作业5

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

这次作业要求六个面的颜色不一样，所以在这里没有采用索引来保存顶点之间的关系，而是按面分别设置每个顶点的属性，需要36个点的信息（如下代码所示）。

```
//顶点输入
float vertices[] = {
    //坐标                          //颜色
    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, 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, 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, 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,
    };
```

把cube放置在(-1.5, 0.5, -1.5)位置只需要做一个平移即可

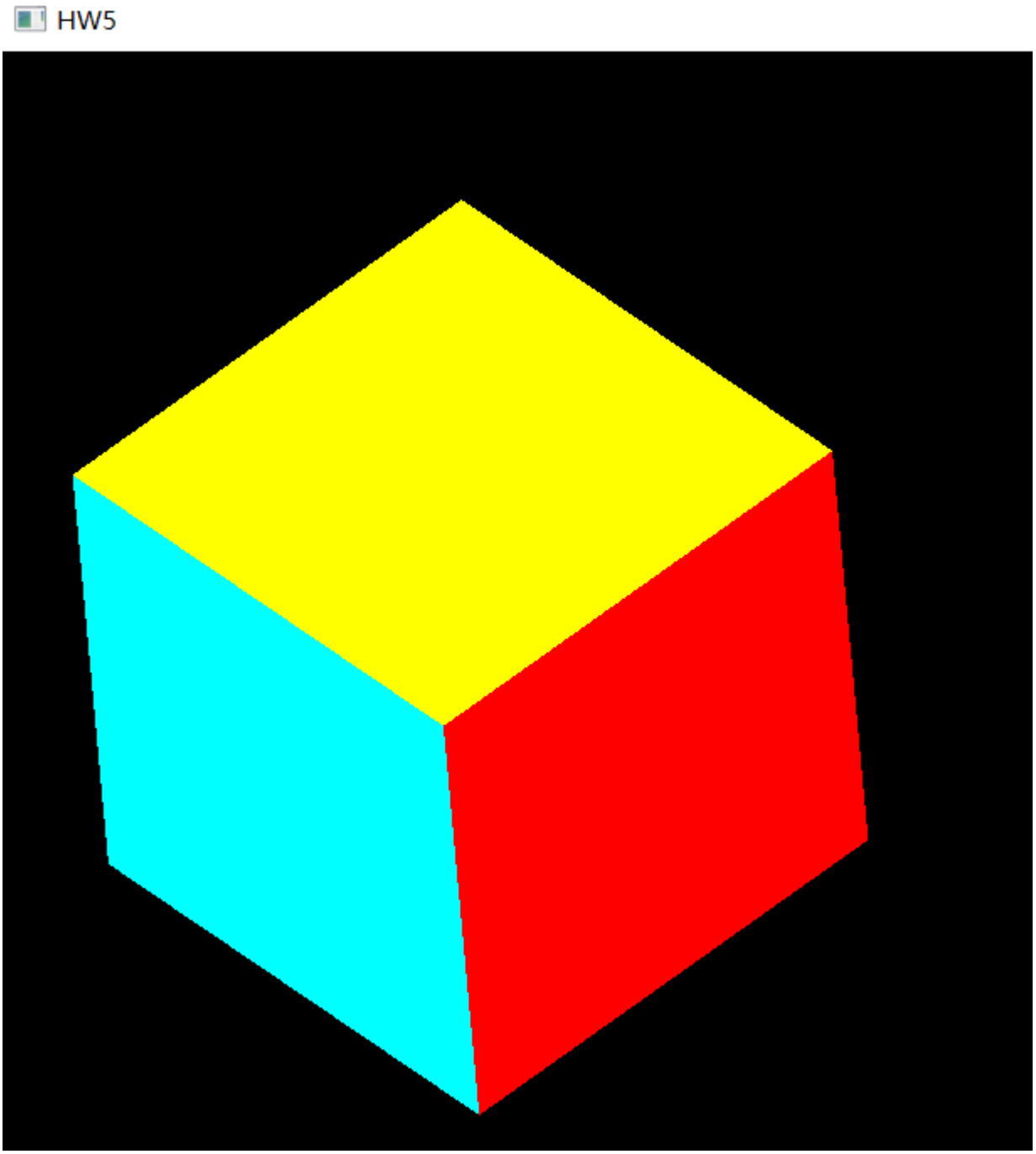
```
model = glm::translate(model, glm::vec3(-1.5f, 0.5f, -1.5f));
```

效果见Basic1-1

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

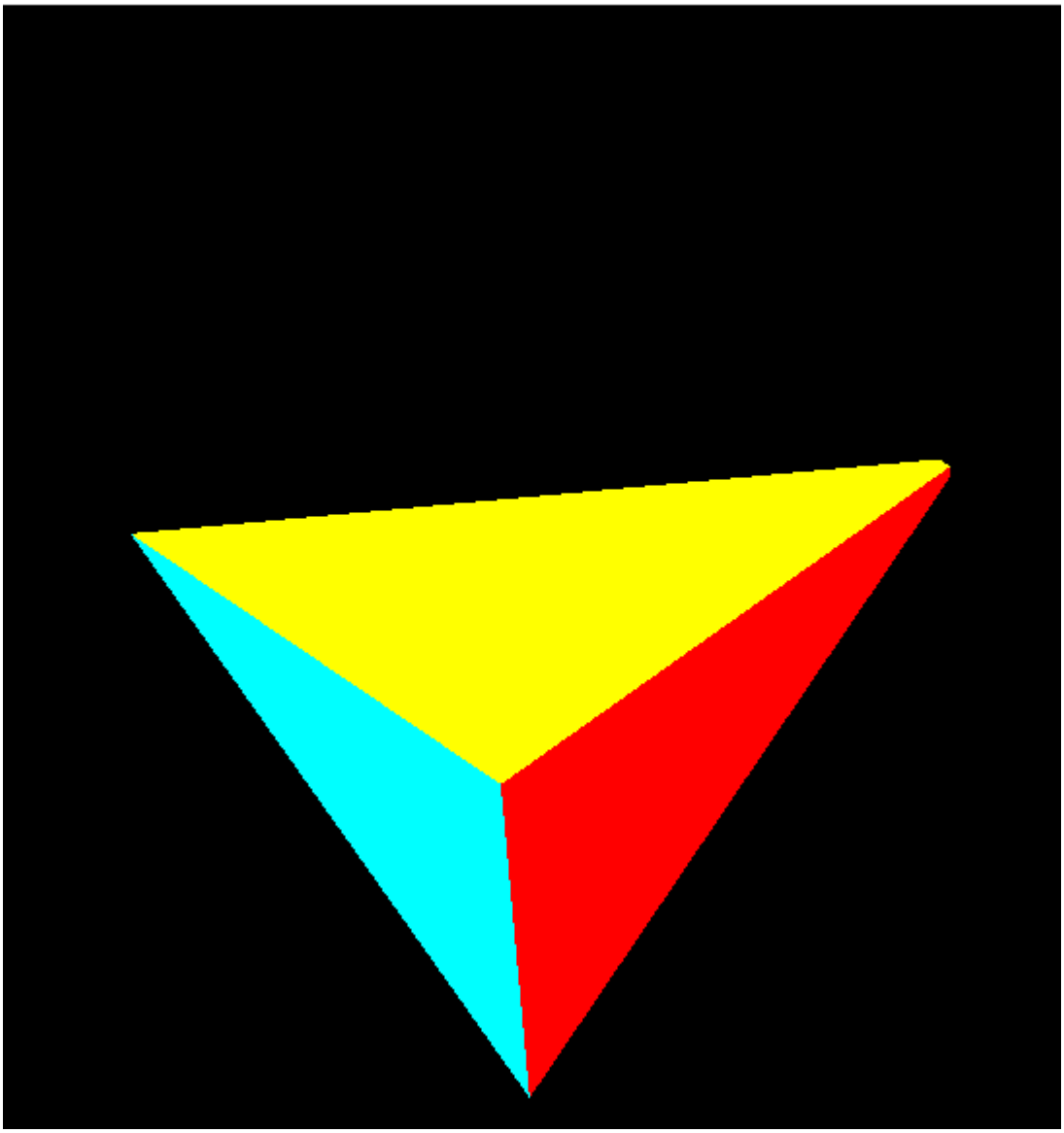
由下方多图可以看出，当显示的物体的位置在完全处于正交投影的范围内的话，会显示的相对正常。否则都会在某些方面有拉伸或者形状的变化。

projection = glm::ortho(-5.0f, 5.0f, -5.0f, 5.0f, 0.1f, 100.0f);

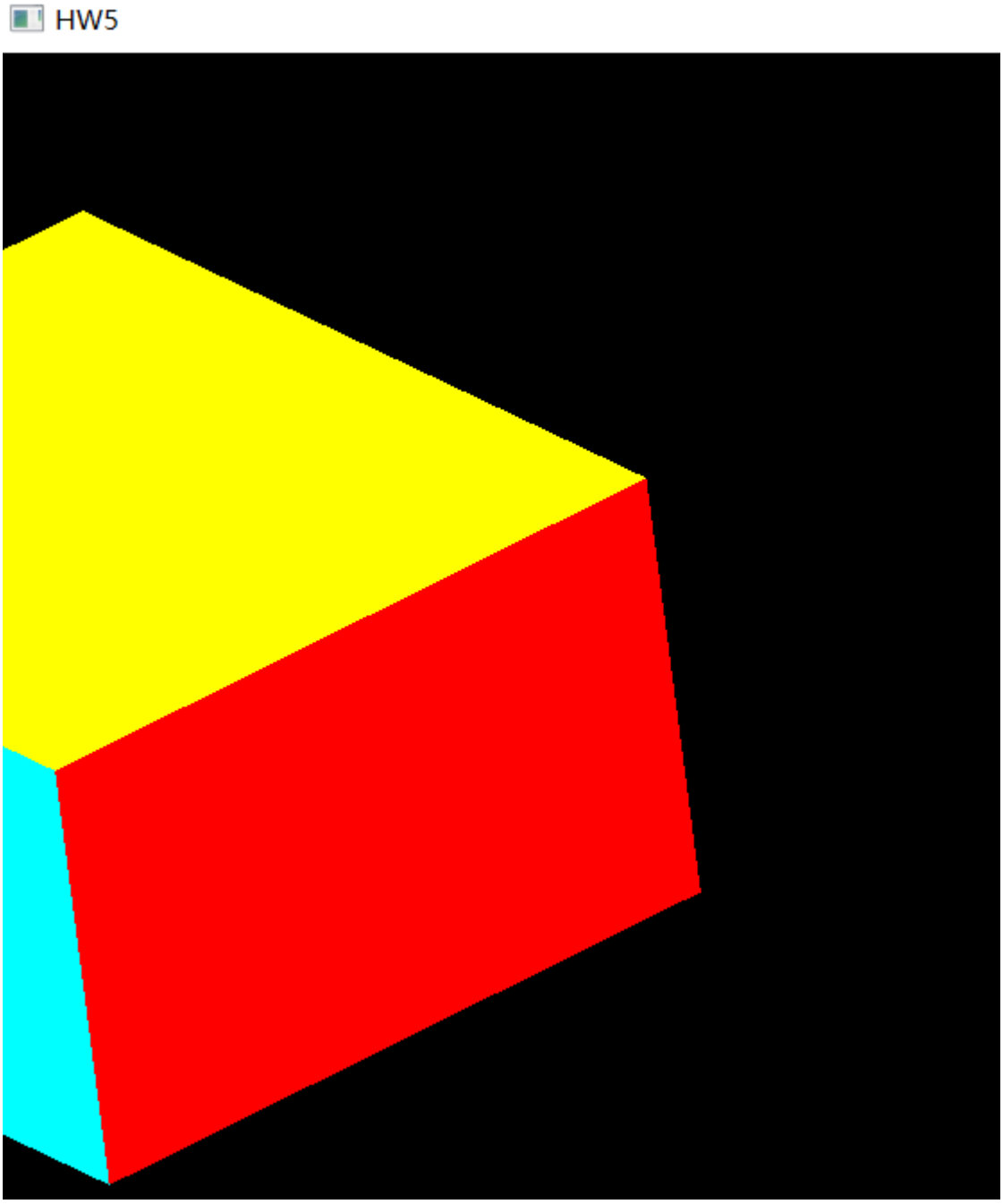


projection = glm::ortho(-5.0f, 5.0f, -5.0f, 5.0f, 0.1f, 15.0f);

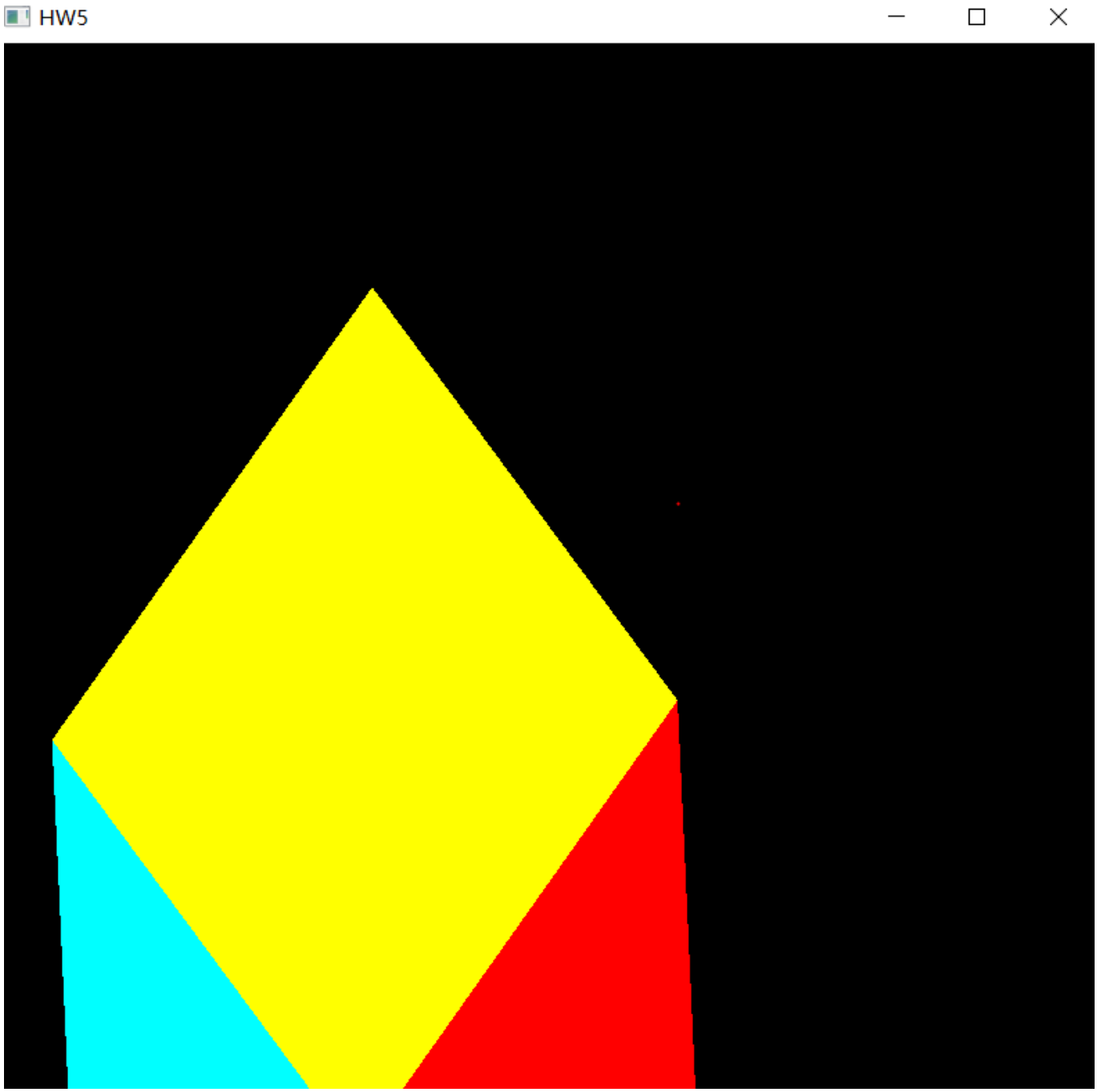
HW5



projection = glm::ortho(-2.0f, 5.0f, -5.0f, 5.0f, 0.1f, 100.0f);



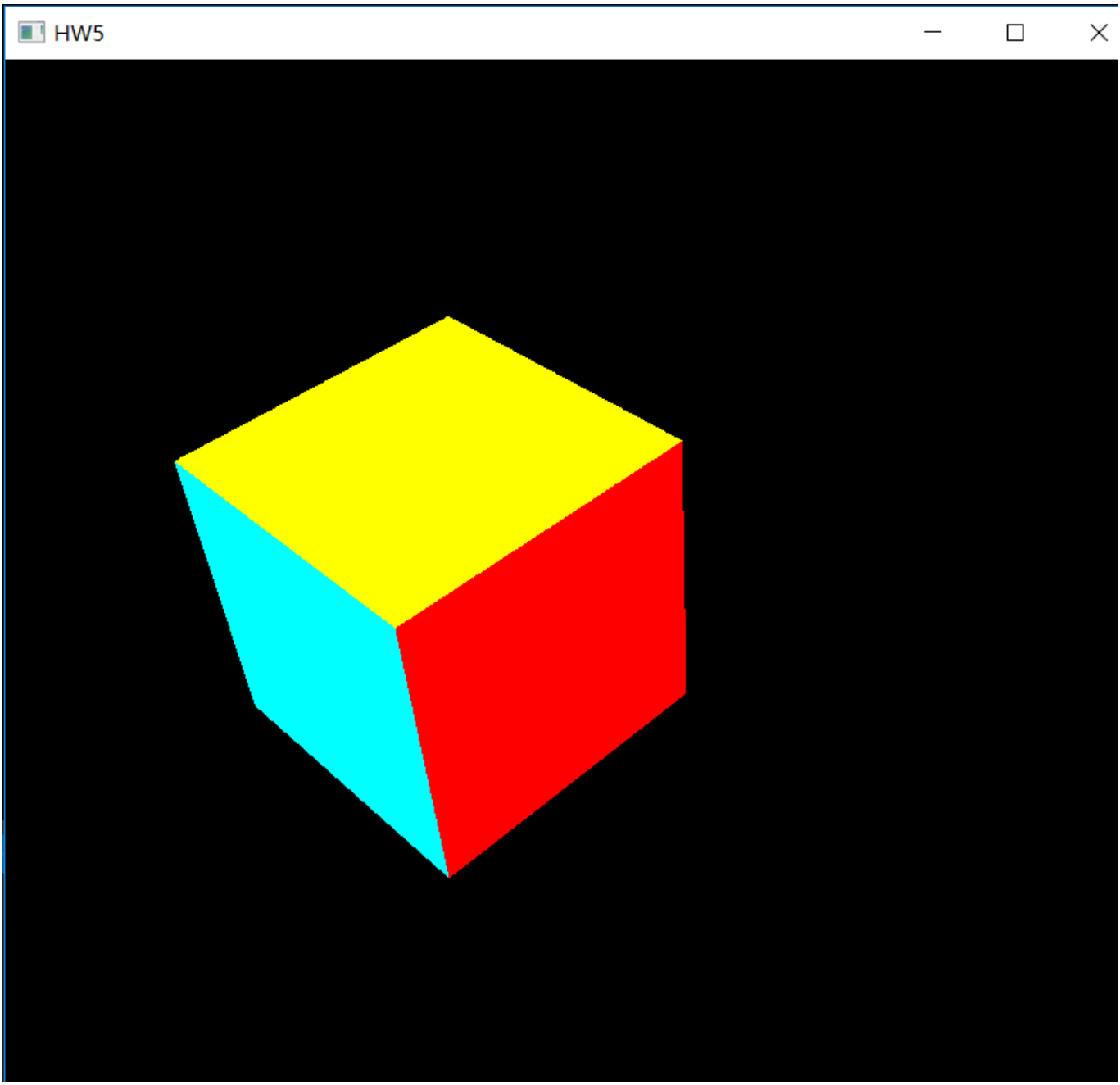
projection = glm::ortho(-5.0f, 5.0f, 0.0f, 5.0f, 0.1f, 100.0f);



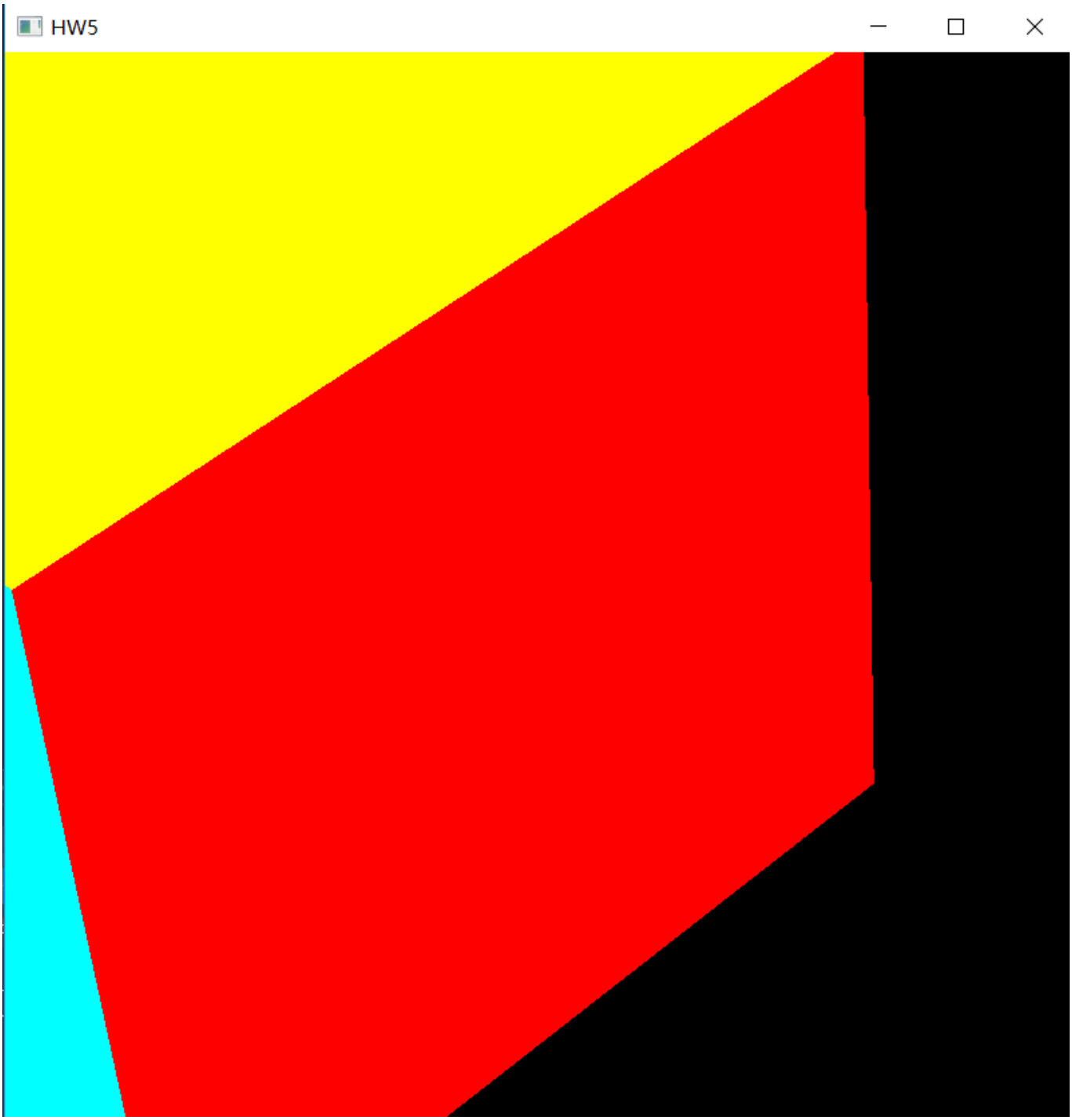
透视投影(perspective projection)：实现透视投影，使用多组参数，比较结果差异

右下方多图可以看出，在改变参数的时候，物体的形状并不会发生变化，仅有物体在屏幕中的大小会发生改变

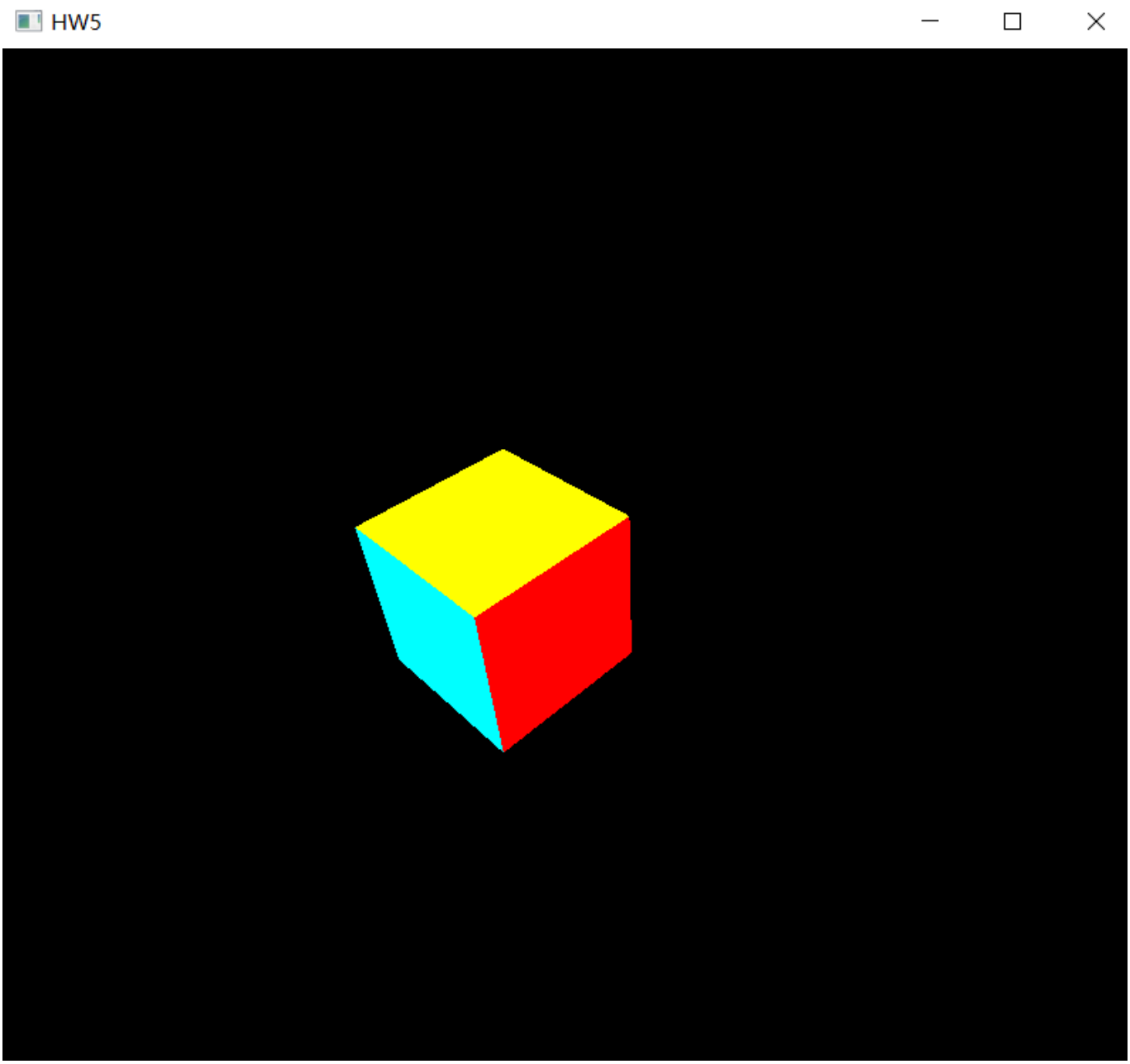
projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);



projection = glm::perspective(glm::radians(15.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);



projection = glm::perspective(glm::radians(75.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);



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

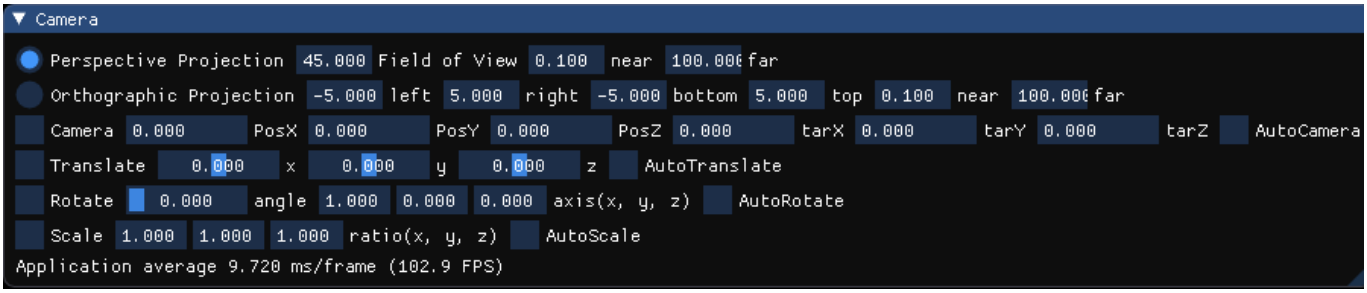
效果见Basic-2。具体代码如下所示。glm::LookAt函数需要一个位置、目标和上向量。它会创建一个观察矩阵。

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

3. 在GUI里添加菜单栏，可以选择各种功能。

实现的功能如下图所示。具体效果见Basic-3

- 第一行为选择透视投影，它有三个参数。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。如果想要一个真实的观察效果，它的值通常设置为45.0f，但想要一个末日风格的结果你可以将其设置一个更大的值。第二和第三个参数设置了平截头体的近和远平面。我们通常设置近距离为0.1f，而远距离设为100.0f。所有在近平面和远平面内且处于平截头体内的顶点都会被渲染。
- 第二项是选择正射投影，它有六个参数。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。这个投影矩阵会将处于这些x，y，z值范围内的坐标变换为标准化设备坐标。
- 第三项是对摄像机调整，它有六个参数。前三个参数调整的为摄像机的位置，后三个参数为摄像机对准的目标。
- 其余选项为上次作业就有的功能



Bonus

1.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

具体效果见Bonus。视频中首先演示了当移动鼠标时能够视觉移动，随后为输入w，a，s，d，up，down六个键让摄像机进行前后上下左右的移动。camera类在Camera.h和Camera.cpp中实现。下面为Camera.h的代码

```
#pragma once

#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

#include <vector>

enum Camera_Movement {
        FORWARD,
        BACKWARD,
        LEFT,
        RIGHT,
        UP,
        DOWN
};

const float YAW = -90.0f;
const float PITCH = 0.0f;
const float SPEED = 2.5f;
const float SENSITIVITY = 0.1f;
```

```
const float ZOOM = 45.0f;


class Camera
{
public:
        glm::vec3 Position;
        glm::vec3 Front;
        glm::vec3 Up;
        glm::vec3 Right;
        glm::vec3 WorldUp;

        float Yaw;                                      // 欧拉角的偏航角 左右
        float Pitch;                                // 欧拉角的俯仰角 上下


        float MovementSpeed;            //移动速度
        float MouseSensitivity;         // 鼠标灵敏度
        float Zoom;                                     // 缩放

        //构造函数1
        Camera(glm::vec3 position = glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3 up =
glm::vec3(0.0f, 1.0f, 0.0f), float yaw = YAW, float pitch = PITCH);
        //构造函数2
        Camera(float posX, float posY, float posZ, float upX, float upY, float
upZ, float yaw, float pitch);
        // 获取view矩阵
        glm::mat4 GetViewMatrix();
        //获取键盘输入
        void ProcessKeyboard(Camera_Movement direction, float deltaTime);
        // 鼠标移动，改变摄像机的欧拉角
        void ProcessMouseMovement(float xoffset, float yoffset, GLboolean
constrainPitch = true);
        // 鼠标缩放，改变视野
        void ProcessMouseScroll(float yoffset);

private:
        void updateCameraVectors();
};
```