练 习 题 报 告

课程名称 计算机图形学

项目名称 层级建模

学　　院 计算机与软件学院

专　　业 软件工程（腾班）

指导教师 熊卫丹

报 告 人 洪子敬 学号 **2022155033**

一、 练习目的

1. 了解层级建模基本概念
2. 掌握简单平移，缩放，旋转的矩阵构建
3. 了解变换矩阵在层级模型父子节点间的传递关系
4. 掌握根据层级结构深度遍历层级树的方法
5. 掌握采用堆栈的方式在父子和兄弟节点直接传递变换矩阵的方法。

二．练习完成过程及主要代码说明

此报告包含两个实验的补充，包括层级机械手臂和人形机器人的层级建模。

1. 机械手臂的层级建模

（1）绘制每个节点，完成 base()，upper_arm()，lower_arm()三个函数。

解答：实验代码中已经给出了 base()函数的代码，此函数中是定位底座为原点开始绘制的，在 y 轴方向进行绘制，平移原点只 BASE_HEIGHT 的位置，相当于绘制后使得底座达到落地的效果；详细代码如下：

```
// 绘制底座
void base(glm::mat4 modelView)
{
    // 按长宽高缩放正方体，平移至合适位置
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, BASE_HEIGHT / 2, 0.0));
    instance = glm::scale(instance, glm::vec3(BASE_WIDTH, BASE_HEIGHT, BASE_WIDTH));

    // 绘制，由于我们只有一个立方体数据，所以这里可以直接指定绘制painter中存储的第0个立方体
    painter->drawMesh(0, modelView * instance, light, camera);

}                                                                    HZJ
```

同理对于 upper_arm()函数，我们可以得到代码如下：

```
// 绘制大臂
void upper_arm(glm::mat4 modelView)
{
    // @TODO: 参考底座的绘制，在此添加代码绘制大臂
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, UPPER_ARM_HEIGHT / 2, 0.0));
    instance = glm::scale(instance, glm::vec3(UPPER_ARM_WIDTH, UPPER_ARM_HEIGHT, UPPER_ARM_WIDTH));

    // 绘制，由于我们只有一个立方体数据，所以这里可以直接指定绘制painter中存储的第0个立方体
    painter->drawMesh(0, modelView * instance, light, camera);

}
```

同理对于 lower_arm()函数，我们可以得到代码如下：

```
// 绘制小臂
void lower_arm(glm::mat4 modelView)
{
    // @TODO: 参考底座的绘制，在此添加代码绘制小臂
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, LOWER_ARM_HEIGHT / 2, 0.0));
    instance = glm::scale(instance, glm::vec3(LOWER_ARM_WIDTH, LOWER_ARM_HEIGHT, LOWER_ARM_WIDTH));

    // 绘制，由于我们只有一个立方体数据，所以这里可以直接指定绘制painter中存储的第0个立方体
    painter->drawMesh(0, modelView * instance, light, camera);

}                                                                    HZJ
```

（2）构建子节点局部变化矩阵使底座绕 y 轴旋转、大小臂绕 z 轴旋转，且大臂的旋转中心为大臂与底座的关节，小臂的旋转中心为大小臂的关节，根据遍历顺序完成 display()函数。

解答：display 函数中绘制的是整个模型，使用的是整体的坐标系，坐标的位置变换是通过计算 modelView 矩阵，不断对其进行变换得到的结果；初始将底座往下移动-BASE_HEIGHT 使得其在原点之下进行绘制，接着再整体往上移动 BASE_HEIGHT，使得在原点绘制大臂，最后再往上移动 UPPER_ARM_HEIGHT 进行小臂的绘制即可完成模型绘制；其中的绘制顺序是通过是个深度优先遍历得到的（底座、大臂和小臂），详细代码如下：

```
// 绘制底座
glm::mat4 modelView = glm::mat4(1.0);
modelView = glm::translate(modelView, glm::vec3(0.0, -BASE_HEIGHT, 0.0));// 稍微下移一下，让机器人整体居中在原点
modelView = glm::rotate(modelView, glm::radians(Theta[Base]), glm::vec3(0.0, 1.0, 0.0));// 底座旋转矩阵
base(modelView); // 首先绘制底座

// @TODO: 在此添加代码完成整个机械手臂绘制
// 大臂变换矩阵
modelView = glm::translate(modelView, glm::vec3(0.0, BASE_HEIGHT, 0.0));
modelView = glm::rotate(modelView, glm::radians(Theta[UpperArm]), glm::vec3(0.0, 0.0, 1.0));// 底座旋转矩阵
// 绘制大臂
upper_arm(modelView);
// 小臂变换矩阵
modelView = glm::translate(modelView, glm::vec3(0.0, UPPER_ARM_HEIGHT, 0.0));
modelView = glm::rotate(modelView, glm::radians(Theta[LowerArm]), glm::vec3(0.0, 0.0, 1.0));// 底座旋转矩阵
// 绘制小臂
lower_arm(modelView);
```

2. 人形机器人的层级建模

（1）参考机械臂的绘制过程，完成 torso()、 head()、 left_upper_arm(),、left_lower_arm(),、right_upper_arm()、right_lower_arm(),、left_upper_leg()、 left_lower_leg()、right_upper_leg()和 right_lower_leg()函数的代码补充。

解答：根据前面的实验我们可以知道，这里绘制的都是局部坐标系，都是从局部坐标系的原点开始绘制的；这里为了使人形机器人整体位于中心，把躯体和头部绘制在原点上方，其他部分均绘制在下方，其他部分均与前面类似。详细代码如下：

躯干代码：

```
// 躯体
void torso(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, 0.5 * robot.TORSO_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.TORSO_WIDTH, robot.TORSO_HEIGHT, robot.TORSO_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, Torso, TorsoObject);
}
```

HZJ

头部代码：

```
// 头部
void head(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, 0.5 * robot.HEAD_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.HEAD_WIDTH, robot.HEAD_HEIGHT, robot.HEAD_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, Head, HeadObject);
}
```

HZJ

左大臂代码：

```
// 左大臂
void left_upper_arm(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.UPPER_ARM_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.UPPER_ARM_WIDTH, robot.UPPER_ARM_HEIGHT, robot.UPPER_ARM_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh( modelMatrix * instance, LeftUpperArm, LeftUpperArmObject);
}
```

HZJ

左小臂代码：

```
// @TODO: 左小臂
void left_lower_arm(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.LOWER_ARM_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.LOWER_ARM_WIDTH, robot.LOWER_ARM_HEIGHT, robot.LOWER_ARM_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, LeftLowerArm, LeftLowerArmObject);
}
```

HZJ

右大臂代码：

```
// @TODO: 右大臂
void right_upper_arm(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.UPPER_ARM_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.UPPER_ARM_WIDTH, robot.UPPER_ARM_HEIGHT, robot.UPPER_ARM_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, RightUpperArm, RightUpperArmObject);
}
```

HZJ

右小臂代码：

```
// @TODO: 右小臂
void right_lower_arm(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.LOWER_ARM_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.LOWER_ARM_WIDTH, robot.LOWER_ARM_HEIGHT, robot.LOWER_ARM_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, RightLowerArm, RightLowerArmObject);
}
```

HZJ

左大腿代码：

```
// @TODO: 左大腿
void left_upper_leg(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.UPPER_LEG_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.UPPER_LEG_WIDTH, robot.UPPER_LEG_HEIGHT, robot.UPPER_LEG_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, LeftUpperLeg, LeftUpperLegObject);
}
```

HZJ

左小腿代码：

```
// @TODO: 左小腿
void left_lower_leg(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.LOWER_LEG_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.LOWER_LEG_WIDTH, robot.LOWER_LEG_HEIGHT, robot.LOWER_LEG_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, LeftLowerLeg, LeftLowerLegObject);
}
```

HZJ

右大腿代码：

```
// @TODO: 右大腿
void right_upper_leg(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.UPPER_LEG_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.UPPER_LEG_WIDTH, robot.UPPER_LEG_HEIGHT, robot.UPPER_LEG_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, RightUpperLeg, RightUpperLegObject);
}
```

HZJ

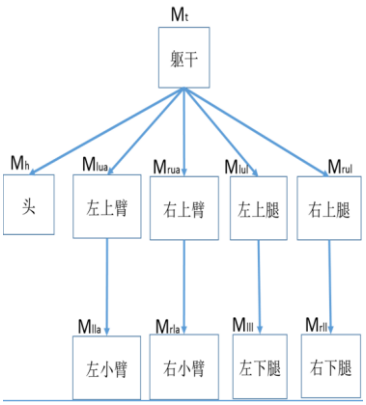
右小腿代码：

```
// @TODO: 右小腿
void right_lower_leg(glm::mat4 modelMatrix)
{
    // 本节点局部变换矩阵
    glm::mat4 instance = glm::mat4(1.0);
    instance = glm::translate(instance, glm::vec3(0.0, -0.5 * robot.LOWER_LEG_HEIGHT, 0.0));
    instance = glm::scale(instance, glm::vec3(robot.LOWER_LEG_WIDTH, robot.LOWER_LEG_HEIGHT, robot.LOWER_LEG_WIDTH));

    // 乘以来自父物体的模型变换矩阵，绘制当前物体
    drawMesh(modelMatrix * instance, RightLowerLeg, RightLowerLegObject);
}
```

HZJ

（2）按深度优先顺序，即“躯干 -> 头 -> 左上臂 -> 左小臂 -> 右上臂 -> 右下臂 -> 左上腿 -> 左下腿 -> 右上腿 -> 右下腿”的顺序完成层级树的遍历，完成 display（）函数。

解答：按照深度优先的遍历顺序进行层级建模绘制，实际上的树结构如下：



从躯干出发，依此遍历每个节点至其叶子节点，实现上由于递归容易导致内存拥塞和其他无法预料的问题，因而这里通过维护一个栈结构来实现对每一部分的绘制，主要是以躯干变换矩阵为中心，往四周进行变换进行各个部分的绘制；唯一需要注意的是节点位置的确定，由于躯干我们绘制在原点，且是在 y 方向进行绘制， 头部，左大小臂，右大小臂，左大小腿，

右大小腿分别是相对躯干的变换，详细坐标如下：

头部： $(0.0, TORSO\_HEIGHT, 0.0)$

左大臂：$(-0.5 * TORSO\_WIDTH - 0.5 * UPPER\_ARM\_WIDTH, TORSO\_HEIGHT, 0.0)$

左小臂：$(0.0, -UPPER\_ARM\_HEIGHT, 0.0)$

右大臂：$(0.5 * TORSO\_WIDTH + 0.5 * UPPER\_ARM\_WIDTH, TORSO\_HEIGHT, 0.0)$

右小臂：$(0.0, -UPPER\_ARM\_HEIGHT, 0.0)$

左大腿：$(-0.5 * TORSO\_WIDTH, 0.0, 0.0)$

左小腿：$(0.0, -UPPER\_LEG\_HEIGHT, 0.0)$

右大腿：$(0.5 * TORSO\_WIDTH, 0.0, 0.0)$

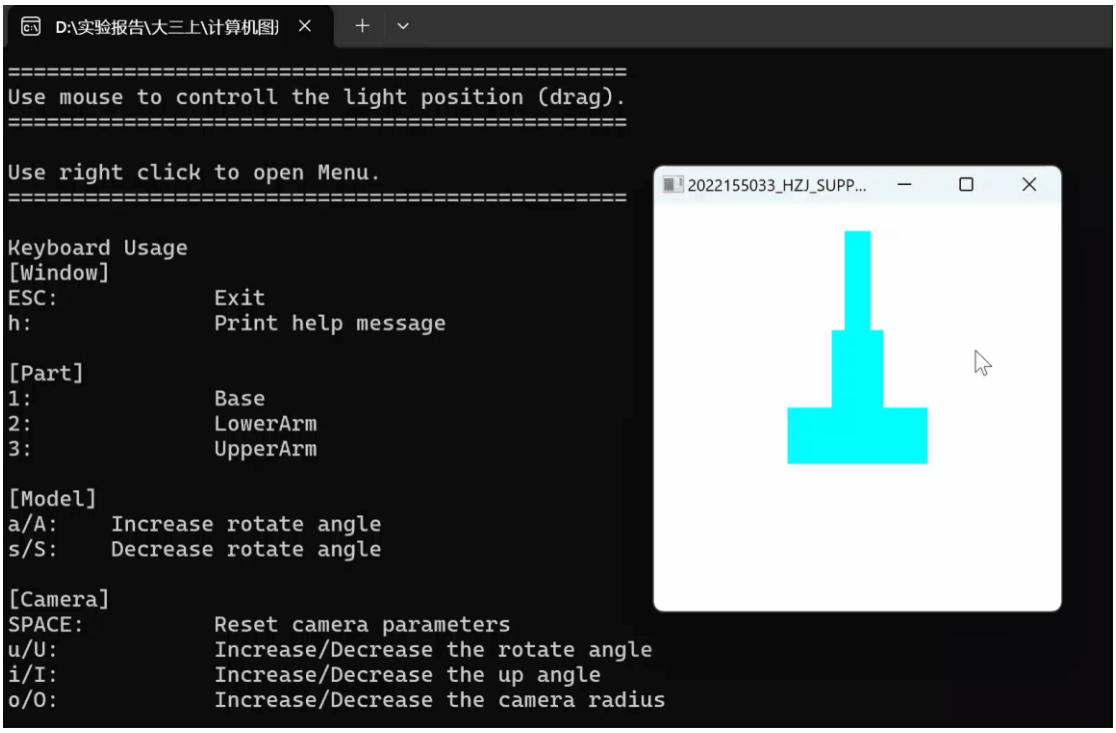
右小腿：$(0.0, -UPPER\_ARM\_HEIGHT, 0.0)$

详细绘制代码如下：

```
// 物体的变换矩阵
glm::mat4 modelMatrix = glm::mat4(1.0);

// 保持变换矩阵的栈
MatrixStack mstack;

// 躯干（这里我们希望机器人的躯干只绕Y轴旋转，所以只计算了RotateY）
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.0, 0.0, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.Torso]), glm::vec3(0.0, 1.0, 0.0));
torso(modelMatrix);

mstack.push(modelMatrix); // 保存躯干变换矩阵
// 头部（这里我们希望机器人的头部只绕Y轴旋转，所以只计算了RotateY）
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.0, robot.TORSO_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.Head]), glm::vec3(0.0, 1.0, 0.0));
head(modelMatrix);
modelMatrix = mstack.pop(); // 恢复躯干变换矩阵


// ========== 左臂 ==========
mstack.push(modelMatrix);   // 保存躯干变换矩阵
// 左大臂（这里我们希望机器人的左大臂只绕Z轴旋转，所以只计算了RotateZ，后面同理）
modelMatrix = glm::translate(modelMatrix, glm::vec3(-0.5 * robot.TORSO_WIDTH - 0.5 * robot.UPPER_ARM_WIDTH, robot.TORSO_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.LeftUpperArm]), glm::vec3(1.0, 0.0, 0.0));
left_upper_arm(modelMatrix);

// @TODO: 左小臂
modelMatrix = glm::translate(modelMatrix, glm::vec3(0, -robot.UPPER_ARM_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.LeftLowerArm]), glm::vec3(1.0, 0.0, 0.0));
left_lower_arm(modelMatrix);
modelMatrix = mstack.pop(); // 恢复躯干变换矩阵                                                                    HZJ

// ========== 右臂 ==========
mstack.push(modelMatrix);   // 保存躯干变换矩阵
// @TODO: 右大臂
// 左大臂（这里我们希望机器人的左大臂只绕Z轴旋转，所以只计算了RotateZ，后面同理）
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.5 * robot.TORSO_WIDTH + 0.5 * robot.UPPER_ARM_WIDTH, robot.TORSO_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.RightUpperArm]), glm::vec3(1.0, 0.0, 0.0));
right_upper_arm(modelMatrix);

// @TODO: 右小臂
modelMatrix = glm::translate(modelMatrix, glm::vec3(0, -robot.UPPER_ARM_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.RightLowerArm]), glm::vec3(1.0, 0.0, 0.0));
right_lower_arm(modelMatrix);
modelMatrix = mstack.pop(); // 恢复躯干变换矩阵

// ========== 左腿 ==========
mstack.push(modelMatrix);   // 保存躯干变换矩阵
// @TODO: 左大腿
modelMatrix = glm::translate(modelMatrix, glm::vec3(-0.5 * robot.TORSO_WIDTH, 0.0, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.LeftUpperLeg]), glm::vec3(1.0, 0.0, 0.0));
left_upper_leg(modelMatrix);

// @TODO: 左小腿
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.0, -robot.UPPER_LEG_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.LeftLowerLeg]), glm::vec3(1.0, 0.0, 0.0));
left_lower_leg(modelMatrix);
modelMatrix = mstack.pop(); // 恢复躯干变换矩阵


// ========== 右腿 ==========
mstack.push(modelMatrix);   // 保存躯干变换矩阵
// @TODO: 右大腿
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.5 * robot.TORSO_WIDTH, 0.0, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.RightUpperLeg]), glm::vec3(1.0, 0.0, 0.0));
right_upper_leg(modelMatrix);

// @TODO: 右小腿
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.0, -robot.UPPER_LEG_HEIGHT, 0.0));
modelMatrix = glm::rotate(modelMatrix, glm::radians(robot.theta[robot.RightLowerLeg]), glm::vec3(1.0, 0.0, 0.0));
right_lower_leg(modelMatrix);
modelMatrix = mstack.pop(); // 恢复躯干变换矩阵                                                                    HZJ
```

3. 实验结果展示：

（1）机械手臂的层级建模



（2）人形机器人的层级建模