HomeWork -- Camera

16340220 王培钰 电子政务

Basic：

1. 投影(Projection):

- **把上次作业绘制的cube放置在(-1.5, 0.5, -1.5)位置，要求6个面颜色不一致。**

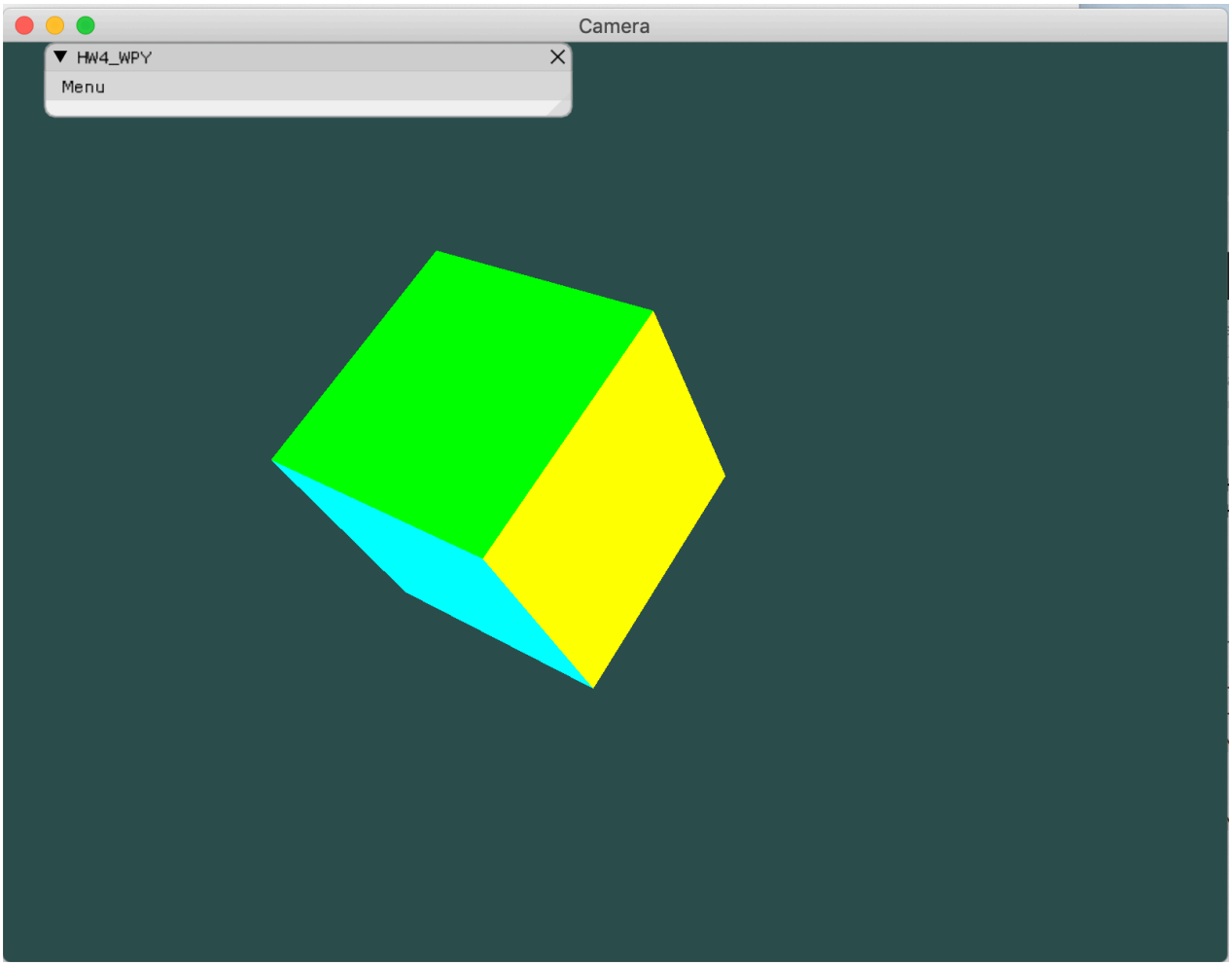
```
float vertices[] = {
        // 顶点位置             // 颜色坐标
        -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
        2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
        2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
        2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
        -2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
        -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f,

        -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        -2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,

        -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 1.0f,
        -2.0f,  2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
        -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
        -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
        -2.0f, -2.0f,  2.0f, 0.0f, 0.0f, 1.0f,
        -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 1.0f,

        2.0f,  2.0f,  2.0f, 1.0f, 1.0f, 0.0f,
        2.0f,  2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
        2.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
        2.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
        2.0f, -2.0f,  2.0f, 1.0f, 1.0f, 0.0f,
        2.0f,  2.0f,  2.0f, 1.0f, 1.0f, 0.0f,

        -2.0f, -2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
        2.0f, -2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
        2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
        2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
        -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
        -2.0f, -2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
```

```
        -2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 1.0f,
        2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 1.0f,
        2.0f,  2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
        2.0f,  2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
        -2.0f,  2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
        -2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 1.0f

    };
```

将6个面分别设置为不同的颜色。

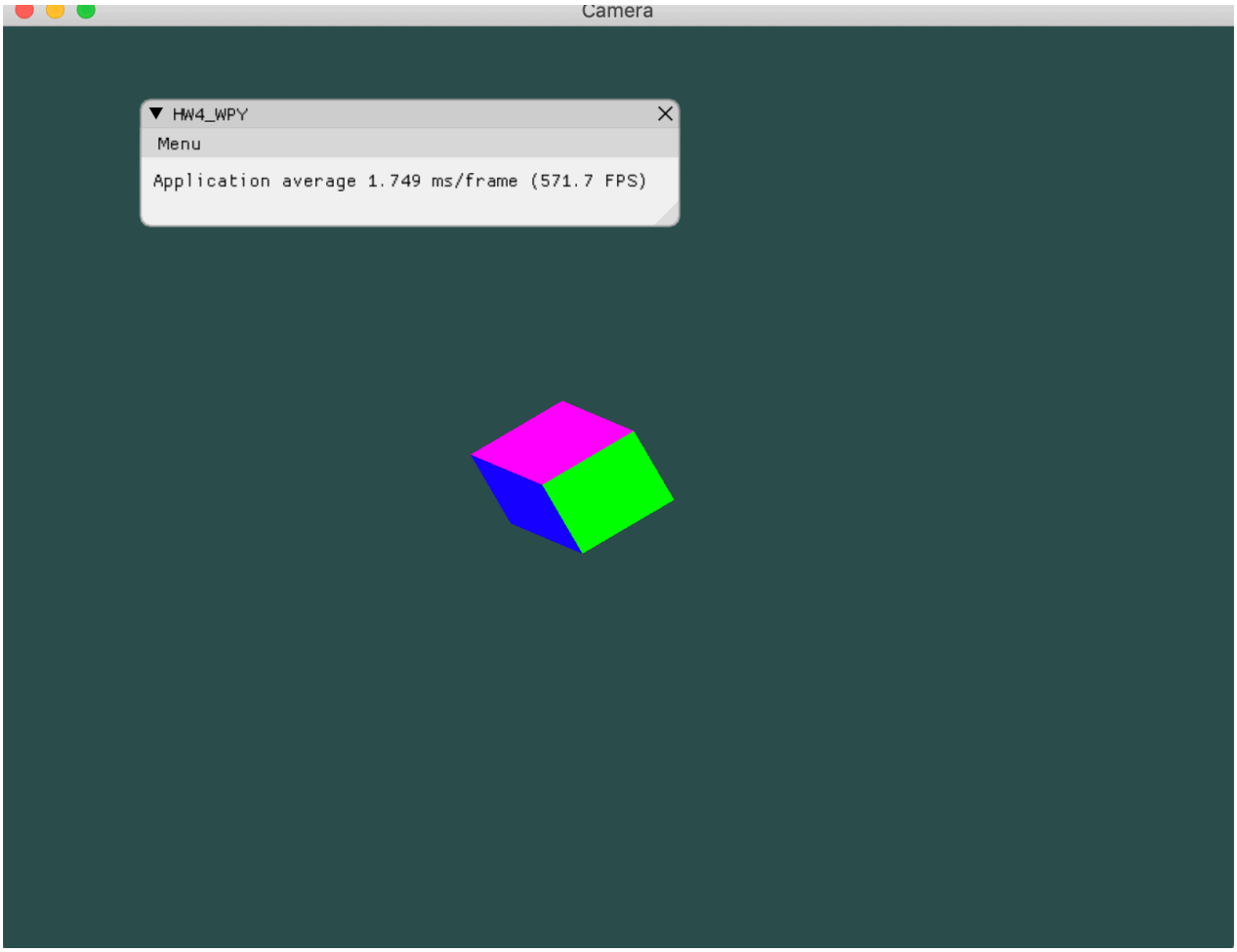
```
model = glm::translate(model, glm::vec3(-1.5f, 0.5f, -1.5f));
```

设置中心坐标为(-1.5, 0.5, -1.5)。

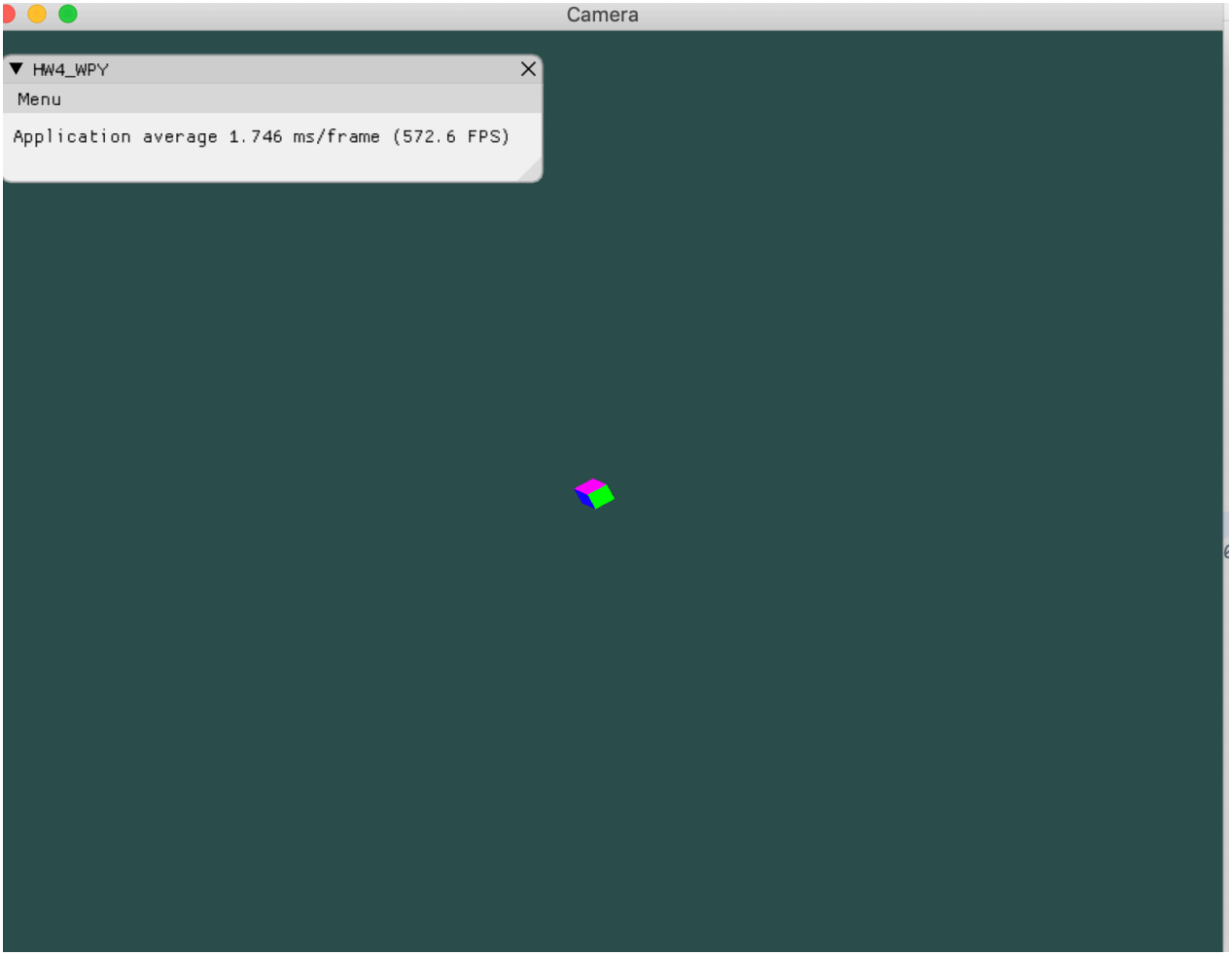


- **正交投影(orthographic projection):实现正交投影，使用多组(left, right, bottom, top, near, far)参数， 比较结果差异。**

选择参数(-10.0f, 10.0f, -10.0f, 10.0f, 0.1f, 100.0f)



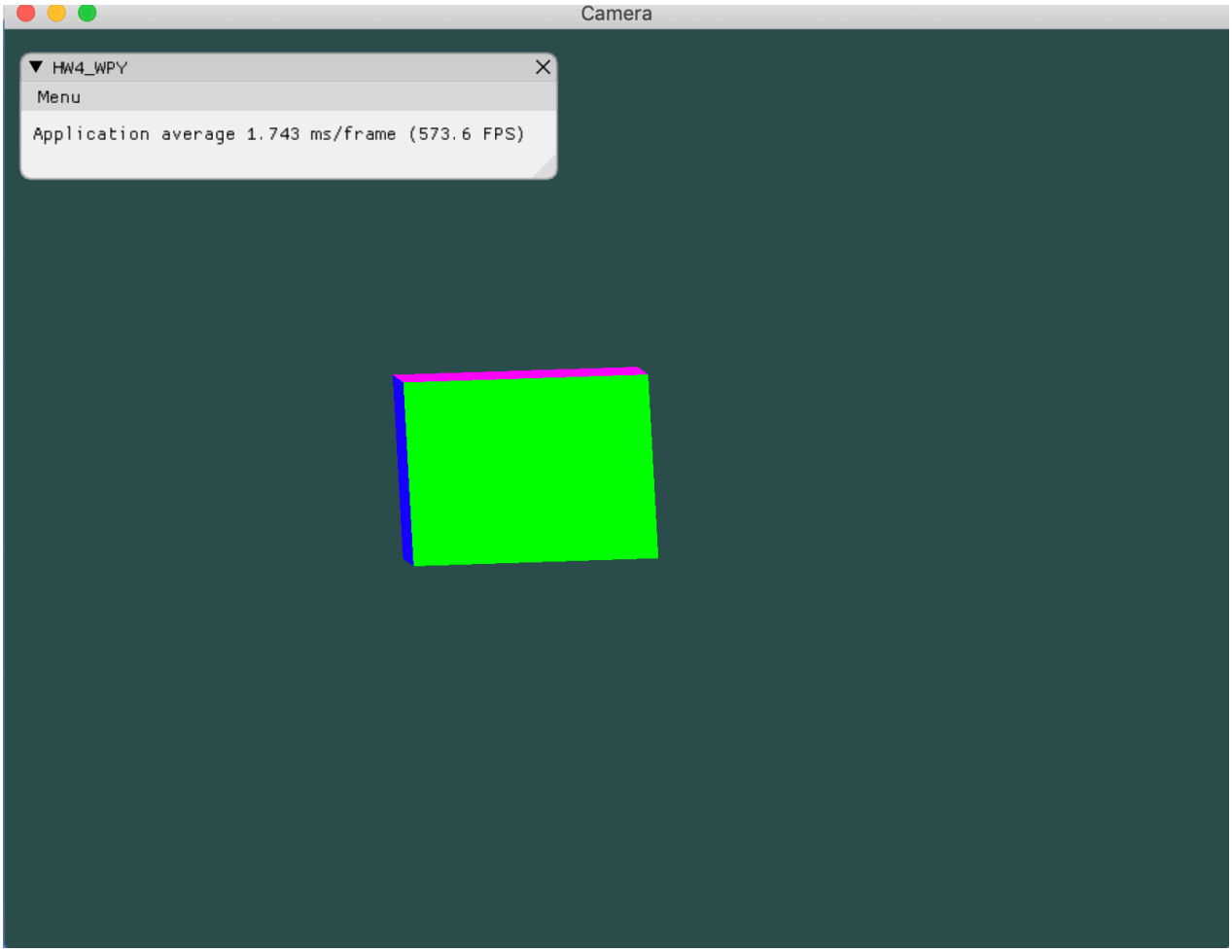
调节前四个参数也就是宽高，会发现投影的范围变大，物体在标准化坐标范围内的视觉效果也变小。



然后调节近平面和远平面的距离，开始时认为这个距离是相对到原点的距离，通过调节发现其实是到摄像机也就是view的距离。

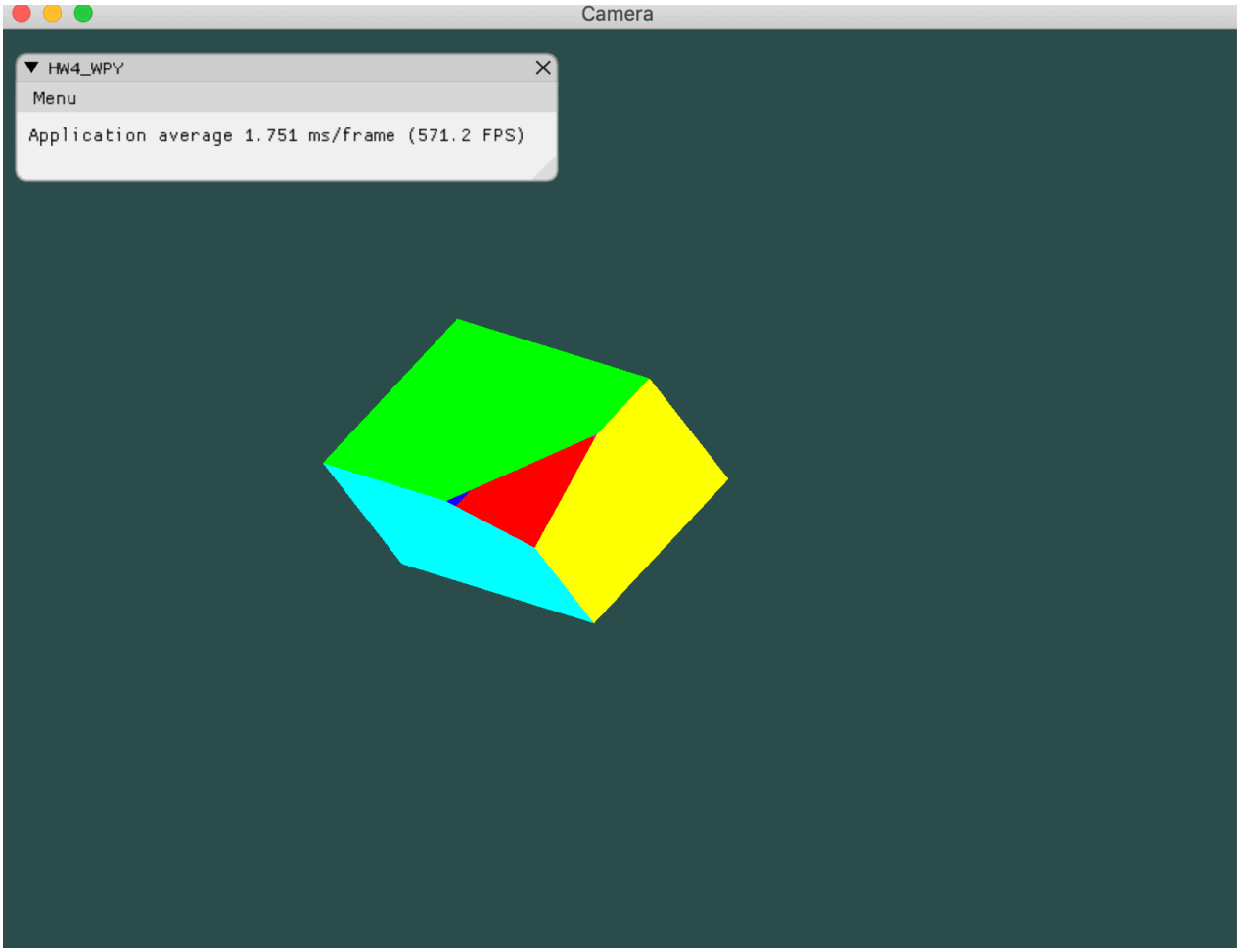
```
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -10.0f));
projection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 8.0f, 50.0f);
```

效果为：




```
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -10.0f));
projection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 9.0f, 50.0f);
```

效果为：

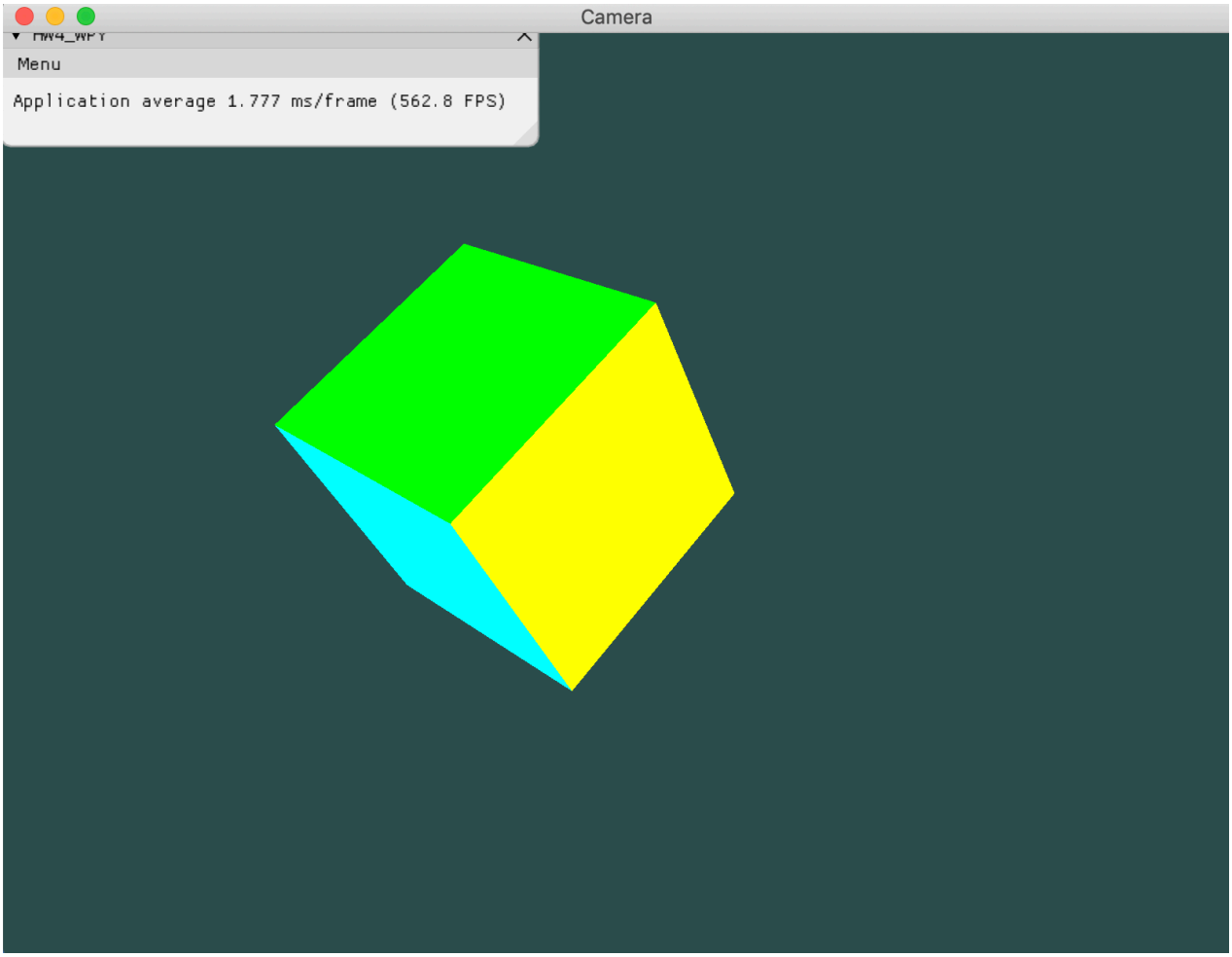


出现如上效果是因为设置的立方体边长为-2~2，9.0f意味着-10.0+9.0=-1.0的位置插入近平面，所以会截掉一部分。

- **透视投影(perspective projection):实现透视投影，使用多组参数，比较结果差异 。**

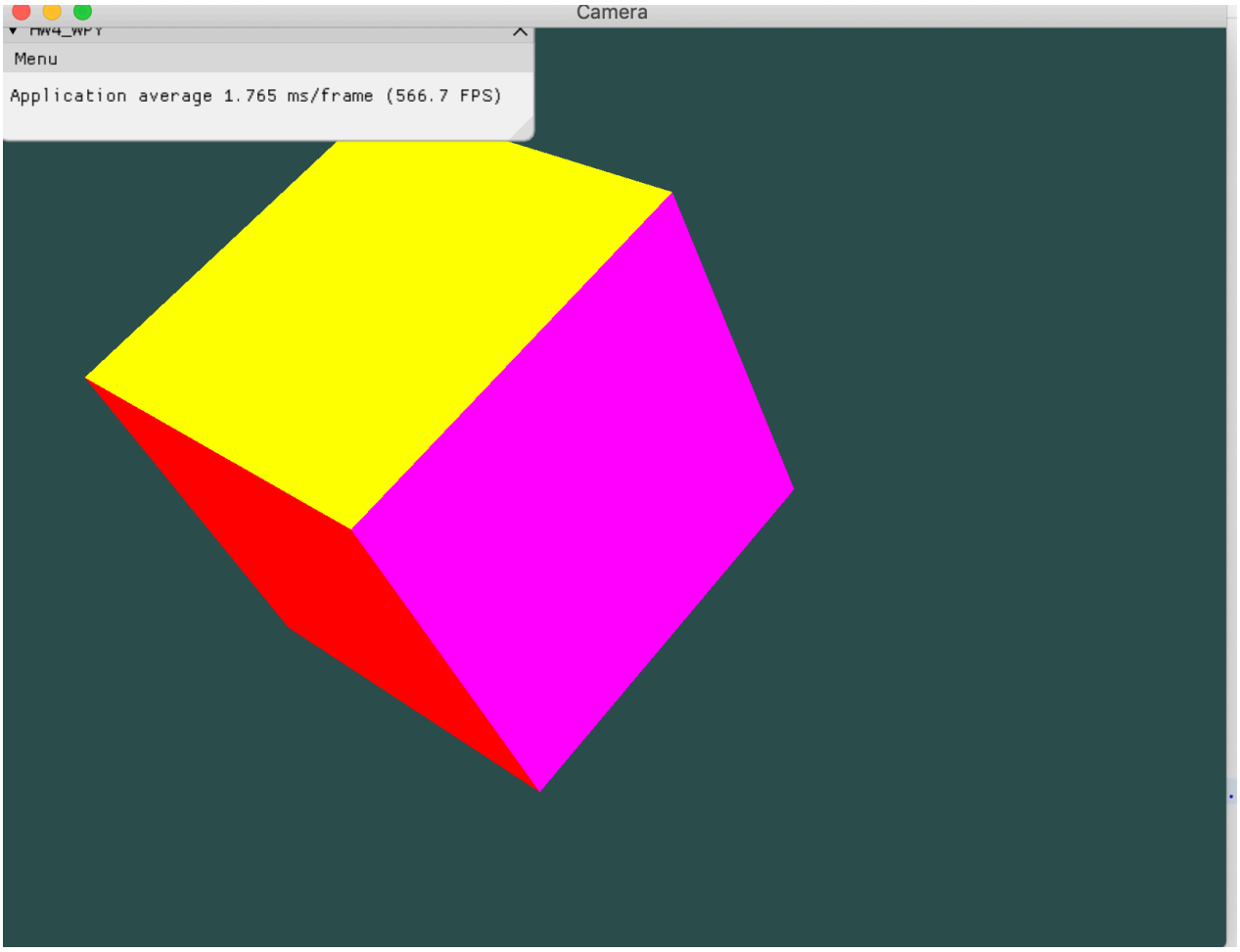
将fov角设置为45度。

```
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -15.0f));
projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH /
(float)SCR_HEIGHT, 0.1f, 100.0f);
```



保持其他参数不变，将fov角调为30度。

```
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -15.0f));
projection = glm::perspective(glm::radians(30.0f), (float)SCR_WIDTH /
(float)SCR_HEIGHT, 0.1f, 100.0f);
```

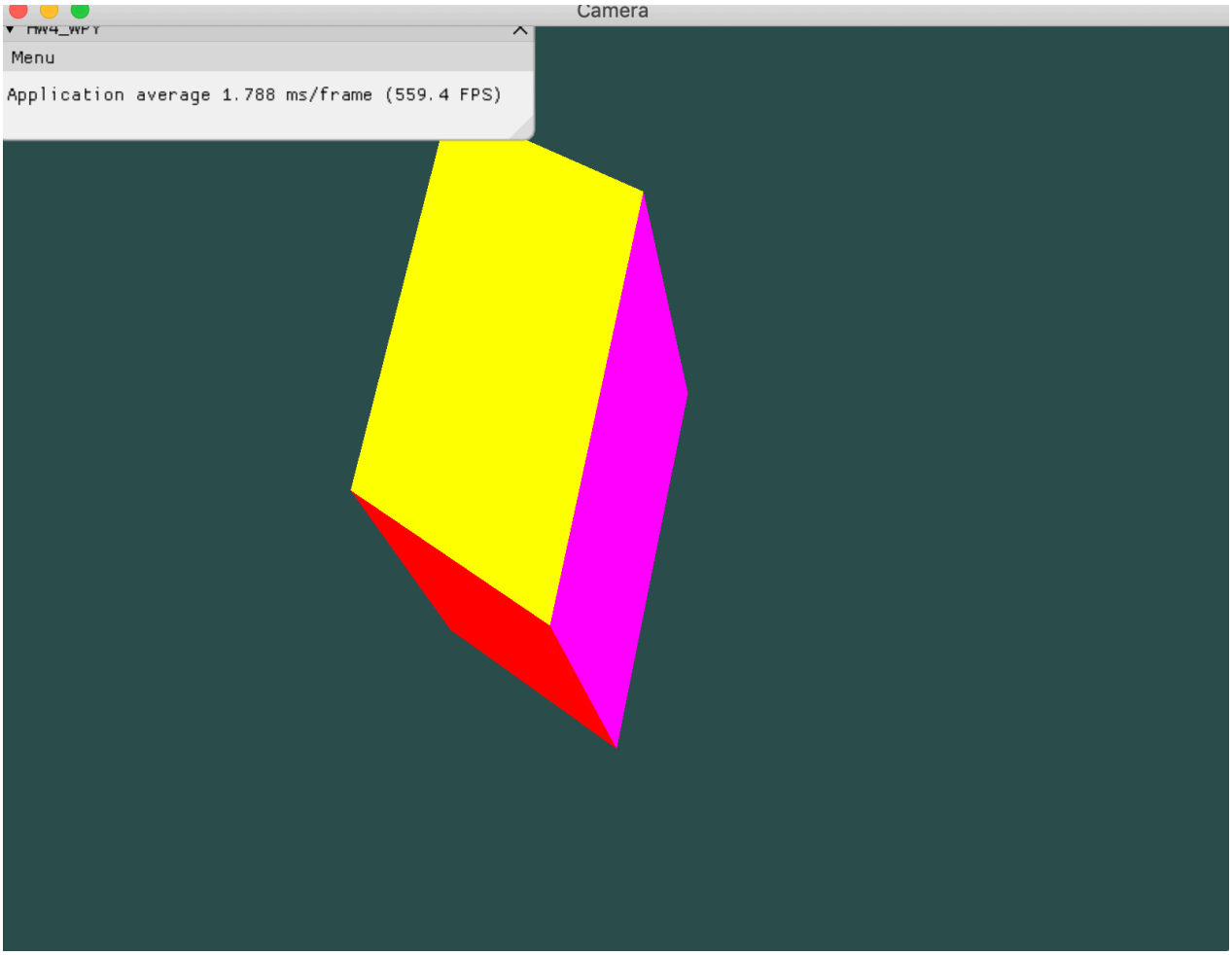


fov角度越小，透视投影出的立方体越大。

接着调整宽高比的大小。

```
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -15.0f));
projection = glm::perspective(glm::radians(30.0f), 2 * (float)SCR_WIDTH /
(float)SCR_HEIGHT, 0.1f, 100.0f);
```

效果如下：



也就是将宽度所占的比重调高，显然可以看出物体沿屏幕高度方向被拉伸，所以要想保持合适的视觉效果最好宽高比就还是设置为屏幕的真正宽高比。

2. 视角变换(View Changing):

- **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。**

用LookAt矩阵创建摄像机的观察空间。LookAt矩阵的三个参数分别是摄像机位置，目标位置，上向量。

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

D是方向向量

```
glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f); //目标位置
glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget);
```

R是右向量

```
glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f); //上向量
glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection));
```

U是上轴

```
glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight);
```

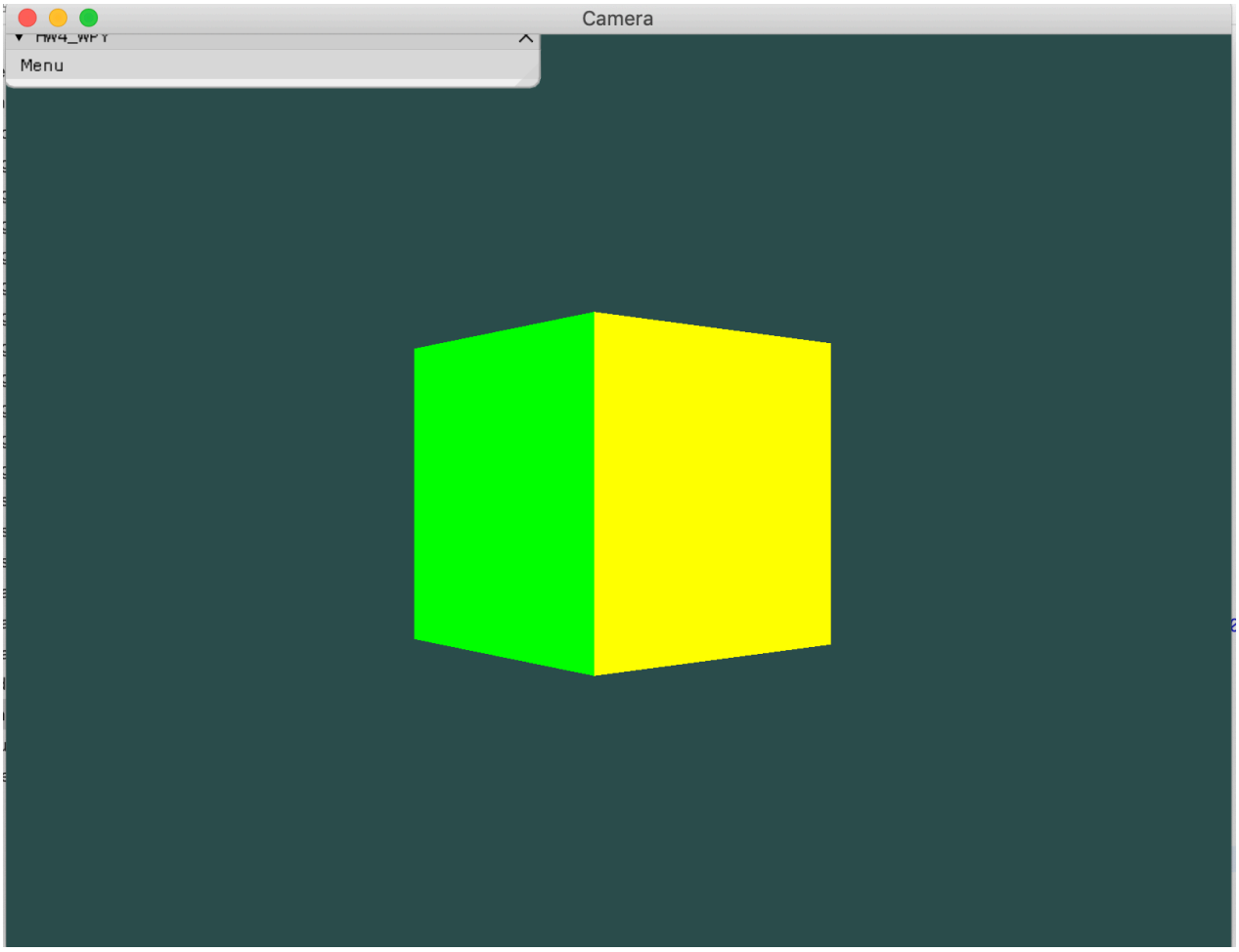
当然这些都已经被LookAt封装好，我们只需要提供三个参数即可。

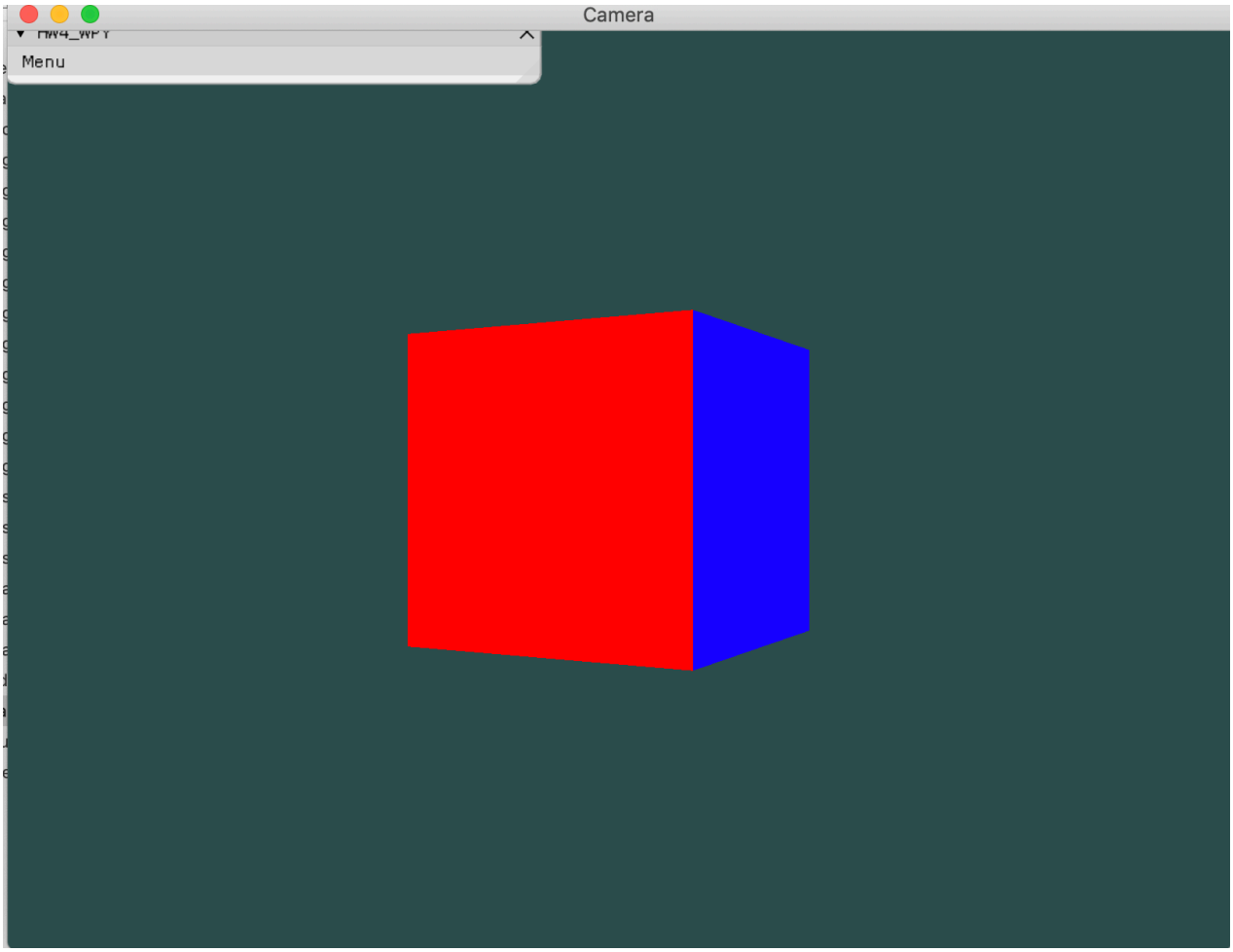
```
glm::mat4 view;
view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f),
           glm::vec3(0.0f, 0.0f, 0.0f),
           glm::vec3(0.0f, 1.0f, 0.0f));
```

然后通过圆的sin，cos设置，使得摄像机围中心点绕x-z平面旋转。

```
float radius = 15.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;
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));
```

截取其中两帧效果如下：

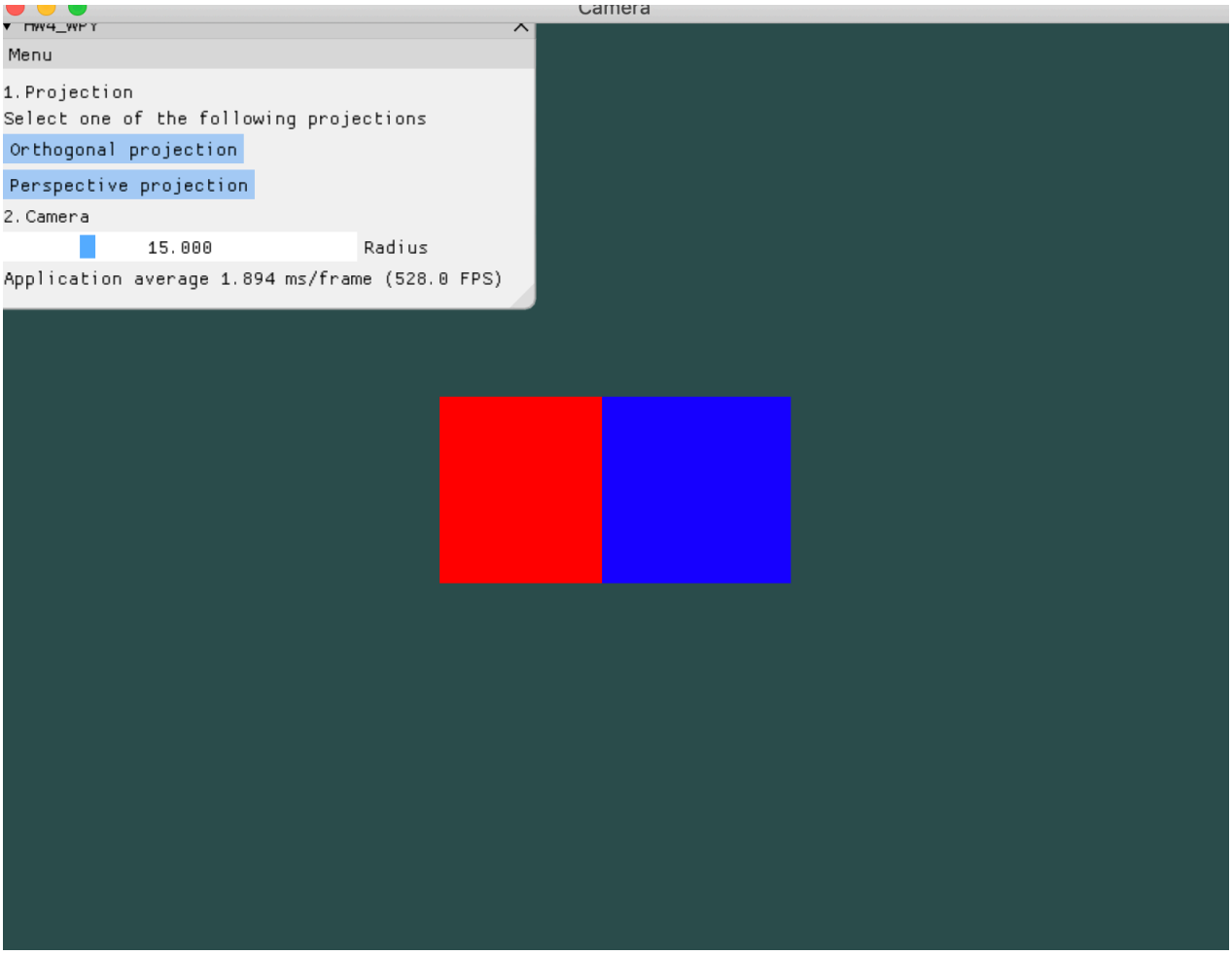


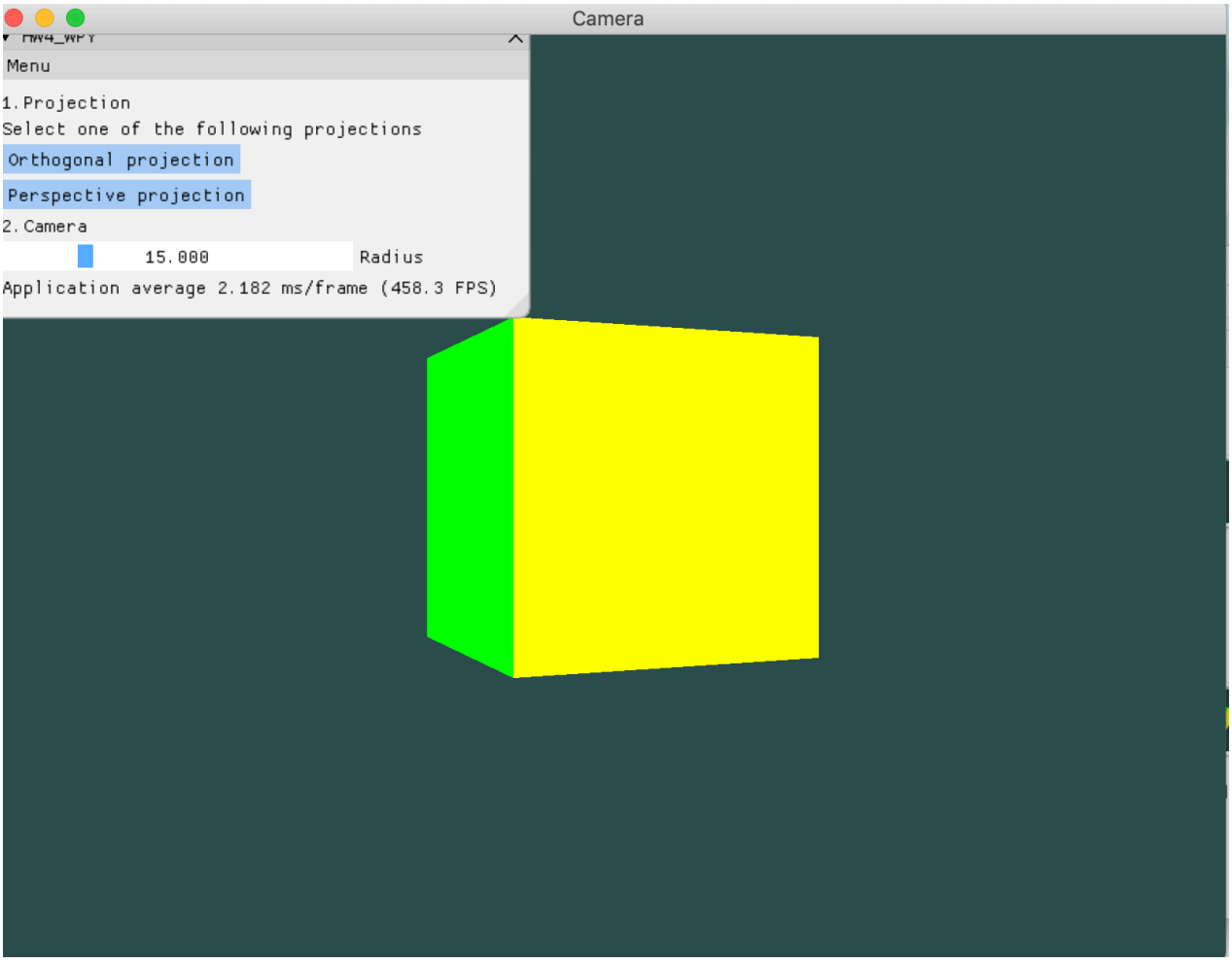


3. 在GUI里添加菜单栏，可以选择各种功能。 Hint: 使摄像机一直处于一个圆的位置。

实现GUI功能如下：

- 菜单点击start进入选择。
- 界面分为两部分，一部分是projection，另一部分是Camera。
- Projection对应两个按钮，可以选择是正交投影还是透视投影。
- Camera对应滑动块，可以调节摄像机距离物体中心的旋转半径。


Camera
Menu
1. Projection
Select one of the following projections
Orthogonal projection
Perspective projection
2. Camera
15.000
Radius
Application average 1.894 ms/frame (528.0 FPS)


Camera
Menu
1. Projection
Select one of the following projections
Orthogonal projection
Perspective projection
2. Camera
15.000
Radius
Application average 2.182 ms/frame (458.3 FPS)

配合ImGui的实现效果：

运行效果/1.gif

4. 思考题

- **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?在报告中写入。(Hints:你可能有不止一个摄像机)。**

坐标变换矩阵栈(ModelView)

用来存储一系列的变换矩阵，栈顶就是当前坐标的变换矩阵，进入OpenGL管道的每个坐标(齐次坐标)都会先乘上这个矩阵，结果才是对应点在场景中的世界坐标。

> VertexData —>(Object Coordinates)—>ModelViewMatrix —>(Eye Coordinates)—>ProjectionMatrix —>(Clip Coordinates)—>Divide by w —>(Normalized Device Coordinates)—>ViewportTransform—>WindowCoordinates

因为一个场景中可能存在多台摄像机，也就是多个视角的结合，如果为每一个都分配一个view可能容易比较混乱，ModelView是一个矩阵栈，可以存储model的各种变化以及不同的view视角，这样既不会造成视觉混乱同时也能减少计算量。

Bonus：

camera类:

- **实现一个camera类，当键盘输入 w,a,s,d ，能够前后左右移动;当移动鼠标，能够视角移动("look around")， 即类似FPS(First Person Shooting)的游戏场景。**

实现内容：

- 键盘WASD按键移动。
- 鼠标移动实现视角移动
- 滚轮实现物体缩放。

（1）键盘WASD按键实现自由移动。

- 计算前一帧与当前帧的时间差。
- 因为图形程序和游戏通常会跟踪一个时间差(Deltatime)变量，它储存了渲染上一帧所用的时间。我们把所有速度都去乘以deltaTime值。结果就是，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，所以这一帧的速度需要变得更高来平衡渲染所花去的时间。使用这种方法时，无论你的电脑快还是慢，摄像机的速度都会相应平衡，这样每个用户的体验就都一样了。

```
float currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;
```

- 在cpp文件中调用Camera类来对键盘移动进行操作。

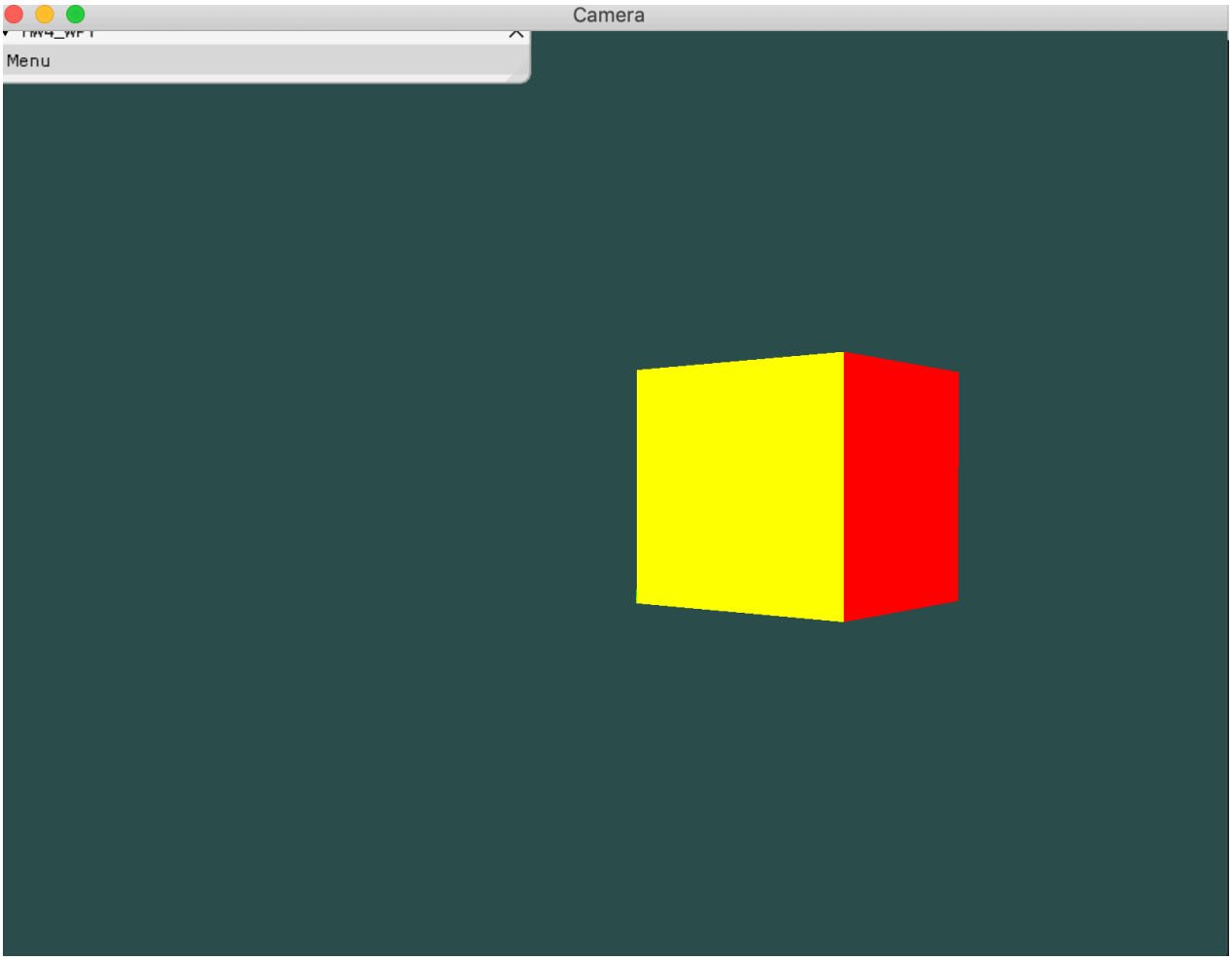
```
void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    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);
}
```

- Camera类中键盘移动的函数实现。

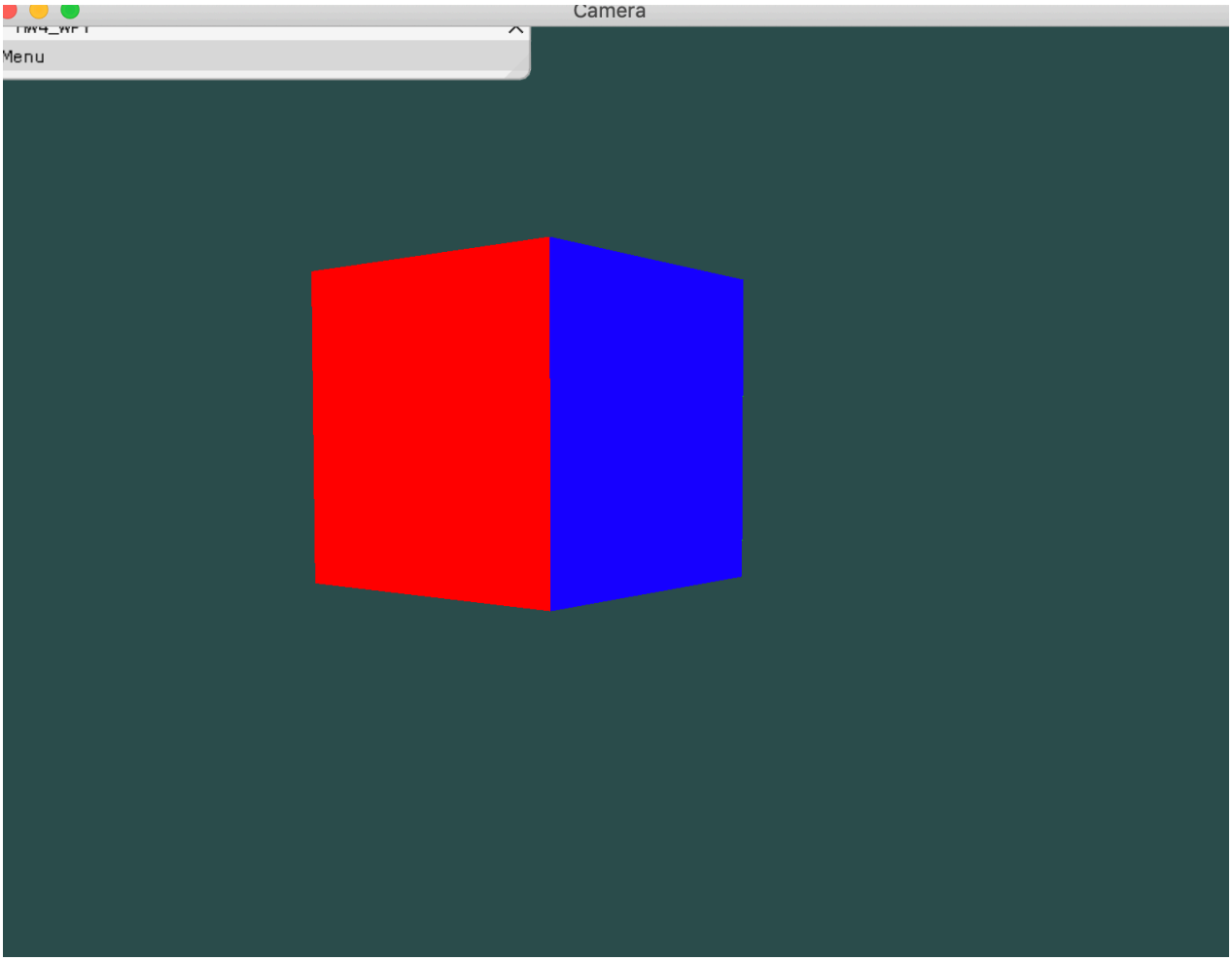
```
void ProcessKeyboard(Camera_Movement direction, float deltaTime)
{
    float velocity = MovementSpeed * deltaTime;
    if (direction == FORWARD)
        Position += Front * velocity;
    if (direction == BACKWARD)
        Position -= Front * velocity;
    if (direction == LEFT)
        Position -= Right * velocity;
    if (direction == RIGHT)
        Position += Right * velocity;
}
```

效果：

通过移动AD键实现左右方向的移动：



通过移动WS键实现物体的前后移动：



（2）利用鼠标实现视角移动

视角移动也就是实现对cameraFront向量的改变。

通过对欧拉角的改变来实现对方向向量的改变。

- 欧拉角由俯仰角和偏航角还有滚转角(用不到)实现
  - 计算鼠标至上一帧的偏移量

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos;

    lastX = xpos;
    lastY = ypos;

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

```
}
```

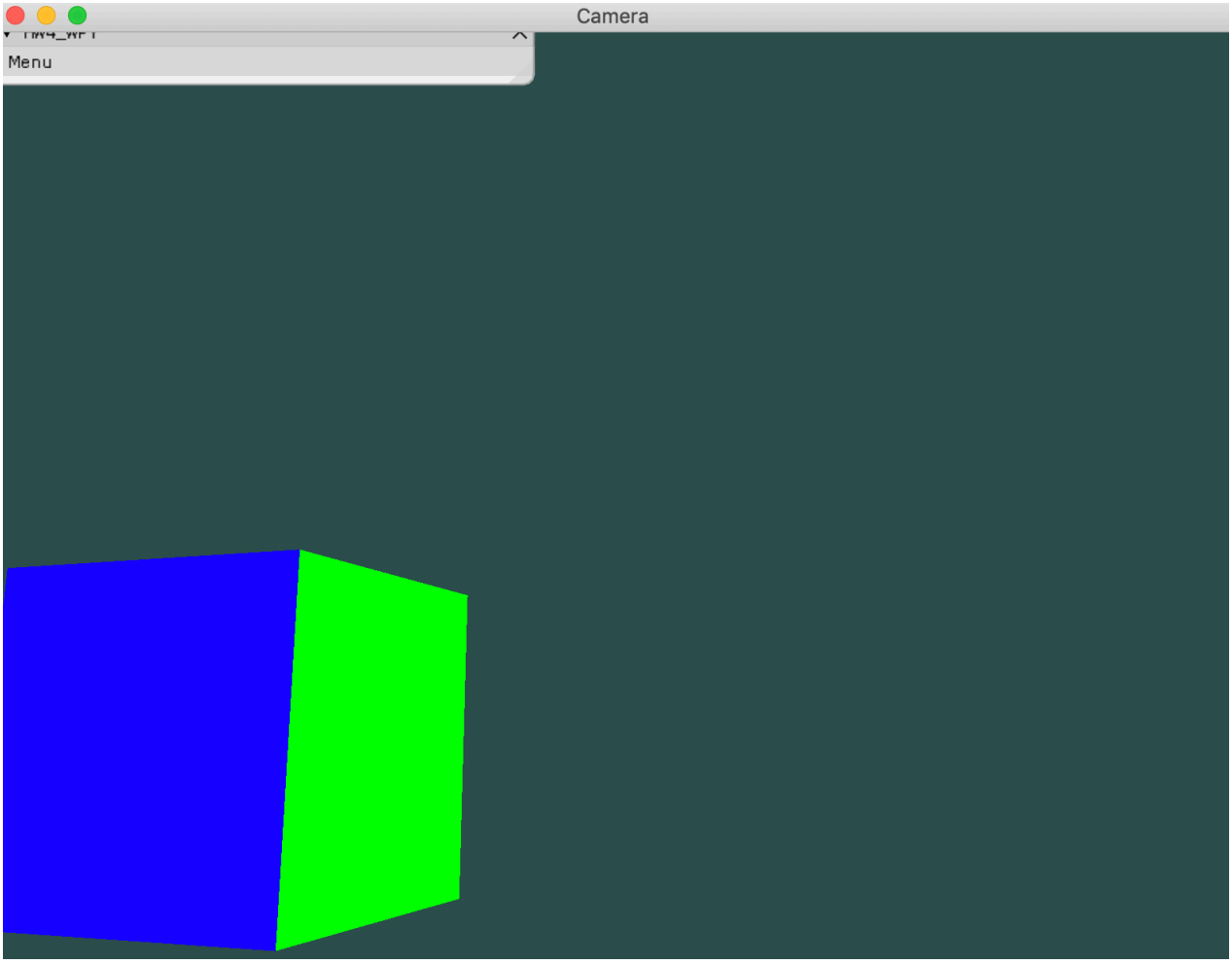
- 将偏移量乘上sensitivity(灵敏度)加到pitch和yaw上。
- 设置偏航角限制。

```
void ProcessMouseMovement(float xoffset, float yoffset, GLboolean
constrainPitch = true)
{
    xoffset *= MouseSensitivity;
    yoffset *= MouseSensitivity;

    Yaw   += xoffset;
    Pitch += yoffset;

    if (constrainPitch)
    {
        if (Pitch > 89.0f)
            Pitch = 89.0f;
        if (Pitch < -89.0f)
            Pitch = -89.0f;
    }

    updateCameraVectors();
}
```

- 把偏移量加到全局变量pitch和yaw上：

```
glm::vec3 front;
front.x = cos(glm::radians(Yaw)) * cos(glm::radians(Pitch));
front.y = sin(glm::radians(Pitch));
front.z = sin(glm::radians(Yaw)) * cos(glm::radians(Pitch));
Front = glm::normalize(front);
```

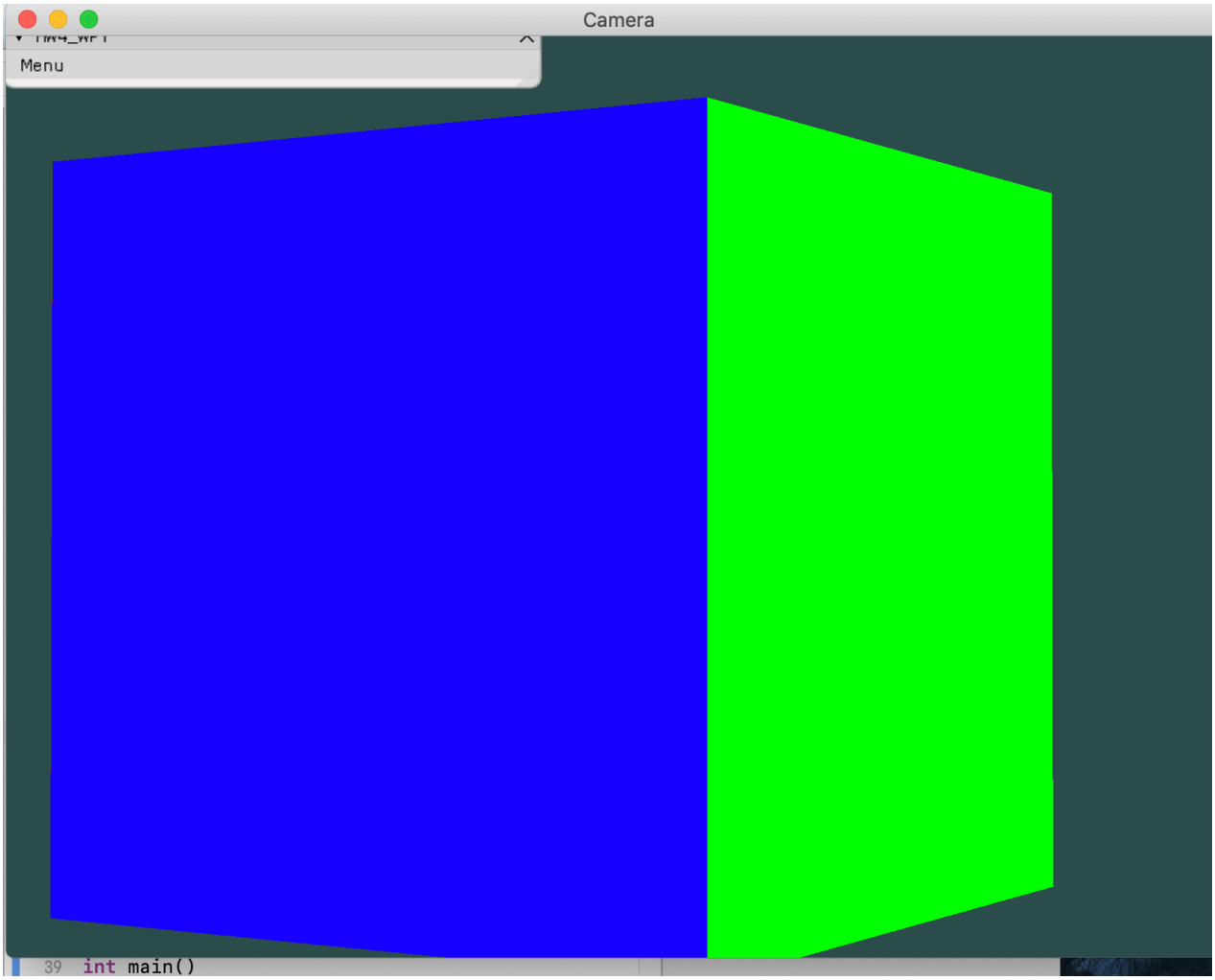
效果：



（3）滚轮实现物体的缩放

通过改变视野来实现，应用到透视矩阵上也就是fov角度的大小。

```
void ProcessMouseScroll(float yoffset)
{
    if (Zoom >= 1.0f && Zoom <= 45.0f)
        Zoom -= yoffset;
    if (Zoom <= 1.0f)
        Zoom = 1.0f;
    if (Zoom >= 45.0f)
        Zoom = 45.0f;
}
```



最终实现效果：

运行效果/2.gif