作业 3：路径追踪

1 总览

在这次的练习中， 我们将于使用光线追踪来渲染图像，实现完整的 **Path Tracing 算法**，**光线与三角形求交**并且用 **BVH 加速结构**对于求交过程进行加速。

在光线追踪中最重要的操作之一就是找到光线与物体的交点。一旦找到光线与物体的交点，就可以执行着色并返回像素颜色。

请认真阅读本文档，按照本文档指示的流程完成本次实验。

2 框架说明与实现

2.1 编译运行

2.1.1 虚拟机

Ubuntu 虚拟机链接：
https://pan.baidu.com/s/1lZNLiJbuu60OhOPOCl6U_A?pwd=6666
提取码：6666

基础代码只依赖于 CMake，下载基础代码后，执行下列命令，就可以编译这个项目：

```
$ mkdir build
$ cd ./build
$ cmake ..
$ make
```

在此之后，你就可以通过 ./Raytracing 来执行程序。请务必确保程序可以正常编译之后，再进入下一节的内容。

2.1.2 Windows

可参照相关指引配置环境。

https://blog.csdn.net/qq_43419761/article/details/127740207

https://github.com/star-hengxing/GAMES101-xmake

2.2 框架分析

2.2.1 调通框架

1. 从 **main** 函数开始。我们定义场景的参数，添加物体（球体或三角形）到场景中，并设置其材质，然后将光源添加到场景中。

2. 调用 **Render(scene)** 函数。在遍历所有像素的循环里，生成对应的光线并将返回的颜色保存在帧缓冲区（framebuffer）中。在渲染过程结束后，帧缓冲区中的信息将被保存为图像。

3. 在生成像素对应的光线后，我们调用 **CastRay** 函数，在该函数中实现Path Tracing算法 。

现在我们对代码框架中的一些类做概括性的介绍：

- **global.hpp**：包含了整个框架中会使用的基本函数和变量。
- **Vector.hpp**: 由于我们不再使用 Eigen 库，因此我们在此处提供了常见的向量操作，例如：dotProduct，crossProduct，normalize。
- **Object.hpp**: 渲染物体的父类。**Triangle 和 Sphere** 类都是从该类继承的。
- **Scene.hpp**: 定义要渲染的场景。包括设置参数，物体以及灯光。
- **Renderer.hpp**: 渲染器类，它实现了所有光线追踪的操作。

• **Material.hpp**: 我们从将材质参数拆分到了一个单独的类中，现在每个物体实例都可以拥有自己的材质。
• **Intersection.hpp**: 这个数据结构包含了相交相关的信息。
• **Ray.hpp**: 光线类，包含一条光的源头、方向、传递时间 **t** 和范围 **range**.
• **Bounds3.hpp**: 包围盒类，每个包围盒可由 **pMin** 和 **pMax** 两点描述（请思考为什么）。**Bounds3::Union** 函数的作用是将两个包围盒并成更大的包围盒。与材质一样，场景中的每个物体实例都有自己的包围盒。
• **BVH.hpp**: **BVH** 加速类。场景 **scene** 拥有一个 **BVHAccel** 实例。从根节点开始，我们可以递归地从物体列表构造场景的 **BVH**.

2.2.2 开始实现

一、你需要修改的函数是：
• **castRay(const Ray ray, int depth)**in Scene.cpp: 在其中实现 Path Tracing 算法 。

• **Triangle::getIntersection** in Triangle.hpp: 这里使用*Möller Trumbore* 算法进行光线-三角形相交，请你补充完整该函数。

• **intersect(const Ray &ray)** in Scene.cpp：这个函数的作用是寻找与光线ray相交的物体，你需要使用BVH.cpp的Intersect函数替代当前的遍历算法。

• **IntersectP(const Ray& ray, const Vector3f& invDir,**

const std::array<int, 3>& dirIsNeg) in the Bounds3.hpp: 这个函数的

作用是判断包围盒 BoundingBox 与光线是否相交，你需要按照课程介绍的算法实现求交过程。 注意检查 t__enter = t__exit 的时候的判断是否正确。

• **getIntersection(BVHBuildNode* node, const Ray ray)**in BVH.cpp: 建立 BVH 之后，我们可以用它加速求交过程。该过程递归进行，你将在其中调用你实现的 Bounds3::IntersectP.

• **main(int argc, char** argv) in main.cpp: 当你成功实现BVH加速结构后，请在main中调用buildBVH() in Scene.cpp构造当前scene的BVH。**

二、关于castRay(const Ray ray, int depth)可能用到的函数有：

• **intersect(const Ray ray)**in Scene.cpp: 求一条光线与场景的交点

• **sampleLight(Intersection pos, float pdf)** in Scene.cpp: 在场景的所有光源上按面积 uniform 地 sample 一个点，并计算该 sample 的概率密度

• **sample(const Vector3f wi, const Vector3f N)** in Material.cpp: 按照该材质的性质，给定入射方向与法向量，用某种分布采样一个出射方向

• **pdf(const Vector3f wi, const Vector3f wo, const Vector3f N)** in Material.cpp: 给定一对入射、出射方向与法向量，计算 sample 方法得到该出射方向的概率密度

• **eval(const Vector3f wi, const Vector3f wo, const Vector3f N)** in Material.cpp: 给定一对入射、出射方向与法向量，计算这种情况下的 f_r 值
可能用到的变量有：

• **RussianRoulette** in Scene.cpp: P_RR, Russian Roulette 的概率

2.2.3 Path Tracing 的实现说明

Path Tracing 伪代码如下

```
shade(p, wo)
 Uniformly sample the light at xx (pdf_light = 1 / A)
  Shoot a ray from p to x
  If the ray is not blocked in the middle
    L_dir = L_i * f_r * cos_theta * cos_theta_x / |x-p|^2
  / pdf_light

  L_indir = 0.0
  Test Russian Roulette with probability P_RR
  Uniformly sample the hemisphere toward wi (pdf_hemi = 1
  / 2pi)
  Trace a ray r(p, wi)
  If ray r hit a non-emitting object at q
    L_indir = shade(q, wi) * f_r * cos_theta / pdf_hemi /
  P_RR

  Return L_dir + L_indir
```

按照本次实验给出的框架，我们进一步可以将伪代码改写为：

```
shade(p, wo)
  sampleLight(inter, pdf_light)
  Get x, ws, NN, emit from inter
  Shoot a ray from p to x
  If the ray is not blocked in the middle
    L_dir = emit * eval(wo, ws, N) * dot(ws, N) * dot(ws, NN) / |x-p|^2 / pdf_light

  L_indir = 0.0
  Test Russian Roulette with probability RussianRoulette
  wi = sample(wo, N)
  Trace a ray r(p, wi)
  If ray r hit a non-emitting object at q
    L_indir = shade(q, wi) * eval(wo, wi, N) * dot(wi, N) / pdf(wo, wi, N) / RussianRoulette

  Return L_dir + L_indir
```

请确保你已经清晰地理解 Path Tracing 的实现方式。

3 结果与分析

本节得到结果与调试过程中需要特别注意的一些问题。

3.1 注意事项

1. 本次实验代码的运行非常慢，建议调试时调整main.cpp 中的场景大小或Render.cpp 中的 SPP 数以加快运行速度； 此外， 还可以实现多线程来进一步加快运算。

2. 注意数值精度问题，尤其注意 pdf 接近零的情况， 以及 sampleLight 时判断光线是否被挡的边界情况。这些情况往往会造成渲染结果噪点过多，或出现黑色横向条纹。

3. 如果严格按照上述算法实现， 你会发现渲染结果中光源区域为纯黑。请分析这一现象的原因，并且修改 Path Tracing 算法使光源可见。

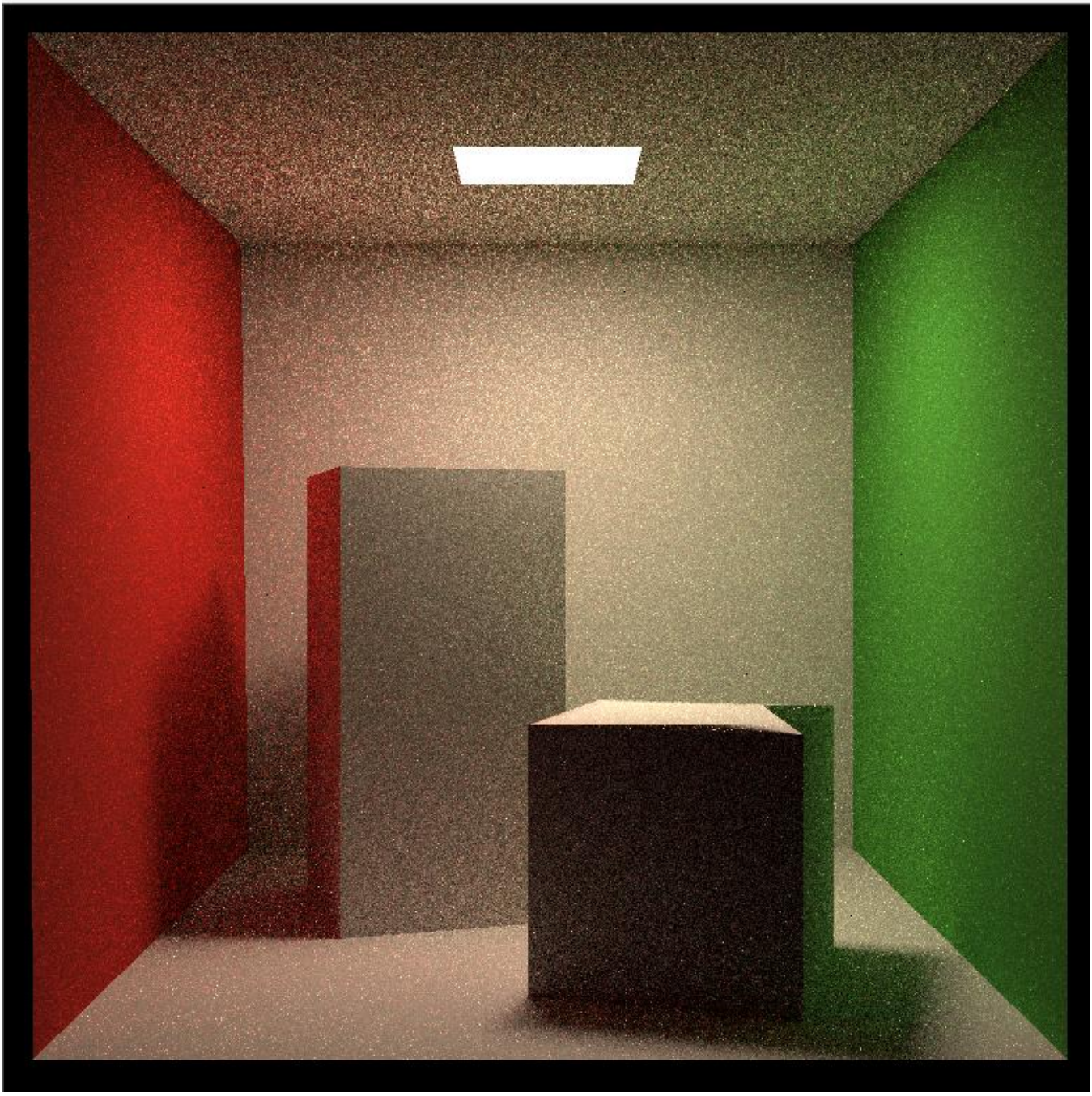
4. 相关补充

Games101 作业7 绕坑引路 (Windows) – 计算机图形学与混合现实在线平台 (games-cn.org)

作业7发布公告 – 计算机图形学与混合现实在线平台 (games-cn.org)

3.2 参考结果

最终结果如下：



4 提交与评分

评分：

- 提交格式正确，包含所有需要的文件；代码可以正确编译运行。
- **[70%]Path Tracing**：正确实现 Path Tracing 算法，并提交分辨率不小于 512*512，采样数不小于 8 的渲染结果图片； **光线与三角形相交**：正确实现 Moller-Trumbore 算法.
- **[30%] BVH加速结构**： 正确实现光线与包围盒求交函数；正确实现BVH加速的光线与场景求交。

提交：

• 当你完成作业后，请清理你的项目，在你的文件夹中需要包含所有的程序文件（无论是否修改)。同时，请提交一个 README 文件写下完成情况，并写清楚自己完成了上述得分点中的哪几点，并说明提交结果的分辨率与采样数、计算时间；

• 同时，请新建一个 /images 目录，将所有实验结果图片保存在该目录下；

• 最后，将上述内容打包，并用“学号_姓名_作业3.zip”的命名方式提交；