实验 7：路径追踪

要求：

- **独立完成**
- **在截止时间之前提交**

---

1 总览

在之前的练习中，我们实现了 Whitted-Style Ray Tracing 算法，并且用 BVH 等加速结构对于求交过程进行了加速。在本次实验中，我们将在上一次实验的基础上实现完整的 Path Tracing 算法。至此，我们已经来到了光线追踪版块的最后一节内容。

在本次实验中，你只需要修改这一个函数:

• **castRay(const Ray ray, int depth)**in Scene.cpp: 在其中实现 Path Tracing 算法

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 代码框架

相比上一次实验，本次实验对框架的修改较大，主要在以下几方面：

• 修改了 main.cpp，以适应本次实验的测试模型 CornellBox

• 修改了 Render，以适应 CornellBox 并且支持 Path Tracing 需要的同一 Pixel 多次 Sample

• 修改了 Object，Sphere，Triangle，TriangleMesh，BVH，添加了 area 属性与 Sample 方法，以实现对光源按面积采样，并在 Scene 中添加了采样光源的接口 sampleLight

• 修改了 Material 并在其中实现了 sample, eval, pdf 三个方法用于 Path Tracing 变量的辅助计算

你需要从上一次实验中直接拷贝以下函数到对应位置：

• **Triangle::getIntersection** in Triangle.hpp: 将你的光线-三角形相交函数粘贴到此处，请直接将上次实验中实现的内容粘贴在此。

• **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，请直接将上次实验中实现的内容粘贴在此处，并且注意检查 t_enter = t_exit 的时候的判断是否正确。

• **getIntersection(BVHBuildNode* node, const Ray ray)**in BVH.cpp: BVH 查找过程，请直接将上次实验中实现的内容粘贴在此处.

3 Path Tracing 的实现说明

课程中介绍的 Path Tracing 伪代码如下 (为了与之前框架保持一致， wo 定义与课程介绍相反):

```
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
```

4 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。