

可视计算大作业实验报告

信息科学技术学院 夏廷轩 2200012930

2024 年 1 月 21 日

1 大作业选题

我的选题是C.4 Ray Tracing Acceleration与C.5 Path Tracing相结合。

Case 1中我实现了基于多线程CPU并行加速的Whitted-Style光线追踪¹，使用了BVH和SAH加速结构（并且添加了很多算法层面剪枝）。

Case 2中我在Case 1的加速优化基础上，对所给模型进行调整，实现了Path Tracing²（同样支持多线程BVH、SAH加速）。

2 Ray Tracing Acceleration

2.1 BVH加速结构

根据讲义和GAME S101提供的优化思路实现了基于对齐包围盒的BVH数据结构。

建树部分没直接取重心的跨度最大维度中位值对应三角形来分割，保证了每层size减半，使层高保持 $O(\log n)$ 。

支持了两种query：普通射线与场景三角面片求交IntersectRay，以及用于shadow ray的IntersectShadowRay。

应用到了如下剪枝：

- 1.若进入当前节点包围盒时间已经比当前最近交点时间晚，则不再进入子节点；
- 2.若当前节点包围盒与射线不相交，则不再进入子节点。
- 3.shadow ray中若已经找到了一个光源前的非透明交点，则直接返回。

具体代码实现见tasks.h

¹这是基于lab 3代码的

²是GAMES 101中所教的版本，也是基于lab 3代码的

2.2 SAH加速结构

改写BVH的build函数，根据表面积比和数量来计算期望代价，找到最优划分点³。

2.3 多线程

我所实现的BVH与SAH都是线程安全数据结构，本身支持多线程访问。

修改CaseRayTracing.cpp中的OnRender函数，使用刚在ICS学的生产者消费者模型，为每个像素新建一个线程。

用逻辑核数concurrent count和正在工作线程数cnt1来实现隐式线程池。

配合互斥锁mtx（保护cnt1）和consume（用于唤醒主线程）实现多线程渲染。

2.4 效果

在我本地⁴开启SAH,max depth=7下的渲染时间如下，实际测试可能需要关闭小核来获取更佳性能

渲染效果图为0.3倍缩放，原图见文末链接

White Oak, max depth=7, 渲染时间 2min41s(参考渲染时间 10min)



图 1: White Oak

Breakfast Room, max depth=7, 渲染时间2min52s(参考渲染时间 20min)



图 2: Breakfast Room

³这是基于讲义公式的

⁴12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz, 16GB内存, 20逻辑核数, 大小核均开启

Sponza, max depth=7, 渲染时间2min39s(参考渲染时间 20min)



图 3: Sponza

可以看到速度优化十分明显，几乎快了10倍。

3 Path Tracing

参考GAMES101以及GAMES101 Lab7中提供的部分代码⁵实现了路径追踪

3.1 关键算法

CasePathTracing.cpp中的OnRender函数为每个像素生成 $SampleRate^2$ 个初始视线，并求得与场景的交点传给PathTrace函数。

核心算法参见tasks.cpp中的PathTrace函数。

1.如果光线与物体的交点 (hit_p) 是自发光的 ($IntersectEmission > 0$)，则直接返回这个Emission。

2.计算直接光照 ($l_{directly}$)。这部分的计算使用了光源采样思想，先随机找当场景中光源lights⁶中的任意一个，然后调用SampleLight函数根据面积随机采样光源上的一个点 (q)，计算从交点 (p) 到这个点的光线 (w_i)，并检查这个光线是否被其他物体阻挡。如果没有被阻挡，就计算这个光线的贡献，并加到直接照明上。

3.进行俄罗斯轮盘赌，如果随机数大于*Russian Roulette*，则直接返回直接光照，否则进行间接光照的计算。

4.计算间接光照，调用SampleRay函数在交点 (p) 处采样一个反射视线(w_i)，然后计算wi与场景碰撞(hit)，如果交点存在且不自发光，则递归调用PathTrace计算这部分的贡献，加到间接光照上。

3.2 直接光照贡献

根据反射方程实现

$$L(p, w_o) = f_r(p, w_i, w_o)L(p, w_i) \cos \theta_i$$

⁵主要参考了采样方法

⁶在Cornell Box中，我设置了两个三角面片面光源

见如下代码，第一行是计算Intensity衰弱后的强度，第二行是除以概率pdf，乘上由 f_r 函数BRDF采样得到的反射到视线方向的辐射通量。第三行在计算各种夹角投影贡献的cos值。

```

1     l_directly += light . Intensity / glm :: dot ( q - p , q - p )
2     / pdf * f_r ( wi , -ray . Direction , hit_p )
3     * std :: max ( 0.0f , glm :: dot ( n , wi ) ) * std :: max ( 0.0f ,
        glm :: dot ( -wi , light . Direction ) );

```

3.3 间接光照贡献

见如下代码，第一行是计算由 f_r 函数BRDF采样得到的反射到视线方向的辐射通量，第二行是PathTrace追踪反射光线wi的值，第三行计算夹角贡献的cos，第四行除以俄罗斯轮盘赌阈值来使其满足期望，除以采样反射方向对应的概率 get_{pdf} 。

```

1     l_indirectly = f_r ( wi , -ray . Direction , hit_p )
2     * PathTrace ( intersector , hit , r )
3     * std :: max ( 0.0f , glm :: dot ( n , wi ) )
4     / RussianRoulette / get_pdf ( wi , ray . Direction , hit_p );

```

3.4 各种采样方法

3.4.1 SampleLight

实现了对两种光源的采样，一种是三角形面光源的采样，另一种是球体近似的点光源采样。提供的pdf均为面积倒数 $\frac{1}{S}$ ，采样结果为在光源表面上随机取一点。

3.4.2 SampleRay

实现了漫反射的反射方向采样，这里由于是漫反射所以随机一个反射方向即可

3.4.3 get_pdf

返回SampleRay对应的概率，为 $\frac{1}{2\pi}$

3.5 其他工作

3.5.1 修改场景

在scenes中加入了Box_ver_path场景，在Cornell Box场景基础上修改了材质，使其更加适合Path Tracing的漫反射。

修正了原场景中light.obj的拓扑结构错误，正确添加了光源的矩形面片⁷。

⁷两个三角面片构成

3.5.2 修改数据结构

在Materials中加入了自发光Emission信息来保证正确渲染光源。

在Lights中加入了面光源Area和球光源Spot的支持，并且相应配置了loader.cpp、Material结构体和yaml文件。

3.6 效果

SampleRate=10下的效果为

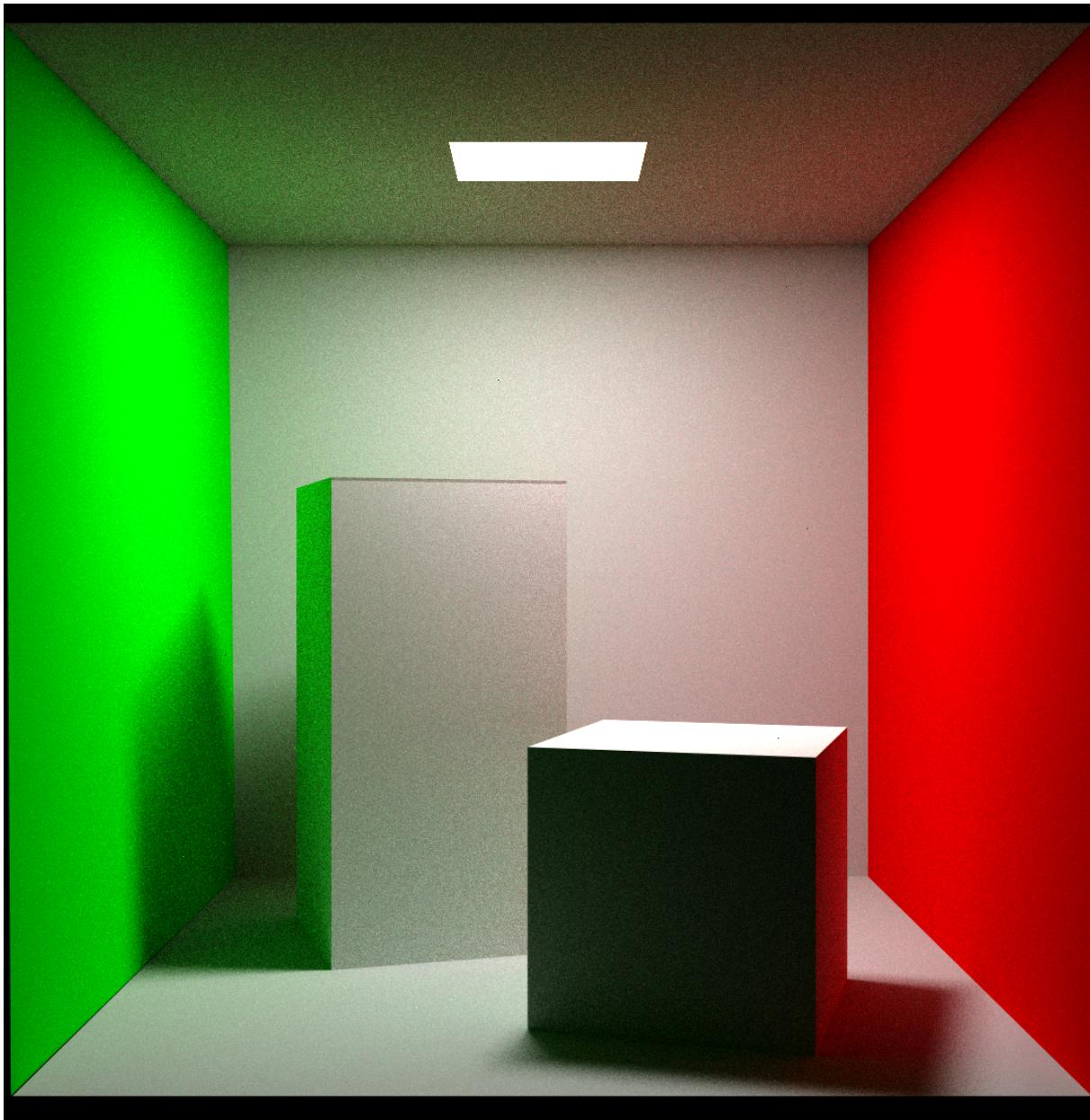


图 4: SampleRate=10

SampleRate=20下的效果为(这个是之前版本渲染的错误光源效果，想看最终效果
请您手动渲染)

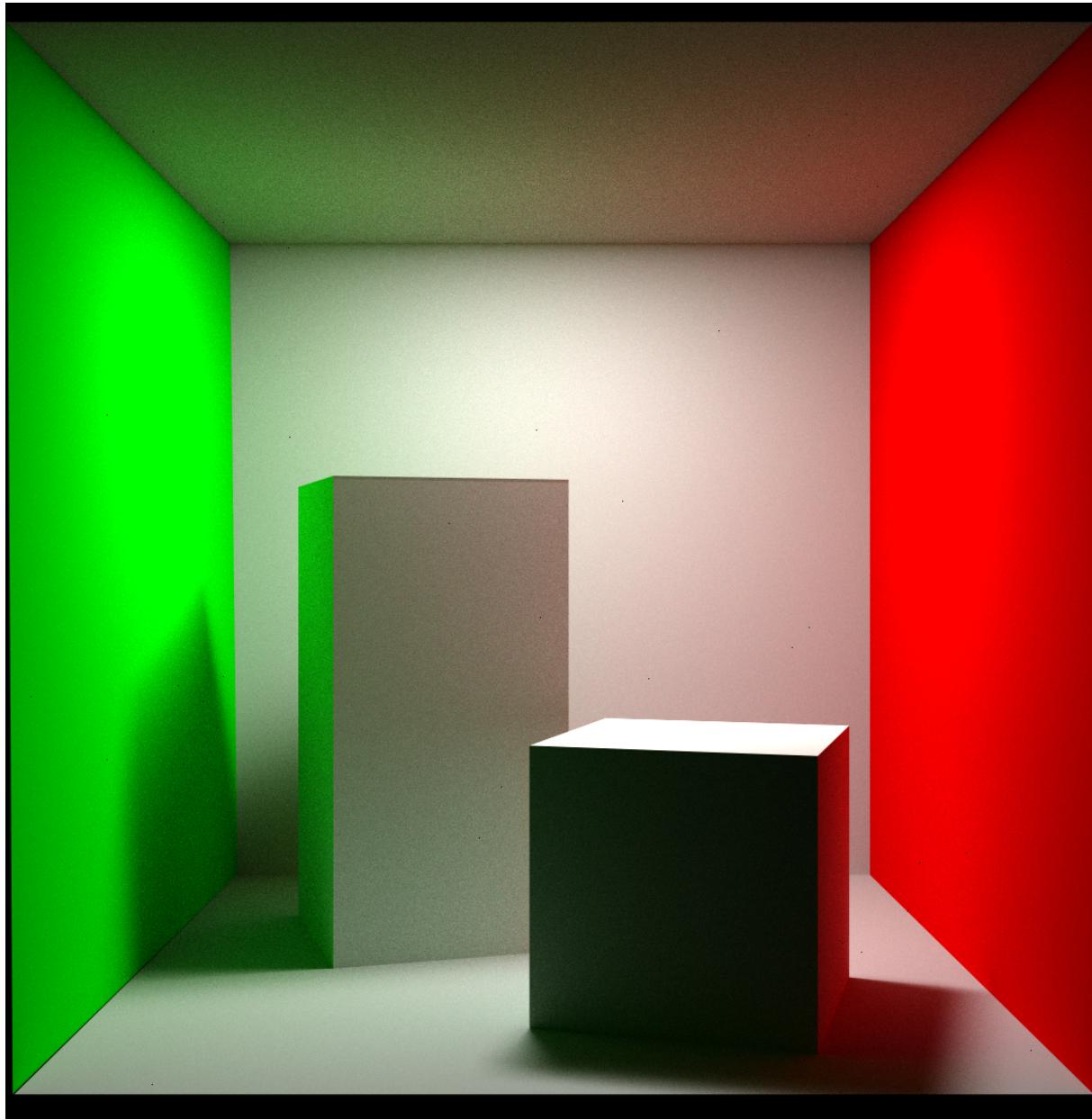


图 5: SampleRate=20

4 提示

完整效果图见压缩文件解压后根目录下。
编译命令为xmake build lab3