PPT制作+汇报（5）	实验报告撰写（20）	创意+技术创新性（15）	实验内容完成度（60）			小组平均得分
			模型创建（20）	光照与材质（20）	相机控制及用户交互（20）	
5	12	7	20	20	20	84

实验报告较为详细，但较为缺乏创新性

北京交通大學

BEIJING JIAOTONG UNIVERSITY

计算机图形学第二次
实验报告

组　　号：6

小组成员：于蕊宁 刘荧 王美靖 冯文涵 曹雯佳 张悦

指导老师：吴雨婷

目 录

1. 实验内容总结

- 已达成要求:
  - ü 三维场景绘制
  - ü 基础光照
  - ü 纹理映射
- 项目创新:
  - ü 模型制作与修改
  - ü 阴影映射
  - ü 移动光源
  - ü 键鼠交互
  - ü 不同材质光照效果

2. 实验目的

- 学习和理解纹理映射，尝试为场景和模型创建纹理，丰富场景画面。
- 学习和理解透视投影，尝试将场景渲染到屏幕空间内输出。
- 了解和学习立方体贴图，尝试使用立方体贴图创建天空盒作为三维背景。
- 学习 Assimp 模型加载库，尝试通过 assimp 导入模型，丰富场景内容。
- 学习和理解 Phong 光照模型，尝试给场景模型添加光照。
- 学习和理解帧缓冲，尝试创建帧缓冲对象并渲染缓冲对象附件。
- 学习和理解键鼠控制，尝试使用键盘鼠标控制相机及物体的位置与方向。
- 学习创建简单菜单 UI，尝试使用 UI 控件控制场景效果。
- 学习和理解光线追踪算法，尝试使用光线追踪算法构建场景。

通过以上的学习内容，完成自选 3D 场景的绘制，包括简单光照、纹理映射和透视投影效果，并在此基础上进行探索创新，还实现了基于光线追踪的小球场景。

3. 实验原理

3.1 模型加载

模型导入是将不同格式的模型文件转换为可供 OpenGL 读取并理解的数据格式，在模型导入过程中我们使用了 Assimp 库来将所有的模型数据加载至 Assimp 的通用数据结构中。这样无论什么模型格式，只要 Assimp 支持，最终都会转换为 Assimp 通用的数据格式，我们只需要编写相应的实体类，将 Assimp 数据格式解析为 OpenGL 能够理解的顶点、法线、网格等元素，再封装一些必要的函数，即可实现模型的加载。

3.2 纹理映射

纹理映射是将纹理空间中的纹理像素映射到屏幕空间像素的过程。纹理生成过程的实质是将所定义的纹理，映射为某个三维物体表面的属性，并参与后续的光照计算。实现纹理映射主要是建立纹理空间与模型空间、模型空间与屏幕空间之间的映射关系。

纹理贴图是使用图像、函数或其他数据源来改变物体表面外观。与纹理贴图类似，法线贴图是通过 RGB 通道来标记法线方向，从而在光滑的表面上形成凹凸不平的视觉效果。

3.3 透视投影

透视投影是投影变换中的一种，也是创建具有真实透视效果的三维场景必不可少的一个环节。透视投影使用一个透视投影矩阵实现，本实验中的实现方式使用 glm 库的 glm::perspective 函数来生成透视投影矩阵。我们需要定义相机的视野、宽高比、远近裁剪面这些参数，最终生成的透视投影矩阵在渲染循环中传入着色器，对顶点坐标进行变换，即可实现透视投影的过程。

3.4 光照模型

使用 phong 光照模型给场景添加光照，phong 光照模型将物体的光线分为三大类，分别是：环境光 ambient，漫反射光 diffuse，镜面高光 specular

环境光 Ambient，Ka 是环境光系数，由物体的材质决定。环境光照系数的设置通常是一个介于 0 到 1 之间的值，表示环境光在整个场景中的强度。较小

的系数会使环境光照的贡献较弱，而较大的系数会增加环境光照的贡献。

$$Ambient=Ka$$

漫反射光 Diffuse，θ 表示入射光的反方向和法线的夹角，漫反射和光源角度，物体法向量有关。漫反射光需要考虑光源和物体的位置关系。光线直射物体的时候，反射的光最多，而光线平视物体的时候，我们几乎无法接收到反射光。漫反射的系数 Kd 可以用来控制光源的强度对漫反射光的影响程度。较大的系数会增强漫反射光的亮度，而较小的系数会减弱漫反射光的亮度。光滑的表面可能会反射更强的漫反射光，而粗糙的表面可能会减弱漫反射光的强度。

$$Diffuse=\cos\theta * K_d$$

镜面反射 Specular，镜面反射和视线方向，光源角度有关。η 表示反射光线的反方向和视线之间的夹角。

$$Specular=\cos\eta * K_s$$

3.5 缓冲帧

缓冲帧是管理显存的方式，一个帧缓冲是由一系列“画纸”组成的，这些画纸叫做附件 attachment。这些附件通常都是纹理。在 OpenGL 中，纹理既可以被当作被写入的对象，也可以被当作读的对象。一个帧缓冲可以有多个颜色附件和一个深度附件，但是默认最终只有 0 号帧缓冲会被输出到屏幕。

3.6 键鼠交互

相机方向控制主要通过键盘和鼠标实现，当按下键盘键时，可以切换相机的自由旋转状态或固定旋转状态；当鼠标移动时，可以根据当前状态计算相机的旋转角度。相机/物体位置控制主要通过键盘实现，可以通过按下不同的键来控制相机的移动方向，从而改变坐标。

3.7 菜单控制

在创建 OpenGL 上下文后，需要创建 ImGui 上下文，并进行窗口输入、主题色等初始化设置。在场景循环渲染的回调函数中，可以新建 ImGui 窗口 帧，并进行页面的设置，最后进行窗口渲染。通过 ImGui 的控件，实现一些交互操作，如模型加载控制等。

4. 实验内容

4.1 基础场景搭建

本三维场景以生活中的家为设计来源，致力于营造出一种舒适的氛围。细节处的装饰和绿植增加了生活的气息，让人一进门就感受到家的温暖与美好。

4.1.1 场景基本介绍

场景构成元素：场景由客厅、餐厅、卧室等基本生活空间组成，其中放置沙发、茶几、电视柜、床、餐桌椅、书桌、置物架、展示柜、装饰画、落地灯等常见元素。

色彩搭配：主色调选择黄色和米白色两种颜色，点缀色用鲜艳的颜色如绿色、粉色，通过靠垫、装饰品和植物点缀，活跃整体氛围。

光影效果：通过方向光模拟阳光，并设置合适的光照方向和强度。使用环境光和全局光照实现场景开关灯。

4.1.2 模型加载

模型构建部分，我们在model.h中，定义VertexData结构体，存储顶点的位置、纹理坐标和法线信息，并声明LoadObjModel函数，从OBJ文件加载这些顶点数据。顶点数据包括以下几项：顶点位置：一个三维向量，表示顶点在空间中的位置。纹理坐标：一对值，表示顶点在纹理图像中的位置，用于渲染过程中贴图。法线：一个三维向量，表示表面某点的切面方向，用于计算光照和阴影。

```
/*
    存储顶点信息的结构体
    float position   顶点位置信息 v
    float texcoord   顶点纹理坐标 vt
    float normal     顶点法线信息 vn
*/
struct VertexData
{
    float position[3];  // 顶点位置信息
    float texcoord[2];  // 顶点纹理坐标
    float normal[3];    // 顶点法线信息
};

struct ModelData
{
    std::vector<VertexData> vertices;
    std::vector<unsigned int> indexes;
};
```

在model.cpp中，声明了LoadObjModel函数，首先将OBJ文件的内容读入内存。接着，使用两个结构体暂存和定义顶点数据。最后，解析OBJ文件的内容，创建最终的顶点数组，以构建顶点缓冲区（VBO）。

```
VertexData*LoadObjModel(const char* filePath, unsigned int **indexes, int&vertexCount, int&indexCount)
{
    //获取文本内容
    char*fileContent = LoadFileContent(filePath);
    if (fileContent!=nullptr)
    {
        //坐标
        struct VertexInfo
        {
            float v[3];
        }; //等同于使用glm库的 glm::vec3

        //表示该面片的第i个顶点的 位置索引/纹理坐标索引/法向量索引
        struct VertexDefine
        {
            int positionIndex;
            int texcoordIndex;
            int normalIndex;
        };
```

顶点缓冲区可以被图形管线中的多个阶段使用，例如顶点着色器阶段，用于进一步处理顶点数据，如变换、光照和纹理映射，最终渲染出三维模型。

4.2 纹理映射

纹理映射部分，我们首先进行纹理采样器的绑定，然后由顶点着色器提供纹理坐标并传递到片段着色器，接着进行纹理采样，最后与光照值结合计算出最终像素颜色。

4.2.1 纹理采样器的绑定

着色器中声明了两个纹理采样器，分别绑定到不同的纹理单元：

U_MainTexture：主纹理，用于给物体表面提供颜色。

U_ShadowMap：阴影贴图，用于阴影判断。

```
//binding的作用：直接在着色器gls1中绑定纹理单元！！！
layout (binding = 0) uniform sampler2D U_MainTexture; //纹理采样器 通过CPU中roomTexture传入
layout (binding = 1) uniform sampler2D U_ShadowMap; //通过CPU中shadowMap传入
```

4.2.2 纹理坐标

顶点着色器：我们从顶点属性中直接读取了坐标，并计算了每个顶点的纹理坐标。计算纹理坐标时，使用了一些变换，如纹理坐标平移（movement）、缩放（scaling）和旋转（rotation），以确保纹理在模型表面上正确地铺展。

顶点属性传输：计算纹理坐标后，通过顶点缓冲区对象（VBO）和顶点属性缓冲（VAO）来实现将它们作为顶点属性传递到片段着色器。

片段着色器：片段着色器使用传递过来的纹理坐标来采样纹理图像。

```
layout (location = 0) in vec3 pos; //顶点坐标
layout (location = 1) in vec2 texcoord; //纹理坐标
layout (location = 2) in vec3 normal; //法线
```

4.2.3 采样纹理

在片段着色器中使用纹理坐标和纹理采样器获取纹理上对应位置的颜色值，与计算出的光照值结合，生成最终的像素颜色。

```
//diffuse
vec4 DiffuseLightColor=vec4(2.0,2.0,2.0,1.0);
vec4 DiffuseMaterial=vec4(0.8,0.8,0.8,1.0);
vec4 diffuseColor=DiffuseLightColor*DiffuseMaterial*max(0.0,dot(L,n));
```

4.2.4 结合光照

获取到的纹理颜色值会与片段着色器中计算的光照值结合:包括环境光（Ambient）、漫反射光（Diffuse）和镜面反射光（Specular）三个分量。

纹理颜色与计算出的光照效果结合，包括环境光、漫反射光和镜面反射光，以及阴影的影响。光照与纹理颜色相乘，产生最终的片段颜色。

```
if(lightOn == 1) //灯打开
{
    color = ambientColor+(diffuseColor+specularColor)*texture2D(U_MainTexture, V_Texcoord);
    color = color*vec4(vec3(1.0-CalculateShadow()),1.0);
}
```

4.3 光照和阴影的实现

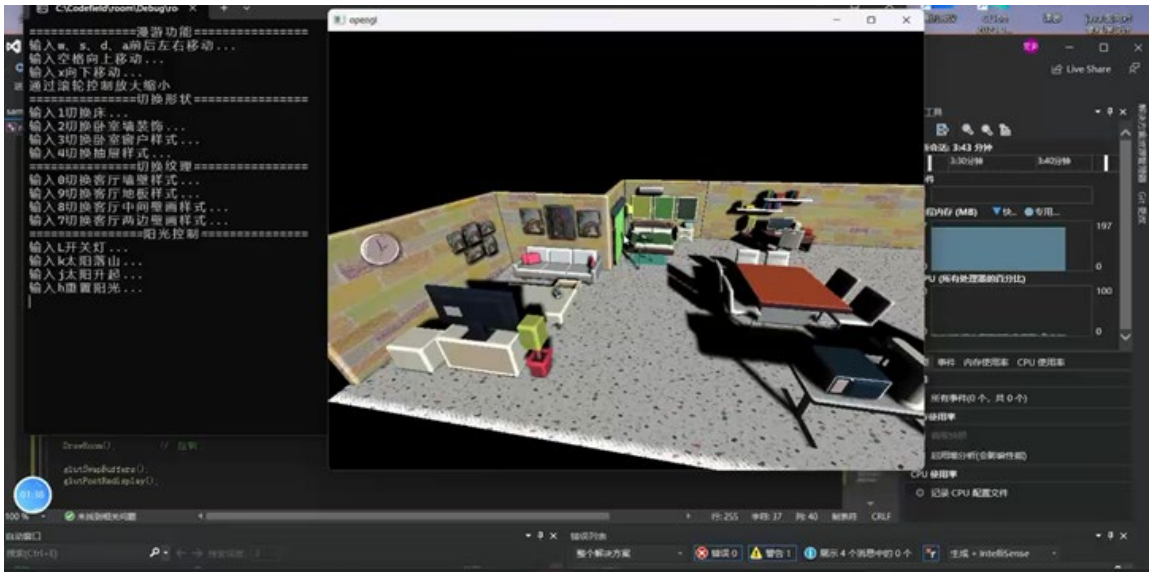
这个过程中我们使用了两个着色器：顶点着色器和片段着色器。

顶点着色器在渲染过程的早期阶段运行，主要负责处理和转换顶点属性，如位置、法线和纹理坐标。它使用变换矩阵（如投影矩阵和视图矩阵）来将顶点位置从世界空间转换到裁剪空间，并可能执行一些顶点属性的标准化操作。

片段着色器在渲染过程的后期阶段运行，它负责进行光照计算和阴影判断。

4.3.1光照的实现

光照计算部分包括环境光、漫反射光和镜面反射光的处理，其中环境光为基础光照，确保场景中所有地方都有光。通过简单的乘法实现。漫反射光依赖于光源方向和法线的角度，实现真实的光照效果。通过光源方向和法线的点积来计算，点积越大，表面接收的光越多。镜面反射光依赖于观察方向和光源方向，为光滑表面添加高光。



此外，片段着色器还将计算得到的光照效果应用于纹理颜色，并通过lightOn变量控制是否启用完整光照模型或仅使用环境光照。下方是光照的代码展示。

```
//binding的作用：直接在着色器glsl中绑定纹理单元！！！
layout (binding = 0) uniform sampler2D U_MainTexture; //纹理采样器 通过CPU中roomTexture传入
layout (binding = 1) uniform sampler2D U_ShadowMap; //通过CPU中shadowMap传入



in vec3 V_Normal; //法线位置
in vec4 V_WorldPos; //世界坐标，即摆放好了场景
in vec2 V_Texcoord;  //纹理坐标
in vec4 V_LightSpaceFragPos;

uniform vec3 ViewPos;
uniform int lightOn;

uniform float sunlight;

```

阴影的实现

片段着色器则在渲染过程的后期阶段运行，它负责进行光照计算和阴影判断。在阴影处理方面，片段着色器使用PCF（Percentage-Closest Filtering）技术。PCF通过在深度纹理中以当前片段为中心，采样周围多个点的深度值，以此来平滑阴影边缘，减少由于深度突变导致的硬边缘现象，从而增强渲染场景的视觉效果。具体来说，片段着色器会比较当前片段的深度值与阴影贴图中存储的深度值，如果当前片段的深度值大于阴影贴图中的深度值，则认为该片段处于阴影中。

```
//binding的作用：直接在着色器glsl中绑定纹理单元！！！
layout (binding = 0) uniform sampler2D U_MainTexture; //纹理采样器 通过CPU中roomTexture传入
layout (binding = 1) uniform sampler2D U_ShadowMap; //通过CPU中shadowMap传入



in vec3 V_Normal; //法线位置
in vec4 V_WorldPos; //世界坐标，即摆放好了场景
in vec2 V_Texcoord;  //纹理坐标
in vec4 V_LightSpaceFragPos;

uniform vec3 ViewPos;
uniform int lightOn;

uniform float sunlight;
```

4.4 键鼠交互与菜单控制

接下来是交互设计，通过键盘，用户可以控制摄像机的移动，改变场景中的光照条件，以及切换不同的场景元素。鼠标可以拖动来旋转视角，通过滚轮调整视场大小，增加用户的互动体验。

4.4.1相机控制

实现键盘交互函数： WASD 控制上下左右移动，空格视角向上移动，X视角向下移动，鼠标滚轮控制放缩。

```
void keyFunc(GLubyte key, int x, int y)          // 键盘交互函数,    ws移动摄像机    c切换方案,    l开关灯
{
        cameraRight = glm::normalize(glm::cross(up, cameraDirection));//根据朝向改变右轴

        switch (key)
        {
                //摄像机移动------------------------------------------------------------
        case 'w': case 'W'://前移
                cameraPos += cameraSpeed * cameraDirection;
                cameraTarget += cameraSpeed * cameraDirection;
                break;
        case 's': case 'S'://后移
                cameraPos -= cameraSpeed * cameraDirection;
                cameraTarget -= cameraSpeed * cameraDirection;
                break;
        case 'a': case 'A'://左移
                cameraPos += cameraSpeed * cameraRight;
                cameraTarget += cameraSpeed * cameraRight;
                break;
        case 'd': case 'D'://右移
                cameraPos -= cameraSpeed * cameraRight;
                cameraTarget -= cameraSpeed * cameraRight;
                break;
        case ' ': //飞天
                cameraPos += cameraSpeed * up;
                cameraTarget += cameraSpeed * up;
                break;
        case 'x'://遁地
                cameraPos -= cameraSpeed * up;
                cameraTarget -= cameraSpeed * up;
                break;
        case 'k': case 'K':
                sunlight -= 0.5;
                //printf("%f", sunlight);
                if (sunlight <= 0) {
                        sunlight = 0.5;
                }
                break;
        case 'j': case 'J':
                sunlight += 0.5;
                //if (sunlight >= 50.0) sunlight = 50.0;
                break;
        case 'h':case 'H':
                sunlight = 50.0;
                break;
```

4.4.2光源控制

输入1-4切换形状、输入7-10切换纹理、输入L开关灯、输入KJ实现太阳升起落下、输入H实现重置阳光 代码实现：

```
                //物品样式切换------------------------------------------------------------
        case '1'://切换卧室床样式
                bed = (bed + 1) % 2;
                break;
        case '2'://切换卧室正墙装饰
                wallDeco = (wallDeco + 1) % 2;
                break;
        case '3'://切换卧室窗户样式
                window = (window + 1) % 4;
                break;
        case '4'://切换抽屉样式
                drawer = (drawer + 1) % 2;
                break;
        case '0'://切换客厅墙壁纹理
                wallTexture++;
                wallTexture %= wallsSize;
                break;
        case '9'://切换客厅地板纹理
                floorTexture++;
                floorTexture %= floorsSize;
                break;
        case '8'://切换客厅中间挂画纹理
                paintTexture++;
                paintTexture %= paintingsSize;
                break;
        case '7'://切换客厅两边纹理
                otherPaintTexture++;
                otherPaintTexture %= OtherPaintingsSize;
                break;
                //开关灯------------------------------------------------------------
        case 'l': case 'L':
                lightOn = lightOn == 1 ? 0 : 1;
                break;
        }
        //viewMatrix = glm::lookAt(cameraPos, cameraCenter, glm::vec3(0.0f, 1.0f, 0.0f));
        viewMatrix = glm::lookAt(cameraPos, cameraTarget, glm::vec3(0.0f, 1.0f, 0.0f));
        lightViewMatrix = glm::lookAt(glm::vec3(150.0f, 150.0f, 100.0f), glm::vec3(0.0f, 0.0f, -50.0f)

}
```

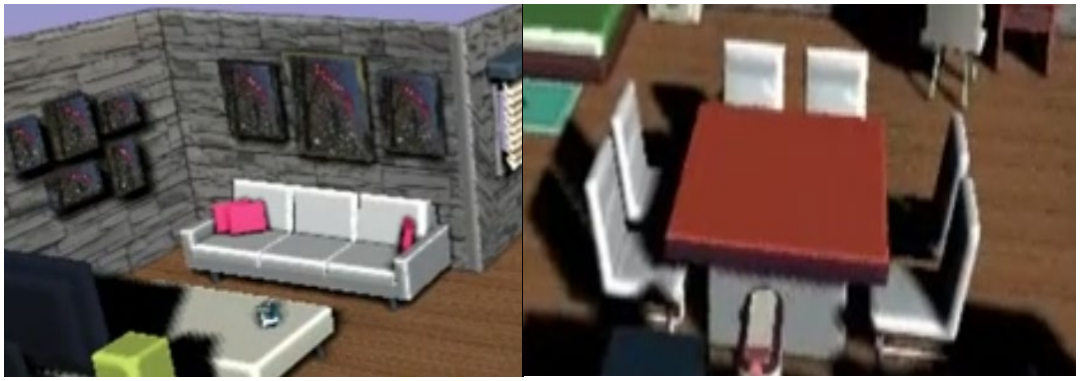
5. 实验结果

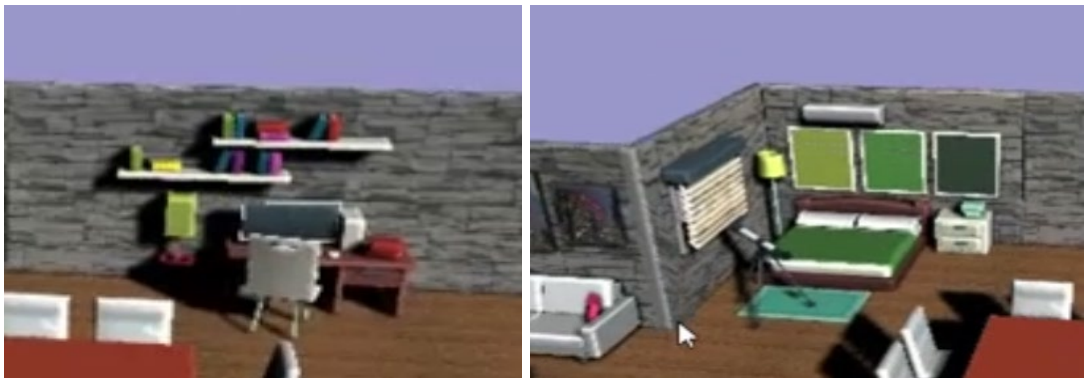
5.1 模型加载

5.1.1 实验结果

在模型加载实验中，我们成功地使用 Assimp 库导入了多种不同格式的模型文件，并将其转换为 OpenGL 可识别的顶点、法线和纹理坐标数据。

每个模型的细节都完整地加载到了 OpenGL 中，并显示在我们构建的三维场景中。以下是我们的模型的加载效果截图：





5.1.2 讨论分析

模型加载的成功取决于以下几个关键点：

1. **Assimp 库的使用**：Assimp 提供了对多种常见模型格式的支持，简化了模型导入的过程。我们利用 Assimp 将模型数据转换为统一的数据结构，这使得模型的解析和渲染更加高效。
2. **数据结构的设计**：我们定义了 VertexData 结构体来存储顶点位置、法线和纹理坐标。这种数据结构的设计使得我们能够轻松地在 OpenGL 中使用这些数据。
3. **VBO 和 VAO 的使用**：顶点缓冲区对象（VBO）和顶点数组对象（VAO）用于存储和管理模型的顶点数据，这使得我们能够高效地在渲染循环中使用这些数据。

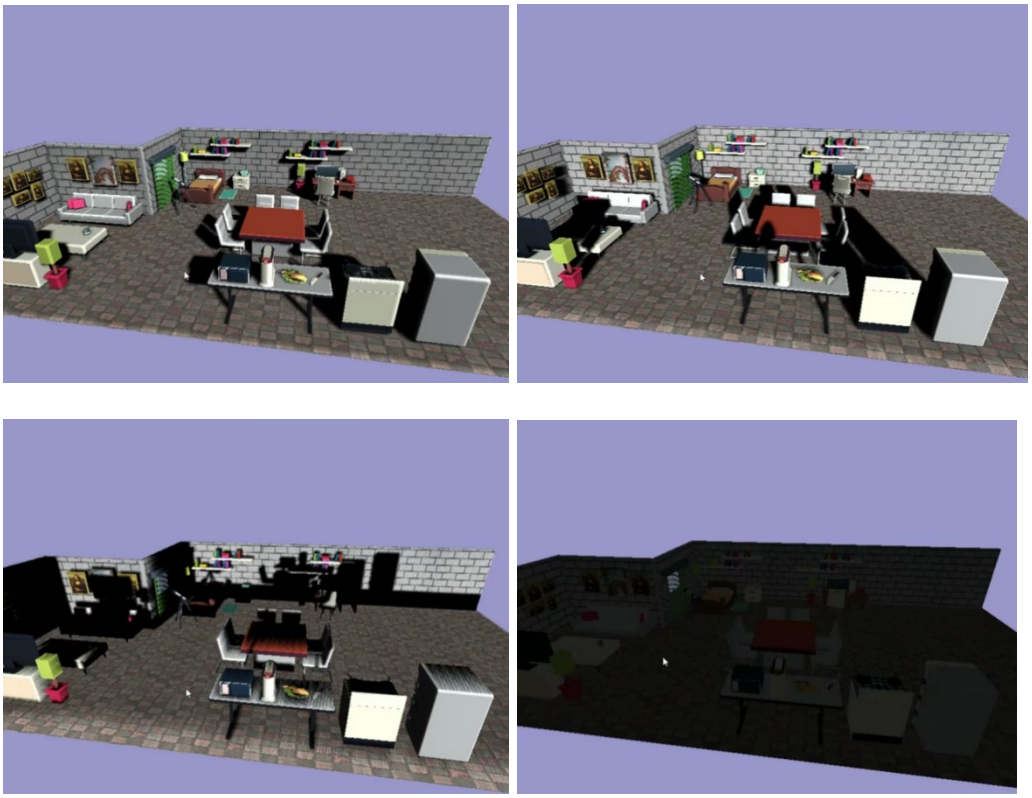
5.2 纹理映射，光照，阴影

5.2.1 实验结果

在纹理映射、光照和阴影实现方面，我们取得了以下成果：

1. **纹理映射**：成功将纹理映射应用于模型表面，生成了逼真的视觉效果。例如，我们将木纹纹理映射到桌面模型上，将砖块纹理映射到墙壁模型上。以下是纹理映射效果截图：

2. **光照效果**：实现了 Phong 光照模型，包括环境光、漫反射光和镜面反射光。通过调整光源的位置和强度，我们能够生成不同的光照效果，使得场景更加真实。以下是光照效果截图：



阴影效果：通过 PCF 技术实现了阴影效果。我们在片段着色器中比较当前片段的深度值与阴影贴图中的深度值，以确定片段是否处于阴影中。

5.2.2 讨论分析

在纹理映射、光照和阴影实现过程中，我们遇到了一些挑战和学习到的经验：

1. **纹理映射**：纹理映射的关键在于正确设置纹理坐标，并将其传递到片段着色器中。我们使用顶点着色器提供的纹理坐标，并在片段着色器中进行纹理采样，最终生成了逼真的纹理效果。
2. **光照模型**：Phong 光照模型的实现需要考虑环境光、漫反射光和镜面反射光的相互作用。通过调整光源的位置和强度，我们能够生成不同的光照效果，使得场景更加真实。
3. **阴影处理**：阴影处理是渲染中较为复杂的一部分。我们使用 PCF 技术平滑阴影边缘，减少硬边缘现象，增强了渲染场景的视觉效果。阴影处理的关键在于深度比较和采样技术，通过调整采样次数和平滑程度，我们能够生成不同质量的阴影效果。

5.3 实现相机控制，添加用户交互

5.3.1 实验结果

在该模块，我们成功地实现了相机的控制，用户可以通过鼠标和键盘操作，进行视角切换。同时我们添加了用户交互功能，用户可以和场景进行一定的交互，如开关灯，切换家具材质和样式等。

（1）**相机的控制**：实现了键盘交互函数，按下 WASD 控制相机上下左右移动，空格可以使视角向上移动，X 则可以控制视角向下移动，鼠标滚轮控制放缩。

（2）**用户交互**：实现了键盘交互函数，输入 1-4 可以切换物体形状，如切换卧室墙上的装饰、切换卧室窗户样式等，输入 7-10 可以切换物体纹理，如切换墙壁、地板的样式。输入 L 控制灯的开关、输入 KJ 实现太阳升起落下、输入 H 则可以实现重置阳光。

5.3.2 讨论分析

在该模块的实现过程中，我们遇到了一些挑战和学习到的经验：

（1）相机控制：在实现相机控制时，我们遇到了一些跳转问题，特别是在处理视角向上和向下移动时，可能会出现视角跳转不流畅的情况。通过调整相机控制的灵敏度和平滑度，我们学会了如何优化视角的移动，使其更加流畅和自然，从而提升用户体验。

（2）用户交互：在用户交互功能方面，切换物体形状和纹理时，可能会出现切换不及时的情况。对用户交互功能进行细致的调试和优化后，我们学会了如何处理用户输入并及时更新场景的状态，确保用户操作能够准确反映在场景中，从而提升交互的实时性和准确性。

6. 实验环境

6.1 基础场景

- 库文件
  - C++标准库
  - GLEW（OpenGL 扩展加载库）
  - SOIL：加载和处理图像数据，用于纹理映射
  - GLM（OpenGL 数学库）
- 系统配置

		版本
System Info	操作系统	Windows 10 家庭中文版 64-bit
	渲染器	Intel(R) UHD Graphics Family
	处理器	intel(R) Core(T1M) 7-10870H CPU @2.20GHz
	屏幕显示	1920*1080*32bpp（144Hz）
OpenGL		4.3
Visual Studio		Microsoft Visual Studio Community 2019 版本 16.11.26

7. 实验总结

7.1 于蕊宁

7.1.1 实验难点

1. 镜面反射：镜面反射的效果受到几何体表面法线方向和光源位置的影响，这需要在每个表面点上进行向量计算，增加了复杂性。
2. 漫反射材质：漫反射光的强度受到材质参数的影响，例如表面的漫反射系数。选择合适的参数对于获得期望的光照效果至关重要。
3. 阴影算法的性能：PCF等平滑阴影技术可能会增加渲染成本，特别是在移动设备或性能较低的硬件上。因此，需要在阴影效果和性能之间进行权衡，并选择适当的算法和优化策略。
4. 阴影与动态场景的处理：对于动态场景，阴影映射需要在每一帧重新计算，这会增加渲染负担。
5. 深度精度和分辨率：阴影贴图是通过从光源视角渲染场景来生成的深度纹理。在这个过程中，需要考虑深度精度和纹理分辨率的平衡。低分辨率的阴影贴图可能导致阴影边缘的锯齿状效果，而高分辨率的贴图可能会增加渲染成本。

7.1.2 实验感想

本次实验有许多的难点和需要注意的点，比如光照种的镜面反射、漫反射材质，还有阴影映射中阴影算法性能、动态场景阴影处理以及深度精度与分辨率等。

首先，镜面反射和漫反射材质的处理需要精确的向量计算和参数选择，以实现逼真的光照效果。对于阴影算法，需要在保证视觉效果的前提下，平衡渲染成本，这要求深入理解不同算法的性能特点，并进行有效的优化。处理动态场景的阴影映射更是一项复杂任务，需要在每一帧中重新计算阴影，增加了渲染负担。

在光照和阴影映射的处理中，PCF技术的应用也是一项重要的挑战。PCF技术通过在深度纹理周围采样多个点来平滑阴影边缘，从而增强视觉效果。在处理本次实验的动态场景时，PCF技术需要在每一帧重新计算，增加了渲染负担，因此我们查阅资料，也不断的尽心优化，才完成了现在的渲染效果。在面对这些挑战时，需要不断学习和探索，结合理论知识和实践经验，寻找最佳的解决方案。通过持续的努力和创新，才能有效应对光照和阴影映射中的各种难点，提升图形渲染的质量和性能。

7.2 刘荧

7.2.1 实验难点

本次实验的难点在于:

1. **纹理映射：** 纹理坐标需要精确映射到几何体上，当模型的顶点数据修改时，纹理坐标也需要相应调整。如果纹理坐标不准确，纹理可能会拉伸、收缩或产生缝隙。同一个几何体表面可能需要多个纹理层，需要正确组合这些纹理。

2. **透视投影：**透视投影矩阵是将 3D 坐标转换为 2D 屏幕坐标的核心。构建该矩阵需要理解和正确设置视锥体参数。深度缓冲用于处理图形的深度信息，以确保正确的遮挡关系。然而，有限的深度缓冲精度可能导致 Z-fighting 现象，调整近、远剪裁面的距离和深度缓冲精度以减少 Z-fighting，同时保持合理的深度范围。在大场景或复杂场景中，透视投影涉及大量的矩阵计算和深度测试。需要优化这些计算以确保渲染性能。

3. **使用合适的数学库来提高计算效率**。

4. **键鼠交互：** 通常需要结合其他库来处理输入事件。在不同平台上配置合适的输入库是一个挑战。键鼠输入需要实时响应，确保用户操作能够立即反映在图形界面上。需要高效的事件处理机制，避免输入滞后或丢失。在处理键鼠交互时，避免输入操作与界面更新之间的延迟或冲突要协调主循环中的输入处理、逻辑更新和渲染操作，确保每个帧都能及时响应用户输入并更新显示内容。实现摄像机的平移、旋转和缩放操作时将鼠标移动转换为摄像机视图变换，并确保操作直观且平滑。需要解决摄像机的视角限制和边界处理，避免因鼠标操作导致摄像机进入不合理的位置或角度。

7.2.2 实验感想

通过本次实验，我有以下几个方面的感想。

具体来说，我们在模型加载方面，学会了使用Assimp库来导入和管理多种格式的三维模型，从而简化了模型处理的复杂性。在纹理映射方面，我们探索了不同的纹理映射技术，并通过GLFW和SOIL库实现了高效的纹理加载和应用，使得模型表面更加逼真和细腻。光照模型的实现过程中，我们深入理解

了Phong光照模型的原理，增强了场景的立体感和真实性。
团队合作方面，我们充分发挥了每个成员的特长，通过定期的讨论和协调，解决了开发过程中遇到的各种问题。从代码编写到调试优化，每个环节都体现了团队的高度协作和共同努力。最终，我们成功地将各个模块整合成一个完整的系统，呈现出一个视觉效果出色、性能优异的实验项目。
这次实验不仅巩固了我们对计算机图形学理论的理解，更让我们在实际操作中积累了丰富的经验。通过解决实际问题，我们的团队合作能力和问题解决能力得到了显著提升，为今后的学习和科研打下了坚实的基础。这次宝贵的经历，将激励我们在计算机图形学领域继续深入探索和创新。

7.3 王美靖

7.3.1 实验难点

在此次实验中，主要的难点在于实现复杂场景的光照计算和材质渲染。特别是在处理复杂的光照模型时，既要保证渲染的逼真度，又要优化计算性能；在材质渲染时，如何精确调整材质参数来真实呈现不同物体的质感，同时确保渲染结果的一致性和高质量，也是一个很大的挑战。

7.3.2 实验感想

通过本次实验，我深入探索了三维图形的建模和渲染技术，并深刻理解了光照、材质和纹理对场景视觉效果的影响。在实验初期，我通过学习基本的建模技术，逐步掌握了使用各种工具创建复杂几何形状的能力。这一过程中，我意识到细节在三维建模中的重要性，无论是一个小的曲面处理，还是一个微小的纹理调整，都会对最终效果产生显著影响。光照是渲染中至关重要的一部分。通过实验，我了解到不同类型的光源、光照方向和强度如何影响场景的整体氛围和物体的立体感。尤其是在处理阴影和反射时，光照的设置显得尤为关键。为了模拟自然光照效果，我尝试了多种光源配置和参数调整，从而实现了更逼真的渲染效果。总体而言，本次实验不仅强化了我在三维图形建模和渲染方面的实践技能，还加深了我对这门技术的理论理解。这次实验的收获将对我今后的学习产生积极作用。

7.4 冯文涵

7.4.1 实验难点

在三维场景模型渲染中，纹理和材质的细节处理是非常重要的一环。由于纹理和材质的种类繁多，不同的纹理和材质需要采用不同的渲染技术进行处理，这增加了实验的难度。

光照和阴影的模拟是三维场景模型渲染中的另一个重要难点。光照和阴影的模拟需要考虑到光源的位置、强度、颜色等因素，以及物体之间的遮挡关系。在实际的实验中，我们需要采用不同的光照模型和阴影算法来模拟真实的光照和阴影效果。

7.4.2 实验感想

通过完成三维场景模型渲染的实验，我收获了以下成果和经验：

成功搭建了三维场景模型，掌握了建模工具的使用方法和技巧，能够根据需求构建出满足实验要求的模型。

通过纹理映射技术，将二维纹理图像成功地映射到三维模型表面，增加了模型的细节和真实感。我们学习了不同的纹理映射方法，并解决了纹理映射中遇到的问题，如纹理拉伸和接缝的处理。

实现了光照和阴影效果，通过选择合适的光照模型和阴影算法，设置了合适的光源，并处理了光照和阴影的细节，使场景更加逼真和立体。

在实验过程中，我也遇到了一些挑战和困难，但通过不断尝试和改进，最终克服了这些难点，并取得了一定的成果。这个实验不仅提高了我的技术水平，还加深了对三维场景模型渲染原理和方法的理解。

7.5 曹雯佳

7.5.1 实验难点

本次实验的难点在于:

1. **纹理映射：** 实现纹理映射涉及纹理采样器的绑定、纹理坐标的传递、采样纹理和光照效果的结合。需要确保纹理贴图能正确地映射到模型表面，并且光照效果与纹理颜色结合得自然。
2. **光照和阴影：** 实现 Phong 光照模型和阴影效果是较为复杂的部分。需要计算环境光、漫反射光和镜面反射光，并将其应用于模型表面，同时还要考虑阴影的生成和渲染，确保场景光照效果逼真。

7.5.2 实验感想

通过本次实验，我们深入了解了计算机图形学中的重要概念和技术，包括模型加载、纹理映射、光照模型和阴影算法等。在实践中，我们不仅加深了对OpenGL图形编程接口的理解，还学会了如何利用各种工具和库来实现复杂的图形效果。在团队合作中，我们分工明确、配合默契，克服了各种困难，最终完成了一个令人满意的实验项目。这次经历让我们对计算机图形学领域有了更深入的认识，并提升了我们的团队合作能力和问题解决能力。

7.6 张悦

7.6.1 实验难点

在实现相机控制时，我们遇到了一些跳转方面的问题，特别是在处理视角向上和向下移动时，可能会出现视角跳转不流畅的情况。通过调整相机控制的灵敏度和平滑度，我们学会了如何优化视角的移动，使其更加流畅和自然，从而提升用户体验。在用户交互功能方面，切换物体形状和纹理时，可能会出现切换不及时的情况。对用户交互功能进行细致的调试和优化后，我们学会了如

何处理用户输入并及时更新场景的状态，确保用户操作能够准确反映在场景中，从而提升交互的实时性和准确性。

7.6.2 实验感想

本次实验对于我们来说是一个全新的挑战，遇到难点时，我们学会了寻找创新的解决方案，不断调整和优化代码以克服困难，这种挑战激发了我们的创造力和解决问题的能力。同时，在用户交互功能方面，通过实验，我们更加深刻地体会了达到良好的软件使用体验同样是一件有挑战性的事情，流畅的相机控制和准确的用户交互功能十分重要。在实验过程中，我们不断学习和改进，通过调试和优化代码，不断提升功能的稳定性和准确性，持续学习的态度我们会持续努力保持下去。

8. 实验分工

序号	学号姓名	主要工作	占比
1	21301160-刘荧	光照，纹理映射， 菜单控制。	18.00
2	21301057-于蕊宁	光照，阴影映射，PPT 制作。	17.50
3	21301049-王美靖	模型加载，键鼠交互，PPT 制作。	17.50
4	21301033-冯文涵	模型加载，文档撰写，分工安排规划。	16.00
5	21301058-张悦	场景构思与搭建，文档撰写。	15.00
6	20301001-曹雯佳	菜单控制的设计实现，文档撰写。	15.00