基于 Moer 的 Stochastic Progressive Photon Mapping 实现

> 学号：[redacted]
> 姓名：央金同学
> 邮箱：[redacted]@smail.nju.edu.cn

摘要

本文基于 Moer 开源渲染器，在 **macOS（Apple Silicon + Rosetta 2 x86_64 交叉编译）** 环境下完成全部开发与渲染，实现了 Stochastic Progressive Photon Mapping（SPPM）全局光照算法。其中包含：光子发射与路径追踪、KD-Tree 空间索引、Camera Pass 可见点收集、辐射度密度估计、渐进半径收缩、焦散光子图（L+S+D 路径分离存储），以及 Camera Pass / Photon Pass 双趟多线程并行。

1. 引言

1.1 选题动机

全局光照是真实感渲染中的核心问题。Moer 渲染器原生支持路径追踪（Path Tracing），它在大多数场景下能够获得较好的渲染效果，但在焦散（Caustics）和复杂间接光照等场景中收敛速度较慢。Photon Mapping 作为经典的双趟算法，通过空间密度估计替代高方差的蒙特卡洛采样，在这类场景中具有明显优势；同时，该技术涉及光子追踪、空间索引和渐进更新等多个图形学核心知识，综合性强，具有较高的学习与实践价值。

基于此，本项目选择实现 **Stochastic Progressive Photon Mapping（SPPM）**。在经典 Photon Mapping 的基础上引入随机采样与渐进更新机制，能够在固定内存条件下逐步逼近真实解，也是目前 Photon Mapping 中应用较广、工程价值较高的实现方案。

1.2 相关工作

Photon Mapping 最早由 Jensen (1996) 提出，将光照计算拆分为光子追踪与密度估计两个阶段，首次实现了高效的全局光照与焦散渲染。其后 Hachisuka 等 (2008) 提出 **Progressive Photon Mapping**

（PPM），通过多轮迭代逐步缩小搜索半径，在固定内存下实现渐进收敛，解决了经典 PM 的内存瓶颈。Hachisuka & Jensen (2009) 进一步提出 **Stochastic Progressive Photon Mapping（SPPM）**，每轮随机重建光子图并更新可见点，使算法兼具渐进一致性与随机降噪能力。

在工程实现方面，PBRT (Pharr et al., 2016) 第 16 章给出了 SPPM 的参考实现，是本项目的重要参考。空间索引方面，经典选择为 KD-Tree (Bentley, 1975) 和哈希网格；本项目采用 KD-Tree，其 $O(\sqrt{N} + k)$ 的查询复杂度在中等光子规模下足够高效。

2. 算法原理

2.1 Photon Mapping 基本思想

全局光照的核心问题是求解渲染方程。Moer原始架构采用的路径追踪（Path Tracing, PT）从相机出发反向追踪光路，对大多数场景表现良好，但在 **SDS 路径**（Specular–Diffuse–Specular，例如玻璃后的焦散）上收敛极慢，需要极高采样数才能得到清晰结果。

Photon Mapping（光子映射）采用不同的思路：将光照计算拆分为两个阶段：

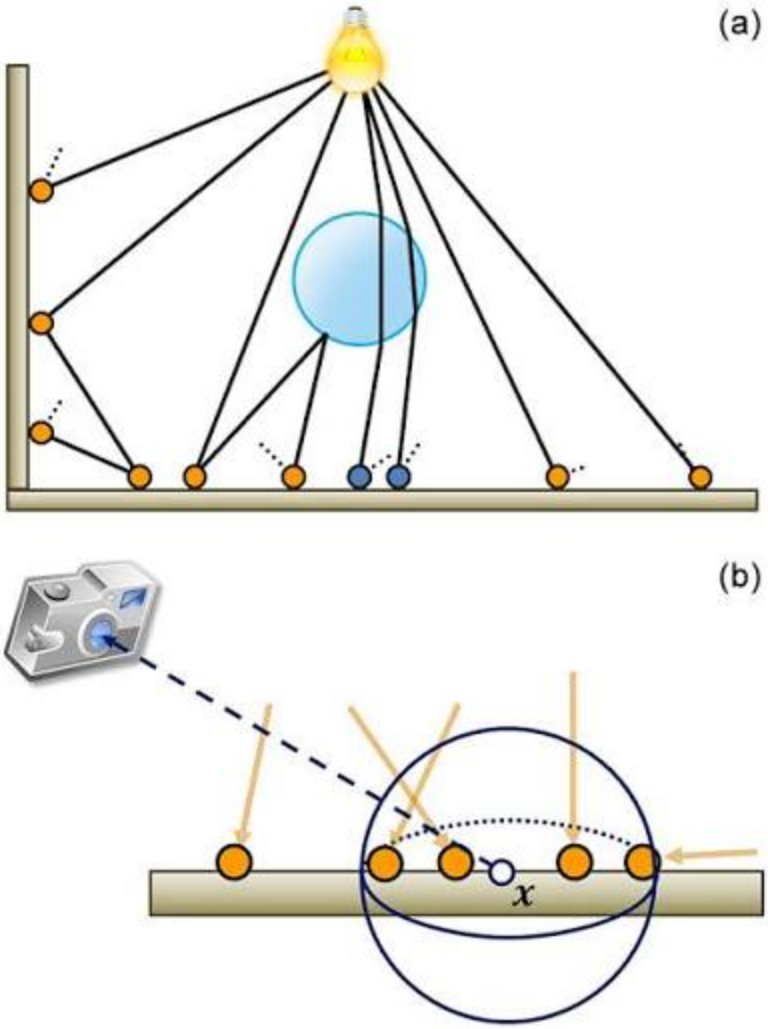
1. **Photon Pass（光子追踪）**：从光源出发正向发射光子，沿场景传播并在非镜面表面存储，形成 **光子图（Photon Map）**。
2. **Camera Pass（相机追踪）**：从相机出发找到可见点，在光子图中查询邻域光子，用 **密度估计** 估算该点的间接光照。

与 PT 的本质区别在于：PT 用蒙特卡洛积分直接估计辐射度；Photon Mapping 先建立光子的空间分布，再通过对邻域光子的统计来估计辐射度。这使得 Photon Mapping 特别适合焦散、间接光照等 PT 难以处理的效应。

2.2 Photon Tracing

光子追踪是 Photon Pass 的核心。每条光子路径可形式化描述为：

1. **光源采样**：按概率 p_i 选择光源 i，在其上采样发射位置与方向，得到初始功率 $\mathbf{\Phi}_0$。
2. **路径延伸**：光子与场景求交，在交点处按 BSDF 采样下一段方向，更新功率并继续传播。
3. **光子存储**：当光子到达 **非镜面** 表面时，将 $(\mathbf{p}, \omega_i, \mathbf{\Phi}, \mathbf{n})$ 存入光子图。
4. **路径终止**：达到最大深度，或 Russian Roulette 判定终止。



2.2.1 光源采样与 BSDF 采样

初始光子功率的无偏估计为：

$$\Phi_0 = \frac{L_e(x,\omega)}{p_{\text{light}} \cdot p_{\text{pos}} \cdot p_{\text{dir}}}$$

其中 L_e 为光源辐射度，p_{light}、p_{pos}、p_{dir} 分别为选光、位置、方向的采样概率。

路径延伸时，在表面点按 BSDF 重要性采样出射方向 ω_o，功率更新为：

$$\Phi_{k+1} = \frac{\Phi_k \cdot f(\omega_i,\omega_o) \cdot |\cos\theta|}{p_{\text{BSDF}}}$$

光子传输使用 **伴随（adjoint）模式**，因为光子从光源出发，传输方向与相机路径相反。

2.2.2 Photon 存储

经典 Photon Mapping 仅在 **漫反射（diffuse）** 表面存储光子。本项目的实现条件为 **非 delta 镜面**（ `roughness < 10⁻⁴` 视为 delta，不存储；其余表面均存储），因此：

材质类型	是否存储	说明
Delta 镜面/透射	不存储	`isSpecularBxDF`，光子直接穿过
Lambert 漫反射	存储	存入全局图或焦散图
塑料（Plastic）	存储	roughness ≈ 0.04，非 delta
光泽金属（roughness 0.05–0.5）	存储	存入全局图；同时标记 `hadSpecularBounce`
焦散接收面（roughness ≥ 0.5）	存储	L+S+D 路径光子存入焦散图（§2.6）

存储的光子记录：

- **位置** $\mathbf{p}$：世界坐标交点
- **入射方向** ω_i：光子到达该点的方向（BSDF 约定：从表面指向光源，用于后续 $f(\omega_o, \omega_i)$ 查询）
- **功率** Φ：光子携带的辐射功率
- **法线** $\mathbf{n}$：用于查询时的法线一致性检查

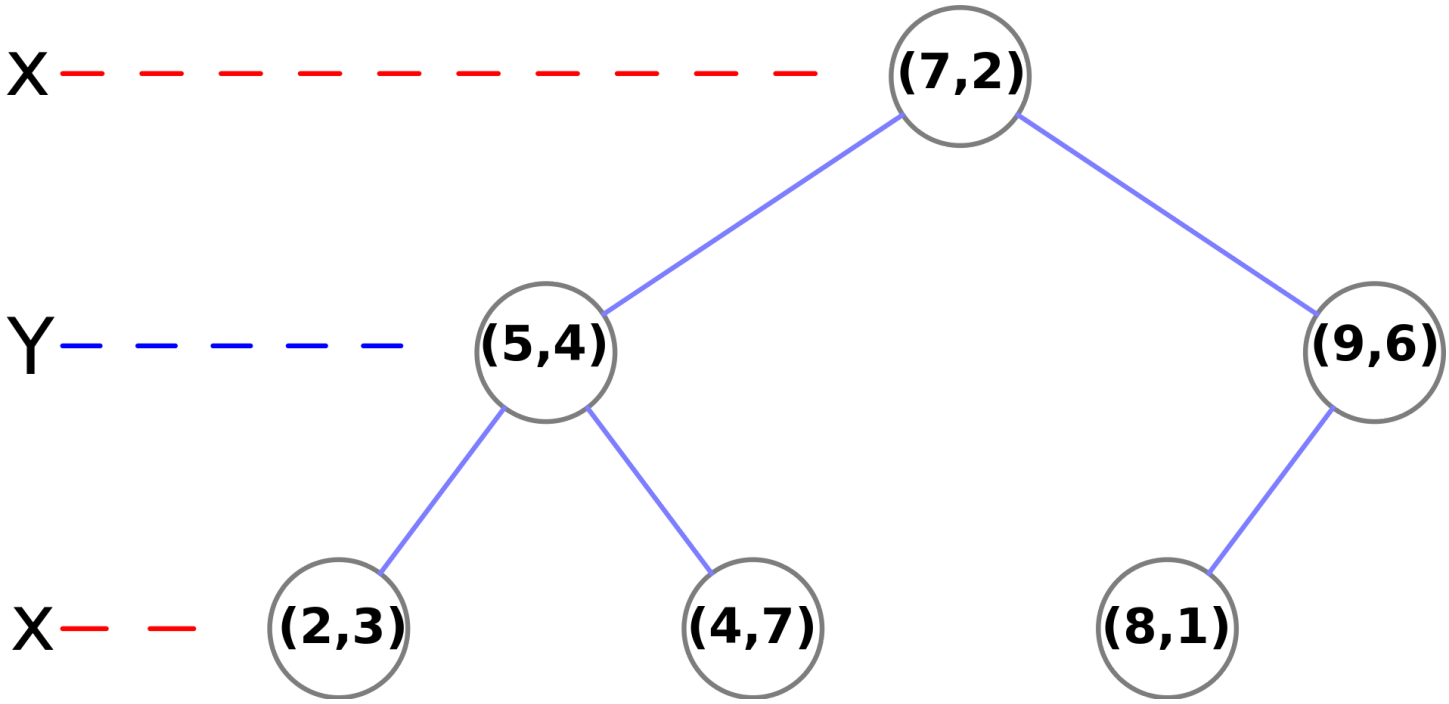
2.2.3 Russian Roulette

为避免路径过长，在深度超过阈值后以概率 p_{survive} 继续追踪；若存活，功率补偿为 $\Phi / p_{\text{survive}}$，保证估计无偏。镜面路径通常不施加 RR，因为镜面反射/折射是确定性过程。

2.3 空间索引结构 KD-Tree

光子图存储后，Camera Pass 需要高效查询某点附近的光子。暴力遍历所有光子为 $O(N)$，在光子数较大时不可接受，因此需要空间索引结构。

KD-Tree 将 k 维空间递归划分为超平面，每个内部节点对应一个分割轴和分割位置，叶节点存储数据点。



建树：在每一层选择 **跨度最大的轴**，按该轴坐标的中位数划分，使左右子树规模尽量平衡，树高约为 $O(\log N)$。

半径查询：给定查询点 $\mathbf{p}$ 和半径 r，从根节点递归下降。若查询点到分割面的距离大于 r，则只需搜索分割面近侧子树；仅当 $|\text{dist}| \leq r$ 时才搜索远侧子树。这种 **分割面剪枝** 可大幅减少访问节点数。

时间复杂度：建树 $O(N \log N)$，单次半径查询期望 $O(\sqrt{N} + k)$（k 为结果数）。

2.4 Camera Pass 与 Visible Point

Camera Pass 从每个像素发射相机光线，穿过镜面/透射表面，在第一个 **非 delta 镜面** 表面记录 **可见点（Visible Point）**：

- 位置 $\mathbf{p}$、法线 $\mathbf{n}$、出射方向 ω_o（指向相机）
- 路径 throughput T（从相机到该点的 BSDF 乘积）
- 该点 BSDF f，用于后续光子贡献计算

同时通过 Next Event Estimation（NEE）计算直接光照，累加到 `directIllumination` 。

2.5 Stochastic Progressive Photon Mapping (SPPM)

SPPM 在 Progressive Photon Mapping 基础上引入随机性，每轮迭代重建光子图，通过渐进缩小搜索半径实现收敛。它**与经典 PM 的执行顺序略有差异**，每轮先执行 Camera Pass 获取当前可见点，再发射新一轮光子与之匹配累积，最后执行 Accumulate 更新统计量。

2.5.1 辐射度密度估计

对可见点 $\mathbf{p}$，在半径 r 内查询光子，累加 BSDF 加权通量：

$$\Phi_{\text{new}} = T \cdot \sum_{i \in \mathcal{N}(r)} f(\omega_o, \omega_i) \cdot |\cos\theta_i| \cdot \Phi_i$$

其中 $\mathcal{N}(r)$ 为以 $\mathbf{p}$ 为圆心、r 为半径的邻域光子集合，T 为相机路径 throughput。

2.5.2 渐进半径收缩（Equation 10）

每轮迭代后，对每个像素更新统计量（Hachisuka & Jensen 2009）：

$$N_{\text{new}} = N + \alpha \cdot M$$

$$r_{\text{new}} = r \cdot \sqrt{\frac{N_{\text{new}}}{N + M}}$$

$$\tau_{\text{new}} = (\tau + \Phi_{\text{new}}) \cdot \frac{N_{\text{new}}}{N + M}$$

其中 M 为本轮查询到的光子数，$\alpha = 2/3$ 为默认缩减率，τ 为累积通量。

2.5.3 最终图像合成

全部迭代结束后，像素辐射度为：

$$L = \frac{L_{\text{direct}}}{N_{\text{iter}}} + \frac{\tau}{N_{\text{iter}} \cdot N_{\text{photon}} \cdot \pi \cdot r^2}$$

直接光照取各轮 NEE 结果的平均；间接光照由渐进统计量 τ 和最终半径 r 归一化得到。

2.6 焦散光子图

标准光子图混合了 L+D（直接漫反射）和 L+S+D（经镜面/光泽反射后到达漫反射面）路径的光子。焦散效果主要来自 L+S+D 路径，需要更小的搜索半径才能获得锐利边界。

本项目实现独立的 **焦散光子图**：

1. 光子追踪时记录是否经过镜面/光泽弹跳（ `hadSpecularBounce` ）
2. L+S+D 光子（经 specular/glossy 弹跳后落在 Lambert 接收面）**仅**存入 `causticsPhotons` ，不进入全局图，避免双重计数
3. 焦散图使用更小的初始半径 `sppm_caustic_initial_radius` （默认 $r_0 \times 0.25$）
4. 独立的渐进统计量： `causticTau` 、 `causticRadius` 、 `causticPhotonCount`

5. 最终图像叠加焦散贡献：$L_{\text{caustic}} = \tau_c / (N_{\text{iter}} \cdot N_{\text{photon}} \cdot \pi \cdot r_c^2)$

表面分类策略：

函数	条件	用途
`isSpecularBxDF`	roughness < 1e-4	delta 镜面，不存储光子
`isGlossySpecularBounce`	0.05 ≤ roughness < 0.5	标记 L+S 路径（金属等）
`isCausticReceiverBxDF`	roughness ≥ 0.5	Lambert 接收面，存储焦散光子

3. 系统设计与实现

3.1 Moer 渲染器架构

本项目在Moer上新增 `SPPMIntegrator` 模块，复用现有的场景、相机、材质、光源与求交基础设施，仅替换光照求解策略。

所有光照算法均实现 `Integrator` 基类的 `render()` 与 `save()` 接口：

```
Integrator                    ← 基类：持有 Camera + Film
├── MonteCarloIntegrator      ← 蒙特卡洛积分器：Sampler + TileGenerator + 多线程
│   ├── AbstractPathIntegrator
│   │   ├── PathIntegrator    ← 表面路径追踪（PT）
│   │   └── VolPathIntegrator ← 体渲染路径追踪（默认积分器）
│   └── GuidedPathIntegrator  ← 路径引导
├── NormalIntegrator          ← 法线可视化调试
└── SPPMIntegrator            ← 本项目新增：随机渐进光子映射
```

Moer 默认使用 `VolPathIntegrator` 进行路径追踪。SPPM 以独立子模块形式接入，不侵入原有 Path/VolPath 积分器：

```
Moer/src/FunctionLayer/Integrator/PhotonMapping/   ← 新增目录
├── Photon.h / PhotonKDTree.*                      ← 光子数据与空间索引
├── SPPMPixel.h                                    ← 逐像素渐进状态
└── SPPMIntegrator.h/.cpp                          ← SPPM 主逻辑

Moer/src/main.cpp                                  ← 添加 integrator 分支
Moer/src/FunctionLayer/Light/InfiniteSphereLight.cpp  ← 补充 sampleEmit
```

这种接入方式保证原有 PT 渲染路径不受影响，同时便于实验对比

3.2 SPPM 积分器设计

`SPPMIntegrator` 继承 `Integrator`，重写 `render()` 实现迭代式两趟渲染。完整主循环如下：

```
void SPPMIntegrator::render(std::shared_ptr<Scene> scene) {
    for (int iter = 0; iter < nIterations; ++iter) {
        cameraPass(scene, iter);      // Pass 1: 可见点 + 直接光照
        photonPass(scene, iter);      // Pass 2: 光子发射 + KD-Tree 构建
        accumulatePhotons();          // 查询光子 + 渐进更新 N, r, τ
    }
    computeFinalImage();              // 合成最终 HDR
}
```

积分器通过 `main.cpp` 读取 `scene.json` 中 `integrator: "sppm"` 及相关参数激活。主要 SPPM 参数：

参数	JSON 键	说明
迭代次数	`sppm_iterations`	默认 64
光子数/轮	`sppm_photons_per_iter`	默认 50000
初始半径	`sppm_initial_radius`	全局图 r_0
缩减率	`sppm_alpha`	默认 2/3
焦散开关	`sppm_enable_caustics`	默认true
焦散初始半径	`sppm_caustic_initial_radius`	通常 $r_0 \times 0.25$

3.3 光子发射与路径追踪

光子发射由 `SPPMIntegrator::photonPass` 驱动：清空光子缓冲，多线程并行调用 `tracePhotonPathLocal`，合并局部缓冲后构建 KD-Tree。

光源选择与初始功率

```
auto [light, pdfChooseLight] = chooseLight(scene, localSampler->sample1D());
LightSampleResult emit = light->sampleEmit(
    localSampler->sample2D(), localSampler->sample2D(), 0.0f);
const double pdfEmit = pdfChooseLight * emit.pdfEmitPos * emit.pdfEmitDir;
Spectrum flux = emit.s / pdfEmit;
```

- `chooseLight` 复用 Moer `PathIntegrator` 的 **均匀选光** 策略，与现有代码风格一致。
- `flux = emit.s / pdfEmit` 对应 2.2.1 节的蒙特卡洛无偏估计，将辐射度除以联合 PDF 得到光子功率。

环境光（Infinite Sphere）处理

原场景光源为 HDR 环境贴图（`infinite_sphere`），Moer 中 `InfiniteSphereLight::sampleEmit` 未实现。我补充了基于 `Distribution2D` 的方向采样：

```
Point2d uv = distribution->SampleContinuous(directionSample, &pdf);
ans.s = emission->eval(textureCoord2D(uv));
ans.wi = UvToDirection(uv, sinTheta);
ans.pdfEmitDir = pdf * width * height / (2π² sinθ);
```

在 `tracePhotonPath` 中，环境光光子从场景包围盒外沿 ω_i 方向射入：

```
if (light->lightType == ELightType::INFINITE) {
    origin = center - direction * sceneRadius;
}
```

原因是无限远处光源没有有限发射点，需人为构造起点，从包围盒外发射可保证光子进入场景且与 Embree 求交兼容。

存储条件与焦散分类

```
const bool deltaSpecular = isSpecularBxDF(bxdf);  // roughness < 1e-4
if (deltaSpecular || isGlossySpecularBounce(bxdf))
    hadSpecularBounce = true;

if (!deltaSpecular) {
    const Vec3d wi = -ray.direction;  // 入射方向，BSDF 约定
    Photon photon(its.position, wi, flux, its.geometryNormal, depth);
    const bool isCausticPhoton = enableCaustics && hadSpecularBounce
                              && isCausticReceiverBxDF(bxdf);
    if (isCausticPhoton)
        localCaustics.push_back(photon);  // 仅焦散图
    else
        localPhotons.push_back(photon);   // 全局图
}
```

- 具体判断条件如2.6分类策略所述
- `wi = -ray.direction` 记录光子**入射方向**（从表面指向光源），供辐射度估计时查询 BSDF $f(\omega_o, \omega_i)$。
- L+S+D 光子只进焦散图，避免与全局图双重计数。

Russian Roulette 与 BSDF 采样

```
if (!deltaSpecular) {
    double pSurvive = (depth <= 3) ? 1.0 : 0.95;
    if (localSampler->sample1D() >= pSurvive) break;
    flux /= pSurvive;
}
BxDFSampleResult bs = bxdf->sample(wo, localSampler->sample2D(), true);
flux = flux * bs.s * cosTheta / bs.pdf;
```

- 前 3 跳不终止，避免短路径过早截断；之后以 95% 存活率继续。
- `adjoint=true` 对应光子从光源出发的伴随传输。

3.4 Camera Pass 与辐射度估计

Camera Pass（多线程按行分配）

```
for (int t = 0; t < renderThreadNum; ++t) {
    threads.emplace_back([this, scene, t]() {
        auto localSampler = sampler->clone(t);
        for (int y = t; y < resolution.y; y += renderThreadNum) {
            // 每像素: generateRay → traceCameraPath → 记录 VisiblePoint + NEE
        }
    });
}
```

`traceCameraPath` 穿过镜面表面，在第一个非 delta 镜面处记录可见点并计算直接光照后返回。

辐射度估计（accumulatePhotons）

```
kdTree.radiusSearch(pixel.vp.position, pixel.radius, queryResults);
const Spectrum Phi = pixel.vp.throughput * fluxSum;  // fluxSum = Σ f·Φ_i
updatePixelProgressive(pixel, M, Phi, false);
```

查询时做法线一致性检查（可见点法线与光子法线、入射方向同侧），并用可见点 BSDF 加权光子通量。

关键 Bug 修复

现象	原因	修复
输出全黑	`accumulatePhotons` 中对 `photon.wi` 再次取反	`wiWorld = photon.wi`（已按 BSDF 约定存储）
镜面路径中断	`cosOut*cosIn <= 0` 检查误杀反射	移除该检查

3.5 KD-Tree 实现

KD-Tree 实现在 `PhotonKDTree.cpp`，采用 Jensen 光子图的经典 **叶节点存单光子** 结构。

递归建树

```
std::unique_ptr<KDNode> PhotonKDTree::buildRecursive(int start, int end) {
    if (count == 1) {
        node->photonIndex = start;
        return node;
    }
    const int axis = argmaxExtent(*photonStorage, start, end);
    const int mid = start + count / 2;
    std::nth_element(photonStorage->begin() + start,
                     photonStorage->begin() + mid,
                     photonStorage->begin() + end,
                     [axis](const Photon &a, const Photon &b) {
                         return a.position[axis] < b.position[axis];
                     });
    node->splitAxis = axis;
    node->splitPos = (*photonStorage)[mid].position[axis];
    node->left = buildRecursive(start, mid);
    node->right = buildRecursive(mid, end);
}
```

实现细节：

- **argmaxExtent**：在子数组范围内计算 AABB，选跨度最大轴，避免点云呈细长分布时划分失衡。
- **nth_element**：$O(n)$ 找中位数，比全排序更高效；建树总复杂度 $O(N \log N)$。
- **photonStorage 指针**：建树会原地重排 `photons` 向量，叶节点存索引而非拷贝，节省内存。

半径查询与剪枝

```
const double planeDist = center[axis] - node->splitPos;
const KDNode *near = planeDist < 0 ? node->left.get() : node->right.get();
const KDNode *far  = planeDist < 0 ? node->right.get() : node->left.get();
radiusSearchRecursive(near, center, radiusSqr, results, maxPhotons);
if (planeDist * planeDist <= radiusSqr)
    radiusSearchRecursive(far, center, radiusSqr, results, maxPhotons);
```

- 先搜近侧子树，仅当查询球体跨越分割面（$|\text{planeDist}| \leq r$）时才搜远侧。
- 叶节点用 `distSqr <= radiusSqr` 判断，避免开方运算。
- `PhotonQueryResult` 返回光子指针与平方距离，供累积时复用。

3.6 渐进半径收缩

`updatePixelProgressive` 实现 Equation 10，同时支持全局图与焦散图：

```
void SPPMIntegrator::updatePixelProgressive(SPPMPixel &pixel, int M,
                                            const Spectrum &Phi, bool isCaustic) {
    double &N = isCaustic ? pixel.causticPhotonCount : pixel.photonCount;
    double &r = isCaustic ? pixel.causticRadius : pixel.radius;
    Spectrum &tau = isCaustic ? pixel.causticTau : pixel.tau;

    const double N_new = N + alpha * static_cast<double>(M);
    const double denom = N + static_cast<double>(M);
    r *= std::sqrt(N_new / denom);
    tau = (tau + Phi) * (N_new / denom);
    N = N_new;
}
```

每轮 `accumulatePhotons` 后日志输出 `avgR` 、 `avgRc` ，用于验证收缩趋势。

3.7 多线程并行策略

SPPM 两趟渲染均实现了多线程并行：

Camera Pass：按行交错分配（ `y = t; y += renderThreadNum` ），每线程独立 `sampler->clone(t)` ，无共享写入冲突（每像素唯一归属一线程）。

Photon Pass：线程局部缓冲 + 无锁合并：

```
struct ThreadLocalPhotons { std::vector<Photon> global, caustic; };
for (int t = 0; t < renderThreadNum; ++t) {
    threads.emplace_back([&, t, count]() {
        auto localSampler = sampler->clone(t * 1000 + 42);
        for (int i = 0; i < count; ++i)
            tracePhotonPathLocal(scene, localSampler.get(),
                                 local.global, local.caustic);
    });
}
// join 后合并各线程局部向量到 photons / causticsPhotons
```

- 消除旧版 `photonMutex` 锁竞争，每线程写入私有 `vector` ，合并阶段串行但开销极小。
- 线程数默认 12（ `renderThreadNum` ），与 MacBook 逻辑核心数匹配。
- `accumulatePhotons` 和 KD-Tree 构建仍为单线程（只读查询 + 一次性建树，并行收益有限）。

4. 实验与结果

4.1 实验环境

项目	配置
硬件	MacBook（Apple Silicon，通过 Rosetta 2 运行 x86_64）
系统	macOS
编译	CMake + conda 环境 （ `osx-64` ）
渲染器	Moer（ `SPPMIntegrator` ）
测试场景	`teapot` （漫反射 GI）、 `cornell-glass` （Cornell Box + 玻璃球焦散）、 `water-caustic` （波浪水面焦散）
线程数	12（Camera Pass + Photon Pass）

场景说明

- **teapot**：全漫反射塑料茶壶 + 棋盘格地面 + HDR 环境光。不含镜面/透射材质，用于验证 SPPM 基础流程和渐进收敛。
- **cornell-glass**：标准 Cornell Box + 两个理想电介质（玻璃，ior=1.5）球体。光线穿过玻璃球在地面产生聚焦焦散，是验证 L+S+D 焦散路径的经典场景。
- **water-caustic**：波浪水面（dielectric，ior=1.33）。水面由程序化生成的 100×100 正则网格构成（20000 三角面，无退化三角形），水面高度叠加多频正弦波位移。光线透过波浪水面折射，在地板和墙壁上产生焦散纹样（caustic patterns），是 Photon Mapping 最经典的展示场景。

4.2 光子分布验证

在 teapot 场景下发射 10000 个光子，导出 `sppm_debug_iter0_photons.csv` 并生成可视化图。

> teapot 为纯漫反射场景，所有光子均存储在非镜面表面（100% 存储率），便于观察光子的空间分布特征和密度变化。而 cornell-glass 和 water-caustic 含有透射/镜面材质，大量光子穿过 delta 镜面后分散到各处，可视化结果不如纯漫反射场景直观。

定量统计

指标	数值
发射光子数	10000

指标	数值
存储光子数	10000（100% 存储率，首跳即命中非镜面表面）
X 范围	[-71.5, 67.4]
Y 范围	[≈0, 6.2]
Z 范围	[-72.4, 67.6]
平均 flux (R,G,B)	(10.09, 9.33, 10.11)

可视化结果

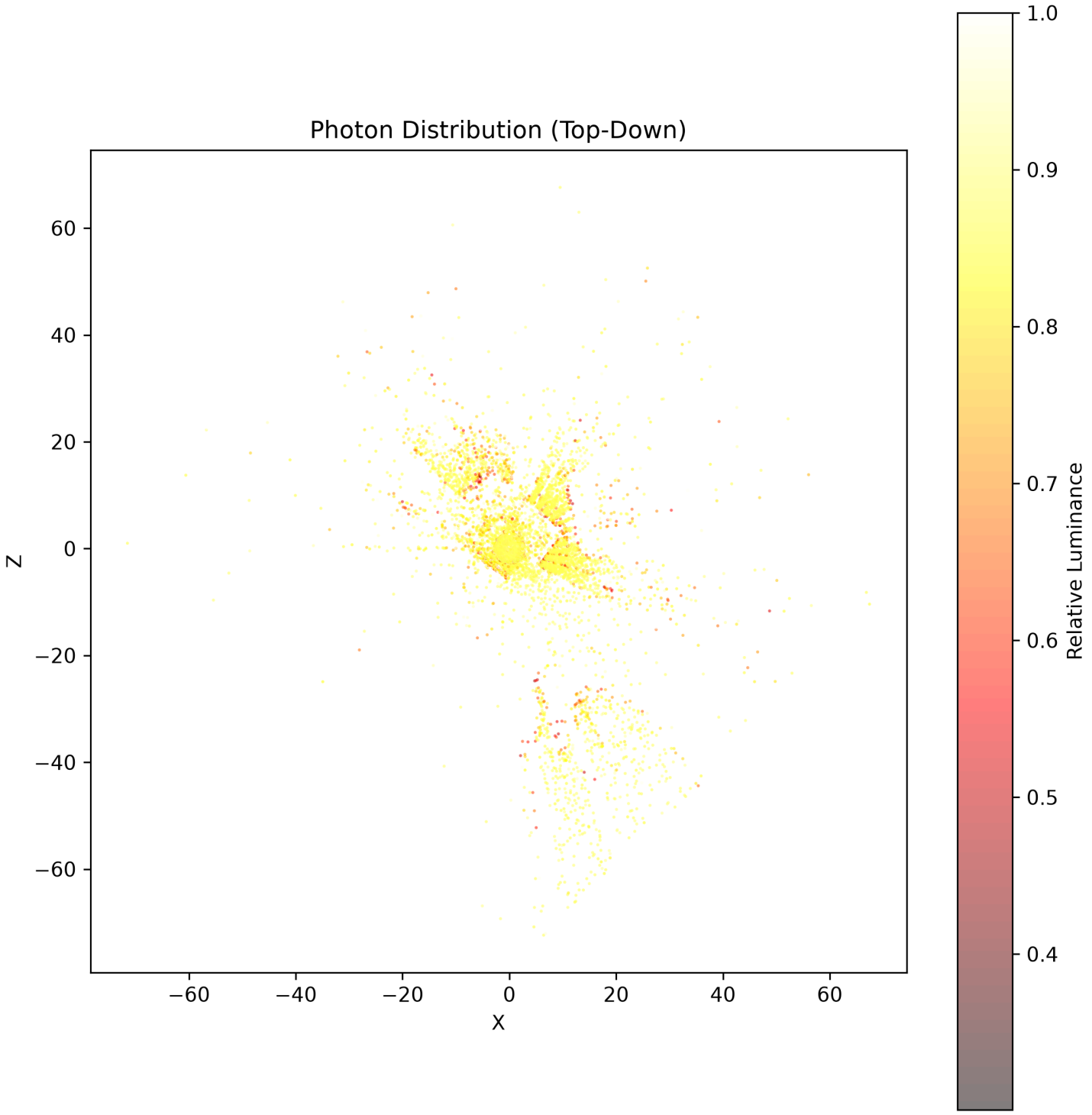


图 4-1：光子分布俯视图（X-Z 平面）。中心高亮区域对应茶壶正下方及地面，四周呈辐射状扩散，与环境光从各方向入射一致。颜色映射表示光子 flux 的相对亮度。

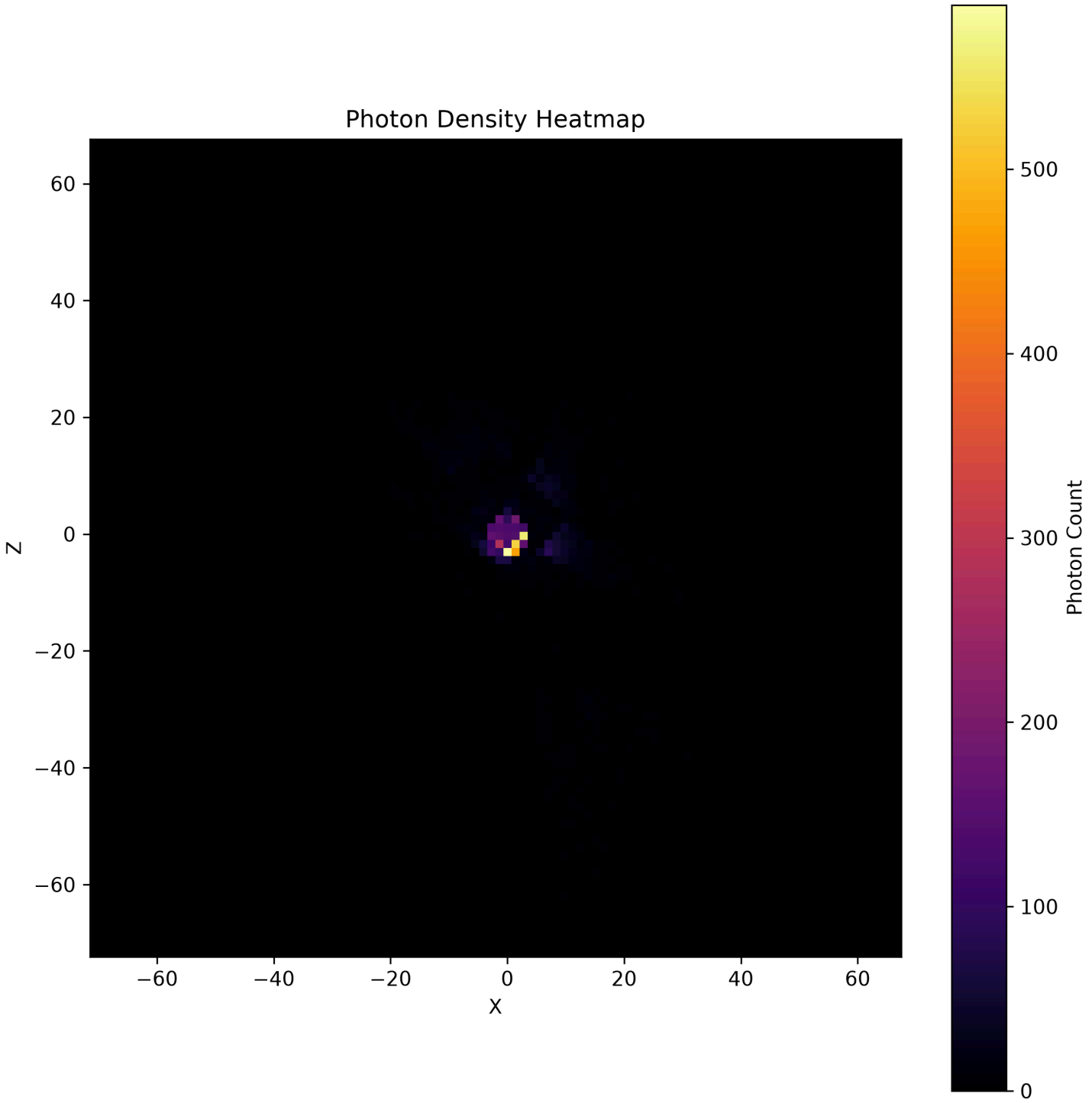


图 4-2：光子密度热图。中心区域（茶壶所在位置）密度最高（>500），向外迅速衰减，符合光源照明下物体附近光子富集的物理规律。

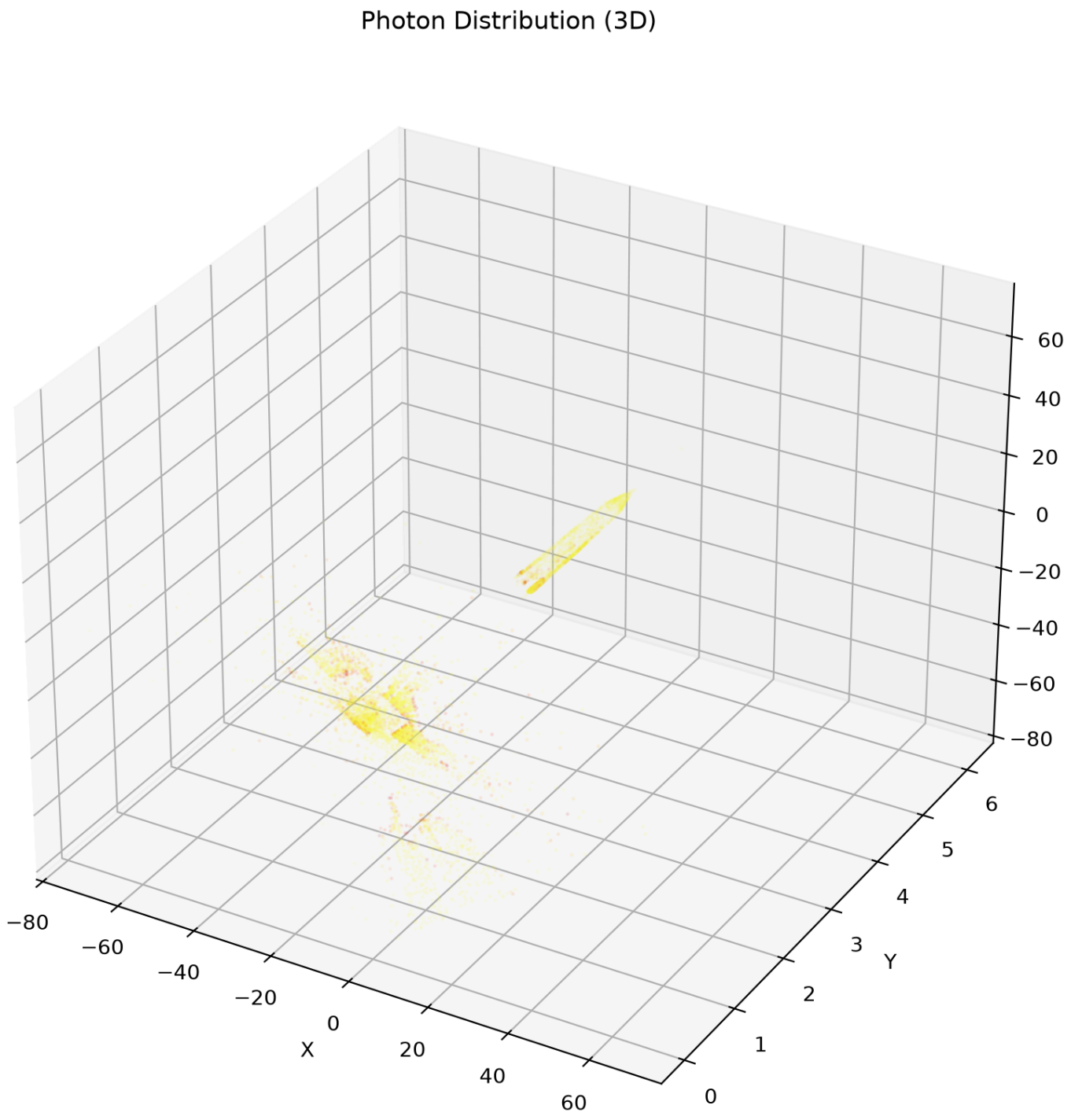


图 4-3：光子分布三维图。光子主要集中在 Y≈0 的地面薄层及 Y∈[0,6] 的茶壶表面；3D 图中可见地面 quad 边缘的斜向高密度带。

结论：光子主要集中在光源可见的漫反射表面（地面、茶壶）附近；中心区域密度显著高于外围，说明光子发射、存储与空间分布合理。Y 方向范围窄是因为场景中大面积接收面为水平地面（quad scale=113），属正常现象。

4.3 SPPM 渐进收敛

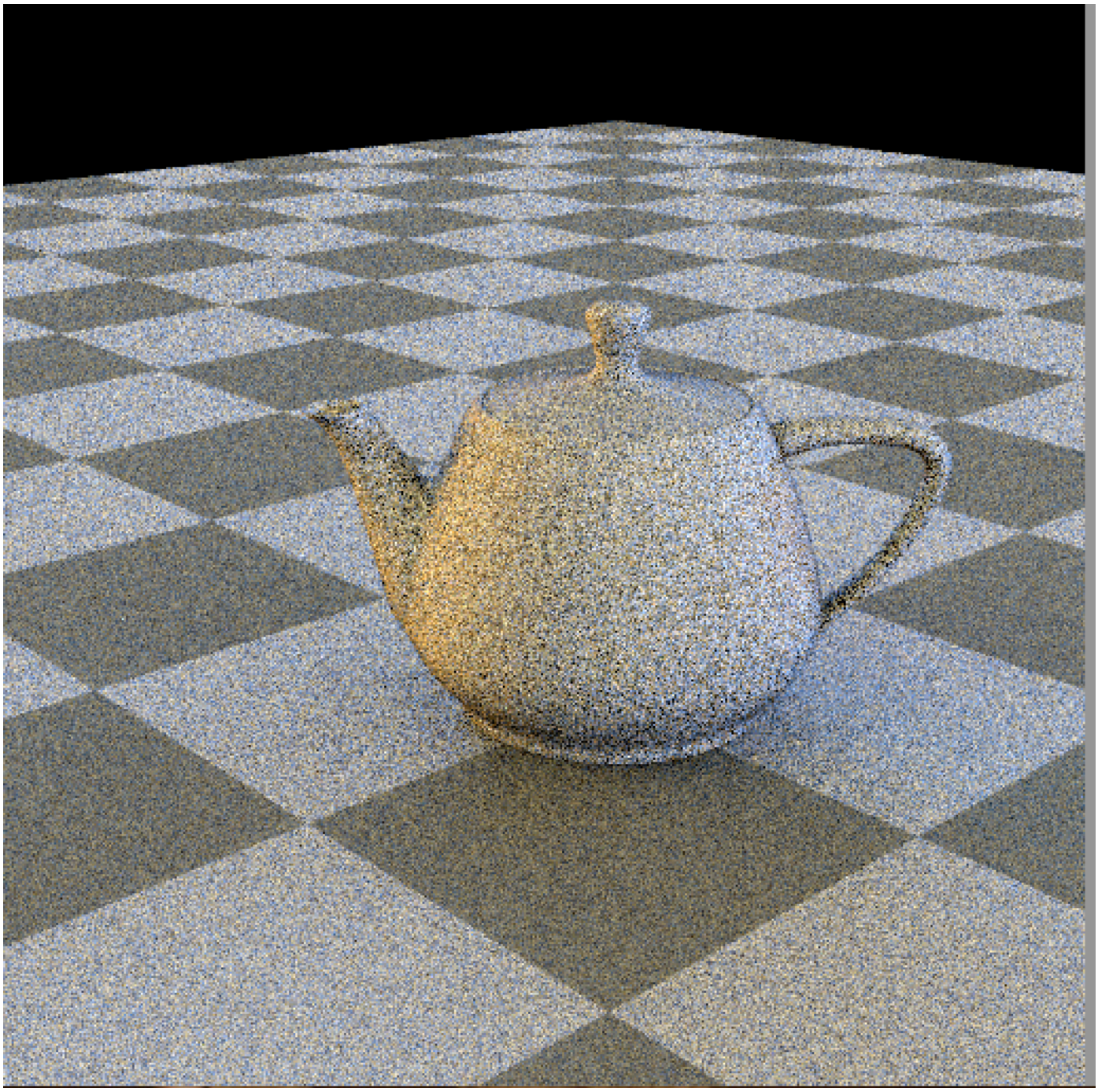
4.3.1 teapot 场景（漫反射 GI）

teapot 为全漫反射塑料材质，用于验证 SPPM 基础流程与渐进收敛。

基础配置：256×256, 8 iter, 10000 photons/iter, $r_0 = 2.0$

迭代	avgR	说明
0	1.70	初始半径收缩开始
7	1.31	8 轮后收缩约 23%

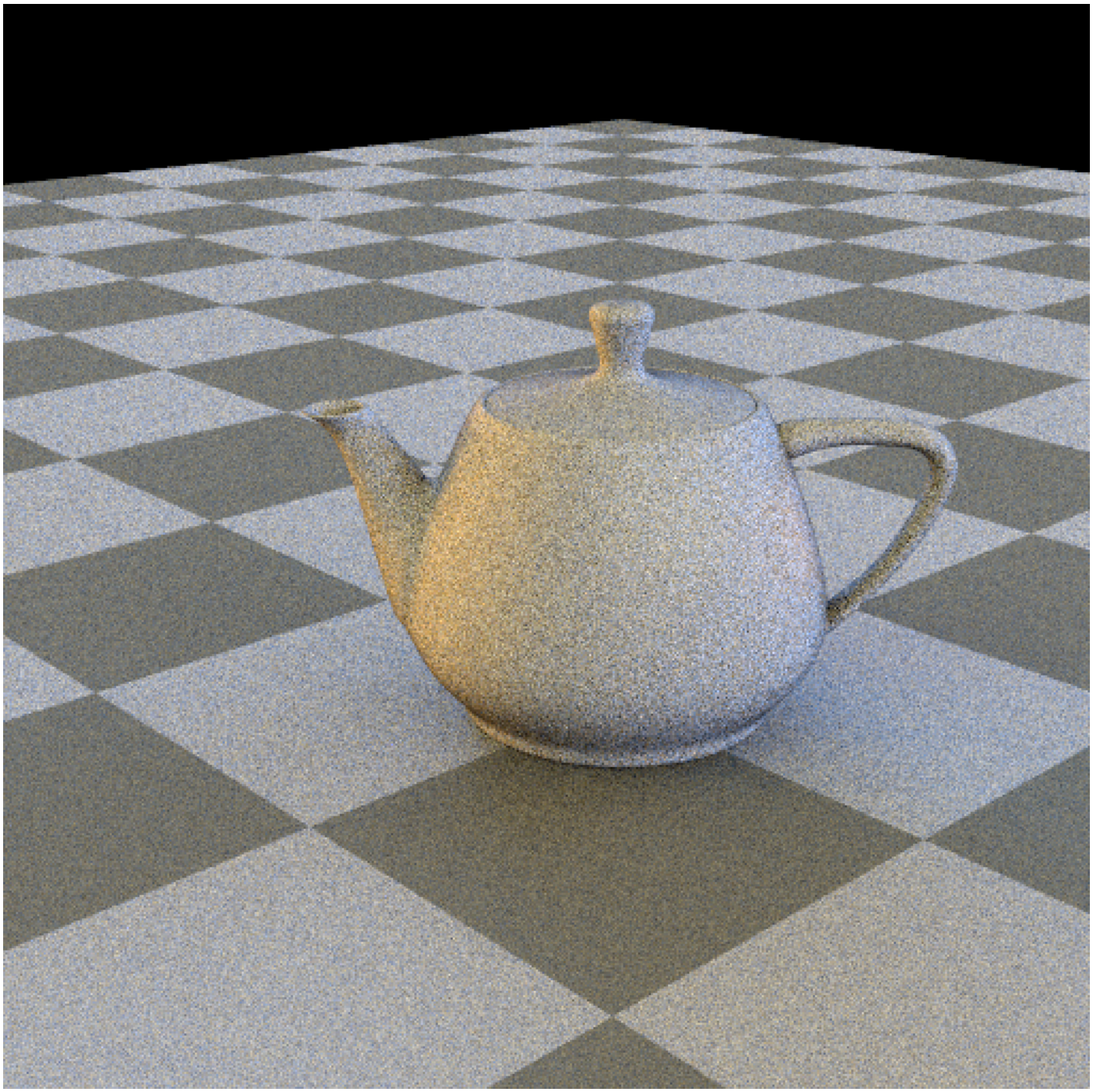
- 输出：



- 噪点明显（参数偏低），但间接光照非零，验证了实现正确性

高质量配置：1280×720, 128 iter, 100000 photons/iter, $r_0 = 0.3$, $\alpha = 0.667$

- 输出：

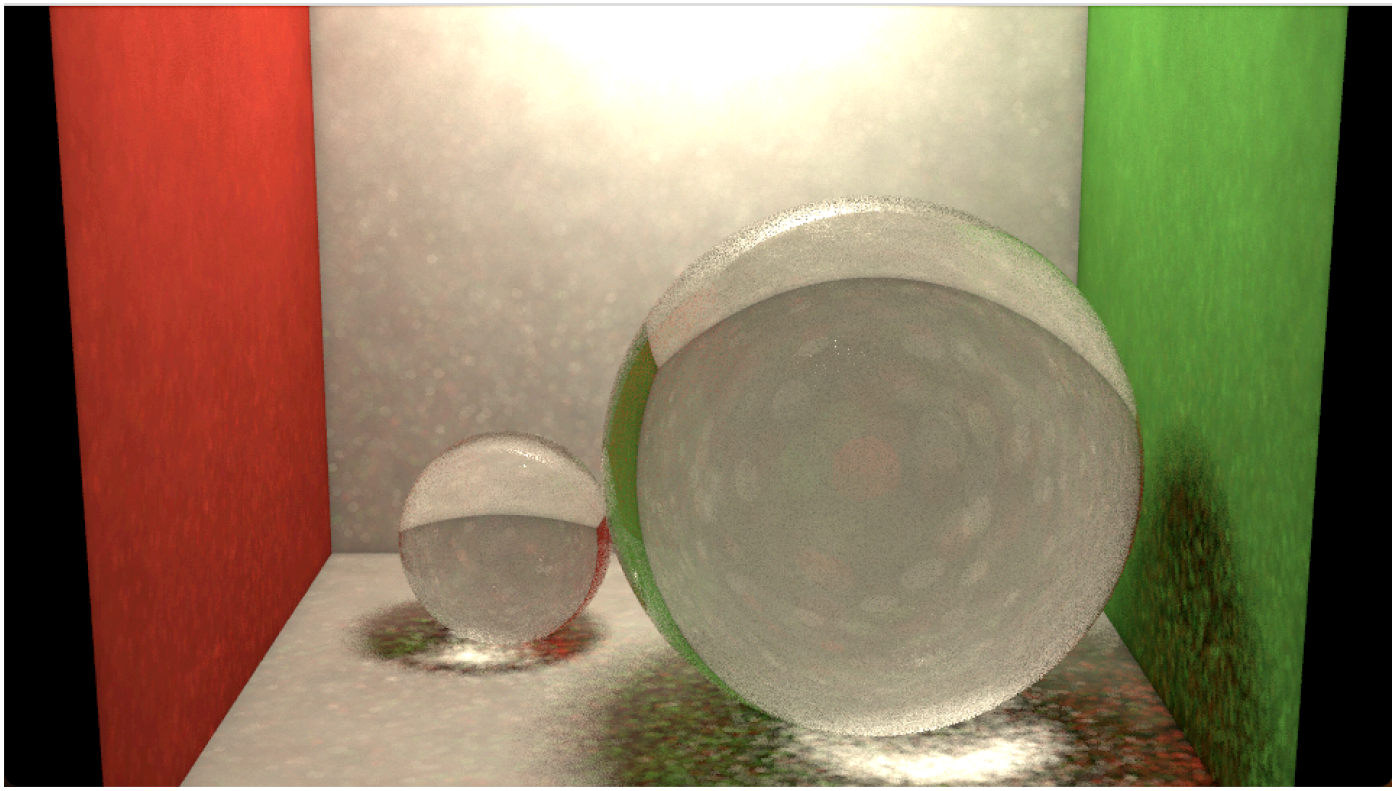


- 渲染时间：约 4 分钟（多线程 Photon Pass）
- 视觉：茶壶+棋盘格清晰可见，间接光照平滑，噪点大幅减少

4.3.2 cornell-glass 场景（玻璃球焦散）

基础配置（`scene_sppm_debug.json`）：512×512, 16 iter, 30000 photons/iter, $r_0 = 0.08$, $r_{c0} = 0.02$

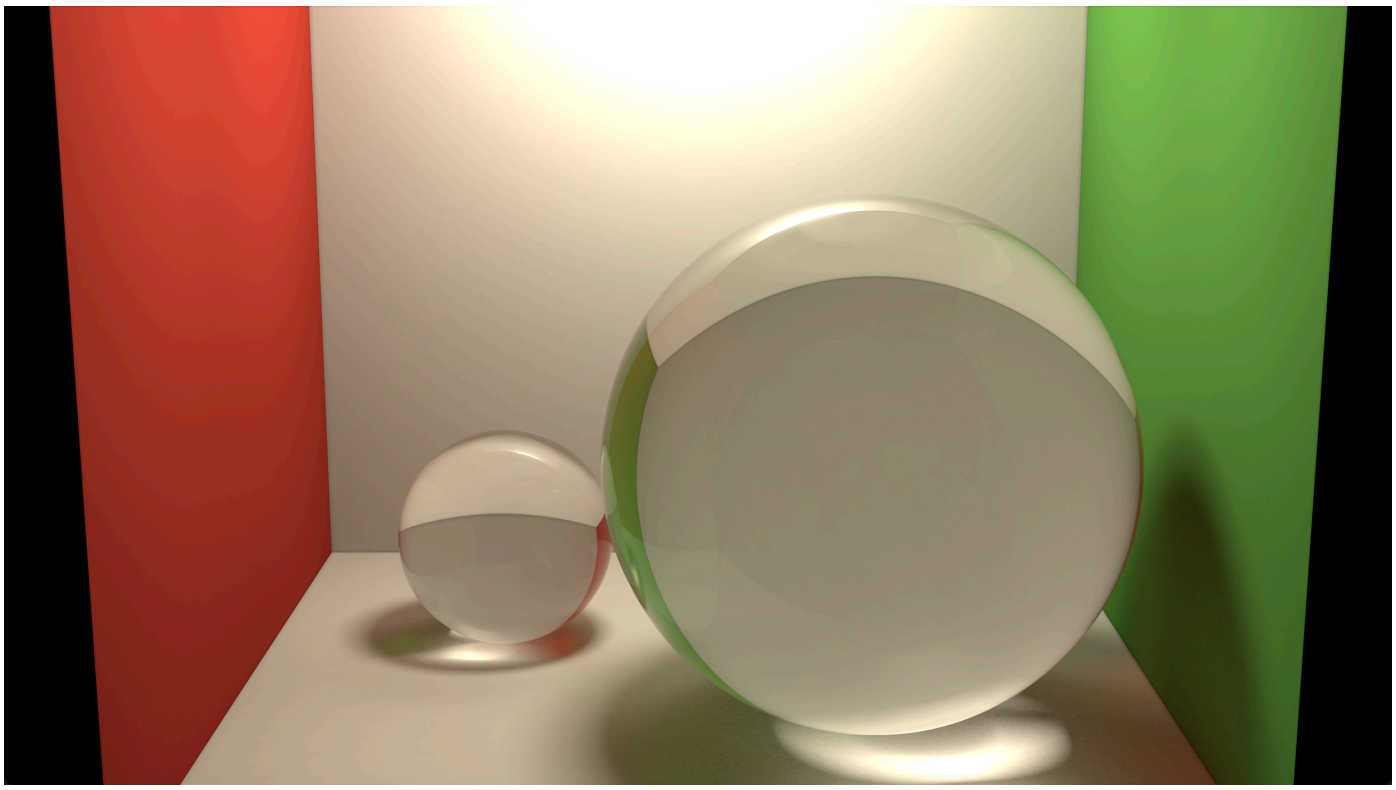
- 输出：



- 渲染时间：约 1–2 分钟
- 视觉：Cornell Box 红绿墙可见，玻璃球透明可辨，地面焦散光斑初现但噪点较多

高质量配置（`scene.json`）：2560×1440, 2048 iter, 300000 photons/iter, $r_0 = 0.05$, $r_{c0} = 0.012$

- 输出：



- 渲染时间：约 6–8 小时（macOS, 12 线程）

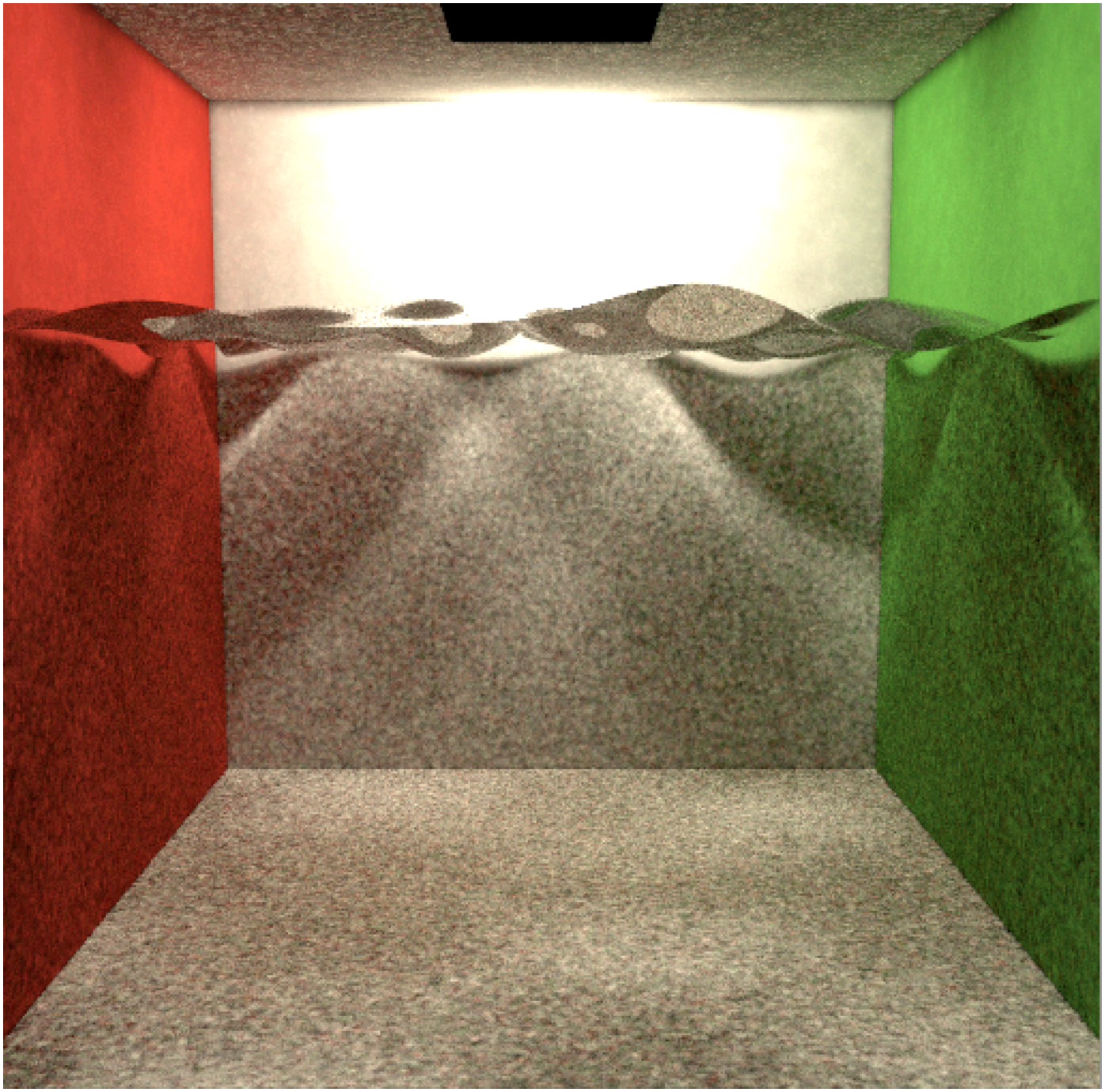
- 视觉：玻璃球产生清晰的聚焦焦散光斑（caustic hotspot），球体内部可见折射色散，红绿墙 color bleeding 自然

4.3.3 water-caustic 场景（波浪水面焦散）

Cornell Box + 程序化生成的波浪水面（dielectric, ior=1.33）。水面为 100×100 正则网格（20000 三角面），叠加多频正弦波位移模拟自然水纹。光线透过水面折射在地板和墙壁上形成复杂焦散纹样，是 Photon Mapping 最具代表性的展示场景。

基础配置（`scene_sppm_debug.json`）：512×512, 32 iter, 50000 photons/iter, $r_0 = 0.06$, $r_{c0} = 0.015$

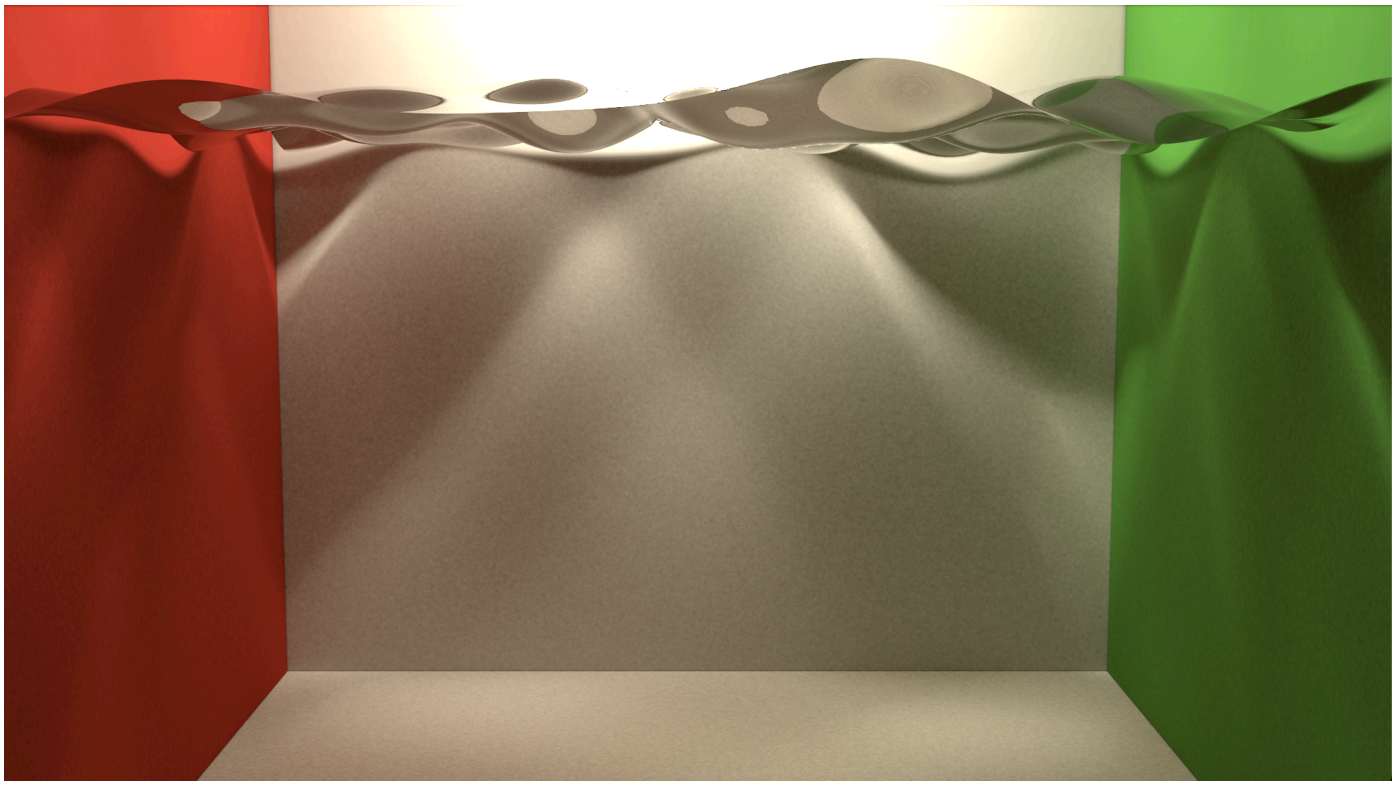
- 输出：



- 渲染时间：约 3–5 分钟
- 视觉：水面形态可见，墙壁和地面上出现焦散明暗纹样雏形

高质量配置（`scene.json`）：1920×1080, 1024 iter, 200000 photons/iter, $r_0 = 0.05$, $r_{c0} = 0.012$

- 输出：

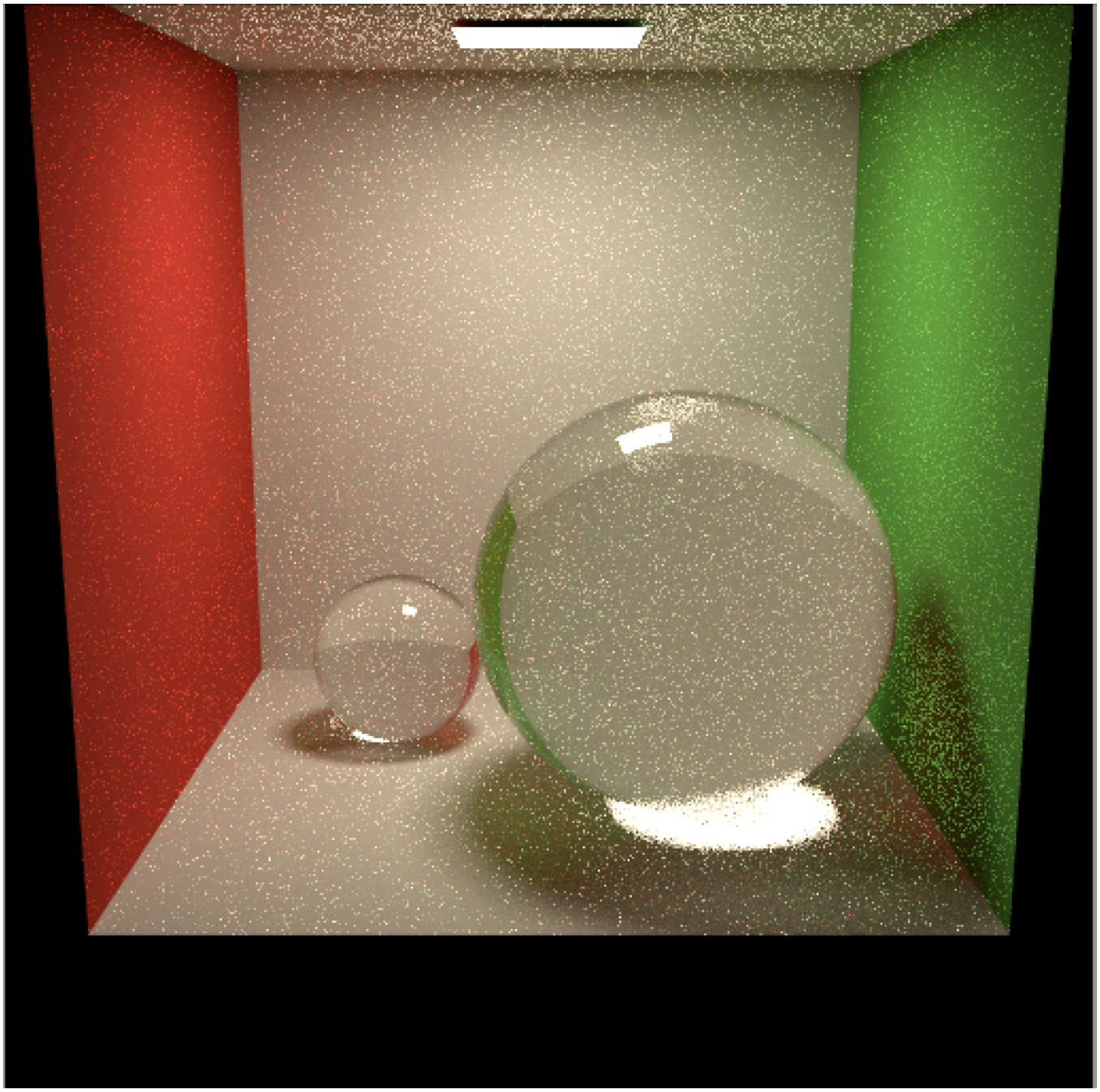


- 预计渲染时间：约 6–10 小时（macOS, 12 线程）
- 视觉预期：水面波纹在地板和后墙上投射出清晰的焦散光纹（caustic patterns），是 Photon Mapping 相比 Path Tracing 的核心优势展示

4.4 PT vs SPPM 定性对比

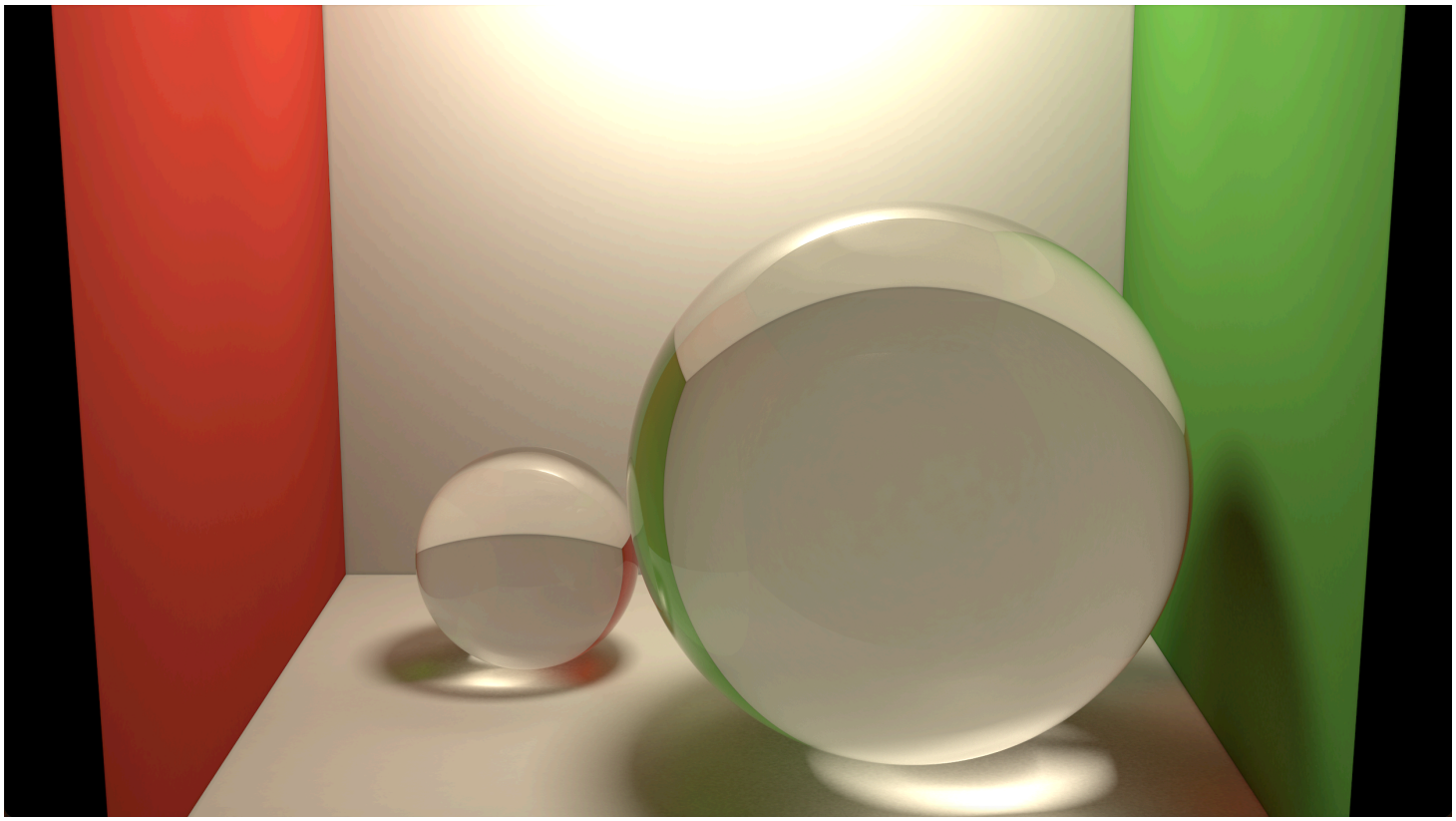
为展示 SPPM 相比传统 Path Tracing（PT）的核心优势，使用 cornell-glass 场景分别用 Moer 原生 `VolPathIntegrator`（PT）和 `SPPMIntegrator` 渲染同一场景。

PT 配置：512×512, spp=256（`"integrator": "volpath"`）
Path Tracing（256 spp），玻璃球整体暗淡，地面焦散区域几乎无光斑，存在大量高频噪点；



SPPM 配置：512×512, 64 iter, 50000 photons/iter（`"integrator": "sppm"`）

SPPM（64 iter），玻璃球透明，地面焦散光斑清晰可见。



分析：
SPPM 的核心优势体现在含焦散（caustics）的场景——PT 依赖从相机出发的随机路径恰好经过镜面反射/折射到达光源，这一概率极低；而 SPPM 从光源正向发射光子穿过镜面，再在漫反射面上收集，天然高效地捕获 L+S+D 路径。

4.5 参数敏感性分析

采用 **控制变量法** 在 teapot （纯漫反射，渲染速度快）场景上进行参数扫描，根据效果选择合理的参数范围进行正式场景渲染。
固定 debug 基线配置：512×512, 64 iterations, 50000 photons/iter, $r_0 = 0.3$, $\alpha = 0.667$。

4.5.1 初始半径 r_0 的影响

r_0	初始 avgR	终态 avgR	nonZeroIndirect	maxLum	渲染时间	视觉特征
0.05	0.048	≈0.03	33198	1.4	19s	间接光区域少， 暗角明显
0.10	0.093	≈0.06	35991	1.3	19s	噪点较多，细节保留
0.30	0.263	≈0.17	36216	1.2	26s	基线， 噪点与模糊平衡

r_0	初始 avgR	终态 avgR	nonZeroIndirect	maxLum	渲染时间	视觉特征
0.50	0.430	≈0.28	36221	1.0	34s	轻微过度模糊
1.00	0.847	≈0.55	36226	0.9	70s	明显模糊， 亮度压低，耗时 3.7×

图 4-8：初始半径 r_0 对渲染结果的影响。从左到右：r_0 从 0.05 增大到 1.0，画面从高噪点→平衡→过度模糊。

分析：r_0 过小（0.05）导致约 3000 个像素（~8%）找不到足够光子，间接光区域缺失；r_0 过大（1.0）导致过度平均，maxLuminance 从 1.4 降至 0.9，细节全部丢失，且因搜索范围大幅增长，渲染耗时增至 3.7 倍。$r_0 = 0.3$ 在噪点与锐度之间取得较好平衡。

4.5.2 缩减率 α 的影响

α	终态 avgR	maxLum	渲染时间	特征
0.50	≈0.15	1.3	22s	收缩快，噪点略多但锐度更高
0.667	≈0.17	1.1	25s	基线
0.80	≈0.20	1.1	29s	收缩慢，轻微模糊，耗时更长

图 4-9：缩减率 α 对渲染结果的影响。差异在 64 轮内较温和，$\alpha = 0.5$ 最锐利，$\alpha = 0.8$ 最平滑。

分析：α 控制每轮半径缩减速度。$\alpha = 0.5$ 半径收缩最快（终态 avgR 最小），适合迭代次数充足时快速收敛；$\alpha = 0.8$ 收缩保守，保留更多低频信息但需更多迭代才能消除模糊。差异在 64 轮内相对温和，理论上在更高迭代次数下区别更显著。

4.5.3 光子数的影响

光子数/轮	终态 avgR	maxLum	渲染时间	特征
10,000	≈0.17	1.1	17s	噪点最多，但速度最快
50,000	≈0.17	1.1	25s	基线
100,000	≈0.17	1.2	34s	噪点减少
500,000	≈0.17	1.2	94s	最平滑，但耗时 3.8×

图 4-10：光子数/迭代对渲染结果的影响。光子数增加降低方差（更平滑），但不影响半径收缩速率（avgR 不变）。

分析：光子数增加不影响半径收缩速率（avgR 不变），但增加每个搜索范围内的光子密度，降低方差。从 10k 到 500k，耗时从 17s 增至 94s（5.5×），边际收益递减。50k 是性价比较高的选择。

4.5.4 迭代次数的影响

迭代次数	withPhotons (末轮)	used (末轮)	avgR (末轮)	normFactor	maxLum	渲染时间
16	124,449	1,311,749	≈0.24	2.5M	1.4	7s
32	116,826	1,022,261	≈0.21	5.0M	1.4	14s
64	108,601	817,214	≈0.17	10.1M	1.2	25s
128	100,129	640,549	≈0.14	20.1M	1.2	47s

图 4-11：迭代次数对渲染结果的影响。这是质量最敏感的参数——16 轮噪点显著，128 轮画面平滑。

分析：迭代次数是最直接影响收敛质量的参数。随着迭代增加，avgR 从 0.24 持续下降至 0.14，`withPhotons`（命中光子的像素数）和 `used`（查询到的光子总数）也相应减少——这是渐进半径收缩的预期行为。16 轮时噪点明显，128 轮后画面明显更平滑。渲染时间与迭代次数近似线性关系。

4.5.5 小结

参数	推荐值	说明
r_0	0.1–0.5	与场景尺度相关；过小导致漏光子，过大导致模糊
α	0.667	经典默认值，平衡收敛速度与稳定性
光子数/轮	50k–100k	边际收益递减，50k 性价比最高
迭代次数	64–128	质量最敏感的参数，直接决定收敛程度

4.6 多线程性能

Photon Pass 多线程实现后，日志显示 `（12 threads）`。在 testball debug 配置下整轮渲染约 3 秒（含编译后首次运行）。

阶段	并行策略	说明
Camera Pass	按行交错，12 线程	D3 已实现
Photon Pass	线程局部缓冲 + 合并	D4 后实现，消除 mutex 竞争
accumulatePhotons	单线程	只读 KD-Tree 查询
KD-Tree build	单线程	每轮一次，$O(N \log N)$

debug 配置下 Photon Pass 占比较小，加速不明显；高质量配置（50000 photons/iter × 64 iter）预期 Photon Pass 多线程可带来显著提速。每轮迭代新增计时日志（ `Iter N done in Xs, ETA Ys` ）便于估算总渲染时间。

5. 总结与展望

5.1 工作总结

本项目在 Moer 渲染器上完整实现了 Stochastic Progressive Photon Mapping，覆盖 Photon Pass 与 Camera Pass 双趟流程、KD-Tree 半径查询、渐进半径收缩（Equation 10）、独立焦散光子图（L+S+D 路径分离），以及双趟多线程并行。在 macOS（Apple Silicon + Rosetta 2 x86_64）环境下完成编译、调试与全部实验渲染。

实验结果验证了实现的正确性与有效性：**teapot** 场景下间接光照随迭代逐步收敛，光子空间分布合理；**cornell-glass** 场景下 SPPM 在有限迭代内即可得到清晰的地面焦散光斑，而同等采样下的 Path Tracing 几乎无法捕获 SDS 路径；**water-caustic** 场景下墙壁与后墙可见焦散明暗纹样，表明焦散图管线整体可用。参数敏感性实验（§4.5）在 teapot 上系统扫描了 r_0、α、光子数与迭代次数，为正式场景的配置提供了依据。

5.2 讨论

（1）SPPM 在焦散场景中的优势与局限

PT 与 SPPM 的对比（§4.4）直观说明了两类算法的路径采样差异：PT 从相机出发，命中 SDS 路径的概率极低，表现为高方差噪点；SPPM 从光源正向追踪光子，经镜面/折射面后落在漫反射接收面，天然适配 L+S+D 类焦散。cornell-glass 的成功正体现了这一优势——汇聚型焦散（玻璃球透镜效应）即使存在一定核模糊，仍能在接收面上形成高对比度 hotspot。

相比之下，water-caustic 等**分散型、高空间频率**的焦散对 SPPM 的圆盘密度估计更为敏感：接收面上的焦散纹样虽已出现，但整体偏柔和，缺少参考图中常见的细密亮线。经过分析，我发现这并非单一实现错误所致，而是 SPPM 核滤波、场景尺度与参数配置共同作用的结果——也提示我，**同一套参数在 diffuse GI 与 caustic 场景上的最优区间并不相同**，参数扫描结论不宜直接外推。

（2）工程实现与算法特性的匹配

多线程改造（§3.7、§4.6）消除了 Photon Pass 的锁竞争，Camera Pass 与 Photon Pass 均可按行并行，在高质量配置下预期带来可观提速。另一方面，KD-Tree 建树与 `accumulatePhotons` 仍为单线程，且每轮迭代需完整重建光子图——这是 SPPM 的设计选择，保证了渐进一致性的同时，也构成了大规模场景下的性能瓶颈。如何在保持算法正确性的前提下进一步并行化或引入空间哈希等替代结构，是工程层面值得继续探索的方向。

5.3 改进方向

结合上述讨论，后续可从以下几方面推进：

1. **焦散场景的参数与场景协同调优**：针对 water-caustic 等场景单独扫描 r_{c0}、α 与迭代次数，结合 `avgRc` 日志与焦散区域亮度统计建立调参闭环；场景侧可尝试缩小光源面积、提高波浪空间频率或缩短水面到接收面的距离，使焦散特征尺度与 SPPM 搜索半径相匹配。
2. **光照估计能力的扩展**：当前 NEE 仅处理 Lambert 面上的直接可见光，无法估计经折射/反射面到达接收点的 specular 路径；可探索 specular-aware NEE 或 Manifold Next Event Estimation（MNEE），与现有焦散光子图互补，进一步改善含水、含玻璃等复杂材质场景的直接照明与焦散锐度。
3. **性能与可扩展性**：并行化 KD-Tree 构建与光子累积阶段；评估哈希网格（`sppm_map_type: hashgrid`）在大光子规模下的查询效率；引入每轮焦散光子占比、`causticUsed` 等指标的自动化监控，便于快速诊断参数是否合理。
4. **算法层面的拓展**：在 SPPM 基础上尝试 Vertex Connection and Merging（VCM）、Photon Beam Diffusion 等混合方法，或引入时域/空域降噪后处理，在偏差与方差之间取得更灵活的平衡。

5.4 结语

本项目从算法原理到工程实现，完成了 SPPM 在 Moer 上的集成与验证，并通过 teapot、cornell-glass、water-caustic 三类场景覆盖了漫反射 GI、汇聚型焦散与分散型焦散等不同需求。实验表明，SPPM 显著改善了 PT 在焦散问题上的收敛效率，同时也让我们更清楚地认识到密度估计类方法在高频细节上的固有 trade-off。后续若能在参数调优、场景设计与算法扩展上继续深入，有望进一步逼近真实感渲染中焦散与全局光照的综合目标。

参考文献

1. Jensen, H. W. (1996). "Global Illumination using Photon Maps." *Eurographics Workshop on Rendering*.
2. Hachisuka, T., Ogaki, S., & Jensen, H. W. (2008). "Progressive Photon Mapping." *ACM Transactions on Graphics (SIGGRAPH Asia)*.
3. Hachisuka, T., & Jensen, H. W. (2009). "Stochastic Progressive Photon Mapping." *ACM Transactions on Graphics (SIGGRAPH Asia)*.
4. Pharr, M., Jakob, W., & Humphreys, G. (2016). *Physically Based Rendering: From Theory to Implementation*, 3rd edition. Chapter 16.
5. Bentley, J. L. (1975). "Multidimensional Binary Search Trees Used for Associative Searching." *Communications of the ACM*, 18(9).
6. Moer Renderer: https://github.com/NJUCG/Moer

附录

A. 完整参数配置说明

SPPM renderer 块示例（testball 高质量）

```
{
  "renderer": {
    "output_file": "testball_sppm",
    "integrator": "sppm",
    "sppm_iterations": 64,
    "sppm_photons_per_iter": 50000,
    "sppm_max_depth": 8,
    "sppm_initial_radius": 0.3,
    "sppm_alpha": 0.6667,
    "sppm_map_type": "kdtree",
    "sppm_enable_caustics": true,
    "sppm_caustic_initial_radius": 0.1
  }
}
```

B. 源代码结构

新增/修改文件：

文 件	说明
`Integrator/PhotonMapping/Photon.h`	光子数据结构
`Integrator/PhotonMapping/PhotonKDTree.h/.cpp`	KD-Tree 空间索引
`Integrator/PhotonMapping/SPPMPixel.h`	逐像素 SPPM 状态（含焦散统计量）
`Integrator/PhotonMapping/SPPMIntegrator.h/.cpp`	SPPM 积分器完整实现
`Light/InfiniteSphereLight.cpp`	补充 `sampleEmit`
`main.cpp`	积分器选择 + SPPM/焦散参数
`run.sh`	环境检查 + 场景目录渲染
`scenes/*/scene_sppm.json`	高质量场景配置
`scenes/*/scene_sppm_debug.json`	快速调试场景配置
`scenes/cornell-glass/scene_pt.json`	PT 对比渲染配置
`scenes/water-caustic/models/water.obj`	程序化生成的波浪水面 mesh
`scripts/visualize_photons.py`	光子分布可视化
`scripts/run_teapot_sweep.sh`	参数敏感性实验自动化脚本