Final Project

图形绘制技术

Graphic Rendering Technology

Final Project

2026 春季学期

大作业要求

对于大作业我们只提供选题，具体的实现以及效果验证不做定性要求。最终你需要提交一份**PDF 格式的实验报告**，其中应当包括但不限于你对该选题的理解、分析、关键步骤的实现、最终的结果以及对本课程的总结和建议，如果你愿意也欢迎在报告中附上源码仓库地址（该要求不是必须项）。

以下选题分为离线渲染、实时渲染两部分，**你只需要挑选其中一个选题即可**。

离线渲染选题

如果你选择离线渲染选题，我们推荐在Moer上进行实现。

Photon Mapping

光子映射（Photon Mapping）从光源发射光线，并将光线与场景表面相交的信息以 **光子** 形式进行存储，而后通过估计 **着色点处光子的密度** 来求解渲染方程，该方式的估计值虽然会引入一些误差，但非常擅长渲染焦散等效果。

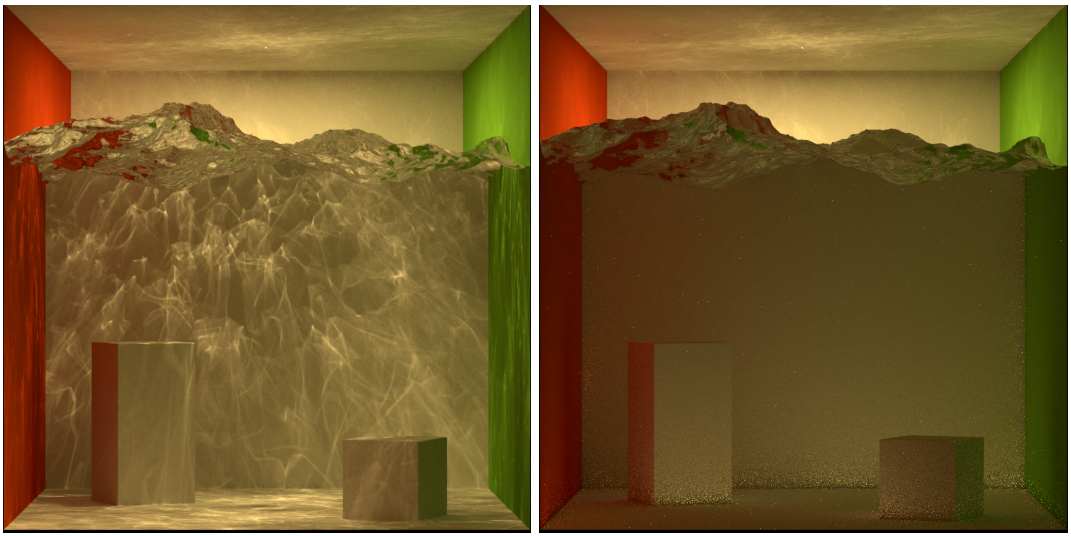

图 1: 随机渐进式光子映射（左），双向路径追踪（右）

实现该选题需要你理解 **如何通过密度估计求解 radiance**，以及**实现一个能够进行快速半径搜索/KNN 的数据结构**（KD-Tree 或 3D HashGrid），对于具体的技术选型我们不做要求，最终你应当能够使用光子映射类的算法渲染一张图像。

以下是可供参考的资料：

- PBRT 对随机渐进式光子映射（SPPM）的理论介绍 Stochastic Progressive Photon Mapping

- Realistic Image Synthesis Using Photon Mapping[3]
- Stochastic Progressive Photon Mapping [1]
- Progressive Photon Mapping [2]

Neural Radiance Cache

图 2: NRC 缓存可视化（左），Path Tracing 结果（右）

Neural Radiance Cache[6] 是一项使用神经网络对实时路径追踪加速的技术，不过其原理也并不复杂————利用一个神经网络去拟合**场景中任意一个点，沿某个出射方向的 radiance 值**即可（你可以通过一个**数据准备阶段**，利用 Path Tracing 来收集这些训练需要的数据）。最终你应当能够将在 Path Tracer 中启用 NRC 功能，在选择查询 cache 时终止路径（你可以在每根光线与场景的第一个交点处查询 cache 值并输出，得到一张如图 2 左侧的图来验证你的网络拟合效果）。

实现该选题需要你理解**如何使用一个神经网络拟合目标函数**，以及**在渲染器中实现/集成一个能够进行推理和训练的神经网络**。神经网络部分基于 CPU 或 GPU 均可。

以下是可供参考的资料：

- NVIDIA 官方基于现代图形 API 的 SDK RTXGI，不过其中的 NRC 实现部分并未开源，你可以参考该 SDK 来理解 NRC 的工作流程
- VkNRC 基于 Vulkan compute shader 实现的 nrc，如果你有意自己实现神经网络的推理和训练，你可以参考该项目如何利用分块矩阵乘法优化 GEMM
- 如何通过 Encoding[5, 4] 提升神经网络拟合一个函数的能力，基于 CUDA 的实现可以参考 TCNN
- 如果你不想自行实现神经网络的部分，那么基于 CPU 的集成推荐使用 libtorch（pytorch 的 c++ 版，重量级）或是MiniDNN（一个基于 Eigen 的 header only 库）；基于 GPU 的集成推荐TCNN

实时渲染选题

如果你选择实时渲染选题，我们推荐在 MoerEngine 上进行实现。

体积雾

体积雾的核心在于模拟介质中的光散射与吸收过程，从而在引擎中实现雾气、体积光、体积散射等效果。

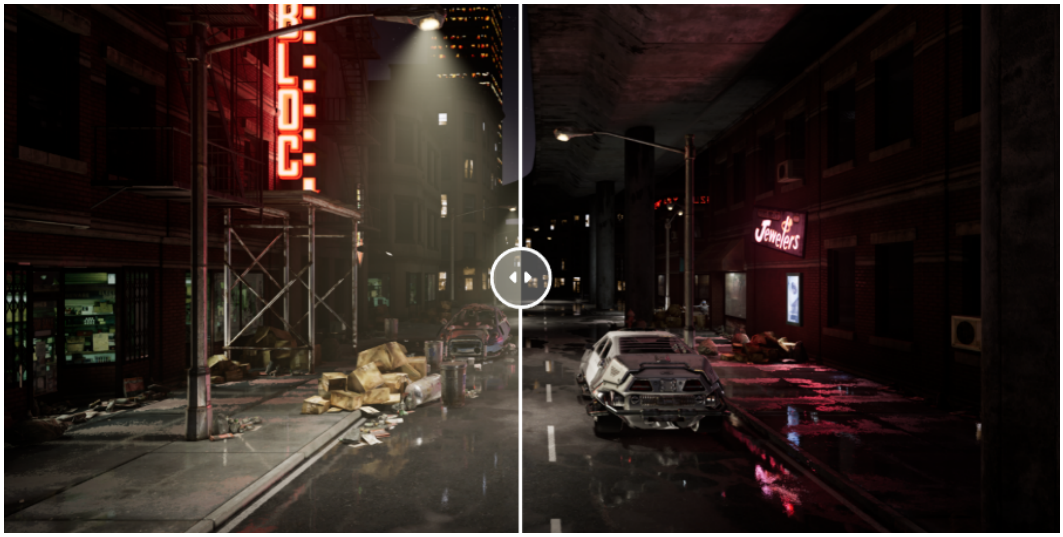

图 3: 启用体积雾 (左)，禁用体积雾 (右)

在实时渲染中，我们通常会通过体素化（Voxelized）和光线步进（Ray Marching）的方法来实现高性能的体积散射。

该选题要求至少实现体积雾对一种光源（如点光源或平行光）的散射。例如，实现图3左侧中，路灯投射出的雾蒙蒙的效果。

以下是可供参考的资料：

- 虚幻引擎体积雾介绍：虚幻引擎文档
- 虚幻引擎体积雾实现原理：体渲染探秘（二）基于 VoxelVolume 实现体积雾 - 知乎
- 在 Unity 中如何实现体积雾：Unity 实现体积雾 - 知乎
- 虚幻引擎源码：虚幻引擎源码下载

硬件路径追踪

在第一次作业中，我们主要接触了离线渲染的路径追踪算法。路径追踪算法的开销极其庞大，通常都需要几秒甚至几个小时才可以渲染出最拟真的图片。但现代 GPU 给我们提供了硬件光线追踪能力，以 3060 显卡为例，它可以让我们在一秒钟内发射 30-150 亿次光线，这使得实时光线追踪成为可能。

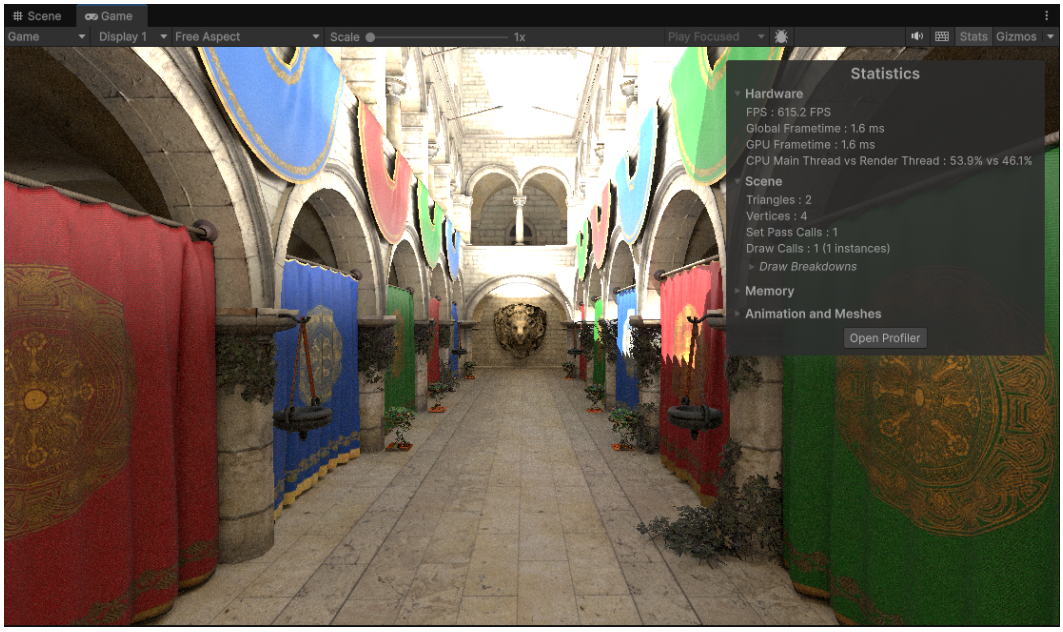

图 4: 基于 Unity SRP 实现的路径追踪器，4070 可以到达每秒 600 帧

MoerEngine 已经提供了实时光线追踪的底层工具链，在 RTAO 算法中也可以看到如何实现硬件光线追踪的实际例子。

该选题要求在 MoerEngine 中实现一个硬件路径追踪器。你可以选择独立搭建一套渲染器，或者基于光栅化渲染器（Raster Renderer）进行拓展来实现路径追踪。此外，你也可以尝试基于光栅化渲染器实现一个 **混合渲染**效果，即只使用光线追踪计算间接光照（光线的二次弹射）。

功能上，要求至少实现 **时域累计**：在不转动摄像机的情况下，渲染出噪点足够少的画面。

以下是推荐的参考代码或资料：

- 基于 Optix SDK 实现的路径追踪器：随 Optix SDK 安装
- 基于 DirectX Raytracing 实现的路径追踪器：链接
- 基于 Vulkan 实现的路径追踪器：链接
- NVIDIA 的 Vulkan 硬件光追教程：链接

注：Optix SDK、Unity SRP、MoerEngine RHI 都可以看做是渲染框架，而 DirectX 和 Vulkan 则是底层图形库。渲染框架包含了底层图形库的封装，所以渲染框架的学习成本会更低，更简单易懂，因此推荐优先参考 Optix SDK 的示例代码。

非真实感渲染

非真实感渲染，又称三渲二。它与基于物理渲染不同，它追求最终渲染效果的风格化、艺术化。非真实感渲染在二次元游戏、动漫电影中十分常见。

图 5: 2020 年公测的二次元游戏《原神》（左）；2024 年公测的二次元游戏《鸣潮》（右）

非真实感渲染和基于物理渲染不同，基于物理渲染的目标就是模拟真实世界，而非真实感渲染的目标由需求决定，不同场景的需求差别较大。

因此，该选题将需求限定在二次元角色渲染上。该选题具体需要至少完成以下功能：

- 导入一个二次元角色模型，并正确渲染角色。
- 二分色阶着色：将连续的光照强度通过阈值化映射为两个离散的色阶亮部和暗部，从而形成明暗分界。
- 描边：在角色外轮廓、场景分界处渲染风格化的轮廓线。

此外，该选题需要在如下 5 个风格化后处理效果中，选择至少 1 个实现（你也可以选择你自己喜欢的风格化后处理效果）：

- 将画面渲染为像素风格。参考资料：Unity 像素化教程
- 将画面渲染为半调或仿色图案。参考资料：从仿色图案到半调 - 知乎
- 给画面渲染为故障艺术风格。参考资料：Photoshop 故障风效果教程
- 将画面渲染为素描风格。
- 将画面渲染为水彩风格。

关于如何导入模型：在引擎的左上角菜单中，可以打开 SceneEditing、Hierarchy、Inspector 等子窗口。可以在其中导入模型，并调整导入模型的位置和朝向。此功能暂不稳定，如果在此步遇到 Bug，请将代码分支切换为 main 分支，或及时向助教反馈。

因此，如果你选择非真实感渲染选题，请尽早开始完成作业，避免遇到引擎 Bug、角色模型兼容性等需要耗费时间处理的意外情况。

以下是推荐的参考资料：

- 二次元艺术的渲染技术：链接

- 虚幻引擎终末地的角色渲染教程：链接
- 像素风格化教程：Unity 像素化教程
- 半调或仿色图案风格化教程：从仿色图案到半调 - 知乎
- 故障风格化教程：Photoshop 故障风效果教程

第一章　参考文献

[1] T. Hachisuka and H. W. Jensen. Stochastic progressive photon mapping. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–8. 2009.

[2] T. Hachisuka, S. Ogaki, and H. W. Jensen. Progressive photon mapping. In *ACM SIGGRAPH Asia 2008 papers*, pages 1–8. 2008.

[3] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., USA, 2009.

[4] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[5] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022.

[6] T. Müller, F. Rousselle, J. Novák, and A. Keller. Real-time neural radiance caching for path tracing. *arXiv preprint arXiv:2106.12372*, 2021.