# 程序设计说明书

## 1.项目概述

本项目是一个基于OpenGL、GLFW、GLAD和GLM库的3D图形应用程序，旨在实现一个带有灯光控制和相机控制的3D场景。用户可以通过键盘和鼠标操作来控制相机的移动和视角，调整灯光的强度和位置，并在场景中添加和控制一个机器人的移动和旋转。同时，也可根据模型的骨骼实现机器人的自主运动。

## 2. 功能实现

·**相机控制：**用户可以通过键盘和鼠标控制相机的移动和视角。

·**光照控制：**用户可以调整场景灯光的强度和颜色。

·**变换控制：**用户可以控制机器人的移动和旋转和大小的缩放。

·**视角切换：**用户可以切换到机器人视角，跟随机器人移动和旋转。

·**漫游实现：**用户可通过按键控制对建模后的场景进行Zoom In/Out，Orbit, Zoom To Fit等观 察功能的漫游控制。

·**场景渲染：**渲染一个包含房间和机器人的3D场景。

·**机器人运动：**结合建模时的骨骼相关实现机器人的骨骼运动。

## 3. 模块设计

### 3.1 主程序模块（main.cpp）

·**初始化：**初始化GLFW和GLAD，创建窗口，设置OpenGL版本和配置。

·**场景设置：**创建场景节点和模型，设置初始变换矩阵。

·**渲染循环：**处理用户输入，更新相机和机器人状态，渲染场景。

·**回调函数：**处理窗口大小变化、鼠标移动和键盘输入等事件。

### 3.2 模型模块（model.cpp/model.h）

·**模型加载：**加载OBJ文件和MTL文件，解析顶点、法线和纹理坐标。

·**材质处理：**处理材质属性，加载纹理。

·**网格设置：**设置顶点数组对象（VAO）和顶点缓冲对象（VBO），准备渲染数据。

·**绘制：**使用Shader绘制模型，应用模型、视图和投影矩阵。

### 3.3 场景模块（scene.h）

·**场景节点：**定义场景节点，包含局部变换矩阵、模型列表和子节点列表。

·**场景管理：**管理场景节点，添加节点和模型，递归绘制场景。

### 3.4 着色器模块（shader.h）

·**着色器加载：**加载和编译顶点着色器和片段着色器。

·**着色器使用：**设置着色器参数，绑定着色器程序。

## 4. 类和结构体设计

### 4.1 Model 类

**成员变量：**

vertices：顶点数据。

normals：法线数据。

texCoords：纹理坐标数据。

materials：材质列表。

textureIDs：纹理ID列表。

directory：模型文件目录。

VAO、VBO、EBO：OpenGL缓冲对象。

成员函数：

Model(const std::string& objPath, const std::string& mtlPath)：构造函数，加载模型。

~Model()：析构函数，释放资源。

loadOBJ(const char\* path, const char\* mtlPath)：加载OBJ文件。

processMaterials(const std::vector<tinyobj::material\_t>& materials)：处理材质。

loadTexture(const char\* path)：加载纹理。

setupMesh()：设置网格数据。

Draw(Shader& shader, const glm::mat4& modelMatrix, const glm::mat4& viewMatrix, const glm::mat4& projectionMatrix)：绘制模型。

### 4.2 SceneNode 类

**成员变量：**

transform：局部变换矩阵。

models：模型列表。

children：子节点列表。

**成员函数：**

SceneNode()：构造函数，初始化变换矩阵。

addChild(std::shared\_ptr<SceneNode> child)：添加子节点。

addModel(std::shared\_ptr<Model> model)：添加模型。

setTransform(const glm::mat4& transform)：设置变换矩阵。

getTransform() const：获取变换矩阵。

draw(Shader& shader, const glm::mat4& parentTransform, const glm::mat4& view, const glm::mat4& projection)：递归绘制节点和子节点。

### 4.3 Scene 类

**成员变量：**

root：场景根节点。

**成员函数：**

Scene()：构造函数，创建根节点。

addNode(std::shared\_ptr<SceneNode> node, std::shared\_ptr<SceneNode> parent = nullptr)：添加节点。

draw(Shader& shader, const glm::mat4& view, const glm::mat4& projection)：绘制整个场景。

### 4.4 其他重要内容：

**Main.cpp**

成员变量：

cameraPos：相机位置

cameraFront：相机前方向量

cameraUp：相机上方向量

cameraTarget：相机目标点

yaw：偏航角，控制相机左右旋转

pitch：相机俯仰角

lastX / lastY：上次鼠标位置

函数：

void processInput(GLFWwindow\* window, std::shared\_ptr<SceneNode> robotNode);处理键盘输入

void mouse\_callback(GLFWwindow\* window, double xposIn, double yposIn);处理鼠标移动

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height);处理窗口大小变化

**在主程序中实现的键盘功能包括：**

1. ESC键：退出程序。
2. W、A、S、D键：控制相机前后左右移动。
3. O、P键：控制相机围绕中心点进行轨道运动。
4. Y键：切换视角（机器人视角与环境视角）。
5. +、-键：调节光照强度。
6. L键：启用/禁用光源位置映射。
7. R、G、B键：改变光源颜色（红、绿、蓝）。
8. W键：重置光源颜色为白色。
9. 上下左右箭头键：控制机器人位置上下左右移动。
10. Q键：启动/停止机器人旋转。
11. Z、X键：缩放相机视野（放大/缩小）

**着色器实现**

**顶点着色器概述 （vertex\_shader.glsl）**

顶点着色器负责处理每个顶点的变换，将物体空间的顶点坐标转换为世界空间、视图空间和裁剪空间。此外，顶点着色器将必要的属性传递给片段着色器。

**主要部分**：

* 输入属性：
  + position: 顶点位置。
  + normal: 顶点法线，用于光照计算。
  + texCoord: 纹理坐标，用于纹理映射。
* 输出：
  + fragPos: 世界空间中的片段位置，传递给片段着色器进行光照计算。
  + fragNormal: 世界空间中的法线，用于光照计算。
  + TexCoords: 纹理坐标，传递给片段着色器。

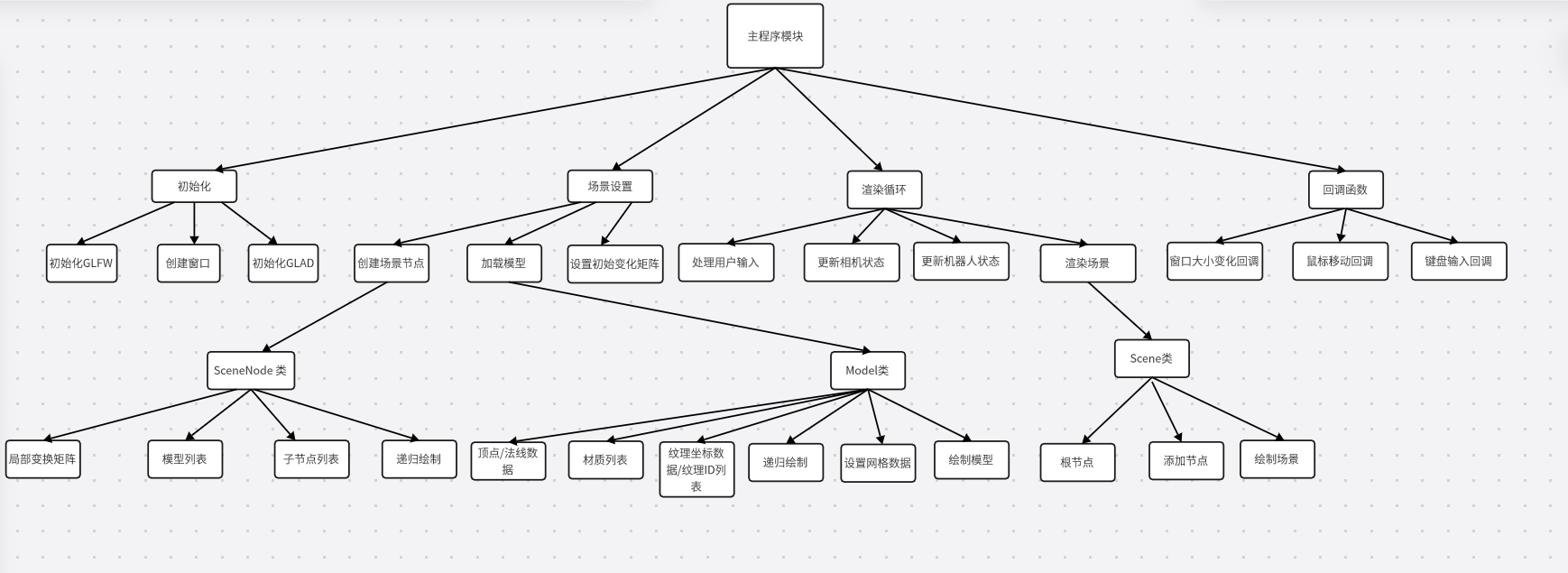
**片段着色器概述 (fragment\_shader.glsl)**

片段着色器负责计算每个像素的最终颜色，使用Phong光照模型（环境光、漫反射、镜面反射），并应用纹理和光照效果。

**主要部分**：

* 材质属性：
  + ambient: 环境光颜色，表示即使没有光源照射时的基本颜色。
  + diffuse: 漫反射颜色，表示表面在光照下的颜色。
  + specular: 镜面反射颜色，控制表面反射光亮的强度和范围。
  + shininess: 材质的光泽度，影响镜面反射的锐利度。
  + diffuseMap, specularMap, normalMap: 纹理贴图，分别控制漫反射、镜面反射和法线。
* 光照属性：
  + light.position: 光源位置。
  + light.color: 光源颜色。
  + constant, linear, quadratic: 光照衰减因子，模拟光照随距离衰减的效果。
* 光照计算：
  + **环境光**：根据ambientStrength调整，是否使用纹理由hasTexture决定。
  + **漫反射**：根据光照方向与法线的点积计算光照强度。
  + **镜面反射**：使用Phong反射模型计算，模拟表面高光。
  + **衰减**：根据光源与片段之间的距离计算光照衰减。
* 光照强度：
  + 所有光照组件（环境光、漫反射、镜面反射）都乘以lightIntensity，允许动态调整光照强度。
* 伽马校正：
  + 采用伽马校正修正最终颜色，调整显示效果以适应非线性显示器。
* 纹理映射：
  + 根据hasTexture标志决定是否使用纹理。如果使用纹理，则从对应的纹理贴图中获取材质属性，否则使用固定的颜色。

## 总体设计图



## 场景内建模