图形绘制技术大作业报告：基于 MoerEngine 的硬件路径追踪实现

> 学生：张同学
> 学号：[redacted]
> 选题：硬件路径追踪

摘要

本项目在 MoerEngine 现有 Vulkan 光线追踪框架上，实现了一个独立的硬件路径追踪渲染分支。原始 MoerEngine 的 Raytracing 模式主要由 GBuffer RayQuery、ReSTIR DI 直接光照、NRD/TAA 和后处理组成，属于面向实时光照的混合式光追管线；本项目新增的 Path Tracing 分支则以路径积分为核心，使用硬件 RayQuery 进行多反弹追踪，并结合材质采样、BRDF 重要性采样、Next Event Estimation、逐帧累积和 firefly clamp，实现了一个可交互、可收敛的硬件路径追踪原型。

实现后的渲染器能够在 Sponza 等测试场景中显示多反弹间接光、环境光、方向光以及场景局部彩色光源。相比初版纯 BRDF 随机路径追踪，加入直接光源采样后，收敛速度明显提升，长时间静止后画面噪声显著减少。

选题背景

路径追踪是一种基于 Monte Carlo 积分的全局光照算法。它从相机发射光线，沿表面反射方向递归采样路径，并累积路径上遇到的光照贡献。相比传统 rasterization 或单次反射的实时光追，路径追踪能够自然表达软阴影、间接光照、镜面反射、粗糙材质高光和环境光影响。

课程作业建议参考 NVIDIA OptiX。OptiX 是 NVIDIA 面向 RTX GPU 的可编程光线追踪框架，强调 RT Core 加速、可编程 ray generation / closest hit / miss、渐进式渲染和 denoising。本项目没有直接引入 OptiX SDK，因为 MoerEngine 已经建立在 Vulkan RayQuery、TLAS/BLAS、bindless resource 和现有 renderer 架构上。项目采用的是同类硬件光追思想：在 MoerEngine 内部通过 Vulkan RayQuery 调用硬件光追能力，实现 path tracing 的核心算法。

参考资料：

- NVIDIA OptiX: https://developer.nvidia.com/rtx/ray-tracing/optix
- MoerEngine 现有 Raytracing / ReSTIR DI 管线代码

原始 MoerEngine 光追管线分析

在本项目开始前，MoerEngine 已经有 Raytracing 模式，但它不是完整路径追踪。其主要流程如下：

```
PrepareLightPass
  -> GBufferPass / GBufferRT.hlsl
  -> LightingPass / ReSTIR DI
  -> optional NRD denoising
  -> CompositionPass
  -> TAA
  -> ToneMapping
  -> Visualize/Present
```

这个管线的重点是实时直接光照和降噪。它用 RayQuery 生成 GBuffer，并通过 ReSTIR DI 对直接光源进行重采样，所以能高效表现大量局部光源和较稳定的直接光照。但它不是从相机射线开始递归追踪路径，也没有在统一路径积分中处理多次间接反弹。因此，本项目的差异化目标是新增一个独立 Path Tracing 分支，而不是简单修改原来的 ReSTIR DI。

项目目标

本项目目标包括：

1. 在 RaytracingRenderer 中新增可切换的 Path Tracing 渲染分支。
2. 使用 Vulkan RayQuery / TLAS 实现硬件加速的 primary ray 和 secondary ray。
3. 在 shader 中实现路径追踪核心循环，包括命中、材质读取、BRDF 采样、throughput 更新、多反弹和 Russian roulette。
4. 支持环境光、方向光、自发光材质和 MoerEngine 场景局部光源。
5. 加入逐帧累积，提高静止画面的收敛质量。
6. 加入 Next Event Estimation 和 firefly clamp，降低噪声与亮斑。
7. 在编辑器 UI 中提供可调参数，方便展示和截图。

实现概览

新增或修改的主要文件如下：

```
source/runtime/render/renderer/raytracing/PathTracingPass.h
source/runtime/render/renderer/raytracing/PathTracingPass.cpp
shaders/pipelines/raytracing/passes/PathTracing.hlsl
source/runtime/render/renderer/raytracing/RaytracingRenderer.cpp
source/runtime/render/renderer/raytracing/RaytracingConfig.h
source/editor/raytracing_ui/RaytracingUI.cpp
source/runtime/render/shaderheaders/shared/ShaderParameters.h
```

新的 Path Tracing 分支流程如下：

```
PrepareLightPass
  -> PathTracingPass / PathTracing.hlsl
  -> ToneMapping
  -> Visualize SceneColor
```

其中 `PrepareLightPass` 被复用来整理场景灯光数据，将 point light、directional light、environment light 和 emissive triangle light 打包到 `light_data_buf` 中。PathTracing shader 再通过 bindless handle 读取这些灯光，用于直接光源采样。

核心算法

1. Primary Ray

每个像素会从相机发出一条或多条 primary ray。为了抗锯齿和降低采样相关性，ray 的像素位置加入随机 jitter：

```
pixel_pos + random_jitter -> clip space -> world space -> ray direction
```

2. RayQuery 命中场景

shader 中通过 `RayQuery` 查询 TLAS。命中三角形后记录：

- instance id
- primitive id
- barycentric coordinates
- hit distance
- front/back face 信息

随后复用 MoerEngine 现有的 `GetGeometryRecordFrom` 和 `SampleGeometryMaterial` 读取几何属性和 PBR 材质，包括 base color、normal、roughness、metalness、emissive 等。

3. BRDF 与路径 throughput

路径追踪的核心是对渲染方程进行 Monte Carlo 估计。每次命中表面后，计算当前材质的 diffuse/specular 权重，并随机采样一个新的反射方向。

路径贡献通过 throughput 传递：

```
throughput *= BRDF * cos(theta) / pdf
```

当路径命中环境光或达到最大反弹次数时，路径终止。为避免深层反弹带来过高成本，shader 中还加入了 Russian roulette。

4. Next Event Estimation

纯路径追踪如果只靠 BRDF 随机方向寻找光源，在小光源、点光源、强高光场景中会产生很高方差，表现为长时间不消失的亮斑。为降低噪声，本项目加入了 Next Event Estimation：

1. 每次命中表面后，从 MoerEngine 的 local light buffer 中随机选择一个灯光。
2. 对该灯光采样一个点或方向。
3. 从当前表面向光源打 shadow ray。
4. 若无遮挡，则直接累加该光源贡献。

其估计形式为：

```
direct += BRDF(surface, light_dir) * light_radiance / light_pdf
```

因为光源是主动采样的，彩色局部光源能够稳定进入结果图像，收敛速度也显著提升。

5. Firefly Clamp

路径追踪中偶发的高能量样本会形成 firefly，即亮斑。项目中提供了 `Firefly Clamp` 参数，在每条路径贡献写入累积前限制其最大亮度。该方法会引入一定偏差，但对实时预览和展示非常有效。

6. Accumulation

每帧 path tracer 输出一个新的 Monte Carlo 估计值。如果相机和渲染参数不变，系统会将当前帧与历史帧做 running average：

```
color = lerp(previous_color, current_sample, 1 / (frame_index + 1))
```

当相机、环境贴图、光照、max bounces、SPP、direct light samples 或 firefly clamp 变化时，累积会自动清零，避免旧结果污染新画面。

UI 参数

在编辑器中打开：

```
Raytracing Config -> Path Tracing
```

主要参数：

- `Enable Path Tracing`：切换到新增的路径追踪管线。
- `Accumulate`：是否进行逐帧累积。
- `Max Bounces`：最大路径反弹数。
- `Samples Per Pixel`：每帧每像素路径数。
- `Direct Light Samples`：每次命中表面主动采样的光源数。
- `Firefly Clamp`：亮斑抑制阈值。
- `Debug Output`：可切换 Path Traced、Base Color、Normal、Hit Distance。

截图使用的参数：

```
Max Bounces = 4
Samples Per Pixel = 1 或 2
Direct Light Samples = 1 或 2
Firefly Clamp = 15 到 25
Accumulate = On
```

实验截图

图 1：原始 MoerEngine Raytracing 模式

截图文件：

```
screenshots/01_original_raytracing_restir.png
```

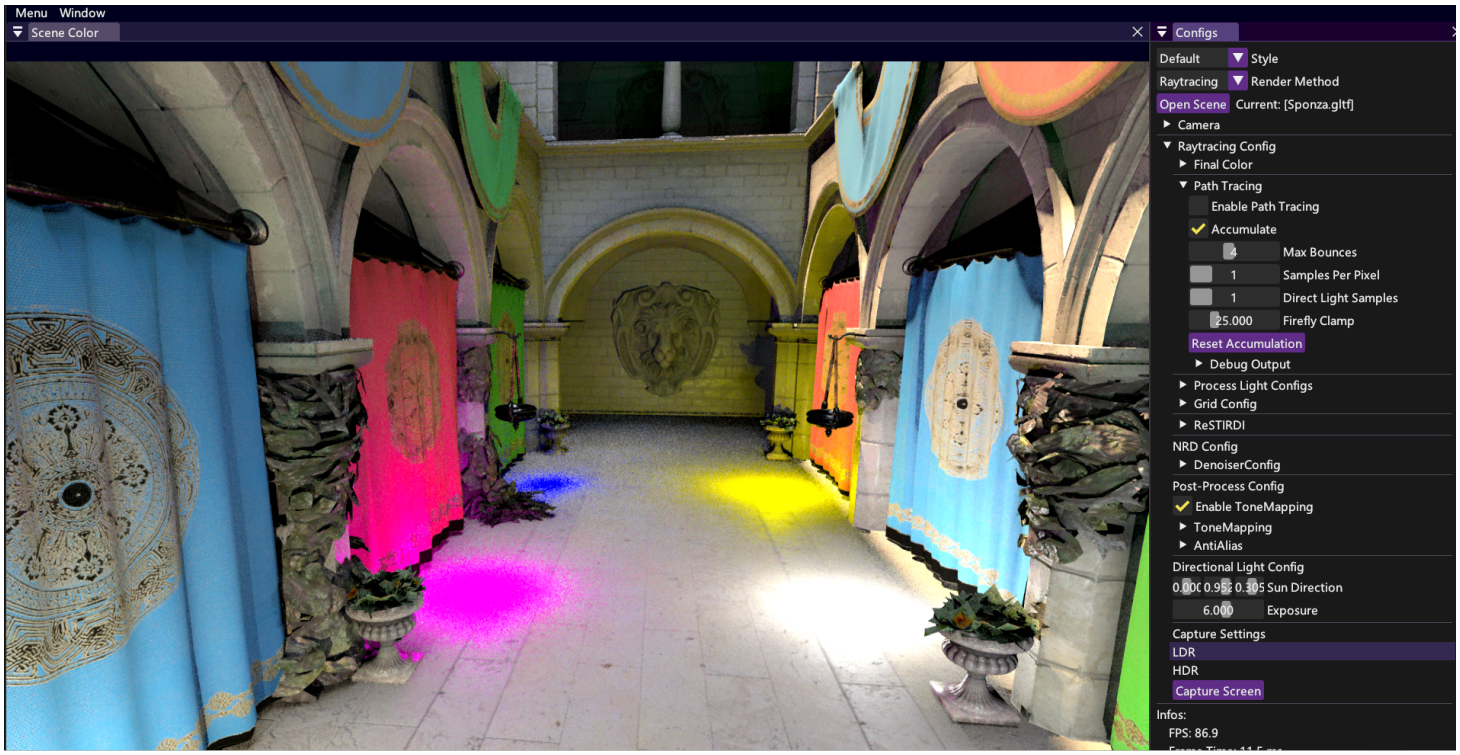
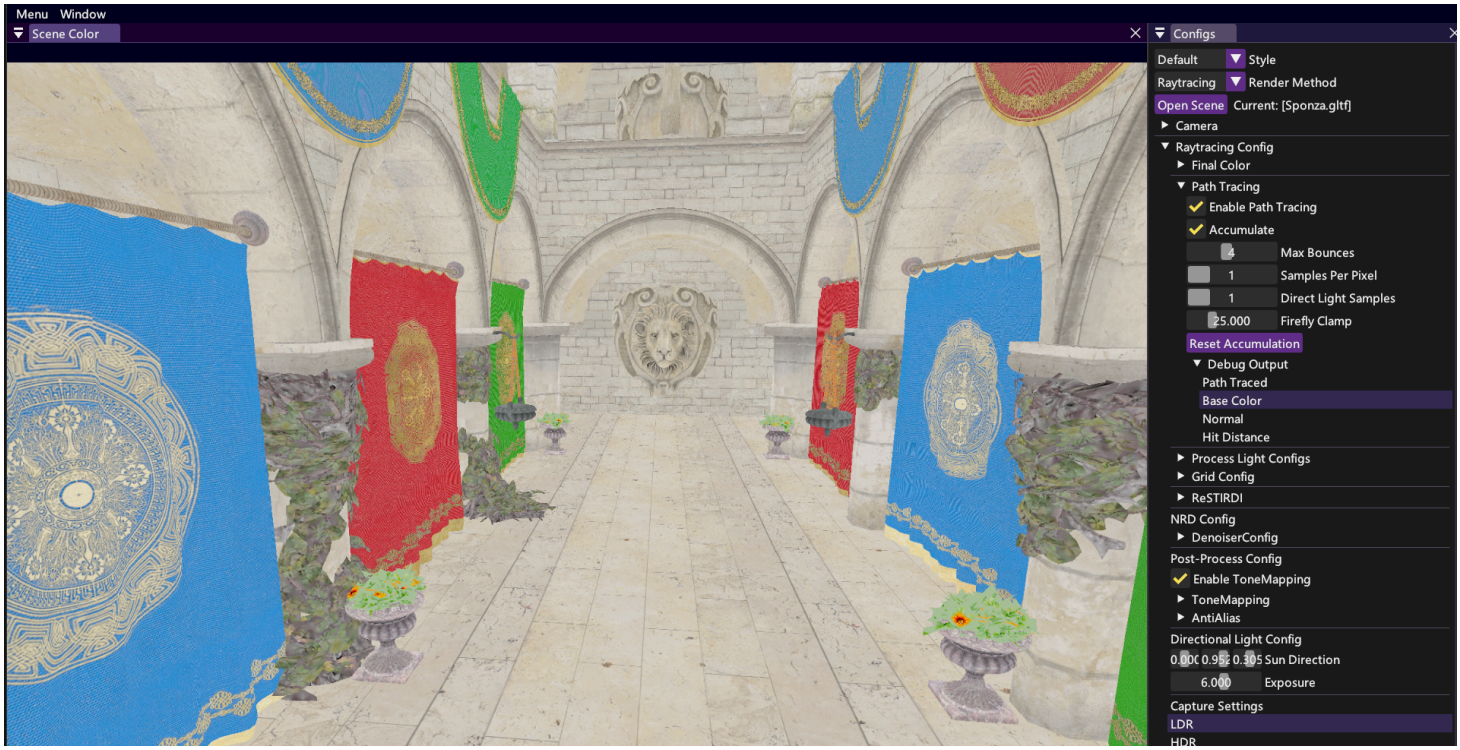
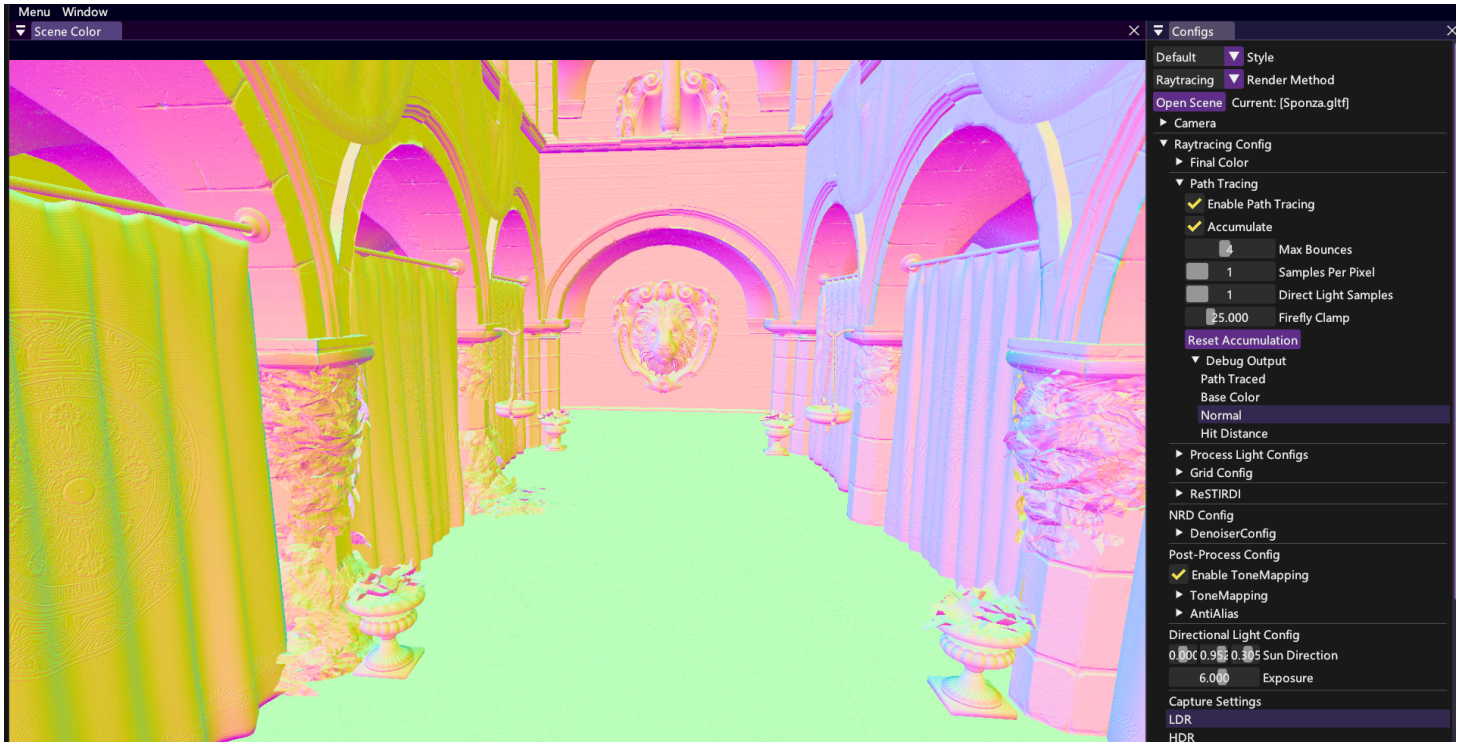


图 2：Path Tracing Debug - Base Color



说明：开启 Path Tracing，将 Debug Output 切到 `Base Color`。该图用于说明 primary ray 命中、几何读取和材质贴图采样正确。

图 3：Path Tracing Debug - Normal

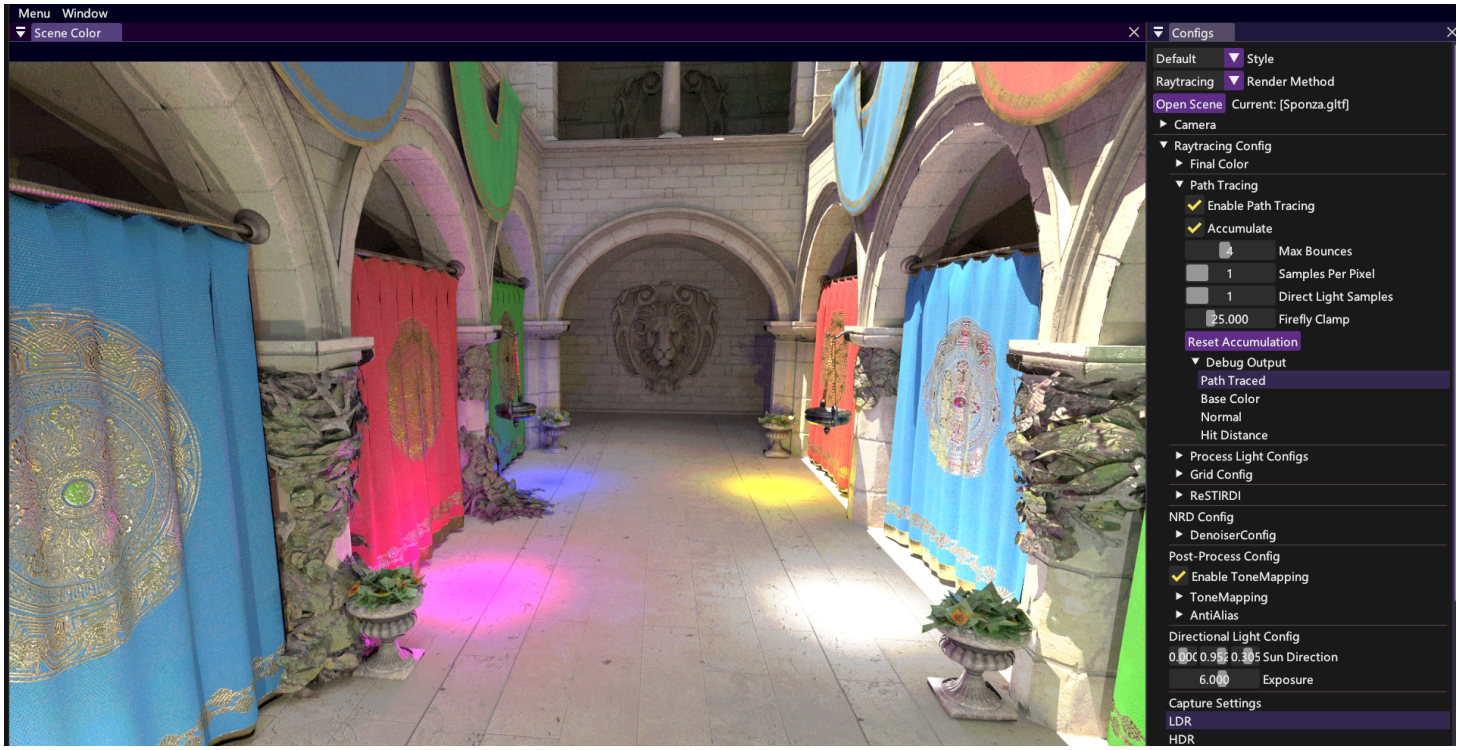


说明：展示法线读取与法线贴图应用结果。

图 4：Path Tracing 初始噪声

说明：开启 `Path Traced`，点击 `Reset Accumulation` 后立即截图。可看到 Monte Carlo 路径追踪的初始噪声。

图 5：Path Tracing 收敛结果



说明：保持相机静止 5 秒后截图。该图展示逐帧累积、Next Event Estimation 和 firefly clamp 后的低噪声结果。

图 6：Direct Light Sampling 对比

截图文件：

```
screenshots/06_direct_light_sampling_comparison.png
```

实验中遇到的问题

彩色光源恢复

初版 Path Tracing 只考虑环境光、方向光和材质自发光，没有读取 MoerEngine 场景中的 point lights。因此开启 Path Tracing 后，原来 ReSTIR DI 管线中的彩色光源会消失。

最终版本在 PathTracing 分支中复用了 `PrepareLightPass`，将场景光源写入 GPU light buffer，并在 shader 中用 `PolymorphicLight` 采样 local lights。因此彩色点光源重新进入路径追踪结果。

噪声收敛改善

初版 path tracer 主要依赖 BRDF 随机采样。如果光源小而亮，路径很难随机命中光源，偶发命中又会产生高亮样本，导致 firefly。最终版本加入 Next Event Estimation 后，每次表面命中都会主动采样光源并测试可见性，因此直接光照噪声明显降低。

Firefly clamp 进一步抑制了极端高亮样本，使长时间累积后的画面更稳定。

局限与后续工作

当前实现已经具备可展示的硬件路径追踪效果，但与工业级实时路径追踪仍有差距：

1. 直接光源采样目前是 uniform light sampling，还没有完整 MIS。
2. Firefly clamp 会引入偏差。
3. 未接入 NRD / OptiX AI Denoiser 等工业级降噪器。
4. 环境贴图重要性采样仍可进一步加强。
5. 对透明材质、折射材质、复杂 alpha material 的支持还较简化。
6. 性能优化仍有空间，例如分离直接光与间接光、加入时空重用或 ReSTIR GI。

后续可继续实现：

- Multiple Importance Sampling
- 更严格的 direct light / BSDF pdf 组合
- 环境光重要性采样
- NRD denoising 接入
- ReSTIR GI 或 path guiding
- 更完整的材质模型

总结

本项目在 MoerEngine 中新增了一个独立的硬件路径追踪管线，完成了从原实时光追管线到 Monte Carlo path tracing 的扩展。实现过程中复用了 MoerEngine 的 TLAS、RayQuery、bindless resource、材质系统和灯光预处理模块，并新增了路径追踪 shader、UI 控制、逐帧累积、Next Event Estimation 和 firefly clamp。

通过该项目，可以较系统地理解现代渲染器中 rasterization、实时 ray tracing、ReSTIR DI 和 path tracing 之间的关系，也能观察到工业级实时路径追踪中必须面对的关键问题：采样方差、光源重要性采样、历史累积、噪声抑制和交互式性能。