# 计算机图形学作业报告

廖蕾 16340135

---

- 画一个立方体，并添加六面不同的颜色：

  这里用到的代码和上周的代码一样： 在这里坐标系的使用用到了三个`glm::mat4`变量：

  - model：将局部坐标系转换成世界坐标系
  - view：将世界坐标系转换成摄像机坐标系
  - projection：将摄像机坐标系进行裁剪，将摄像机坐标系转换成屏幕坐标系

  在画立方体的时候，用到立方体的六个面顶点信息，每个面两个三角形，每个三角形3个顶点，一共需要定义36个顶点信息：

```
float vertices[216] = {
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,

        -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,

        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,

         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,

        -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
```

```
        -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 1.0f,

        -0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
        -0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 1.0f
};
```

在渲染的过程中，先得到这36个顶点和颜色的信息，进行渲染：

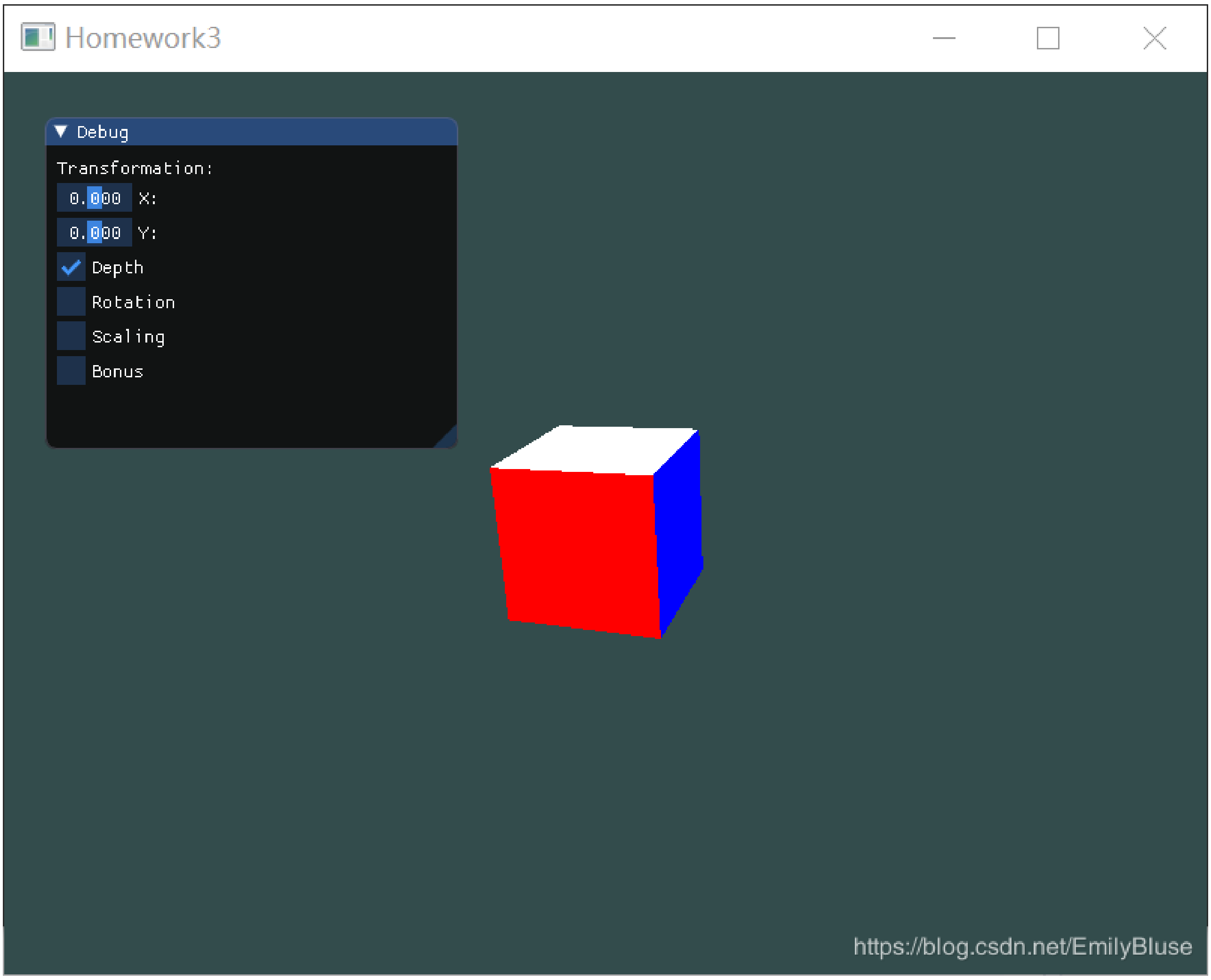
```
glBindBuffer(GL_ARRAY_BUFFER, VAO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

// position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
(void*)0);
glEnableVertexAttribArray(0);
// texture coord attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3
* sizeof(float)));
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// activate shader
glUseProgram(shaderProgram);
```

之后再根据定义的model、view和projection三个变量，得到3d的立方体，这里是利用了透视投影：

```
glm::mat4 model = glm::mat4(1.0f); // make sure to initialize matrix to
identity matrix first
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);

view = glm::translate(view, glm::vec3(0.0f, 0.0f, -5.0f));
view = glm::rotate(view, glm::radians(30.0f), glm::vec3(1.0f, -1.0f, 0.0f));
projection = glm::perspective(glm::radians(60.0f), (float)WINDOW_WIDTH /
(float)WINDOW_HEIGHT, 0.1f, 100.0f);

string name = "view";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(view));
name = "projection";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(projection));

model = glm::translate(model, glm::vec3(translatio_x, translatio_y, 0.0f));
model = glm::scale(model, glm::vec3(scale_x, scale_y, scale_z));
```

```
name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

这样通过渲染可以得到3d的立方体（默认开启深度测试）：




---

- 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
    放置位置只需要修改`model`的属性就好了，这里定义三个变量：`translation_x`, `translation_y`, `translation_z`，这样可以方便之后用GUI修改三个参数的值。完整代码如下：

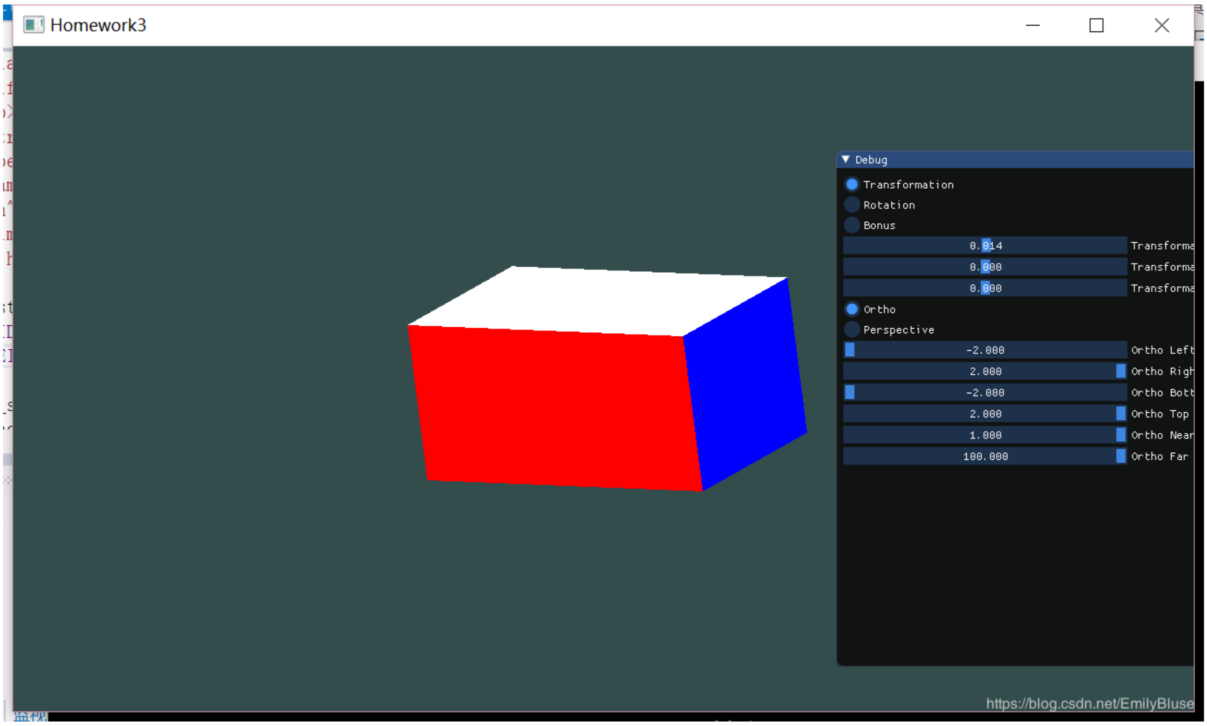
```
float translation_x = -1.5f;
float translation_y = 0.5f;
float translation_z = -1.5f;
model = glm::translate(model, glm::vec3(translation_x, translation_y,
translation_z));
```

  - 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

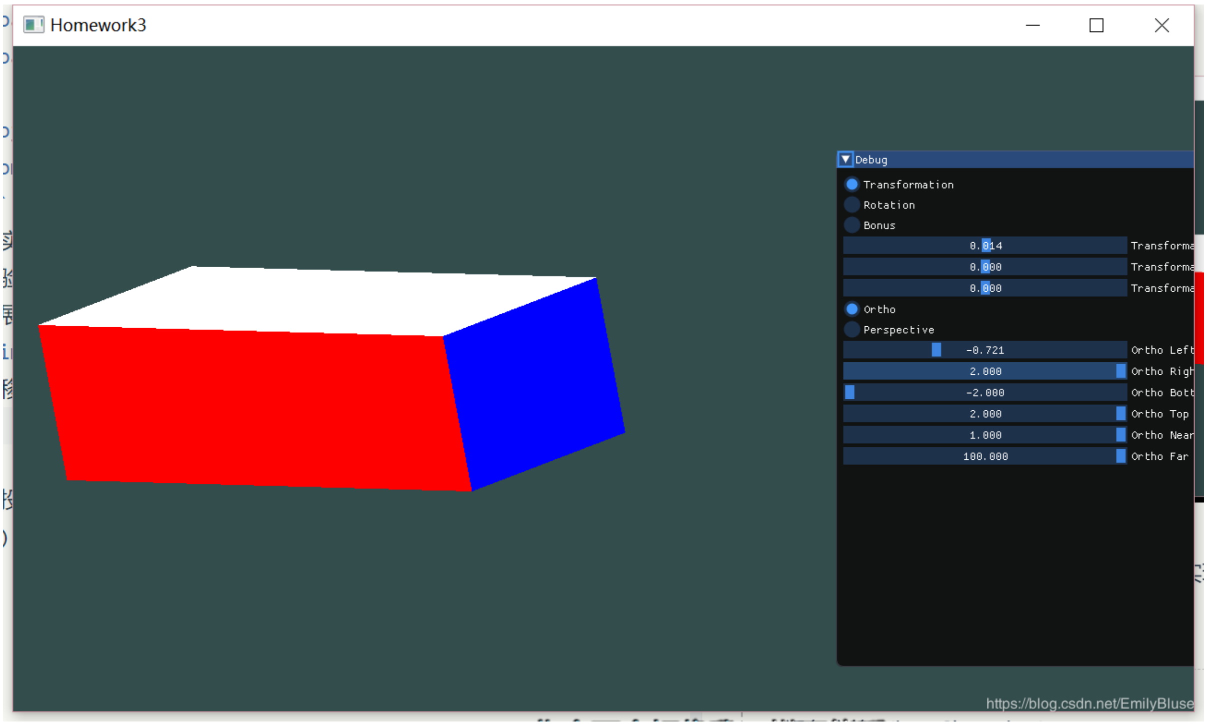
这里定义六个变量，分别调节对应的六个参数。然后利用`glm::ortho`函数对`projection`变量进行计算，代码如下：

```
float ortho_left = -2.0f;
float ortho_right = 2.0f;
float ortho_bottom = -2.0f;
float ortho_top = 2.0f;
float ortho_near = 1.0f;
float ortho_far = 100.0f;'

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

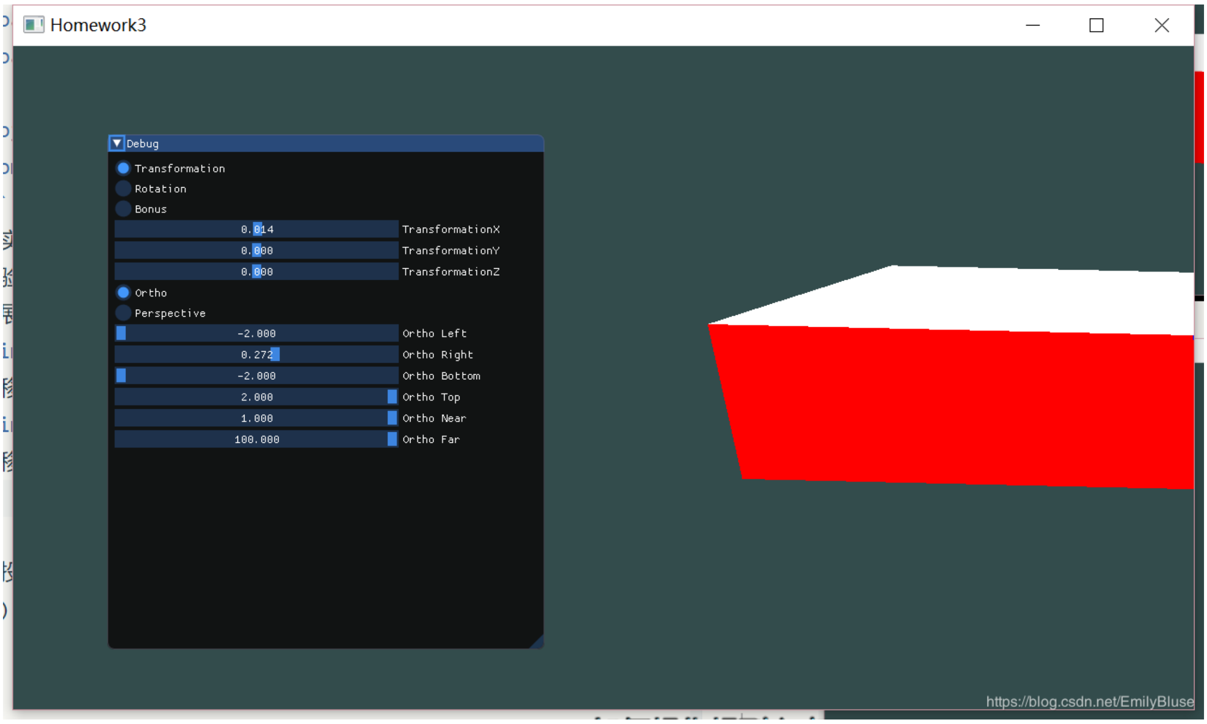
在实验的过程，我发现六个参数分别对应六面体六个面投影的视角变换，实验结果如下：
先展示初始六面体：




当移动left的值的时候：




当移动right的值的时候：

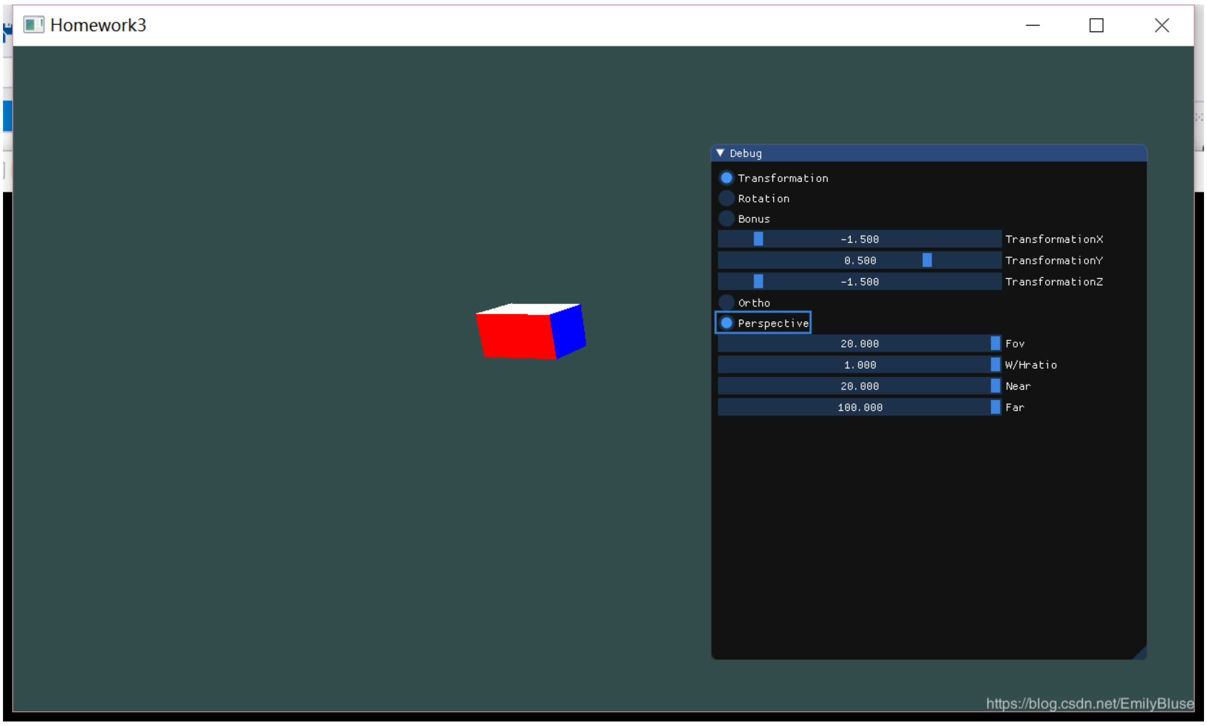



具体移动效果请见同文件夹中mp4文件。

- 透视投影(perspective projection)：实现透视投影，使用多组参数，比较结果差异 这里和正交投影类似，使用了4个变量去记录透视投影需要用到的4个值，然后使用`glm::perspective`对`projection`进行计算，代码如下：

```
float perspective_fov = 20.0f;
float perspective_w = 1.0f;
float perspective_n = 0.1f;
float perspective_far = 100.f;
projection = glm::perspective(perspective_fov, perspective_w,
perspective_n, perspective_far);
```

得到结果如下：




当我们拖动Fov这个参数，控制的是他的旋转角。W/Hratio这个参数是控制立方体横轴宽度的。Near这个是控制他的靠近距离的，Far是控制远离距离的。具体变换请见同文件夹下的mp4文件。

---

- 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

利用圆的公式可以让摄像机在XOZ平面内不移动，然后使用`glm::lookat`函数对`view`进行计算，代码实现如下：

```
int radius = 5;
float camPosX = sin(glfwGetTime()) * radius;
float camPosZ = cos(glfwGetTime()) * radius;
view = glm::lookAt(glm::vec3(camPosX, 0.8f, camPosZ), glm::vec3(0.0f, 0.0f,
0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
projection = glm::perspective(45.0f, 1.0f, 0.1f, 100.0f);
model = glm::translate(model, glm::vec3(translation_x, translation_y,
translation_z));
```

得到的效果请见同文件夹下的展示视频。

---

- 在GUI里添加菜单栏，可以选择各种功能。
  这里的实现效果可以在上面的几幅截图中看出。这里就不另放图了，主要涉及的变量和函数使用如下：

```
int tastMark = 0;
int tastMark2 = 0;

float translation_x = -1.5f;
```

```
float translation_y = 0.5f;
float translation_z = -1.5f;

float perspective_fov = 20.0f;
float perspective_w = 1.0f;
float perspective_n = 0.1f;
float perspective_far = 100.f;

float ortho_left = -2.0f;
float ortho_right = 2.0f;
float ortho_bottom = -2.0f;
float ortho_top = 2.0f;
float ortho_near = 1.0f;
float ortho_far = 100.0f;

while (!glfwWindowShouldClose(window))
{
    ImGui_ImplGlfwGL3_NewFrame();
    {
        ImGui::RadioButton("Transformation", &tastMark, 0);
                ImGui::RadioButton("Rotation", &tastMark, 1);
                ImGui::RadioButton("Bonus", &tastMark, 2);
                if (tastMark == 0) {
                        ImGui::SliderFloat("TransformationX",
&translation_x, -2.0f, 2.0f);
                        ImGui::SliderFloat("TransformationY",
&translation_y, -1.0f, 1.0f);
                        ImGui::SliderFloat("TransformationZ",
&translation_z, -2.0f, 2.0f);
                        ImGui::RadioButton("Ortho", &tastMark2, 0);
                        ImGui::RadioButton("Perspective", &tastMark2, 1);
                        if (tastMark2 == 0) {
                                ImGui::SliderFloat("Ortho Left",
&ortho_left, -2.0f, 2.0f);
                                ImGui::SliderFloat("Ortho Right",
&ortho_right, -2.0f, 2.0f);
                                ImGui::SliderFloat("Ortho Bottom",
&ortho_bottom, -2.0f, 2.0f);
                                ImGui::SliderFloat("Ortho Top", &ortho_top,
-2.0f, 2.0f);
                                ImGui::SliderFloat("Ortho Near",
&ortho_near, -1.0f, 1.0f);
                                ImGui::SliderFloat("Ortho Far", &ortho_far,
50.0f, 300.0f);
                                //···
                        }
                        else if (tastMark2 == 1) {
                                ImGui::SliderFloat("Fov", &perspective_fov,
10.0f, 20.0f);
                                ImGui::SliderFloat("W/Hratio",
&perspective_w, 0.0f, 1.0f);
                                ImGui::SliderFloat("Near", &perspective_fov,
0.0f, 0.5f);
                                ImGui::SliderFloat("Far", &perspective_far,
```

```
50.0f, 300.0f);
                                        //...
                                }
                                //...
                        }
                        else if(tastMark == 1)
                        {
                            //...
                        }
                        else if (tastMark == 2) {
                            //...
                        }
            }
    }
```

---

- 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）

  OpenGL有六种坐标：物体或模型坐标系(Object or Model Coordinates)，世界坐标系(World Coordinates)，眼坐标或相机坐标(Eye (or Camera) Coordin，ates)，裁剪坐标系(Clip Coordinates)，标准设备坐标系(Normalized Devices Coordinates)，屏幕坐标系(Window (or Screen) Coordinates)。OpenGL显示的过程就是很多步的投影过程，让物体能投影到坐标系中，便于观察。 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。
  这样在多个摄像机的时候，我们比较容易切换观察视角，也比较容易快速的找到需要变换的矩阵。

---

## Bonus

- 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

  此处的camera我是参考官网上写的，主要有的函数如下：

```
// Defines several possible options for camera movement. Used as abstraction
to stay away from window-system specific input methods
enum Camera_Movement {
        FORWARD,
        BACKWARD,
        LEFT,
        RIGHT
};

// Default camera values
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;


// An abstract camera class that processes input and calculates the corresponding Euler Angles, Vectors and Matrices for use in OpenGL
class Camera
{
public:
        // Camera Attributes
        glm::vec3 Position;
        glm::vec3 Front;
        glm::vec3 Up;
        glm::vec3 Right;
        glm::vec3 WorldUp;
        // Euler Angles
        float Yaw;
        float Pitch;
        // Camera options
        float MovementSpeed;
        float MouseSensitivity;
        float Zoom;

        // Constructor with vectors
        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) : Front(glm::vec3(0.0f, 0.0f, -1.0f)), MovementSpeed(SPEED), MouseSensitivity(SENSITIVITY), Zoom(ZOOM){}

    // Constructor with scalar values
        Camera(float posX, float posY, float posZ, float upX, float upY, float upZ, float yaw, float pitch) : Front(glm::vec3(0.0f, 0.0f, -1.0f)), MovementSpeed(SPEED), MouseSensitivity(SENSITIVITY), Zoom(ZOOM){}

        // Returns the view matrix calculated using Euler Angles and the LookAt Matrix
        glm::mat4 GetViewMatrix(){}

        // Processes input received from any keyboard-like input system. Accepts input parameter in the form of camera defined ENUM (to abstract it from windowing systems)
        void ProcessKeyboard(Camera_Movement direction, float deltaTime){}

        // Processes input received from a mouse input system. Expects the offset value in both the x and y direction.
        void ProcessMouseMovement(float xoffset, float yoffset, GLboolean constrainPitch = true){}

        // Processes input received from a mouse scroll-wheel event. Only requires input on the vertical wheel-axis
        void ProcessMouseScroll(float yoffset){}

private:
        // Calculates the front vector from the Camera's (updated) Euler Angles
```

```
        void updateCameraVectors(){}
};
```

在使用的时候，也需要添加几个函数：

```
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
float lastX = WINDOW_WIDTH / 2.0f;
float lastY = WINDOW_HEIGHT / 2.0f;
bool firstMouse = true;

float deltaTime = 0.0f;
float lastFrame = 0.0f;

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
void processInput(GLFWwindow* window);
```

这几个函数是用来处理输入的，读入键盘和鼠标的信号。
在初始化的时候，需要对他们初始化：

```
glfwMakeContextCurrent(window);
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
glfwSetCursorPosCallback(window, mouse_callback);
glfwSetScrollCallback(window, scroll_callback);
```

在渲染的时候，用三个向量记录摄像机的位置，然后使用`glm::lookAt`函数对`view`处理，然后得到`zoom`的值，对`projection`计算透视投影的值。 通过改变`model`的值，可以渲染出多个立方体。
所以在渲染时候的代码为：

```
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom),
(float)WINDOW_WIDTH / (float)WINDOW_HEIGHT, 0.1f, 100.0f);
name = "projection";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(projection));
// camera/view transformation
glm::mat4 view = camera.GetViewMatrix();
name = "view";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(view));

for (unsigned int i = 0; i < 10; i++) {
        model = glm::translate(model, cubePosition[i]);
        float angle = 20.0f * i;
        model = glm::rotate(model, glm::radians(angle), glm::vec3(1.0f,
0.3f, 0.5f));
```

```
        name = "model";
        glUniformMatrix4fv(glGetUniformLocation(shaderProgram, 
name.c_str()), 1, GL_FALSE, glm::value_ptr(model));
        glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

得到的结果请见同一文件夹下的mp4文件。

---