲染，最重要的就是如何求解渲染方程（辐射度量学太难了）：

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega f_r(p,\omega_i,\omega_o) L_i(p,\omega_i) n \cdot \omega_i d\omega_i$$

P 点最终渲染得到的颜色就是自发光项加上其反射到相机的光。这些光的来源可能是其他光源的直接光照，也可能是其他物体结果反射/折射后再次碰撞的间接光照。对于方程的积分项，根据大数定理，我们可以采用蒙特卡洛方法来计算积分值：

$$\int_a^b f(x)dx = \frac{1}{n}\sum_{i=1}^{n} f(x_i)(b-a) = \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{p(x_i)}$$

其中，$p(x)$ 是连续随机变量 X 的概率密度函数（Probability Density Function，PDF）。由此可知，前文渲染方程中的一大堆东西就是 $f(x)$。

回归到 Path Tracing 中，我们使用半球余弦加权采样（间接）、光源直接采样等方法来模拟直接光源、镜面反射、漫反射和折射。

对于漫反射，采样余弦加权半球采样，使密度函数与入射角的余弦成正比，即：$PDF(\omega_i) = \frac{\cos\theta}{\pi}$。构建局部坐标系 wuv，根据 BRDF 模型，进行随机采样：

$$\omega = \vec{u}\cos\phi\sin\theta + \vec{v}\sin\phi\sin\theta + \vec{n}\cos\theta$$

其中，$\phi = 2\pi \cdot r_1$，$\sin\theta = \sqrt{r_2}$，$r_1, r_2 \sim U[0,1]$ 为均匀分布，实现在球面上的均匀随机采样[3]。

[1] http://www.kevinbeason.com/smallpt/

[2] https://zhuanlan.zhihu.com/p/370162390

[3] https://zhuanlan.zhihu.com/p/503163354

相比之下，（理想）镜面反射简单得多，只需要自发光加上交点颜色与反射光线返回的颜色的叠加即可：$L = E + C \cdot F$

对于光源直接采样，计算其立体角来显式计算光源贡献（重要性采样）。光源方向通过球坐标采样生成，目标是分布在光源的立体角范围内。我们对光源方向角进行限制：$\cos\theta_{max} = \sqrt{1 - \frac{r^2}{d^2}}$，此时得到密度概率函数为：

$$p(\omega_i) = \frac{1}{\Omega_s} = \frac{1}{2\pi(1 - \cos\theta_{max})}$$

在透明材质上的反射/折射，首先定义反射光线，判断是离开物体还是进入物体。同时定义两侧介质的折射率之比 nnt，计算入射光线与表面法线的夹角余弦 ddn。如果发生了全反射（根据 Snell's law 折射定律）：

$$n_c \sin\theta_i = n_t \sin\theta_t \Rightarrow \cos^2\theta_t = 1 - nnt^2(1 - ddn^2)$$

若 $\cos^2 t < 0$，则发生全反射。反之，则计算折射方向。菲涅尔反射系数用于描述光线在两种不同介质交界面处反射和透射的比例，通过 Schlick 近似公式求解：

$$F_{Schlick}(\mathbf{v}, \mathbf{n}) = F_0 + (1 - F_0)(1 - \cos\theta)^5$$

最后通过俄罗斯轮盘赌来决定反射或者折射。俄罗斯轮盘赌不仅用于判断反射/折射，还用于判断什么时候递归终止，利用概率控制递归，同时保持无偏估计。而结束的概率 p 也是动态调整的：

$$p = \max(color_r, color_b, color_g)$$

总体 Path Tracing 算法的伪代码如下：具体实现在 `task1.cpp` 和 `task1.h` 中。模型需要的 Vec、Sphere 类都集成在 `Sphere.h` 中，实现模块化管理。

Algorithm 3 Path Tracing Main Loop

```
for each pixel (i,j) do
    Vec3 C = 0
    for (k=0; k < samplesPerPixel; k++) do
        Create random ray in pixel:
            Choose random point on lens P_lens
            Choose random point on image plane P_image
            D = normalize(P_image − P_lens)
            Ray ray = Ray(P_lens, D)
        castRay(ray, isect)
        if the ray hits something then
            C += radiance(ray, isect, 0)
        else
            C += backgroundColor(D)
        end if
    end for
    image(i,j) = C / samplesPerPixel
end for
```

2.2 BVH

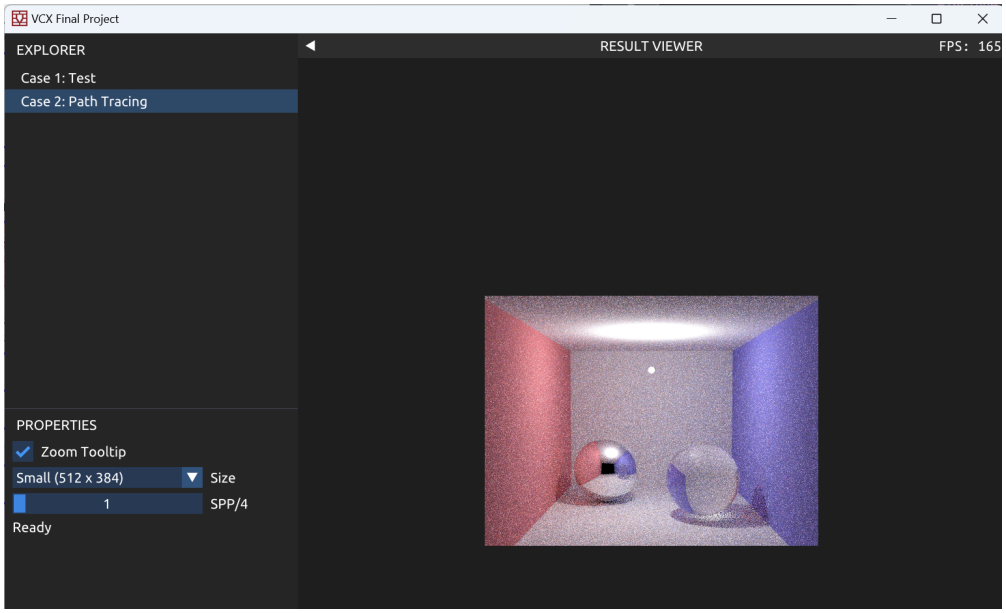
Bounding Volume Hierarchy（BVH，包围体层次结构）[4]，是一种加速渲染的空间分割结构。其基本元素为轴对齐包围盒（Axis-Aligned Bounding Box，AABB），是一个可以包围特定几何形状的矩形/立体框，且

[4] https://blog.csdn.net/qq_35312463/article/details/108419276

该框的边与坐标轴平行。BVH 树是一个二叉树，可以将求交的时间复杂度从 $O(N)$ 降低到 $O(\log N)$ 。

这个过程通过递归分割，将物体分组，然后为每一组创建包围盒，最终形成一棵层次结构的树。最终只对可能相交的少量对象进行逐个检查，提升了光线追踪的效率。具体实现位于 `BVH.cpp` 和 `BVH.H` 中。

3 Result

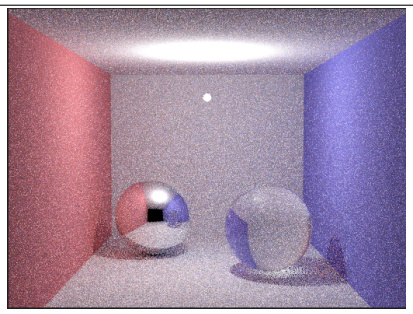
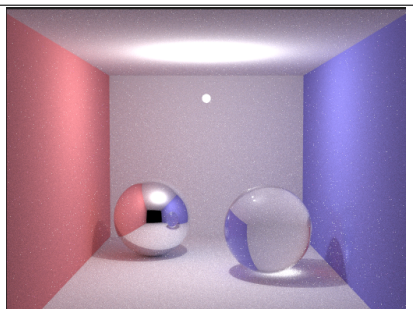
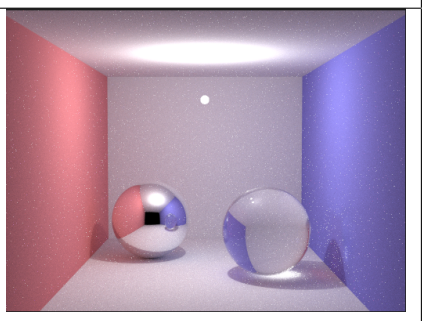


3.1 UI

可以选择图像大小和 SPP（渲染过程中不可调节）。zoom 的功能可以查看细节

3.2 Rendered

在不同条件下渲染结果如下：

SPP	1x4	50x4	100x4
结果			

可见提高采样率可以显著减少蒙特卡洛噪声（或称渲染噪声）。参考代码的结果是在 1250x4 的采样下得到的结果，几乎没有噪声。