深圳大学考试答题纸

(以论文、报告等形式考核专用)  
二○ 二四 ～二○ 二五 学年度第 一 学期

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程编号 | 1500610003 | | 课序号 | | 05 | 课程名称 | | 计算机图形学 | | 主讲教师 | 熊卫丹 | | 评分 |  |
| 学 号 | 2022155028 | | | 姓名 | | 黄亮铭 | 专业年级 | | 2022级软件工程腾班 | | | | | |
|  | | | | | | | | | | | | | | |
| 教师评语： | | | | | | | | | | | | | | |
| 题目： | | 俄罗斯方块  中世纪村庄场景建模 | | | | | | | | | |  | | |

宋体五号，至少八页，可以从下一页开始写。

**成绩评分栏：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **评分项** | **俄罗斯方块文档**  （占12分） | **俄罗斯方块代码**  （占24分） | **俄罗斯方块迟交倒扣分**  （占0分） | **虚拟场景建模文档**  （占16分） | **虚拟场景建模代码**  （占38分） | **演示与答辩**  （占10分） | **虚拟场景建模迟交倒扣分**  （占0分） | **大作业总分** |
| **得分** |  |  |  |  |  |  |  |  |
| **评分人** |  |  |  |  |  |  |  |  |

目录

[1工程架构 3](#_Toc185851431)

[2具体实现 3](#_Toc185851432)

[2.1内容整合 4](#_Toc185851433)

[2.2添加纹理 5](#_Toc185851434)

[2.3添加材质、光照和阴影效果 5](#_Toc185851435)

[2.4任意角度浏览场景 7](#_Toc185851436)

[2.4.1键盘 7](#_Toc185851437)

[2.4.2鼠标 7](#_Toc185851438)

[2.5用户交互控制物体 8](#_Toc185851439)

[2.6机器人的层次建模 10](#_Toc185851440)

[3鼠标和键盘的交互使用方法 12](#_Toc185851441)

[3.1鼠标交互使用方法 12](#_Toc185851442)

[3.2键盘的交互使用方法 12](#_Toc185851443)

[4 模型绘制结果展示 14](#_Toc185851444)

# 1工程架构

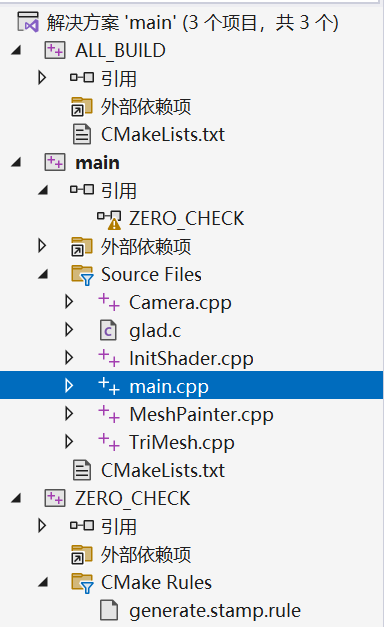


图1总体架构

项目的总体架构如图1所示。本次大作业涉及到的主要文件为：Camera、MeshPainter、TriMesh和main。此外，还包括顶点着色器和片元着色器。

接下来我将介绍每个文件在工程中的作用。

* **Camera**：主要用于在3D图形渲染中管理和控制相机的行为，提供了诸如视图矩阵和投影矩阵的计算方法，并支持键盘和鼠标交互来控制相机的移动和旋转等功能。
* **MeshPainter**：主要用于处理和渲染3D网格对象及其纹理、着色器、光照等，涉及到OpenGL的绘制、纹理加载、光照绑定等操作，常用于3D图形渲染应用程序。
* **TriMesh**：主要用于描述3D网格模型以及与之相关的光源，提供了对网格模型的几何形状、纹理、法线、颜色等属性的管理和处理方法，同时也包含了一些光照和变换功能。
* **InitShader**：提供了与OpenGL着色器相关的一些辅助函数，主要功能是从文件中读取着色器代码并创建一个OpenGL着色器程序。
* **Main**：提供了一个标准的框架来设置OpenGL环境、处理用户输入、进行渲染，并在窗口关闭时清理资源。

# 2具体实现

在经过2.1的内容整合之后，所有的实现，如2.2的添加纹理，2.3的添加光照、材质、阴影等效果，2.4的任意角度浏览场景，2.5的用户交互控制物体等都可以在main文件中进行实现，而无需再次修改其他文件。

## 2.1内容整合

上述工程的代码文件来自前面的各个实验。例如，MeshPainter类来自实验4，着色器（顶点着色器、片元着色器）来自实验3，main文件中的机器人建模来自实验补充二等等。

我将它们整合到一起，同时修改了部分的代码以更好地实现虚拟场景建模。以绘制3D网格图像为例，我在MeshPainter类中重载了drawMesh函数（图2），让其更方便绘制纹理的时候同时绘制阴影（实验四的MeshPainter类没有提供阴影）。



图2重载的drawMesh函数

此外，我还对main文件中的所有3D网格对象进行了统一的处理，方便初始化和绘制（图3）。例如，我将3D网格对象的索引和名字进行存储，然后将3D网格对象的缩放倍率、旋转角度等内容统一存储到一个数组中方便管理。通过上述措施，我就可以通过3D网格对象的名字直接调用对应的缩放倍率等内容，而无需为每个3D网格对象都新增一个缩放倍率变量。



图3统一管理3D网格对象

## 2.2添加纹理

添加纹理部分主要在main文件的init函数中实现，实现的思路如下所示。

1. 读入着色器（顶点着色器和片元着色器）的存储位置（图4）。

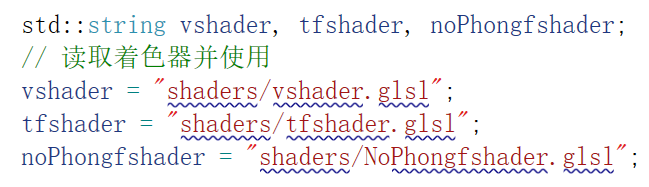


图4读取着色器

1. 初始化场景中所有的3D网格对象，如光源、地板、机器人的各个部位、建筑物和村民等。初始化也即是读取每个3D网格对象对应的obj文件（图5）。

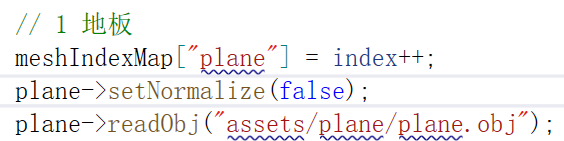


图5初始化对象（以地板为例）

1. 然后设置每一个对象的位置、旋转、缩放等属性（图6）。

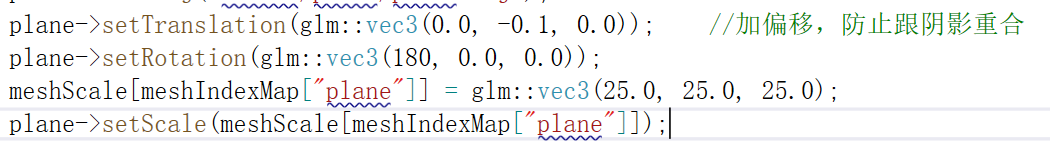


图6设置对象属性

1. 再加载3D网格对象的纹理，并将纹理和对应的对象进行绑定操作（图7）。

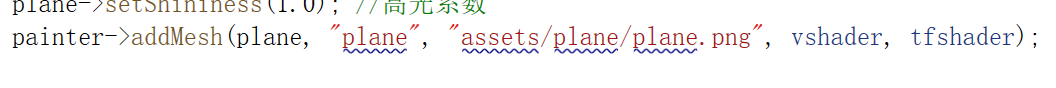


图7加载和绑定

1. 接下来将3D网格对象添加到渲染管线中，方便后续的渲染过程。同时，将当前对象添加到对象列表中，方便后续回收内存，防止内存泄露（图8）。

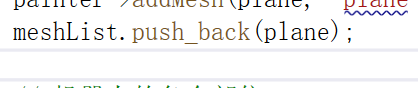


图8添加到对象列表

1. 最后是设置场景的背景颜色。我将背景颜色设置为蓝色，目的是让背景颜色充当天空。

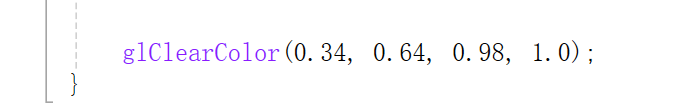


图8设置背景颜色

## 2.3添加材质、光照和阴影效果

以太阳为例，添加材质使用到了setAmbient、setDiffuse和setSpecular这个三个函数，分别代表环境光、漫反射和镜面反射，这三个函数接收的参数均为glm::vec4（图9）。

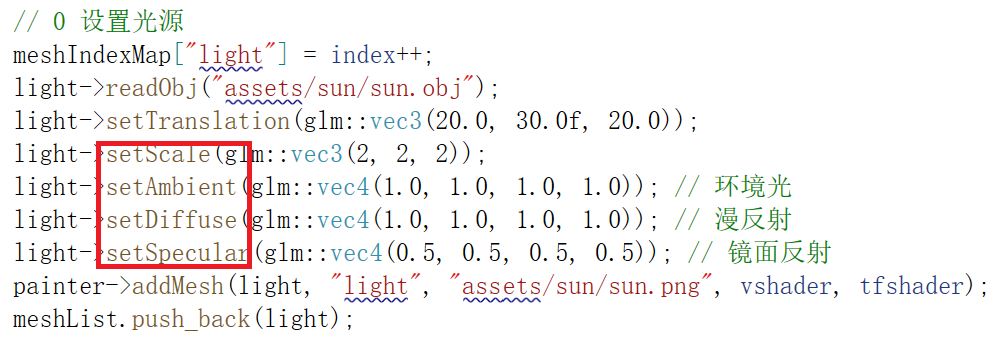


图9设置材质

而添加光照部分则是在片元着色器中实现，即通过模拟光照与物体表面交互的过程来实现基本的光照效果（图10）。具体来说，首先，计算环境光、漫反射光和镜面反射光三个分量。环境光模拟背景光，漫反射光反映光源照射到物体表面并散射的效果，而镜面反射光则模拟光线反射后在平滑表面形成的高光部分。然后着色器通过法线、光源方向和观察方向之间的关系，结合物体的材质属性（如漫反射、镜面反射系数和光泽度）计算出最终的光照颜色。最后，光照分量与纹理颜色相乘，得到物体的最终渲染效果。

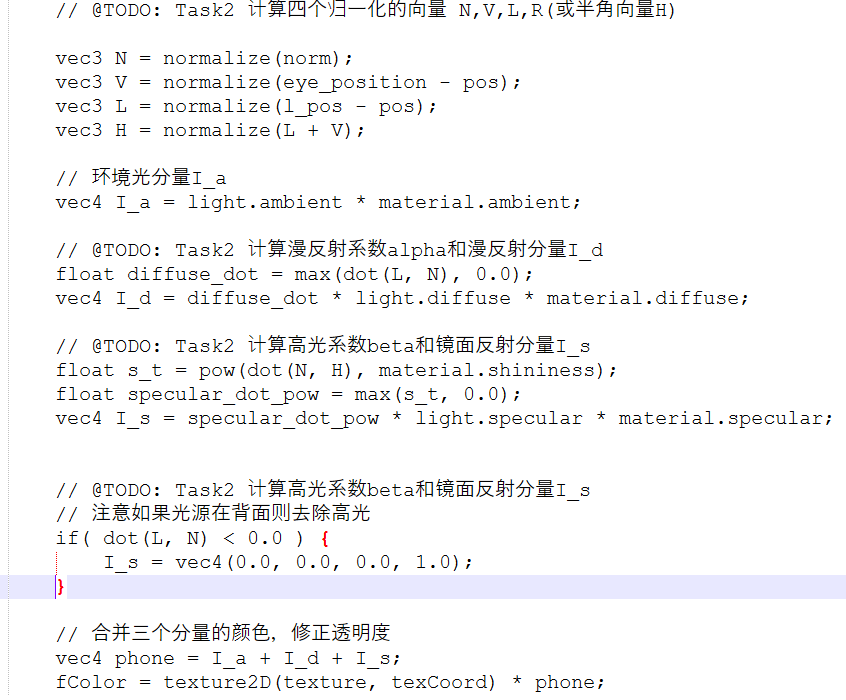


图10添加光照

阴影实际上在2.1的整合的文件中已经实现。在重载的drawMesh中，计算阴影矩阵和当前模视矩阵的乘积得到阴影的绘制位置，然后传递绘制阴影的变量到着色器中，将绘制的颜色改为黑色（图11）。

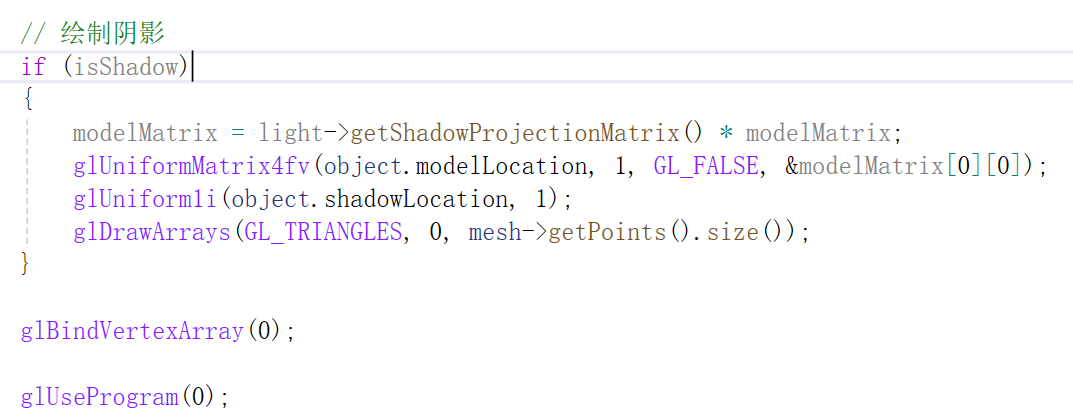


图11添加阴影

## 2.4任意角度浏览场景

该部分的实现在Camera的键盘响应函数和鼠标相应函数中。控制权传递到Camera的过程为（图12）：键盘和鼠标的回调函数会通过判断当前控制的3D网格对象是否为相机，从而决定是否将控制权移交给Camera。

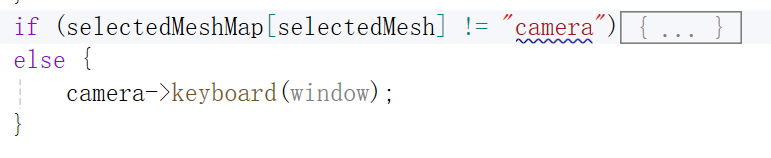


图12移交控制权判断条件

### 2.4.1键盘

键盘部分通过 WASD 键和 CTRL、ALT 键实现相机的平移和上下移动。当按下 W 键时，相机会沿着视线方向（front）前进，而按下 S 键时，则会沿着相反方向后退。按下 A 键会使相机向左移动，按下 D 键则会使相机向右移动，这两个方向是通过计算相机视线方向（front）与上方向（up）的叉积获得的。此外，按下 CTRL 键时，相机会沿着上方向（up）上升，而按下 ALT 键时，相机会沿着下方向下降。为了防止相机越界，代码还对相机的位置进行了限制，确保相机始终在预定的场景范围内。具体的代码如图13所示。

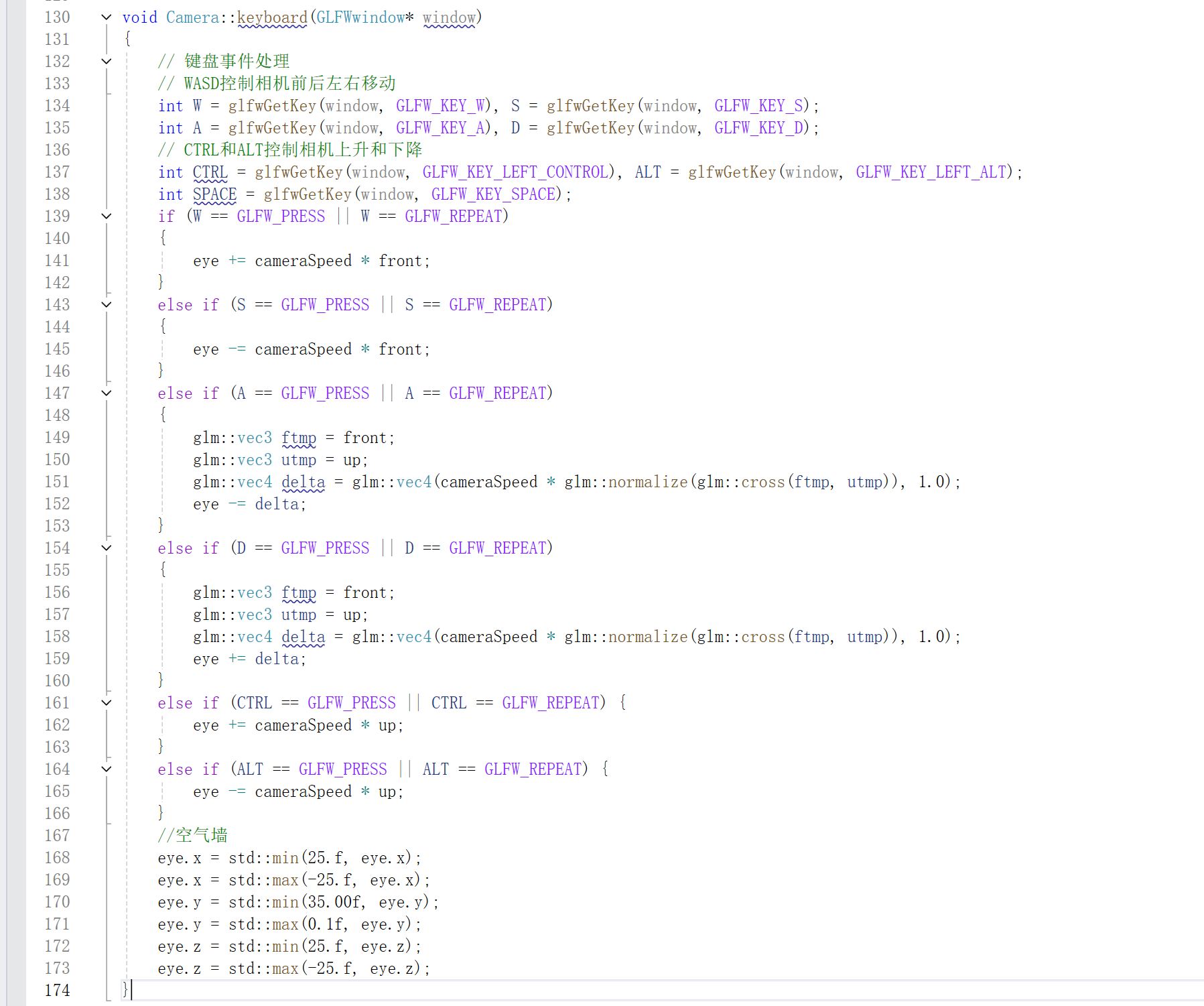


图13相机的键盘响应函数

### 2.4.2鼠标

鼠标部分控制相机的旋转，通过改变相机的俯仰和偏航角度来实现。当鼠标移动时，计算出当前鼠标位置与上次位置的偏移量，分别应用于 水平旋转和垂直旋转。鼠标的水平移动控制相机的左右旋转，而垂直移动则控制相机的上下旋转。为了避免过度俯视或仰视，俯仰角度被限制在 -89.9 到 89.9 度之间。通过这种方式，用户可以通过鼠标的移动精确控制相机视角，实现自由的场景浏览。具体的代码如图14所示。



图14相机的键盘响应函数

## 2.5用户交互控制物体

通过键盘控制物体，如机器人各个部件、三个村民和光源的位置和旋转。

用户可以通过数字键（1-9）选择不同的机器人部件（图15），然后使用CTRL和ALT键来调整选中部件的旋转角度（图16）。

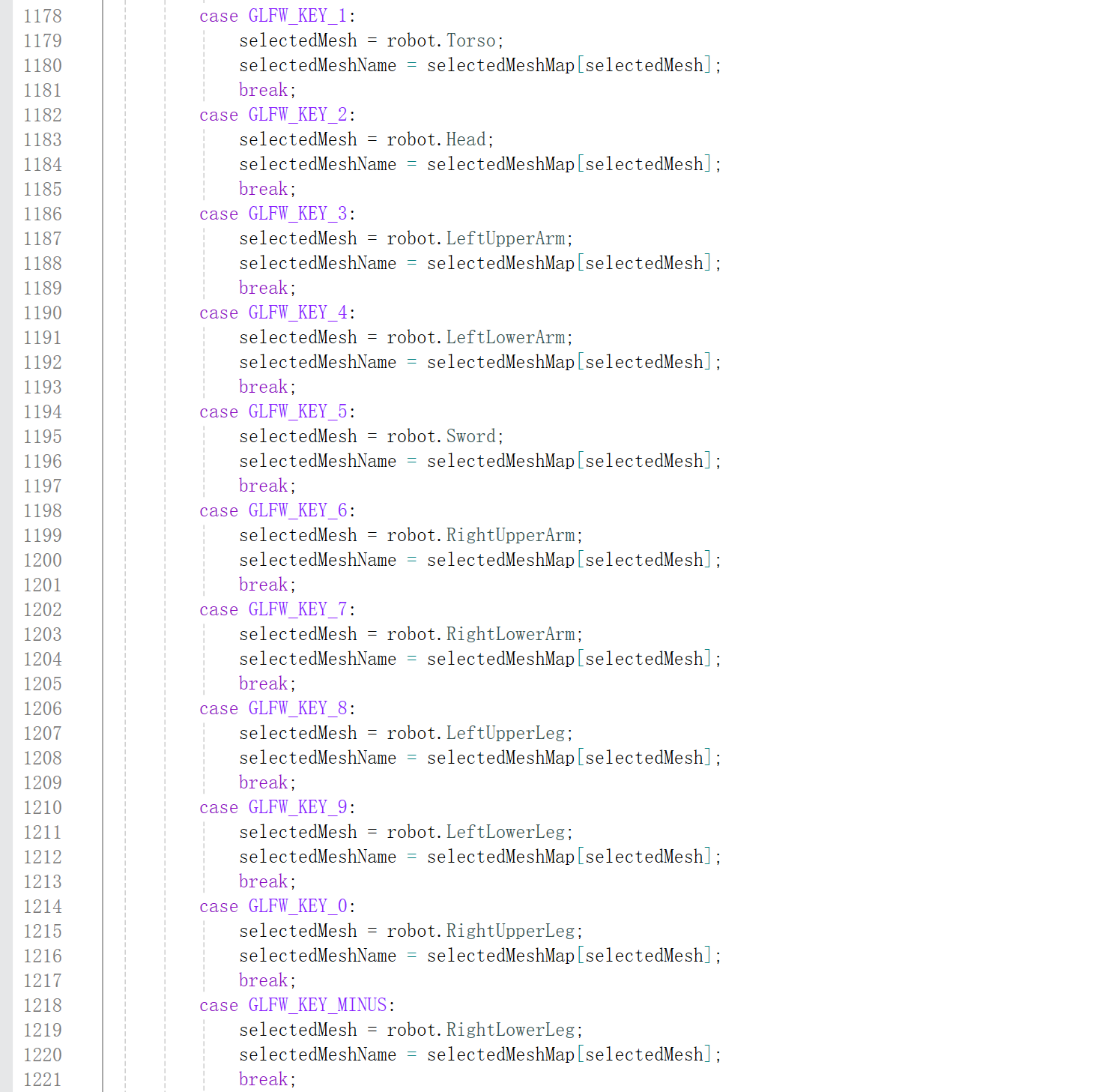


图15数字键选择部件

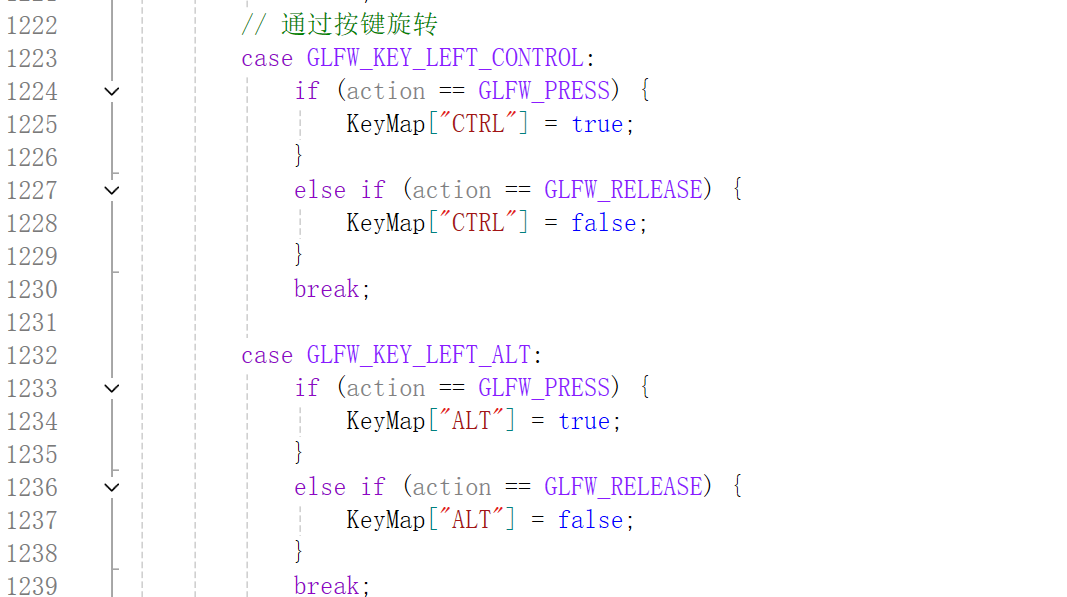


图16调整旋转角度

使用空格键选定控制的3D网格对象（图17）后，按下W、A、S、D键控制物体在三维空间中的前后左右移动（图18a），且每个移动都受到空气墙的限制。



图17空格键选定对象

按下I、J、K、L键则可以移动光源的位置（图18b）。

|  |
| --- |
|  |
| 图18a WASD控制网格对象 |
|  |
| 图18b IJKL控制光源 |

通过这些键盘输入，用户能够实时调整和操控物体的位置、方向和旋转，使得整个场景可以根据用户的需求进行动态变化。

## 2.6机器人的层次建模

一个机器人可以抽象成如图19所示。

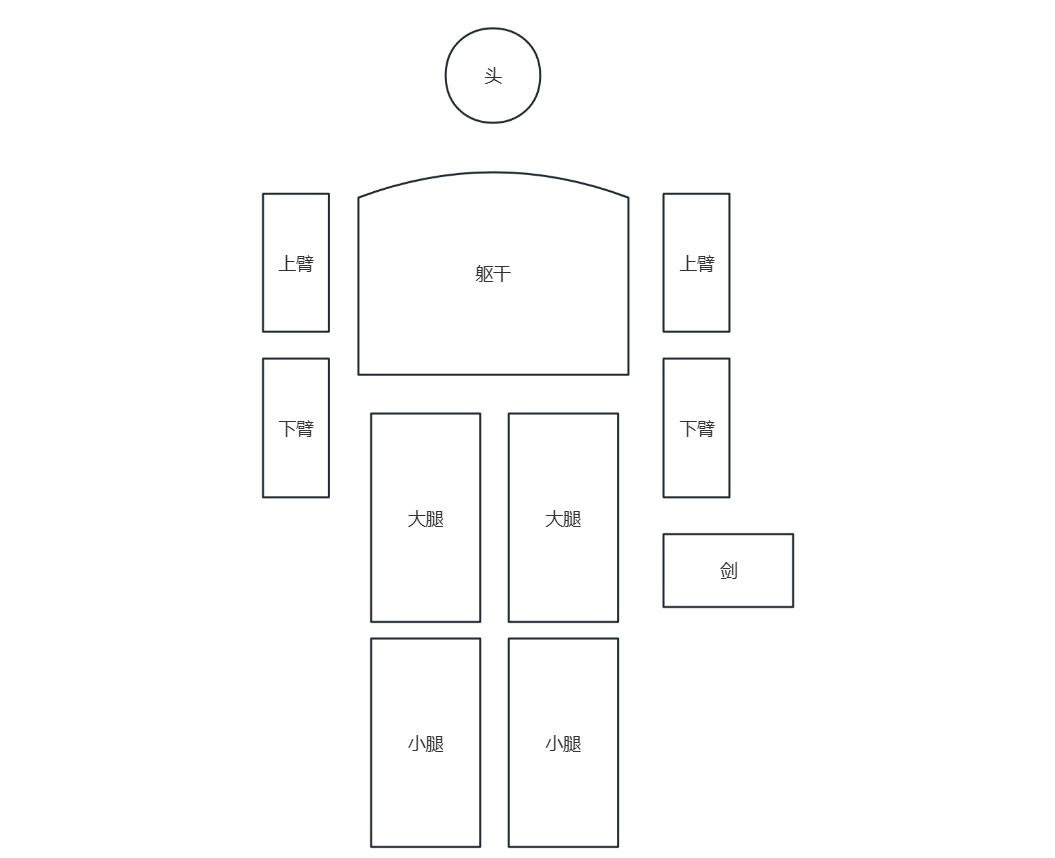


图19抽象机器人

层次建模的关键思想是将每个部件的位置、旋转和缩放与其父部件相对定位，从而形成一个父子关系的层次结构。

机器人的层级结构如图20所示。

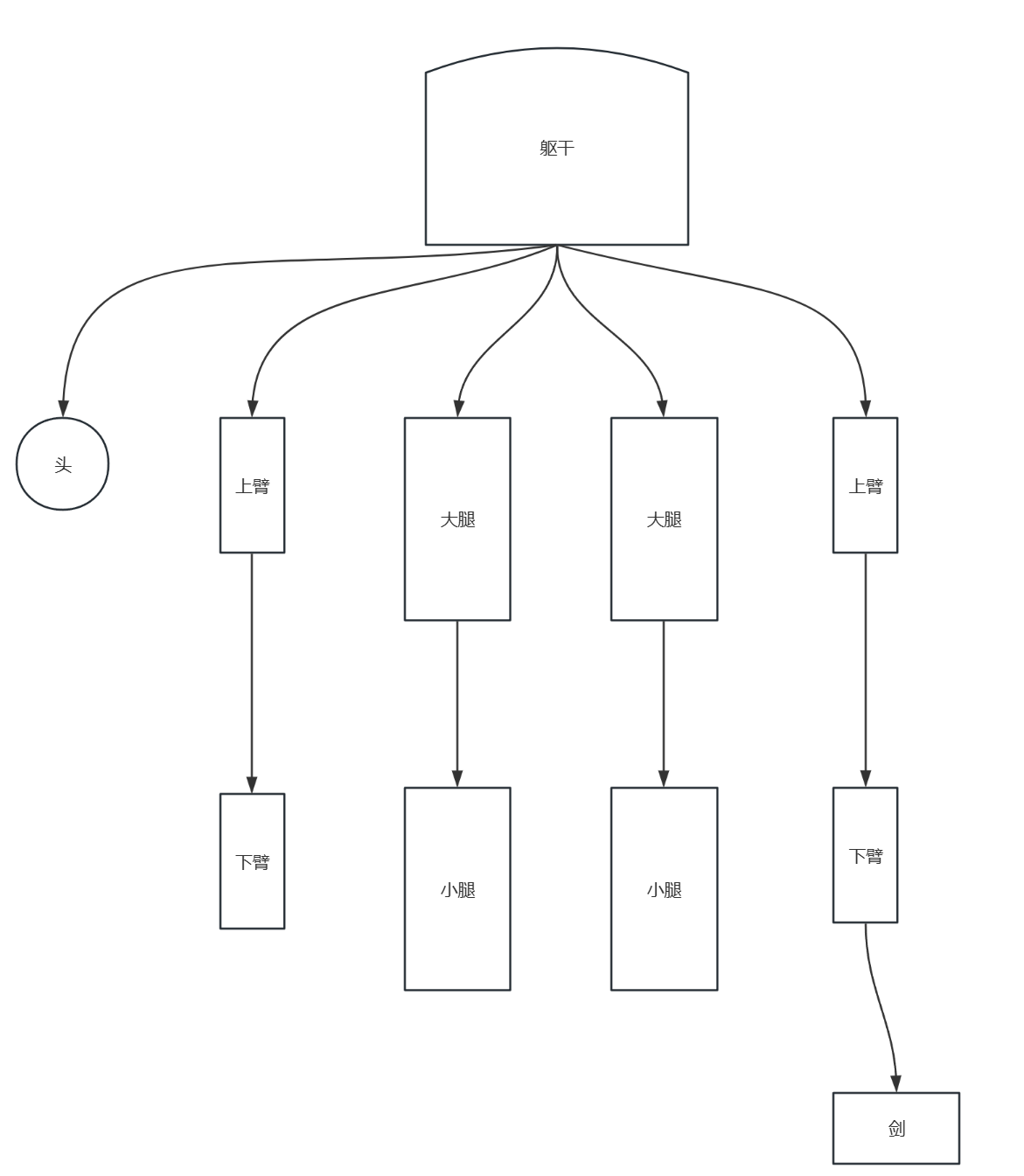


图20机器人的层次结构框架图

首先使用mstack作为一个堆栈来存储和恢复模型矩阵。每个部件，如躯干、头、四肢，都有自己的变换矩阵，这些矩阵通过堆栈进行管理。每当对子部件进行变换时，会先保存当前的模型矩阵（如果还有同一分支的同层次的模型），再应用相应的变换，最后从堆栈中弹出恢复之前的矩阵，确保层次结构正确。

然后是逐层变换，即每个部件的变换都是基于其父部件的变换进行的。通过使用堆栈保存当前的模型矩阵，逐层对每个部件应用相对变换。例如，躯干作为基准，所有其他部件都相对于躯干的位置进行平移、旋转和缩放。每次对子部件进行变换时，先保存当前矩阵，再应用变换后绘制部件，最后恢复父部件的变换矩阵。这种方式保证了每个部件的变换是层次化的，从而确保部件之间的正确相对定位和旋转，最终形成完整的模型。

最后，每次计算完一个部件的变换矩阵，使用drawMesh函数将该部件绘制出来。这个过程会按照层次结构依次执行，确保子部件在父部件变换的影响下进行渲染。

这里给出左上臂、左下臂和剑的层级建模（图21）。

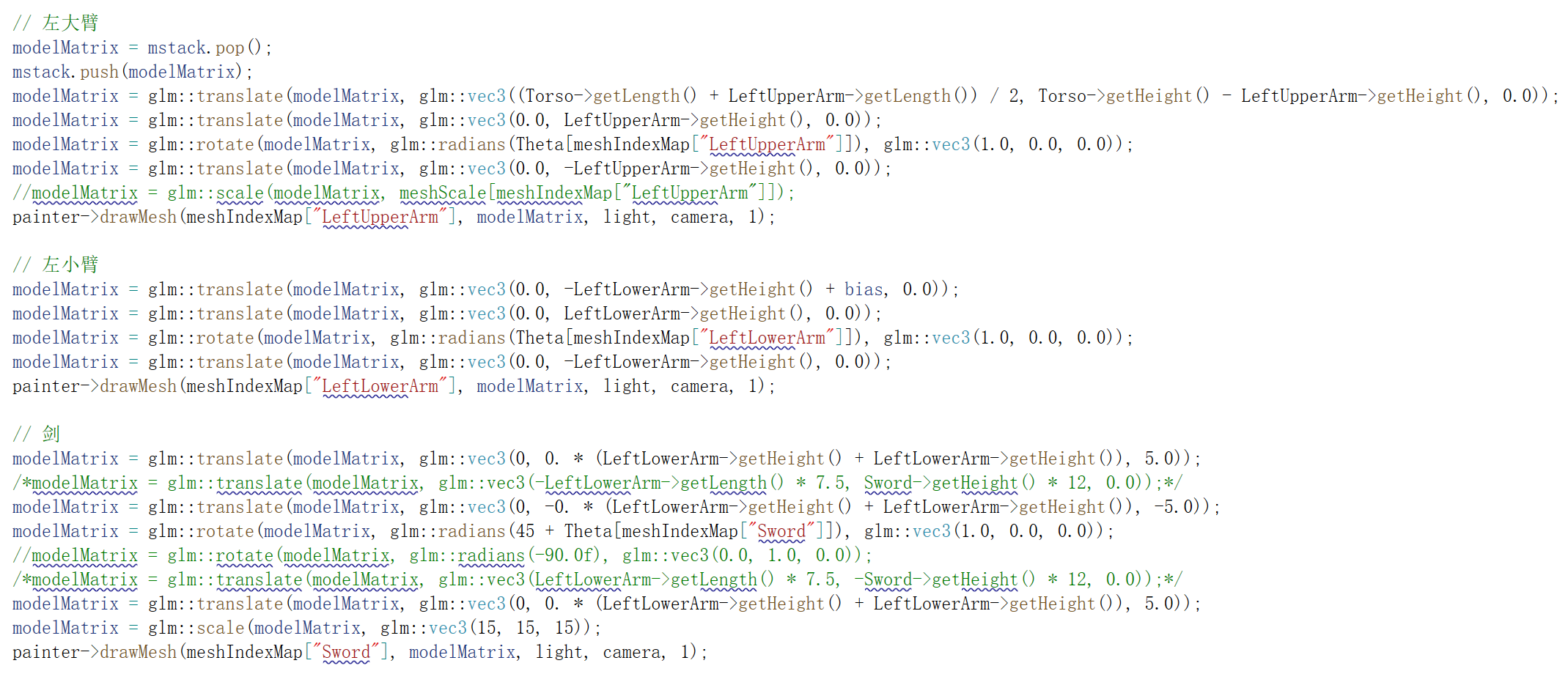


图21机器人的左手层级建模

# 3鼠标和键盘的交互使用方法

## 3.1鼠标交互使用方法

鼠标可以上下左右移动，用于控制相机的视角方向，同时配合键盘按键移动相机位置。

## 3.2键盘的交互使用方法

* **空格**：空格用于切换控制权，分别为相机控制权、三个村民的控制权以及机器人的控制权。
* **WASD**：控制相机、村民或者机器人的移动。

|  |  |
| --- | --- |
|  |  |

图22控制人物移动

* **CTRL和ALT**：1）控制相机的上升与下降，有利于提高相机的操控性，无需鼠标配合键盘就能实现相机的上升与下降；2）控制机器人身体的各个部位的旋转。

|  |  |
| --- | --- |
|  |  |

图23相机上升与下降

* **IJKL**：用于控制太阳模型移动，同时可以改变光照角度，进而改变阴影的朝向。

|  |  |
| --- | --- |
|  |  |

图24太阳移动

|  |  |
| --- | --- |
|  |  |

图25阴影变化

* **数字键以及“-”**：用于将控制权移交给机器人身体的各个部位，按照排列数字键1-9、0、-，它们依次控制的部位为躯干、头部、左上臂、左下臂、剑、右上臂、右下臂、左大腿、左小腿、右大腿、右小腿。

|  |  |
| --- | --- |
|  |  |

图26控制机器人

* **ESC**：退出程序。

# 4 模型绘制结果展示

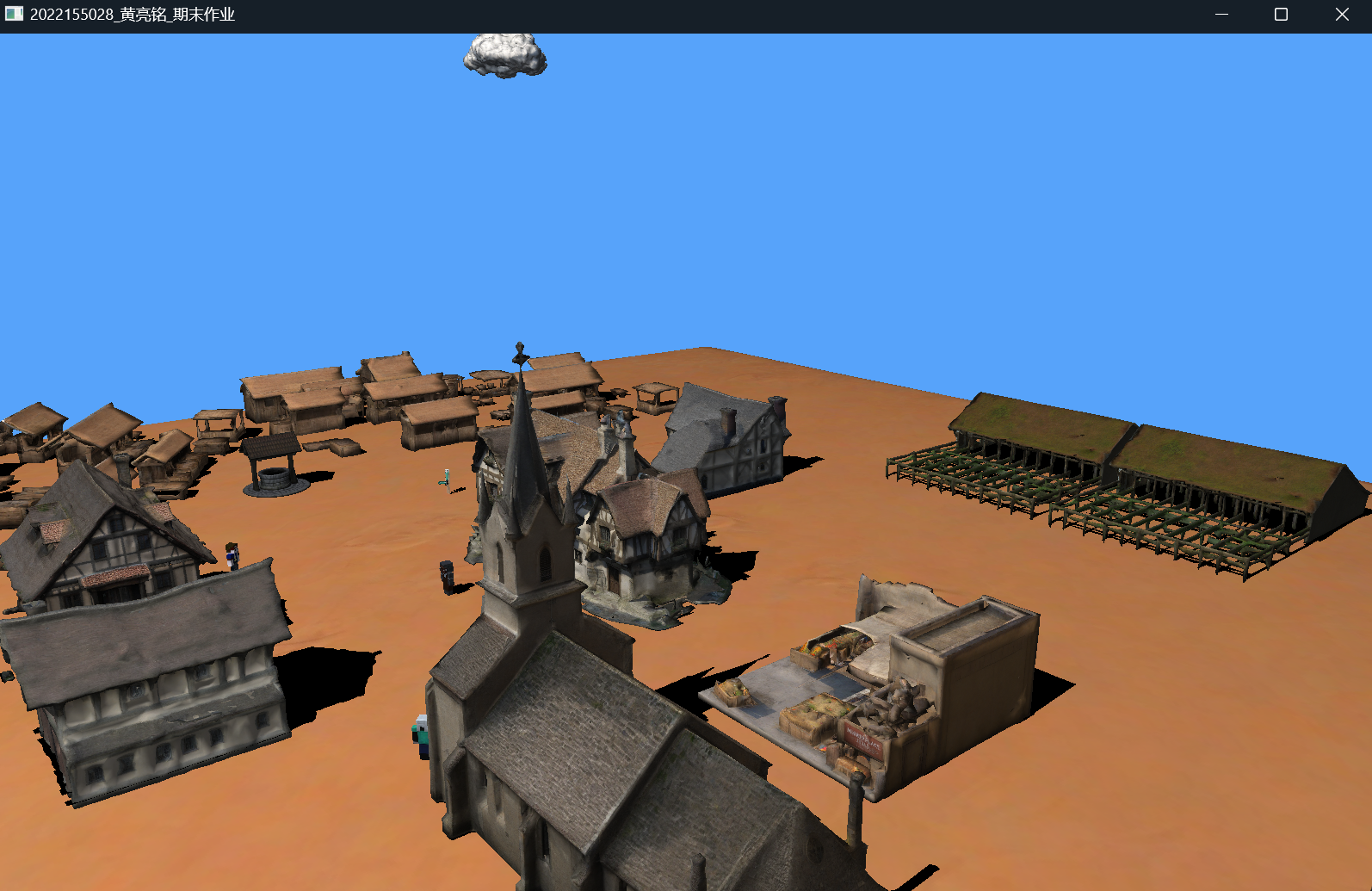


图27场景截图