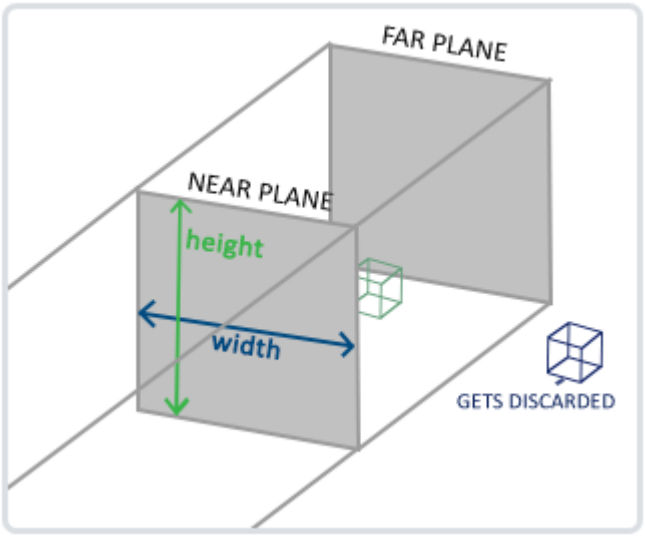
计算机图形学作业（四）：实现正方体的正交、透视投影并实现一个camera类控制摄像头

正交投影

正交投影的示例图如下所示：



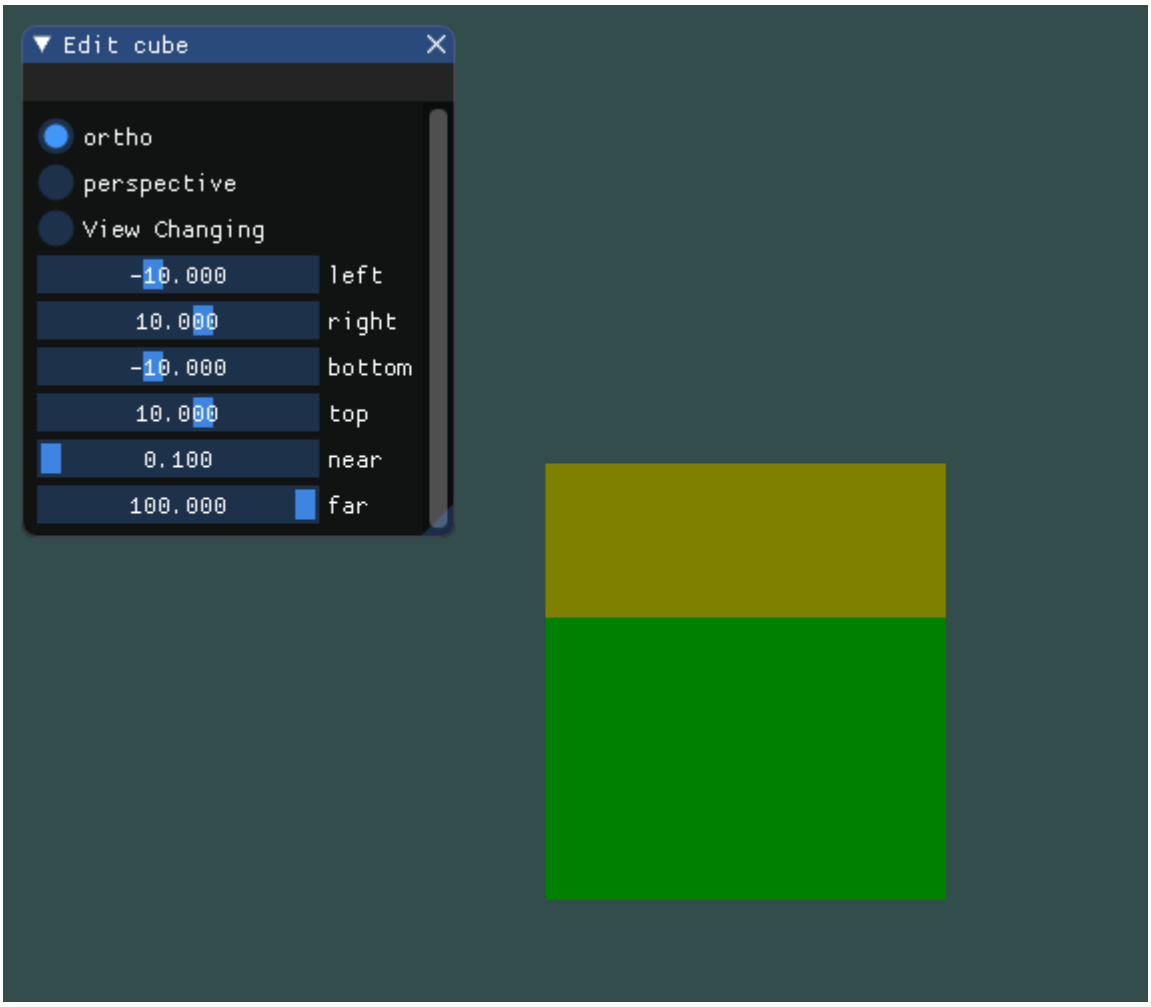
上面的平截头体定义了可见的坐标，它由由宽、高、近(Near)平面和远(Far)平面所指定。任何出现在近平面之前或远平面之后的坐标都会被裁剪掉。这种投影会产生不真实的结果，因为这个投影没有将透视(Perspective)考虑进去。

要在程序中使用正射投影，就用以下代码：

```
projection = glm::ortho(ortho_left, ortho_right, ortho_bottom, ortho_top,
ortho_near, ortho_far);
```

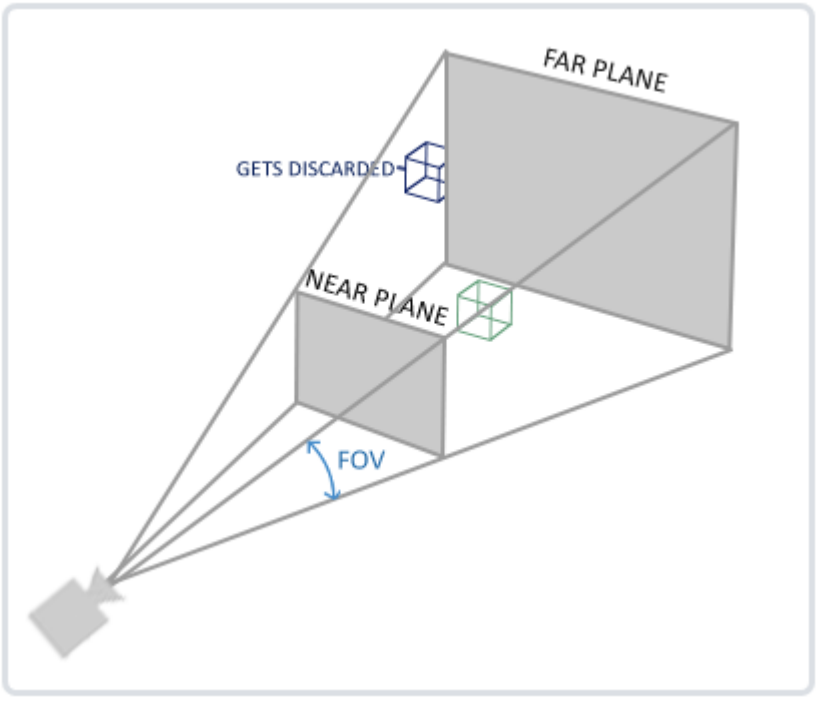
前两个参数指定了平截头体的左右坐标，第三和第四参数指定了平截头体的底部和顶部。通过这四个参数我们定义了近平面和远平面的大小，然后第五和第六个参数则定义了近平面和远平面的距离。

最终效果如下图：



透视投影

透视投影的示例图如下：



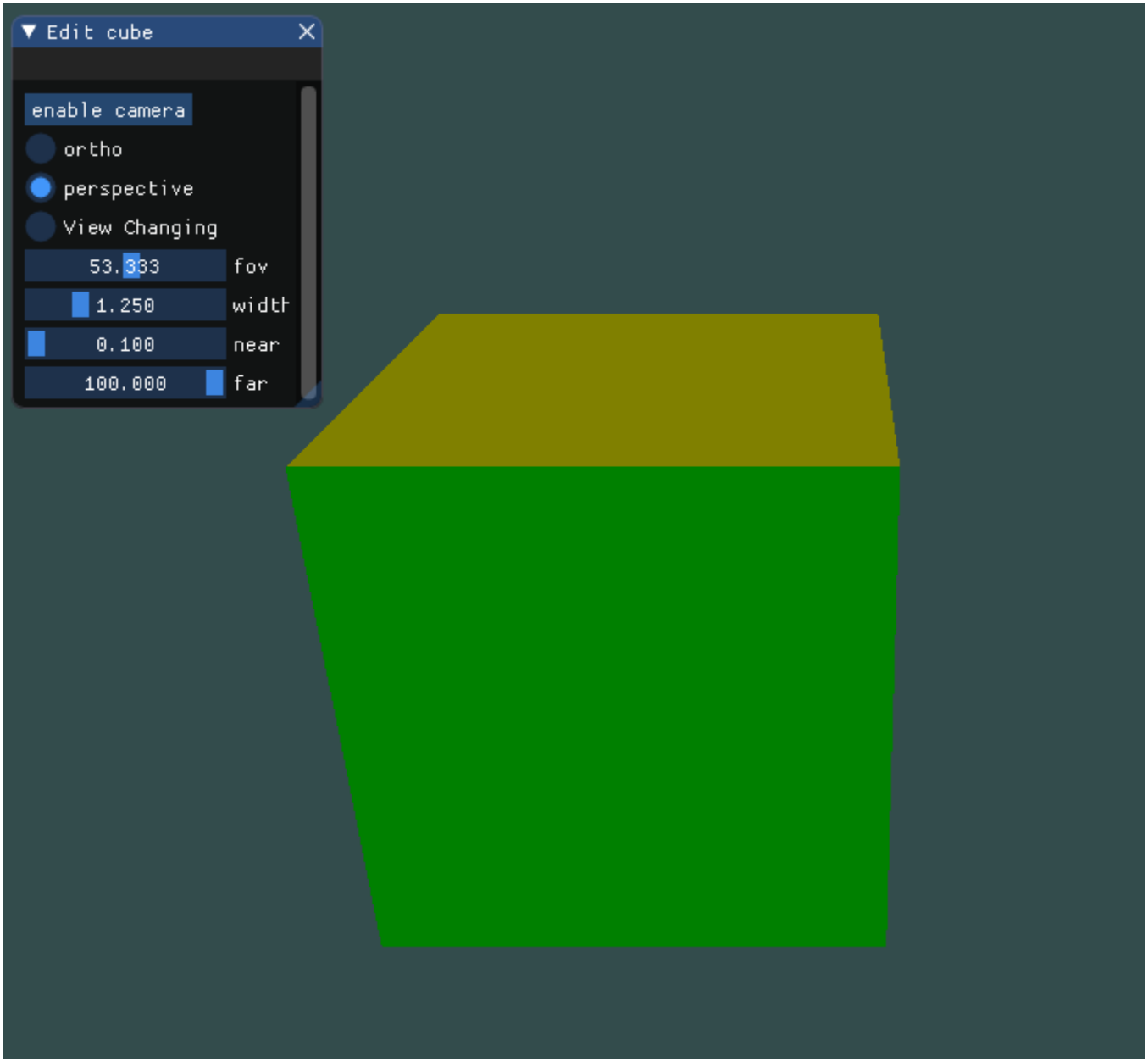
由于透视，你会注意到离你越远的东西看起来更小，正更符合现实生活体验。

要使用透视投影，就是用以下代码：

```
projection = glm::perspective(glm::radians(perspective_fov), perspective_division,
perspective_near, perspective_far);
```

它的第一个参数定义了fov的值，它表示的是视野(Field of View)，并且设置了观察空间的大小，它的值通常设置为45.0f。第二个参数设置了宽高比，由视口的宽除以高所得。第三和第四个参数设置了平截头体的近和远平面。

最终效果图如下：



视角变换

把cube放置在(0, 0, 0)处，做透视投影，使摄像机围绕cube旋转，并且时刻看着cube中心 。

为了达成上述目标，首先我们要创建一个摄像机，也就是view观察矩阵，创建摄像机的观察矩阵，我们会用到lookAt函数，如下所示：

```
glm::mat4 view;
view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f),
           glm::vec3(0.0f, 0.0f, 0.0f),
           glm::vec3(0.0f, 1.0f, 0.0f));
```

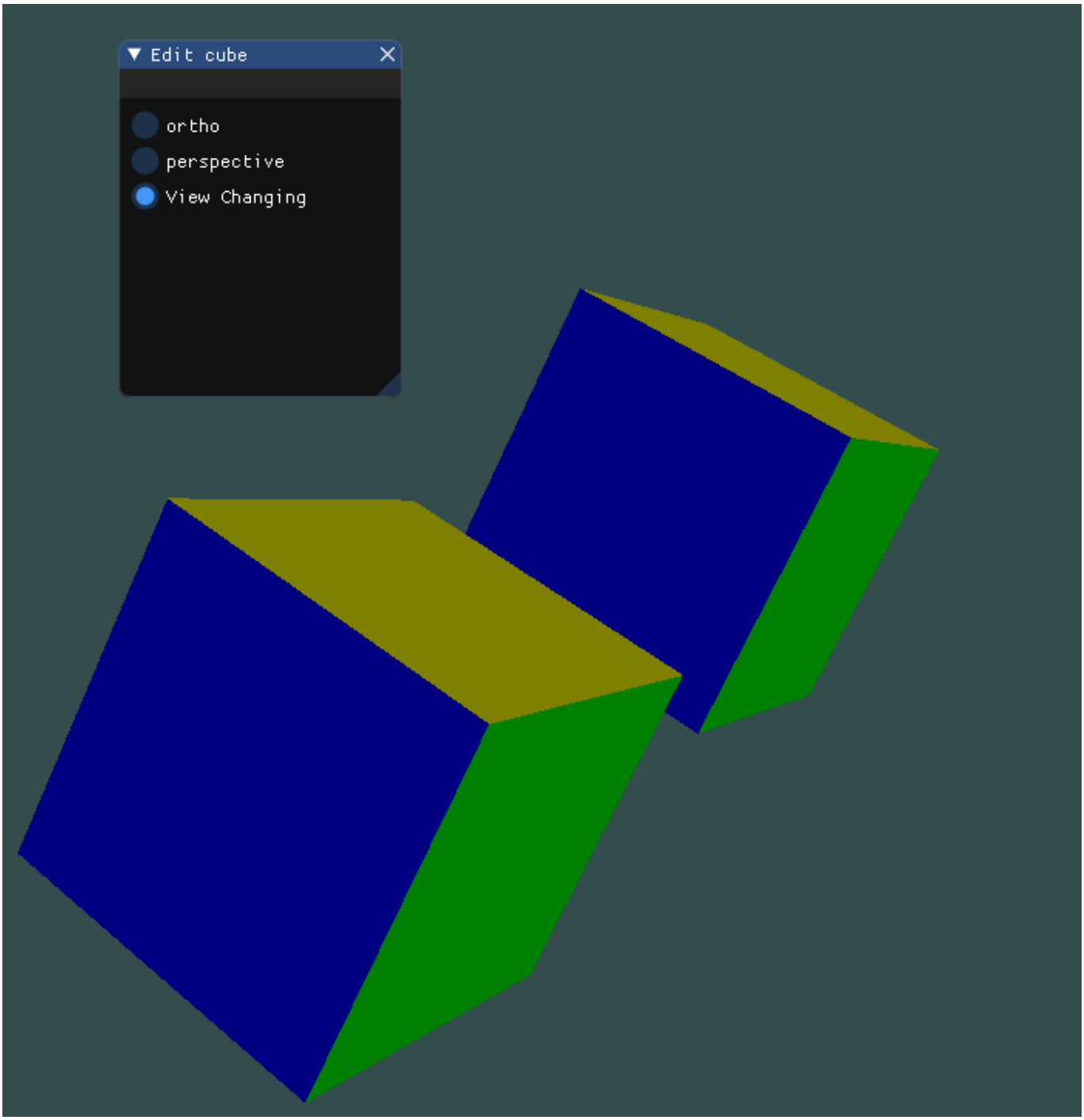
该函数的第一个参数是摄像机位置，第二个参数是要观察的目标的位置，第三个参数是摄像机的上向量。

那么，为了使摄像机绕cube旋转并观察，我们需要修改lookAt函数中的第一个参数，也就是摄像机的位置坐标要呈现出一个圆形的变化。我们用以下方法实现：

```
float radius = 10.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;
glm::mat4 view;
view = glm::lookAt(glm::vec3(camX, 0.0, camZ), glm::vec3(0.0, 0.0, 0.0),
glm::vec3(0.0, 1.0, 0.0));
```

这样的话，摄像机在x0y平面上，就会随时间变化呈现出一个圆形移动。

最终结果如下：



对摄像机空间与物体空间的理解

在现实生活中，我们一般将摄像机摆放的空间View matrix和被拍摄的物体摆设的空间Model matrix分开，但 是在OpenGL中却将两个合二为一设为ModelView matrix，通过上面的作业启发，你认为是为什么呢？

我认为这是OpenGL中固定管线与可编程管线的区别。
在固定管线中，摄像机摆放的空间View matrix和被拍摄的物体摆设的空间Model matrix是合二为一的，固定渲染管线的OpenGLES不需要也不允许你自己去定义顶点渲染和像素渲染的具体逻辑，它内部已经固化了一套完整的渲染流程，只需要开发者在CPU代码端输入渲染所需要的参数并指定特定的开关，就能完成不同的渲染，有一定的局限性。
而在可编程管线中，两种矩阵是分开的，可编程渲染管线的OpenGLES版本必须由开发者自行实现渲染流程，否则无法绘制出最终的画面。开发者可以根据自己的具体需要来编写顶点渲染和像素渲染中的具体逻辑，可最

大程度的简化渲染管线的逻辑以提高渲染效率，也可自己实现特定的算法和逻辑来渲染出固定管线无法渲染的效果。具有很高的可定制性，但同时也对开发者提出了更高的要求。

camera类（加分项）

camera类的源码可以在OpenGL的官方教程中找到：https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/camera.h，只要在项目中引用此头文件即可，重点在于理解和使用该camera类。

移动

为了使用该camera类，首先我们要初始化一个camera类，并初始化一些参数，如下所示：

```
Camera camera(glm::vec3(0.0f, 0.0f, 10.0f));
//初始鼠标位置
float mouseX = 0.0f;
float mouseY = 0.0f;
bool firstMouse = true;
//当前帧与上一帧的时间差
float deltaTime = 0.0f;
//上一帧的时刻
float lastFrame = 0.0f;
```

其中，deltaTime，lastFrame这两个变量是用来衡量当前计算的渲染速度， 如果我们的deltaTime很大，就意味着上一帧的渲染花费了更多时间。当实现camera的移动速度时，我们要结合机器渲染速度，即deltaTime值，使得camera的移动速度较为平滑。

然后我们控制键盘输入的在processInput函数中，增添摄像机移动命令，就可实现摄像机的移动,如下：

```
void processInput(GLFWwindow* window) {
        if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
                glfwSetWindowShouldClose(window, true);
        }

        if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
                camera.ProcessKeyboard(FORWARD, deltaTime);
        if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
                camera.ProcessKeyboard(BACKWARD, deltaTime);
        if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
                camera.ProcessKeyboard(LEFT, deltaTime);
        if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
                camera.ProcessKeyboard(RIGHT, deltaTime);
}
```

视角

然后便是鼠标对摄像机俯视角和偏航角的控制，我们在main.cpp里先声明定义鼠标事件回调函数。如下：

```
//鼠标移动
void mouse_callback(GLFWwindow* window, double xpos, double ypos) {
        if (firstMouse)
        {
                mouseX = xpos;
                mouseY = ypos;
                firstMouse = false;
        }

        float xoffset = xpos - mouseX;
        float yoffset = mouseY - ypos;

        mouseX = xpos;
        mouseY = ypos;

        camera.ProcessMouseMovement(xoffset, yoffset);
}
```

然后再通知glfw使用这回调函数，如下

```
glfwSetCursorPosCallback(window, mouse_callback);
```

还有，最重要的一点是，增加这条配置语句，告诉glfw捕获我们的鼠标移动，否则视角的左右移动将无法大于90度。

```
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
```

放缩

最后是实现滑轮的放缩功能，首先是生命定义滑轮的回调函数，如下：

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset) {
        camera.ProcessMouseScroll(yoffset);
}
```

然后是通知glfw使用滑轮回调函数

```
glfwSetScrollCallback(window, scroll_callback);
```

最后是在cube的投影矩阵，使用camera类的zoom放缩参数

```

```

```
projection = glm::perspective(glm::radians(camera.Zoom), perspective_division,
perspective_near, perspective_far);
```

效果

最终结果如下所示：