中山大學
SUN YAT-SEN UNIVERSITY

《计算机图形学作业》

实验报告

（作业五）

学 院 名 称 ：	数据科学与计算机学院
学 生 姓 名 ：	姚雪辉
学　　号 ：	15355119
专业（班级）：	16 软件工程四（7）班
时　　间 ：	2019 年 4 月 15 日

Basic

投影(Projection):

(1) 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

在上次的基础上增加了颜色参数，所以在 vertexShaderSource 里面加入一层 layout:

```
const char *vertexShaderSource = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"layout (location = 1) in vec3 aColor;\n"
"out vec3 ourColor;\n"
"uniform mat4 transform;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"void main()\n"
"{\n"
"   gl_Position = projection * view * transform * vec4(aPos, 1.0);\n"
"   ourColor = aColor;\n"
"}\0";
```

在 fragmentShaderSource 里面将颜色写成自己的颜色

```
const char *fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"in vec3 ourColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(ourColor, 1.0f);\n"
"}\n\0";
```

相应的做如下改变

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
```

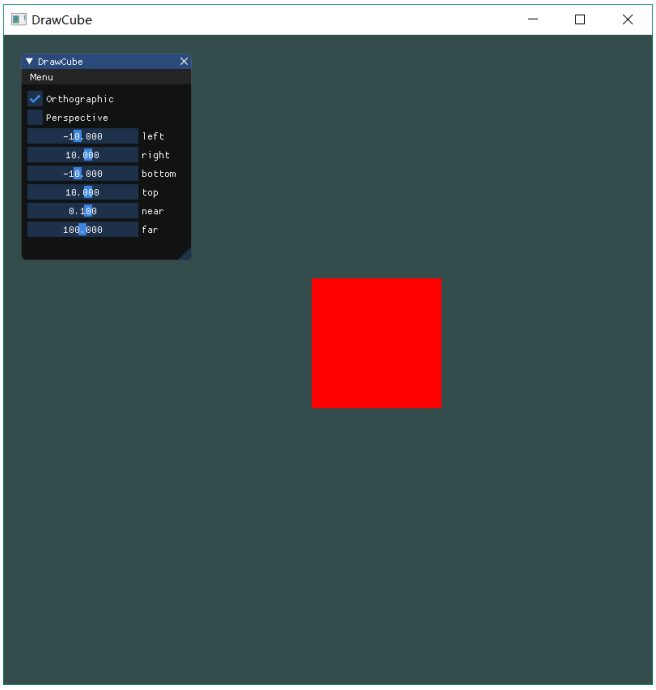
然后

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

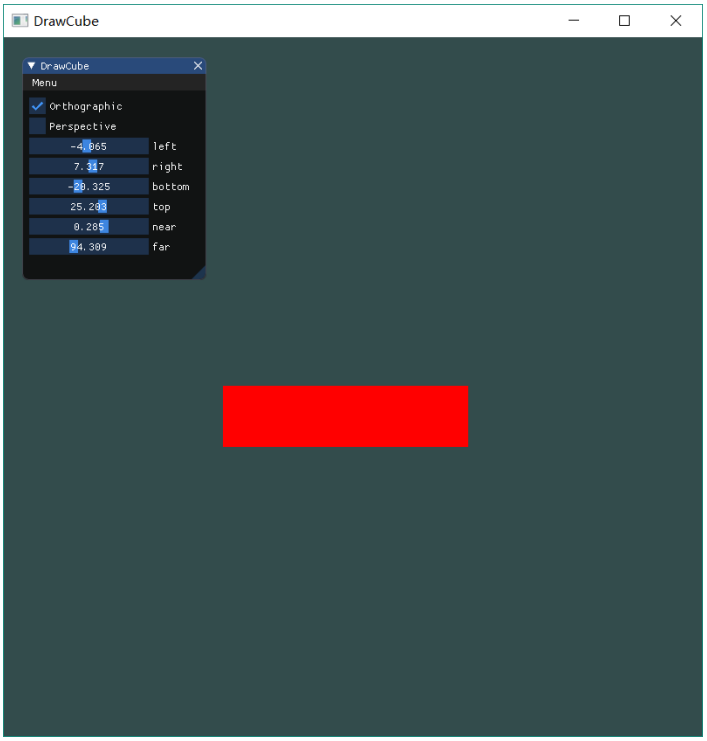
将位置转移到(-1.5f,0.5f,-1.5f)

(2) 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

```
projection = glm::ortho(myLeft, myRight, myBottom, myTop, myNear, myFar);
```



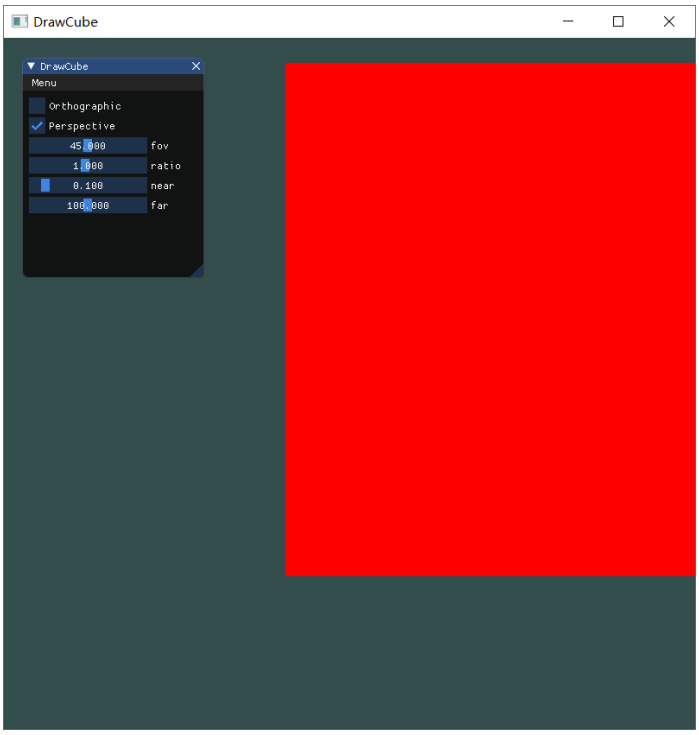
改变 near，far 参数对视图没有影响，改变其他值视图会有形变，也会产生位移



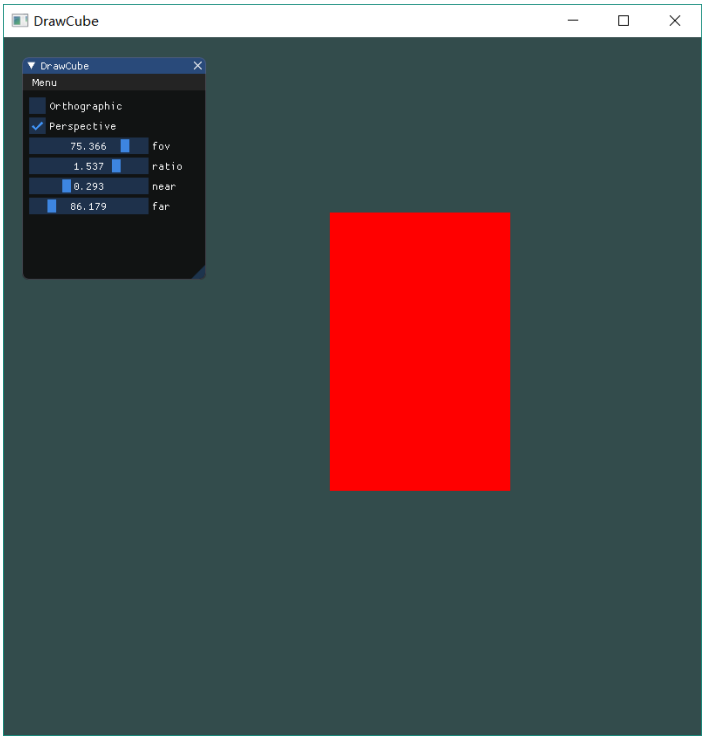
（3） 透视投影(perspective projection)：实现透视投影，使用多组参数，比较结果差异

```
projection = glm::perspective(glm::radians(fov), ratio, myNear2, myFar2);
```

其中 fov 是视野角度，ratio 是长宽比例



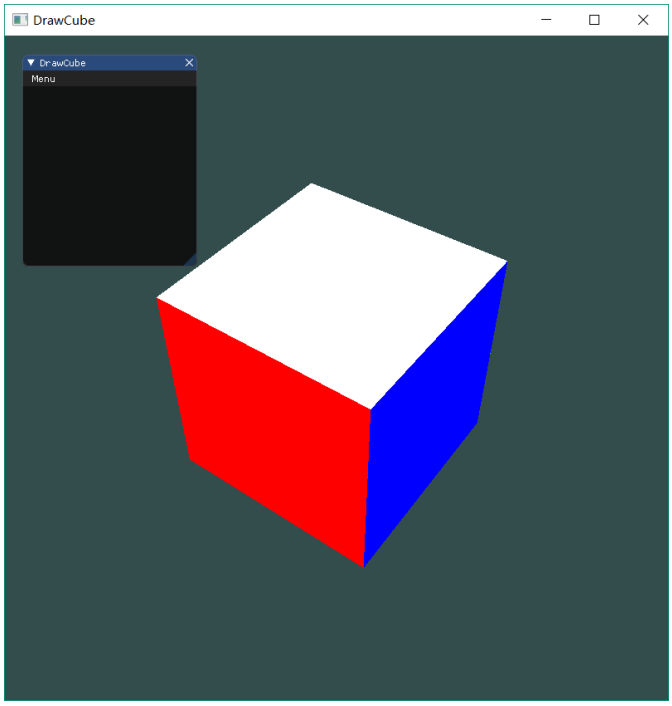
改变参数 near,far 对视图没有变化，，改变视角会按比例缩放视图，调整比例会形变。



2.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

```
projection = glm::perspective(glm::radians(fov), ratio, myNear2, myFar2);
float radius = 10.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;
view = glm::lookAt(glm::vec3(camX, 10.0f, 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。

```
ImGui::Begin("DrawCube",&isOpen, ImGuiWindowFlags_MenuBar);
        if (ImGui::BeginMenuBar()) {
            if (ImGui::BeginMenu("Menu")) {
                if (ImGui::MenuItem("FirstProblem")) {
                    isFirstProbelm = true;
                    isSecondProblem = false;
                    isBouns = false;
                }
                if (ImGui::MenuItem("SecondProblem")) {
                    isFirstProbelm = false;
                    isSecondProblem = true;
```

```
                isBouns = false;
            }
            if (ImGui::MenuItem("Bouns")) {
                isFirstProbelm = false;
                isSecondProblem = false;
                isBouns = true;
            }
            ImGui::EndMenu();
        }
        ImGui::EndMenuBar();
    }
    if (isFirstProbelm) {
        ImGui::Checkbox("Orthographic", &isOrthographic);
        if (isOrthographic) {
            isPerspective = false;
        }
        ImGui::Checkbox("Perspective", &isPerspective);
        if (isPerspective) {
            isOrthographic = false;
        }
    }

    if (isFirstProbelm && isOrthographic) {
        ImGui::SliderFloat("left", &myLeft, -100.0f, 100.0f);
        ImGui::SliderFloat("right", &myRight, -100.0f, 100.0f);
        ImGui::SliderFloat("bottom", &myBottom, -100.0f, 100.0f);
        ImGui::SliderFloat("top", &myTop, -100.0f, 100.0f);
        ImGui::SliderFloat("near", &myNear, -1.0f, 1.0f);
        ImGui::SliderFloat("far", &myFar, 80.0f, 120.0f);
    }
    if (isFirstProbelm && isPerspective) {
        ImGui::SliderFloat("fov", &fov, 0.0f, 90.0f);
        ImGui::SliderFloat("ratio", &ratio, 0.1f, 2.0f);
        ImGui::SliderFloat("near", &myNear2, 0.0f, 1.0f);
        ImGui::SliderFloat("far", &myFar2, 80.0f, 120.0f);
    }
    ImGui::End();
```

4 .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。

OpenGL 本身没有摄像机的概念，但我们可以通过把场景中的所有物体往相反方向移动的方式来模拟出摄像机，产生一种我们在移动的感觉，而不是场景在移动。所以 Model 的移动和 View 的移动是相对的，所以可以合二为一。

Bonus

参考官方教程，顺带一起加入了缩放功能

成员变量申明如下

```
GLfloat Yaw, Pitch;
GLfloat MovementSpeed, MouseSensitivity;
GLfloat Zoom;
glm::vec3 Position;
glm::vec3 Front;
glm::vec3 Up;
glm::vec3 Right;
glm::vec3 WorldUp;
```

在主函数的运用，需要两个回调函数，用来处理鼠标移动事件和缩放事件

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

在本来的按键监听函数中增加逻辑，然后调用 camera 类中的 API

```
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);
    }
}
```

下面解释 camera 各部分函数的逻辑

```
    glm::mat4 GetViewMatrix()
    {
        return glm::lookAt(Position, Position + Front, Up);
    }
```

Position 是摄像机的位置坐标，Position+Front 是位置加上前方的方向向量，可以保证摄像机一直看着前方，Up 只要取个向上的向量即可。

```
    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;
        }
    }
```

通过 deltaTime 更新物体移动，deltaTime 在 unity 中是内置的，而这里的 deltaTime 是每帧之间的时间间隔，为的是解决不同硬件的移动感觉相差太大的问题。利用方向向量乘以速度再在每帧更新就可以形成移动的效果。

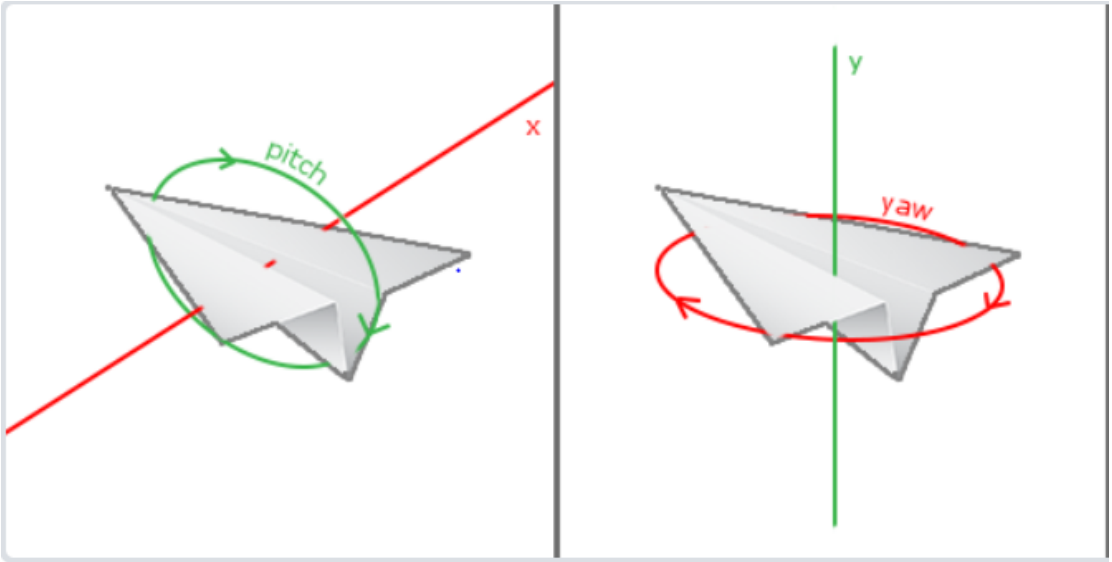
```
void ProcessMouseMovement(float xoffset, float yoffset, GLboolean constrainPitch =
true)
```

```
{
    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();
}
```



俯仰角(Pitch)、偏航角(Yaw),

最后加了角度的约束，其中 offset 是鼠标移动的回调函数得到每次调用后的鼠标的偏移量，通过偏移量改变旋转角度。

```
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;
    }
```

缩放函数，也加了缩放的约束，是鼠标缩放的回调函数得到每次调用的偏移量。

```
void updateCameraVectors()
    {
        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);

        Right = glm::normalize(glm::cross(Front, WorldUp));
        Up = glm::normalize(glm::cross(Right, Front));
    }
```

最后是一个辅助函数，设置 Front,Right,Up 三个向量，首先得到 Front，然后通过 Front 和 WorldUp 叉乘，得到 Right，再通过 Right 和 Front 叉乘得到 Up