深圳大学考试答题纸

(以论文、报告等形式考核专用)

二〇 二四 ～二〇 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 2022155033 姓名 洪子敬 专业年级 22 级软件工程（腾班）

教师评语：

题目： **俄罗斯方块与虚拟场景建模**

宋体五号，至少八页，可以从下一页开始写。

成绩评分栏：

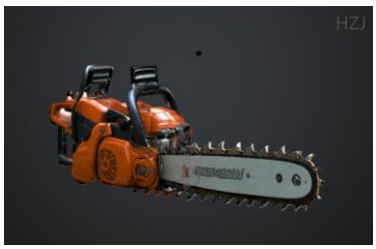
评分项	俄罗斯方块文档（占 12 分）	俄罗斯方块代码（占 24 分）	俄罗斯方块迟交倒扣分（占 0 分）	虚拟场景建模文档（占 16 分）	虚拟场景建模代码（占 38 分）	演示与答辩（占 10 分）	虚拟场景建模迟交倒扣分（占 0 分）	大作业总分
得分								
评分人								

由于俄罗斯方块在前面报告中已经详细叙述过了，这里只展示期末大作业虚拟场景建模；虚拟场景建模我选用在实验 4.2 的基础上进行建模，下面是详细的建模过程：

1. 场景设计和纹理显示

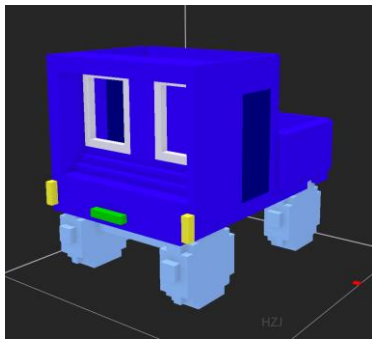
本项目我建立的场景是森林中伐木工的小屋，在 init 函数中用补充实验中的方法对 obj 文件进行读取后，添加到画板上，并导入纹理文件进行绘制；至于对模型的平移、旋转和缩放，在 display 函数中层级建模的方法指定视图矩阵后，根据深度优先的层级建模方式进行绘制即可。

至于本项目选用的模型文件包括大树、苹果树、草地、房子、伐木工、电锯、海绵宝宝、冰墩墩和雪融荣、太阳和皮卡，其中房子和电锯是通过免费的网站生成的，因而有很强的写实感；



伐木工我在场景中绘制了两个，一个是整体进行渲染的，一个是按照层级结构进行渲染的，这两个都是通过 MagicVoxel 进行手动绘制的，后者比较麻烦，需要导出每一部分的 obj 和纹理文件；此外，皮卡也是通过此软件进行绘制的，本来是分开绘制，想控制轮子移动的，但由于时间有限，

没调成功，因而场景中也是直接加载；

而草地是直接调用 generateSquare 函数生成一个长方体，通过加载网上下载的草地图片纹理即可；剩下的模型文件均是从网上找到对应的 obj 和纹理文件，直接加载即可。

下面以伐木工的**整体直接渲染**和**使用层级结构**进行建模过程为例来说明整体流程：

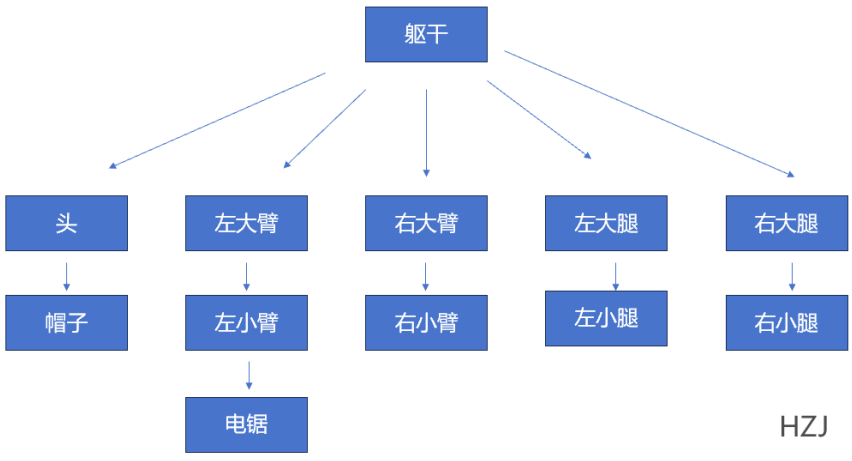
（1）对于伐木工整体的直接渲染：首先实例化一个 TriMesh 对象作为此模型的基础，接着在 init 函数中对此 off 文件进行加载，指定平移旋转等参数后，加载纹理图片进入画板中；这里由于对象太多，所以使用一个 vector 容器进行存放，便于管理；同时使用一个字典文件把模型和索引关联起来，这样可以减少命名，使用方便一些；详细代码如下：

```
IndexMap["qiangge"] = idx++;
qiangge->setNormalize(true);
qiangge->readObj("assets/guangtouqiang/qiangge.obj");
qiangge->setScale(glm::vec3(1.0f, 1.0f, 1.0f));
qiangge->setTranslation(glm::vec3(-0.5f, 0.01f, 10.0f));
painter->addMesh(qiangge, "qiangge", "assets/guangtouqiang/qiangge.png", vshader, tfshader);
meshList.push_back(qiangge);
```

接着在 display 函数中，使用 getModelMatrix 直接调用该模型的模型矩阵（因为不需要其他操作），指定模型，传入画板中进行绘制渲染即可；详细代码如下：

```
modelView = qiangge->getModelMatrix();
painter->drawMesh(IndexMap["qiangge"], modelView, light, camera, 1);
```

（2）对于伐木工的层级结构建模：首先需要确定伐木工的层级结构，如下图所示：



与补充实验一样，使用深度优先的层级建模方式进行绘制，利用栈 stack 结构对模型矩阵进行保存，以躯干为中心进行绘制，这里比较复杂的是第二列，一共有四层需要绘制，在实践时比较难的是确定各个部分的拼接的流畅性，因为这个每个部分都是自己绘制的，与建模好的还是有些差别的，需要去调正 x、y 和 z 轴的偏移实现流畅的拼接。

具体实现时对每一部分均建立一个 TriMesh 对象，各自加载 obj 和纹理文件，指定平移等参数，由于此部分与前面重复，所以不加代码展示；而层级建模的过程编写成一个函数 drawPerson，详细代码如下所示：

```
void drawPerson()
{
    float bias = 0.5;

    // **** 画Body ****
    MatrixStack mstack;
    glm::mat4 modelView = Body->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(0.0, LeftLowerLeg->getHeight() + LeftUpperLeg->getHeight() - bias, 0.0f));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["Body"]]), glm::vec3(0.0, 1.0, 0.0));
    painter->drawMesh(IndexMap["Body"], modelView, light, camera, 1);

    // **** 画头和帽子 ****
    // Head
    mstack.push(modelView);
    modelView = glm::translate(modelView, glm::vec3(0.0, Body->getHeight(), 0.0));
    painter->drawMesh(IndexMap["Head"], modelView, light, camera, 1);
    // Hat
    modelView = glm::translate(modelView, glm::vec3(0.0, Head->getHeight(), 0.0));
    painter->drawMesh(IndexMap["Hat"], modelView, light, camera, 1);

    // **** 画左大臂和左小臂
    // LeftUpperHand
    modelView = mstack.pop();
    mstack.push(modelView);
    modelView = glm::translate(modelView, glm::vec3((Body->getLength() + LeftUpperHand->getLength()) / 2, Body->getHeight() - LeftUpperHand->getHeight(), 0.0));
    modelView = glm::translate(modelView, glm::vec3(- 0.35, LeftUpperHand->getHeight(), 0.0));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["LeftUpperHand"]]), glm::vec3(1.0, 0.0, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, -LeftUpperHand->getHeight(), 0.0));
    painter->drawMesh(IndexMap["LeftUpperHand"], modelView, light, camera, 1);

    // LeftLowerHand
    modelView = glm::translate(modelView, glm::vec3(0.0, -LeftLowerHand->getHeight() + bias -0.5 , 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, LeftLowerHand->getHeight(), 0.0));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["LeftLowerHand"]]), glm::vec3(1.0, 0.0, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, -LeftLowerHand->getHeight(), 0.0));
    painter->drawMesh(IndexMap["LeftLowerHand"], modelView, light, camera, 1);

    // chainsaws
    modelView = glm::scale(modelView, glm::vec3(0.5f, 0.5f, 0.5f));
    modelView = glm::translate(modelView, glm::vec3(LeftLowerHand->getLength() / 12 + 0.40, chainsaws->getWidth()/3 - 0.6 , 2.2));
    modelView = glm::rotate(modelView, glm::radians(-80.0f), glm::vec3(0.0, 1.0, 0.0));
    painter->drawMesh(9, modelView, light, camera, 1);

    // **** 画右大臂和右小臂
    // RightUpperHand
    modelView = mstack.pop();
    mstack.push(modelView);
    modelView = glm::translate(modelView, glm::vec3(-(Body->getLength() + RightUpperHand->getLength()) * 41 / 80, Body->getHeight() - RightUpperHand->getHeight(), 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.35, RightUpperHand->getHeight(), 0.0));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["RightUpperHand"]]), glm::vec3(1.0, 0.0, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, -RightUpperHand->getHeight(), 0.0));
    painter->drawMesh(IndexMap["RightUpperHand"], modelView, light, camera, 1);

    // RightLowerHand
    modelView = glm::translate(modelView, glm::vec3(0.0, -RightLowerHand->getHeight() + bias -0.5, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, LeftLowerHand->getHeight(), 0.0));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["RightLowerHand"]]), glm::vec3(1.0, 0.0, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, -LeftLowerHand->getHeight(), 0.0));
    painter->drawMesh(IndexMap["RightLowerHand"], modelView, light, camera, 1);

    // **** 画左大腿和左小腿 ****
    // LeftUpperLeg
    modelView = mstack.pop();
    modelView = glm::translate(modelView, glm::vec3(0.0, bias / 2, 0.0));
    mstack.push(modelView);
    modelView = glm::translate(modelView, glm::vec3(Body->getLength() * 13.0f / 45.0f, -LeftUpperLeg->getHeight(), 0.0));
    // 为了得到正确的旋转，需要先将腿往下移，绕原点旋转，再恢复原位
    modelView = glm::translate(modelView, glm::vec3(0.0, LeftUpperLeg->getHeight(), 0.0));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["LeftUpperLeg"]]), glm::vec3(1.0, 0.0, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, -LeftUpperLeg->getHeight(), 0.0));
    painter->drawMesh(IndexMap["LeftUpperLeg"], modelView, light, camera, 1);
    // LeftLowerLeg
    modelView = glm::translate(modelView, glm::vec3(0.0, -LeftLowerLeg->getHeight() + bias / 2, 0.0));
    painter->drawMesh(IndexMap["LeftLowerLeg"], modelView, light, camera, 1);

    // **** 画右大腿和右小腿 ****
    // RightUpperLeg
    modelView = mstack.pop();
    mstack.push(modelView);
    modelView = glm::translate(modelView, glm::vec3(-Body->getLength() * 13.0f / 45.0f, -RightUpperLeg->getHeight(), 0.0));
    // 理由同上
    modelView = glm::translate(modelView, glm::vec3(0.0, RightUpperLeg->getHeight(), 0.0));
    modelView = glm::rotate(modelView, glm::radians(Theta[IndexMap["RightUpperLeg"]]), glm::vec3(1.0, 0.0, 0.0));
    modelView = glm::translate(modelView, glm::vec3(0.0, -RightUpperLeg->getHeight(), 0.0));
    painter->drawMesh(IndexMap["RightUpperLeg"], modelView, light, camera, 1);
    // RightLowerLeg
    modelView = glm::translate(modelView, glm::vec3(0.0, -RightLowerLeg->getHeight() + bias / 2, 0.0));
    painter->drawMesh(IndexMap["RightLowerLeg"], modelView, light, camera, 1);
}
```

编写好此函数后直接在 display 函数中进行调用即可，（1）左和（2）右的建立效果如下所示：

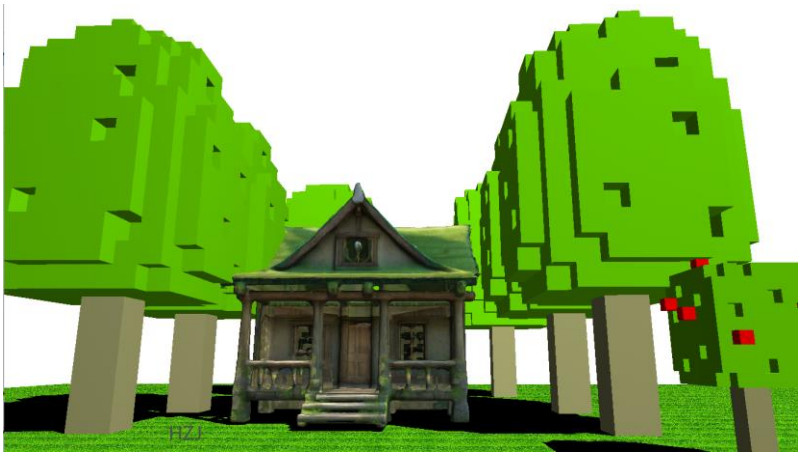

除此之外，我们由于是森林环境，所以需要对树进行多次渲染，所以我在屋子的周围渲染了八棵树（同一模型）；同样地编写成函数 drawTrees，在 display 函数中调用；详细代码如下：

```
void drawTree()
{
    // 房子前面唯一可砍掉的树
    float zoffset = 10.0f;
    glm::mat4 modelView = Tree1->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(3.0f, 0.0f, -5.0f + zoffset));
    modelView = glm::scale(modelView, glm::vec3(1.0f, 1.0f, 1.0f));
    painter->drawMesh(IndexMap["Tree1"], modelView, light, camera, 1);

    //房子右边两棵树
    modelView = Tree2->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(1.0f, 0.0f, -12.5f + zoffset));
    modelView = glm::rotate(modelView, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    modelView = glm::scale(modelView, glm::vec3(5.0f, 5.0f, 5.0f));
    painter->drawMesh(IndexMap["Tree2"], modelView, light, camera, 1);

    modelView = Tree2->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(1.0f, 0.0f, -17.5f + zoffset));
    modelView = glm::rotate(modelView, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    modelView = glm::scale(modelView, glm::vec3(5.0f, 5.0f, 5.0f));
    painter->drawMesh(IndexMap["Tree2"], modelView, light, camera, 1);

    //房子左边两棵树
    modelView = Tree2->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(-11.0f, 0.0f, -12.5f + zoffset));
    modelView = glm::rotate(modelView, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    modelView = glm::scale(modelView, glm::vec3(5.0f, 5.0f, 5.0f));
    painter->drawMesh(IndexMap["Tree2"], modelView, light, camera, 1);

    modelView = Tree2->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(-11.0f, 0.0f, -17.5f + zoffset));
    modelView = glm::rotate(modelView, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    modelView = glm::scale(modelView, glm::vec3(5.0f, 5.0f, 5.0f));
    painter->drawMesh(IndexMap["Tree2"], modelView, light, camera, 1);

    //房子后面两棵树
    modelView = Tree2->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(0.0f, 0.0f, -20.0f + zoffset));
    modelView = glm::scale(modelView, glm::vec3(5.0f, 5.0f, 5.0f));
    painter->drawMesh(IndexMap["Tree2"], modelView, light, camera, 1);

    modelView = Tree2->getModelMatrix();
    modelView = glm::translate(modelView, glm::vec3(-8.0f, 0.0f, -20.0f + zoffset));
    modelView = glm::scale(modelView, glm::vec3(5.0f, 5.0f, 5.0f));
    painter->drawMesh(IndexMap["Tree2"], modelView, light, camera, 1);
}
```

此外，我们可以发现我还在门前绘制了一棵苹果树，这是为了符合伐木工的身份，可以将该树进行“砍掉”，实际上是通过删除模型 vector 中指定索引的模型来实现（不过此处要注意 C++在 vector 中 delete 对象不会删除对应的指针，所以不能进行多次删除，所以需要设置一个变量来控制 delete 次数为 1 次）；这里是通过绑定按键“2”来实现，详细代码如下所示：

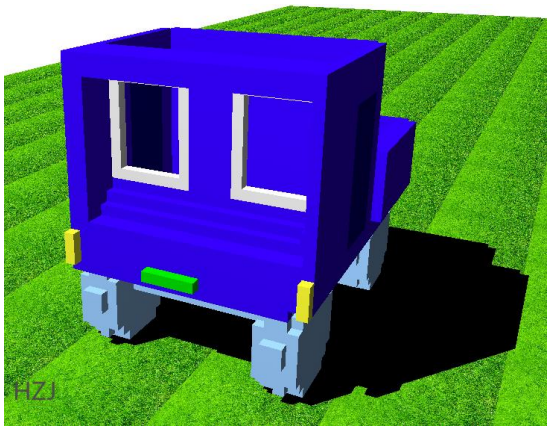
```
case GLFW_KEY_2: // 按下数字2 砍树
    if (action == GLFW_PRESS&&!isTreeChopped)
    {
        delete meshList[15];
        isTreeChopped = true;
    }
    break;
```

渲染后的数目围绕房屋效果如下所示：



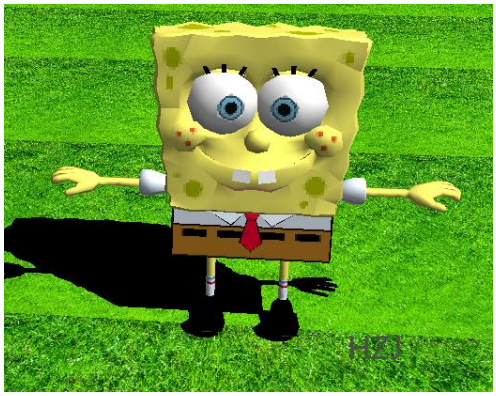
其他模型（底面草地模型不加展示）的渲染效果如下所示：

皮卡（car）模型：



冰墩墩雪融荣（bingdundun）模型：

海绵宝宝（spongebob）模型：



太阳（sun）模型：

HZJ

2. 光照、材质和阴影效果

有前面的实验可知，要实现光照和阴影效果，这里同样在片元着色器 tfshader 中实现，这里为了简单化，当检测到阴影时直接将其绘制成黑色（0.0，0.0，0.0，1.0），否则使用 Phong 光照模型进行环境光、漫反射和镜面反射三种光进行叠加。详细代码如下所示：

```
if (isShadow == 1) {
    fColor = vec4(0.0, 0.0, 0.0, 1.0);
}
else {
    // 将顶点坐标、光源坐标和法向量转换到相机坐标系
    vec3 pos = (vec4(position, 1.0)).xyz;
    vec3 l_pos = (vec4(light.position, 1.0)).xyz;
    vec3 norm = (vec4(normal, 0.0)).xyz;
    vec3 eye = (vec4(eye_position, 1.0)).xyz;

    // @TODO: Task2 计算四个归一化的向量 N,V,L,R(或半角向量H)

    vec3 N = normalize(norm);
    vec3 V = normalize(eye_position - pos);
    vec3 L = normalize(l_pos - pos);
    vec3 H = normalize(L + V);
    /*
    vec3 N = normalize(vNormal);
    vec3 V = normalize(eye_position - vPosition);
    vec3 L = normalize(light.position - vPosition);
    vec3 H = normalize(L + V);
    */

    // 环境光分量I_a
    vec4 I_a = light.ambient * material.ambient;
```

HZJ

```
// @TODO: Task2 计算漫反射系数alpha和漫反射分量I_d
float diffuse_dot = max(dot(L, N), 0.0);
vec4 I_d = diffuse_dot * light.diffuse * material.diffuse;

// @TODO: Task2 计算高光系数beta和镜面反射分量I_s
float s_t = pow(dot(N, H), material.shininess);
float specular_dot_pow = max(s_t, 0.0);
vec4 I_s = specular_dot_pow * light.specular * material.specular;


// @TODO: Task2 计算高光系数beta和镜面反射分量I_s
// 注意如果光源在背面则去除高光
// if( dot(L, N) < 0.0 ) {
//     I_s = vec4(0.0, 0.0, 0.0, 1.0);
// }

// 合并三个分量的颜色，修正透明度
vec4 phone = I_a + I_d + I_s;
fColor = texture2D(texture, texCoord) * phone;
// fColor = vec4(color, 1.0);
// fColor = vec4(normal, 1.0);
// fColor = vec4(normalize(position) , 1.0);
// fColor = vec4(1.0/position.z, position.z, position.z , 1.0);
```

值得注意的是，这里我还定义了另一个片元着色器 NoPhongfshaer，顾名思义，这个是没有使用 Phong 光照模型，主要是为了使底面和太阳绘制出来不要有阴影，否则就和常理出现了违背；因此此处直接是直接加载纹理图片的，不受光照亮度和阴影的影响。详细代码如下：

```
void main()
{
                                                    HZJ
    fColor = texture2D(texture, texCoord);
    // fColor = vec4(color, 1.0);
    // fColor = vec4(normal, 1.0);
    // fColor = vec4(normalize(position) , 1.0);
    // fColor = vec4(1.0/position.z, position.z, position.z , 1.0);
}
```

至于模型的材质，这里均选用了纹理进行贴图，尽管用了材质也看不太出来，所以这里直接不用，全部选用纹理贴图，观感上较为舒服。

此外，为了能够有随时间变化阴影也随之变化的效果，这里设置了光源移动的按键；通过“I”“K”实现 z 轴方向上光源的移动，通过“J”“L”实现 x 轴方向上光源的移动，此时物体的阴影会随之拉长或者变短，按键的设置如下所示：（具体实现请见后面的按键实现函数）

```
//光源控制
case GLFW_KEY_I:
    if (action == GLFW_PRESS) KeyMap["I"] = true;
    else if (action == GLFW_RELEASE) KeyMap["I"] = false;
    break;
case GLFW_KEY_K:
    if (action == GLFW_PRESS) KeyMap["K"] = true;
    else if (action == GLFW_RELEASE) KeyMap["K"] = false;
    break;
case GLFW_KEY_J:
    if (action == GLFW_PRESS) KeyMap["J"] = true;
    else if (action == GLFW_RELEASE) KeyMap["J"] = false;
    break;
case GLFW_KEY_L:
    if (action == GLFW_PRESS) KeyMap["L"] = true;
    else if (action == GLFW_RELEASE) KeyMap["L"] = false;
    break;
}
```

3. 用户交互实现视角切换完成对场景的任意角度浏览

为了实现用户的交互和展现层级建模“动起来”的感觉，这里设置了两个对象“相机”和层级建模的“伐木工”，默认为相机；通过建立一个字典结构{int:string}来表示当前的对象，前面的索引主要是为了在 vector 中找到相应的伐木工躯体，好改变视点位置。这里伐木工的视角实际上是相机的视角加上伐木工的位置来实现对伐木工的视角切换。

具体实现时通过按键“SPACE”进行切换，这里还添加了一个**切换效果**，镜头会往后移动一点距离表示从伐木工切换到了相机，详细代码如下：

```
// 键盘响应函数
std::map<std::string, bool> KeyMap;  // 按键映射，按某个键会激活对应的key，松开某些键会去激活对应的key
bool isTreeChopped = false;
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    float tmp;
    glm::vec4 ambient;
    if (action == GLFW_PRESS || action == GLFW_REPEAT || action == GLFW_RELEASE) {
        switch (key)
        {
            //切换视角
            case GLFW_KEY_SPACE:
                if (action == GLFW_PRESS)
                {
                    CurCtrlMesh = (CurCtrlMesh + 1) % NumCtrlMesh;
                    CurMeshName = CtrlMeshMap[CurCtrlMesh];
                    if (CurMeshName == "camera")
                    {
                        //如果是相机，给个弹出效果提示用户切换到了相机
                        GLfloat angle = Theta[0];
                        glm::vec3 dir = glm::normalize(glm::vec3(sin(glm::radians(angle)), 0.0f, cos(glm::radians(angle))));
                        camera->eye += glm::vec4(-dir + glm::vec3(0.0f, 0.1f, 0.0f), 1.0);
                    }
                }
                break;
```

此外，为了使交互更加流畅，这里通过按键**“W”“S”“A”“D”实现了上下左右的功能**，对于伐木工是实现了平面上行走的功能，对于相机是实现了地面之上任意角度的上升和下降；具体的移动是通过指定移动方向，在该方向上通过 speed*0.05 计算距离后，对相机进行该方向上的距离变换，对于相机对象变换比较直接，直接作用在上面即可；而对于伐木工，实际上就是“有台摄影机跟着伐木工跑”的效果；这里为了防止伐木工跑到平面外面，添加了“**空气墙**”对用户的移动范围进行限制，详细代码如下所示：

```
void process_key_input(GLFWwindow *window)
{
    // 窗口控制
    if (KeyMap["ESCAPE"])
    exit(EXIT_SUCCESS);

    // 光源控制
    if (KeyMap["I"] || KeyMap["K"] || KeyMap["J"] || KeyMap["L"])
    {
        glm::vec3 trans = light->getTranslation();
        // 获得移动方向
        int zdir = 0, xdir = 0;
        if (KeyMap["I"] || KeyMap["K"])
            zdir = KeyMap["I"] ? zdir = -1 : 1;
        else
            xdir = KeyMap["J"] ? xdir = -1 : 1;

        // 移动并加上空气墙限制
        float dist = 0.1 * moveSpeed;
        if (trans.z + zdir * dist < 24.5f && trans.z + zdir * dist > -24.5f) trans.z += zdir * dist;
        if (trans.x + xdir * dist < 24.5f && trans.x + xdir * dist > -24.5f) trans.x += xdir * dist;
        light->setTranslation(glm::vec3(trans.x, trans.y, trans.z));
    }

    // 如果控制的不是相机就进行处理
    if (CtrlMeshMap[CurCtrlMesh] != "camera")
    {
        auto dist = 0.05 * moveSpeed;
        glm::vec3 trans;
        // 获取对应物体的T矩阵
        if (CurMeshName == "Body") trans = Body->getTranslation();
        // 摄像机需要旋转的角度（与人物一致）
        GLfloat angle = Theta[IndexMap[CurMeshName]];
        // 获取人物面向的方向
        glm::vec3 dir = glm::normalize(glm::vec3(sin(glm::radians(angle)), 0.0f, cos(glm::radians(angle))));
        glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);
        // 如果按住左键，则摄像机设置于物体左边，并看向物体，便于观察物体动作
        if (KeyMap["1"])
        {
            camera->eye = glm::vec4(trans - 2.0f * glm::normalize(glm::cross(dir, up)) + glm::vec3(0.0f, 0.5f, 0.0f), 1.0f);
            camera->yaw = -angle + 90.0f;
        }
        // 否则摄像机设置于人物后上方
        else
        {
            camera->eye = glm::vec4(trans - 2.0f * dir + glm::vec3(0.0f, 1.0f, 0.0f), 1.0);
            camera->yaw = -angle;
            camera->upAngle = 15.0f;
        }
    }
    else
    {
        camera->eye = glm::vec4(trans - 2.0f * dir + glm::vec3(0.0f, 1.0f, 0.0f), 1.0);
        camera->yaw = -angle;
        camera->upAngle = 15.0f;
    }
    if ((KeyMap["W"] || KeyMap["S"] || KeyMap["A"] || KeyMap["D"]))
    {
        moveanimation();// W/S控制前进后退，A/D控制左右转向
        if (KeyMap["W"])
            trans += glm::vec3(dist * sin(glm::radians(angle)), 0.0, dist * cos(glm::radians(angle)));
        if (KeyMap["S"])
            trans -= glm::vec3(dist * sin(glm::radians(angle)), 0.0, dist * cos(glm::radians(angle)));
        if (KeyMap["A"])
        {
            camera->yaw = -angle;  // 转弯时强制锁定视角
            if (KeyMap["1"]) camera->yaw += 90.0f;
            // 让当前物体的第0层转向
            Theta[IndexMap[CurMeshName]] += rotateSpeed;
            if (Theta[IndexMap[CurMeshName]] > 360.0f) Theta[IndexMap[CurMeshName]] -= 360.0f;
        }
        if (KeyMap["D"])
        {
            camera->yaw = -angle;  // 转弯时强制锁定视角
            if (KeyMap["1"]) camera->yaw += 90.0f;
            // 让当前物体的第0层转向
            Theta[IndexMap[CurMeshName]] -= rotateSpeed;
            if (Theta[IndexMap[CurMeshName]] < 0.0f) Theta[IndexMap[CurMeshName]] += 360.0f;
        }
        // 记得保存T矩阵
        // 空气墙
        trans.x = std::min(24.5f, trans.x);
        trans.x = std::max(-24.5f, trans.x);
        trans.z = std::min(24.5f, trans.z);
        trans.z = std::max(-24.5f, trans.z);
        if (CurMeshName == "Body")
            Body->setTranslation(trans);
    }
}
// 如果控制的是相机，则将按键传到相机
else camera->keyboard(window);
```

（为了贴合现实世界的效果，这里均使用“透视投影”的效果来实现）

4. 通过交互控制物体

为了能够更加方便地对摄像头进行变换，这里添加了**鼠标事件**对摄像头进行操作，通过移动鼠标可以改变当前对象上下左右视角，但注意为了防止相机上下发生翻转，这里设置其仰角和俯角不能超过 90 度，详细代码如下所示：

```
// 鼠标控制摄像头方向，利用欧拉角
void Camera::mouse(double xpos, double ypos)
{
    // 如果鼠标是第一次移动，直接初始化lastX和lastY
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    // 注意要反过来，因为原点在左上方；
    float yoffset = lastY - ypos;
    lastX = xpos;
    lastY = ypos;

    xoffset *= sensitivity;
    yoffset *= sensitivity;

    yaw += xoffset;
    pitch += yoffset;

    if (pitch > 89.0f)
        //不允许相机颠倒
        pitch = 89.0f;
    if (pitch < -89.0f)
        pitch = -89.0f;
}
```

其次，为了使人的移动更加贴合实际，即不僵硬地移动，这里设置人在进行上下左右移动时会前后臂大小腿会进行**前后摆动**，这里设置摆动幅度不超过 20 度，即看起来像是“走路”的姿态，详细代码如下所示：

```
// 移动时的动画
void moveanimation()
{
    // 摆手动画
    if (CurMeshName == "Body")//当前控制对象时时
    {
        if (Theta[IndexMap["LeftUpperHand"]] > 20.0 || Theta[IndexMap["LeftUpperHand"]] < -20.0)
        {
            //控制摆动方向
            dir = -dir;
            //两个关键帧
            if (dir < 0)
            {
                Theta[IndexMap["LeftUpperHand"]] = 20.00f;
                Theta[IndexMap["RightUpperHand"]] = -20.00f;
                Theta[IndexMap["LeftLowerHand"]] = 0.0f;
                Theta[IndexMap["RightLowerHand"]] = -20.00f;
            }
            else
            {
                Theta[IndexMap["LeftUpperHand"]] = -20.00f;
                Theta[IndexMap["RightUpperHand"]] = 20.00f;
                Theta[IndexMap["LeftLowerHand"]] = -20.00f;
                Theta[IndexMap["RightLowerHand"]] = 0.0f;
            }
        }
        Theta[IndexMap["LeftUpperHand"]] += dir * moveSpeed;
        // 左大臂在身体前面的时候，左小臂才开摆
        if (Theta[IndexMap["LeftUpperHand"]] < 0)
            Theta[IndexMap["LeftLowerHand"]] += dir * moveSpeed;
        Theta[IndexMap["RightUpperHand"]] -= dir * moveSpeed;
        // 右大臂在身体前面的时候，右小臂才开摆
        if (Theta[IndexMap["RightUpperHand"]] < 0)
            Theta[IndexMap["RightLowerHand"]] -= dir * moveSpeed;
        Theta[IndexMap["RightUpperLeg"]] += dir * moveSpeed;
        Theta[IndexMap["LeftUpperLeg"]] -= dir * moveSpeed;
    }
```

此外，我还给对象添加了**加速和减速**的效果，通过按键“Z”“X”实现，但设置了上下界，详细代码如下所示：

```
// 按Z加速
case GLFW_KEY_Z:
    // 相机最高不过 16.0f，物体最高不过 200.0f；
    if (CurMeshName == "camera") camera->cameraSpeedBase = std::min(camera->cameraSpeedBase + 0.5f, 16.0f)
    else
    {
        moveSpeedBase = std::min(moveSpeedBase + 10.0f, 200.0f);
        rotateSpeedBase = std::min(rotateSpeedBase + 10.0f, 200.0f);
    }
    break;
// 按X减速
case GLFW_KEY_X:
    // 相机最低不下 1.0f，物体最低不下 50.0f
    if (CurMeshName == "camera") camera->cameraSpeedBase = std::max(camera->cameraSpeedBase - 0.5f, 1.0f);
    else
    {
        moveSpeedBase = std::max(moveSpeedBase - 10.0f, 50.0f);
        rotateSpeedBase = std::max(rotateSpeedBase - 10.0f, 50.0f);
    }
    break;
```

5. 结果展示

完成上面全部步骤后，基本完成了本项目的虚拟场景建模，除了细微的其他设置没叙述之外，其他均已经详细描述。完成场景后部分全局截图如下所示：（详细按键效果请见提交的 exe 文件）

相机视角砍树前场景 1：

相机视角砍树后场景 2：

伐木工视角走路：

本次项目圆满结束！