大学生创新创业项目

2025年12月17日

基于图形引擎的科学计算可视化与真实感渲染探索

郑泽锋[1] 龚达宣[2] 张洛云[1] 钟卓能[2] 朱鑫盛[2]

[1] *中山大学软件工程学院*

[2] *中山大学土木工程学院*

摘要

为实现科学计算数据（如土木工程多物理场、颗粒介质等）的直观化呈现与高效交互，解决传统模型难以精准传达复杂数据特征及多专业协同低效的问题，本项目开展基于图形引擎的科学计算可视化与真实感渲染技术探索。现有国内外工具存在局限：国外软件（如 Paraview、Ovito）处理大规模数据时交互流畅度不足或硬件需求高，国内软件生态较弱，商业软件封闭性强且二次开发受限。

本项目提出核心解决方案：整合开源生态与 AI 技术，构建“OpenWebUI+Function Calling+Blender”自动化技术路线，通过 GraphRAG 构建领域知识库弥补通用大模型的专业认知缺口，以 JSON 中间态实现自然语言驱动的 BVTKNodes 节点树自动生成与渲染；自主研发基于 NVIDIA OptiX 9.0 的光线追踪渲染器，搭建物理真实感渲染管线，同时开发 SDF-DEM 与 OpenFOAM 耦合的 CFD-DEM 方法，支持任意形状颗粒数值模拟与 MPI 多线程并行计算。

项目已成功构建低成本、可扩展的 AI 科学可视化原型系统，实现简单几何体可视化工作流的自动化闭环，沉淀了图形学底层技术积累。通过典型场景测试（如基础网格显示、简单颜色映射），验证了技术路线的可行性，显著降低了科学计算可视化的操作门槛与算力成本，为土木、水利等领域的工程设计优化、风险评估提供了高效的技术支撑。

研究思路

AI 赋能与光线追踪：

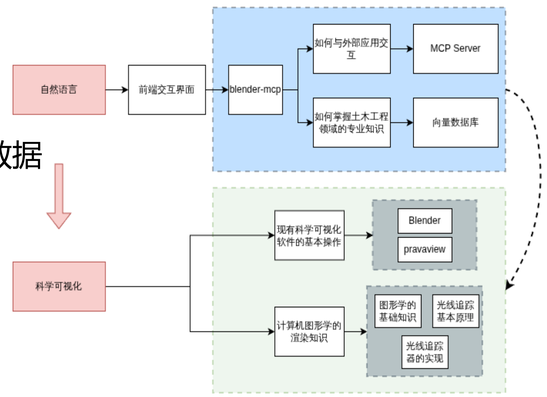
- 易用性：通过自然语言驱动科学可视化
- 高质量：从零构建基于光线追踪的渲染引擎

利用 AI 技术降低软件使用门槛：

- 自然语言交互：驱动大语言模型给出渲染方案
- 构建专业知识库：为大语言模型提供专业知识数据
- 自动化流程构建：自动化实现渲染方案

基于光线追踪技术实现高质量渲染：

- 系统性掌握图形学知识
- 逐步尝试实现光线追踪器
- 构建 CPU 版本光线追踪器
- 从 CPU 计算转向 GPU 计算



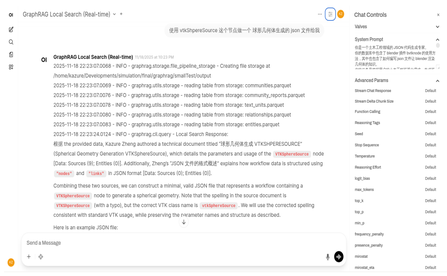
整体研究思路流程图

基于 OpenWebUI 的智能交互前端与知识增强系统

为了解决通用大模型不懂“科学可视化”的问题，我们构建了一个具备领域知识的智能前端。

轻量化 Web 前端部署：

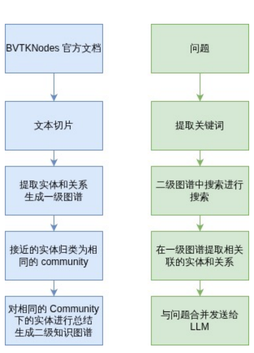
利用 OpenWebUI 搭建了基于浏览器的交互界面。通过Docker 容器化技术，将前端服务、后端推理服务以及数据库服务进行了统一封装，实现了“一键部署”，便于在实验室不同终端上进行测试。



轻量化web部署

GraphRAG 领域知识库构建：

引入 GraphRAG技术。我们将 BVTKNodes 的官方文档、节点定义规范以及 VTK 数据格式说明文档进行了结构化处理，构建了基于图谱的向量知识库。当用户输入“渲染一个流体速度场”时，系统会先检索知识库中关于“流体”、“速度矢量”对应的节点组合模板，再将其注入Prompt，显著提高了 AI 生成工作流的准确性。



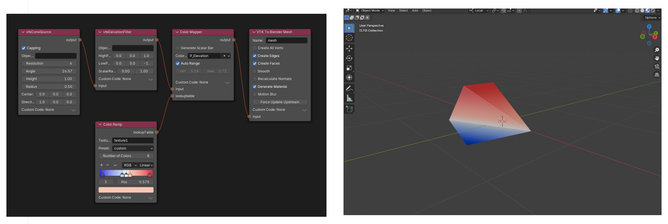
向量知识库构建流程图

可视化工作流

基于 Function Calling 与JSON 中间态的自动化工作流:

在 OpenWebUI 后端注册了自定义函数（Tools），核心功能是“保存工作流文件”。当 LLM 规划好渲染逻辑后，正确输出 JSON 文件之后，可由用户自行选择是否使用该渲染方案，一键提取代码并将其保存到服务器指定的“监听文件夹”中。

自动检测代码内容并保存



指定文件自动导入Blender

Blender 端的文件监听与自动加载插件

Blender 驻留插件的开发:

- 定时轮询机制：该插件件以 1 秒/次 的频率扫描“监听文件夹”。
- 动态解析不重构：一旦检测到新的 JSON 文件，插件立即读取内容，调用BVTKNodes 的API，在 Blender 场景中动态创建对应的节点树（Node Tree）。例如，自动创建 VTK Reader 节点读取数据，连接 Threshold 节点进行过滤，最后连接 BVTK to Mesh 节点生成几何体。
- 流程闭环：用户在网页端说话 - > AI 生成 JSON - > 文件保存 - > Blender 自动捕捉并刷新场景。这一流程完全屏蔽了复杂的节点操作。

```
win = bpy.context.window
if not win:
    raise RuntimeError("No active window context available")
area = None
region = None
for a in win.screen.areas:
    if a.type == "NODE_EDITOR":
        area = a
        region = next((r for r in a.regions if r.type == "WINDOW"), None)
        break
if area is None or region is None:
    raise RuntimeError(
        "No NODE_EDITOR area found. Open a Node Editor and try again."
    )

with bpy.context.temp_override(window=win, area=area, region=region):
    bpy.ops.node.bvtk_node_tree_import(filepath=path, confirm=True)
```

```
def scan_once(path: str) -> float:
    ensure_dirs()
    try:
        for name in list(os.listdir(path)):
            src = os.path.join(path, name)
            if not name.lower().endswith(".json"):
                shutil.move(src, os.path.join(FAILED, name))
                continue
            try:
                import_bvtk_json(src)
                shutil.move(src, os.path.join(PROCESSED, name))
            except Exception:
                traceback.print_exc()
                shutil.move(src, os.path.join(FAILED, name))
    finally:
        return 1.0

@persistent
def register_timer(dummy=None):
    ensure_dirs()

    bpy.app.timers.register(
        lambda: scan_once(INBOX), first_interval=2.0, persistent=True
    )
```

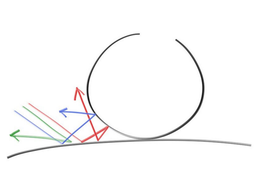
Blender驻留插件代码

基于 NVIDIA OptiX 的真实感渲染引擎底层探索

为了弥补仅使用 Blender 现成引擎可能带来的底层原理认知缺失，项目组同步开展了基于 NVIDIA OptiX 9.0 的渲染器开发，作为图形学基础研究的练手项目

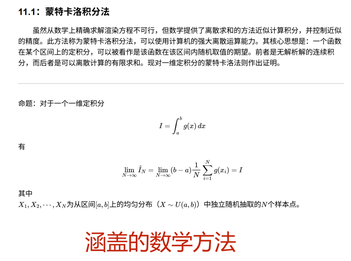
光线追踪管线搭建：

- 深入研究了 GPU 渲染管线，使用 CUDA C++ 编写了 Ray Generation（光线生成）、Closest Hit（最近命中）、Miss（未命中）等核心着色器程序。
- 实现了基于 SBT (Shader Binding Table) 的高效数据绑定，理解了现代光线追踪硬件加速的底层逻辑。



光线追踪渲染基本原理

学习过程



11.1: 蒙特卡洛积分法

虽然从数学上精确求解渲染方程不可行，但数学提供了离散求和的方法近似计算积分，并控制近似的精度。此方法称为蒙特卡洛积分法，可以使用计算机的强大离散运算能力。其核心思想是：一个函数在某个区间上的定积分，可以被看作是该函数在该区间内随机取值的期望。前者是无解析解的连续积分，而后者是可以离散计算的有限求和。现对一维定积分的蒙特卡洛法则作出证明。

命题：对于一个一维定积分

$$I = \int_a^b g(x)\,dx$$

有

$$\lim_{N\to\infty} \hat{I}_N = \lim_{N\to\infty} (b-a)\frac{1}{N}\sum_{i=1}^{N} g(x_i) = I$$

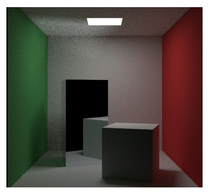
其中 $X_1, X_2, \cdots, X_N$ 为从区间 $[a,b]$ 上的均匀分布（$X \sim U(a,b)$）中独立随机抽取的 N 个样本点。

涵盖的数学方法

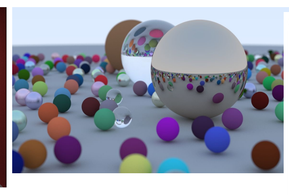
科学数据渲染尝试：

探索了如何将简单的几何数据（如球体、三角形）直接传入显存构建 IAS (Instance Acceleration Structure) 和 GAS (Geometry Acceleration Structure)。虽然该引擎功能尚简陋，但通过亲手实现 Fresnel 反射、Lambert 漫反射等物理模型，团队成员对“真实感渲染”的物理本质（积分方程求览）有了深刻理解，反向指导了 Blender 中材质参数的调优

(a)构建CPU光线追踪渲染器



不同形状



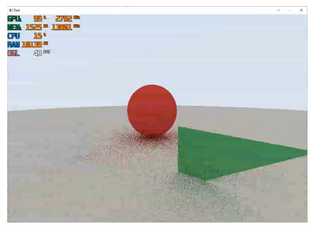
不同材质

```
Color3 rayColor(Camera & cam, const Ray & ray, size_t sampleIndex,
                const BVHTree::BVHTreeNode * tree, const std::pair<PrimitiveType, size_t> * indexArray,
                const Sphere * spheres,
                const Triangle * triangles,
                const Parallelogram * parallelograms,
                const Transform * transforms,
                const Box * boxes,
                const Rough * roughMaterials, const Metal * metalMaterials,
                const DiffuseLight * lightMaterials, const Dielectric * dielectricMaterials,
                const Sphere * hittablePDFSphere, size_t hittablePDFSphereCount,
                const Parallelogram * hittablePDFParallelogram, size_t hittablePDFParallelogramCount)
```

部分实现代码

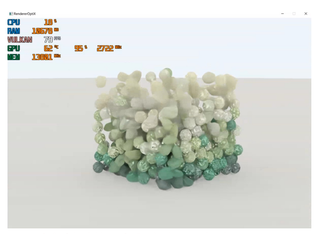
(b)构建2个版本的GPU光线追踪渲染器

1.两层加速结构（实例 + 几何）
2.静态场景60FPS+实时交互
- CUDA内存应用：全局 + 常量 + 页面锁定
- CUDA同步机制：实现双缓冲
- GPU渲染同时，CPU更新加速结构



渲染模型实例（静态）

1.使用NVIDIA OptiX光线追踪API重构渲染器
2.利用RT Core的强大能力，支持动态场景实时交互
3.集成OptiX AI Denoiser，实现实时降噪
4.使用VTK C++库读取场景数据，实现科学可视化
5.使用现代图形API（Vulkan、Direct3D12）呈现

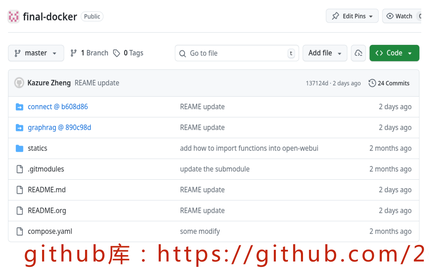


渲染模型实例（动态）

研究小结

项目收获：

1.熟悉了一个项目的完整开发流程
2.熟悉软件项目开发工作流程及技术
3.初步了解了 AIGC 前沿技术
4.深入学习和掌握了图形学知识



github库：https://github.com/2025-simulation/final-docker

改进方向：

- **集成高级光线追踪算法：**Physically Based Rendering
- **GPU应用：**大规模场景的流式加载和分块渲染
- **利用图形API：**DirectX Ray-Tracing、Vulkan Ray-Tracing
- **AI应用：**自动调节渲染参数、通过自然语言生成渲染配置文件

→ 科学可视化

- **渲染器GUI制作：**
- 内置GUI（Dear ImGUI）或外置配置界面（Qt，Electron…）
- **集成交互功能：**对象选取，切片，等值面…
- **混合渲染：**光栅化搭建场景骨架+光线追踪呈现完整场景

→ 完善技术产品