

# 计算机图形学实验作业报告

222222 张三

效果演示见项目中 `效果演示.mp4` 视频。

## 1. 使用手册

本项目采用 **C++ + OpenGL (GLFW + GLAD)** 实现一个带有完整光照、材质、阴影和多几何物体的三维 Cornell Box 场景。工程结构清晰，可在 Windows 与 Linux 下通过 CMake 构建。

### 1.1 运行环境

- C++17
- OpenGL 4.1 Core Profile
- GLFW3 (窗口和输入)
- GLAD (OpenGL 函数加载)
- stb\_image (贴图加载)
- CMake  $\geq 3.10$

### 1.3 编译方式

#### 1.2.1 windows

使用 [minGW](#) 和 [Clang](#) 进行编译，步骤如下

1. 确保已将 MinGW 和 Clang 的 bin 目录添加到系统环境变量 Path 中（避免编译时找不到工具链）。
2. 打开 PowerShell，执行以下命令启动构建：

```
cd cg_project
.\build_windows.bat
```

## 1.4 项目结构

```
cg_project
├─ build_windows.bat - windows 编译脚本
├─ CMakeLists.txt - CMake 配置文件
├─ image - 截图
│   └─ cornell_box.png
├─ include - 头文件
│   ├── glad
│   ├── GLFW
│   ├── glm
│   ├── KHR
│   └─ stb_image.h
├─ lib - 库文件
│   └─ libglfw3.a
├─ readme.md
├─ shader - shader 文件
│   ├── default_fragment_shader.fs - 默认片段着色器
│   ├── default_vertex_shader.vs - 默认顶点着色器
│   ├── normal_fragment_shader.fs - 法线片段着色器
│   ├── normal_vertex_shader.vs - 法线顶点着色器
│   ├── phone_fragment_shader.fs - phone片段着色器，带阴影
│   ├── phone_vertex_shader.vs - phone顶点着色器，带阴影
│   ├── point_shadow_fragment_shader.fs - 点光源阴影片段着色器
│   ├── point_shadow_vertex_shader.vs - 点光源阴影顶点着色器
│   ├── shadow_fragment_shader.fs - 平行光阴影片段着色器
│   └─ shadow_vertex_shader.vs - 平行光阴影顶点着色器
├─ src
│   ├── Camera.cpp
│   ├── Camera.h - 相机实现
│   ├── glad.c
│   ├── light - 光源模型
│   │   ├── DirectionalLight.h - 平行光抽象基类
│   │   ├── Light.h - 光源抽象基类
│   │   └─ PointLight.h - 点光源抽象基类
│   ├── main.cpp
│   ├── material - 各种各样的材质
│   │   ├── Material.h
│   │   ├── PhoneMaterial.cpp
│   │   ├── PhoneMaterial.h - phone材质
│   │   ├── PureColorMaterial.cpp
│   │   ├── PureColorMaterial.h - 纯色材质
│   │   ├── TexturedPhoneMaterial.cpp
│   │   └─ TexturedPhoneMaterial.h - 带纹理的phone材质
│   ├── object - 各种几何体
│   │   ├── Cone.cpp
│   │   ├── Cone.h - 圆锥体
│   │   ├── Cube.cpp
│   │   ├── Cube.h - 立方体
│   │   └─ Cylinder.cpp
```

```
| | |─ Cylinder.h - 圆柱体
| | |─ Object.h - 几何体抽象基类
| | |─ Plane.cpp
| | |─ Plane.h - 平面
| | |─ Sphere.cpp
| | |─ Sphere.h - 球体
| |─ Shader.cpp
| |─ Shader.h - shader 编译与 uniform 设置
| |─ stb_image_impl.cpp
| |─ texture - 纹理
| | |─ Texture.cpp
| | |─ Texture.h
| |─ tool - 工具类
| | |─ Line.cpp
| | |─ Line.h
| | |─ Point.cpp
| | |─ Point.h
|─ texture - 贴图
| |─ brick.jpeg
| |─ earth.jpg
| |─ floor.jpeg
| |─ painting.jpg
| |─ wall.jpg
| |─ wood.jpg
|─ 效果演示.mp4 效果演示视频
|─ 实验报告.md
```

## 1.4 相机控制方式

程序内置类似 FPS 的自由相机：

- **W / S**：前进 / 后退
- **A / D**：左移 / 右移
- **鼠标移动**：调整视角（yaw/pitch）
- **鼠标滚轮**：缩放视场角（FOV）
- **Esc**：退出程序

## 1.5 场景内容概述

程序运行之后，可以看到一个由多个平面围成的景箱（**Cornell Box**），内部布置了：

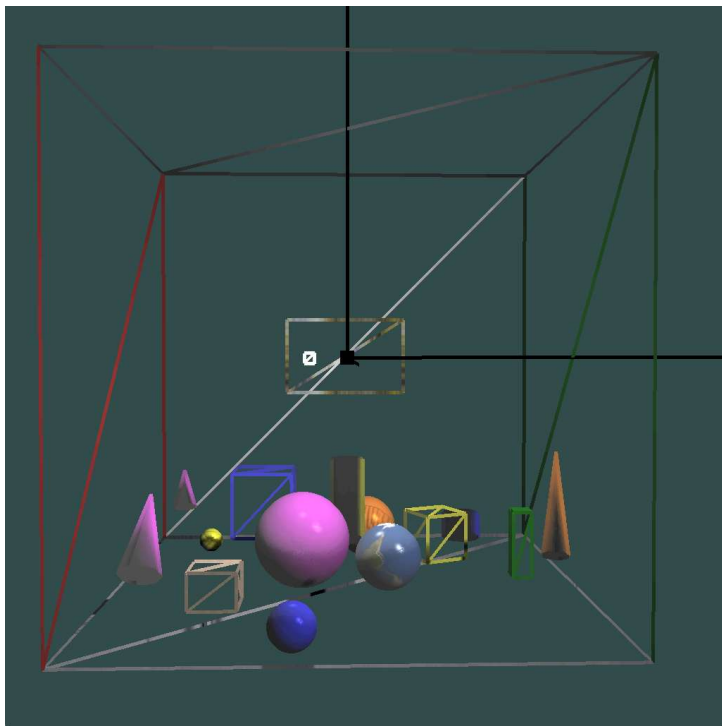


- 左边的红墙和右边的绿墙
- 对面墙面的画
- 具有材质贴图的墙和地板
- 不同材质和贴图的立方体、球体、圆柱、圆锥
- 动态旋转的点光源(图中白色立方体)与平行光源(方向为  $\text{vec3}(-1 - 1 - 1)$ )
- PCF 软阴影
- 使用 Phong 光照模型的高光、漫反射与环境光

整体效果接近一个可交互的三维室内展示空间。

## 1.6 debug mode

在 `main.cpp` 中，通过 `bool debugMode = true;` 可以开启 debug 模式，此时会渲染物体的线框，场景的原点以及三个坐标轴，便于调试。



## 2. 设计思路

本实验目标是：

构建一个具有真实感的三维场景，包含至少两个三维物体，具备光照、阴影、纹理、交互，并能进行实时漫游。

为了达到真实感，我采用了“康奈尔盒 + 多物体 + 软阴影 + 纹理材质 + 点光源阴影 + 平行光阴影”的结构化方案。思路可分为以下部分：

### 2.1 模块化架构

整个程序按照“渲染引擎基本组件”进行模块拆分：

- **Shader**：封装 OpenGL shader 的编译 / 链接 / uniform 设置
- **Camera**：实现 FPS 相机，支持鼠标与键盘操作
- **Light**：DirectionalLight + PointLight，两种光源模型
- **Material**：Phong 材质、纯色材质、纹理 Phong 材质
- **Object**：抽象基类 + Cube、Sphere、Cylinder、Cone、Plane 子类
- **Texture**：封装 stb\_image 纹理加载
- **Shadow**：平行光 ShadowMap + 点光源 ShadowCubeMap

这种拆分方式让场景构建像“搭积木”一样，便于扩展、调试与理解。

### 2.2 场景构建策略

我选择构造一个**扩展版康奈尔盒**：

- 地面、墙体全为 Plane 实体
- 后墙附加一副“贴图画框”

- 场景内部布置超过 10 个不规则体：球体、圆锥、圆柱、立方体等
- 特别加入一个 **贴地球纹理的球体和砖块纹理的立方体**，作为纹理示例
- 物体摆放设计遵循：
  - 分布均匀
  - 高度层次不一
  - 尽量避免穿插与相交

此整体效果类似一个小型艺术展览空间。

## 2.3 光照设计

光照由两部分组成：

### 1. 定向光（模拟太阳光）

- 方向为  $\text{vec3}(-1 - 1 - 1)$
- 用于生成大方向阴影
- 使用正交投影生成 ShadowMap

### 2. 点光源（可旋转）

- 提供动态变化的局部光照，绕 y 轴旋转
- 使用立方体深度贴图生成 point shadow（6 次渲染）

为使阴影更柔和，我实现了基础 PCF（percentage-closer filtering）的阴影过滤，使阴影过渡更自然。

## 3. 实现

### 3.1 渲染流程

程序的主渲染循环分三步：

#### (1) 渲染平行光阴影贴图

- 使用深度着色器（shadow shader）
- 将所有物体渲染到 1024×1024 depth map
- 使用 lightSpaceMatrix 完成从世界到光源空间的变换

#### (2) 渲染点光源立方体阴影贴图

- 每帧渲染六次方向视图（+X, -X, +Y, -Y, +Z, -Z）
- 深度写入 2048×2048 cube map
- 使用 90° FOV 正投影

#### (3) 正常渲染场景（带阴影）

所有物体按如下流程渲染：

- 应用 Model / View / Projection 矩阵
- 应用材质（Phong 或 TexturedPhong）

- 设置光源参数
- 从阴影图中查询是否被遮挡
- 输出最终颜色

## 3.2 主要模块说明（基于你的源码结构）

### （1）Shader 模块

- 封装 shader 的加载、编译、错误检测
- 支持 `setVec3` / `setMat4` / `setInt` / `setFloat` 等 uniform 传递

### （2）Camera 模块

实现：

- FPS 风格相机
- 鼠标控制 yaw/pitch
- 滚轮控制 zoom

### （3）Object 模块

抽象类提供：

- 顶点/索引缓存
- shader 持有指针
- model matrix

子类 Cube、Sphere、Cylinder、Cone、Plane 通过 VAO/VBO 构建几何。

### （4）Light 模块

- DirectionalLight：方向、颜色、环境/漫反射/镜面属性
- PointLight：位置、颜色、衰减因子与阴影 cube map

### （5）Material 模块

- PhongMaterial：经典 Phong 光照公式
- TexturedPhongMaterial：在漫反射中加入纹理采样
- PureColorMaterial：单色材质，适合 debug

### （6）阴影系统

- 使用深度贴图（shadow map）计算遮挡
- 采样时加入 PCF 模糊，使阴影边缘更柔和

## 3.3 场景构建

`main.cpp` 中集中创建物体、设置坐标、缩放、旋转，并装入对象列表。存在基于平行光和点光源的阴影，提高真实感。

## 4. 小结

本次实验从零构建了一个完整的 **OpenGL 三维渲染系统**，涵盖了：

- 几何建模（Cube / Sphere / Cone / Plane / Cylinder）
- 材质系统（Phong、纹理）
- 光照系统（平行光 + 点光源）
- 阴影映射（Shadow Map + Shadow Cube Map）
- PCF 软阴影
- 三维交互式相机
- 贴图与纹理采样
- 康奈尔盒场景搭建与多物体分布设计

整个实验开发过程，让我完整理解了一个实时渲染管线如何从“几何 → 变换 → 光照 → 材质 → 阴影 → 输出”逐步构成，也加深了我对 OpenGL 底层原理与 shader 思维方式的掌握。

如果进一步扩展，我认为可以加入：

- 全局光照算法
- PBR 材质
- 面光源
- PCSS 软阴影

本次项目是一个完整而扎实的 3D 渲染练习，使我对计算机图形学渲染流程有了系统而深入的理解。