CG assignment3

蔡永宁 11710802

Experimental aims

- know how to use VAO and VBO
- write vertex shader and fragment shader
- know how to draw basic 2-D and 3-D shapes

Experimental setting

glfw, glad
windows 10
GPU
openGL 3.3

Experimental content（key functions with code and clear comments)

draw a rectangle with VBO and VAO

1. initialize glfw and glad
2. create a glfw window
3. create compile and link vertex shader and fragment shader, generate shader program
4. create vertex data
5. create VAO and VBO, bind them
6. render to draw a rectangle

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow* window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

const char* vertexShaderSource = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"void main()\n"
"{\n"
"   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
"}\0";
const char* fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
```

```
"void main()\n"
"{\n"
"   FragColor = vec4(1.0f, 1.0f, 0.0f, 1.0f);\n"
"}\n\0";


int main()
{
    //初始化glfw
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    //创建glfw窗口
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    //加载glad
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }


    //生成着色器程序
    //顶点着色器
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    //片段着色器
    unsigned int fragmentShaderYellow = glCreateShader(GL_FRAGMENT_SHADER);
    //着色器程序
    unsigned int shaderProgramYellow = glCreateProgram();
    //编译
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    glShaderSource(fragmentShaderYellow, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShaderYellow);
    //链接生成
    glAttachShader(shaderProgramYellow, vertexShader);
    glAttachShader(shaderProgramYellow, fragmentShaderYellow);
    glLinkProgram(shaderProgramYellow);

    //顶点数据
    float firstTriangle[] = {
        -0.5f, -0.5f, 0.0f,  // left
        0.5f, 0.5f, 0.0f,  // right
        -0.5f, 0.5f, 0.0f  // top
    };
    float secondTriangle[] = {
```

```
        -0.5f, -0.5f, 0.0f,  // left
        0.5f, -0.5f, 0.0f,  // right
        0.5f, 0.5f, 0.0f   // top
    };
    //创建VBO和VAO
    unsigned int VBOs[2], VAOs[2];
    glGenVertexArrays(2, VAOs);
    glGenBuffers(2, VBOs);
    //第一个三角形 绑定
    glBindVertexArray(VAOs[0]);
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(firstTriangle), firstTriangle,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
    //第二个三角形 绑定
    glBindVertexArray(VAOs[1]);
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(secondTriangle), secondTriangle,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
    glEnableVertexAttribArray(0);



    //线框模式
    //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    //渲染
    while (!glfwWindowShouldClose(window))
    {
        //处理用户操作
        processInput(window);

        //渲染背景
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        //渲染三角形
        glUseProgram(shaderProgramYellow);
        glBindVertexArray(VAOs[0]);
        glDrawArrays(GL_TRIANGLES, 0, 3);
        glBindVertexArray(VAOs[1]);
        glDrawArrays(GL_TRIANGLES, 0, 3);

        //切换缓存
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    //释放内存
    glDeleteVertexArrays(2, VAOs);
    glDeleteBuffers(2, VBOs);
    //终止程序
    glfwTerminate();
    return 0;
}
```

```
//处理用户操作
void processInput(GLFWwindow* window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

//适应窗口大小变化
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

draw a cube

1. initialize glfw and glad
2. create a glfw window
3. create compile and link vertex shader and fragment shader, generate shader program
4. create rotation matrix
5. create vertex data and color data
6. create VAO and VBO, bind them
7. use matrix to change view
8. render to draw a cube

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <iostream>
#include <ctime>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform.hpp>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow* window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

//着色器
const char* vertexShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 vertexPosition_modelspace;\n"
"layout(location = 1) in vec3 vertexColor;\n"
"out vec3 fragmentColor;\n"
"uniform mat4 MVP;\n"
"void main() {\n"
"gl_Position = MVP * vec4(vertexPosition_modelspace, 1);\n"
"fragmentColor = vertexColor;\n"
"}\n\0";
const char* fragmentShaderSource = "#version 330 core\n"
"in vec3 fragmentColor;\n"
"out vec3 color;\n"
"void main()\n"
"{\n"
"    color = fragmentColor;\n"
```

```
"}\n\0";


int main()
{
    //初始化glfw
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    //创建glfw窗口
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "CaiYongning
11710802", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    //加载glad
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);


    //生成着色器程序sss
    //顶点着色器
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    //片段着色器
    unsigned int fragmentShaderYellow = glCreateShader(GL_FRAGMENT_SHADER);
    //着色器程序
    unsigned int shaderProgramYellow = glCreateProgram();
    //编译
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);

    glShaderSource(fragmentShaderYellow, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShaderYellow);

    //链接生成
    glAttachShader(shaderProgramYellow, vertexShader);
    glAttachShader(shaderProgramYellow, fragmentShaderYellow);
    glLinkProgram(shaderProgramYellow);



    //矩阵变换
    // Projection matrix : 45?Field of View, 4:3 ratio, display range : 0.1 unit
<-> 100 units
```

```
    glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 4.0f / 3.0f,
0.1f, 100.0f);
    // Camera matrix
    glm::mat4 View = glm::lookAt(
        glm::vec3(4, 3, 3), // Camera is at (4,3,3), in World Space
        glm::vec3(0, 0, 0), // and looks at the origin
        glm::vec3(0, 1, 0)  // Head is up (set to 0,-1,0 to look upside-down)
    );
    // Model matrix : an identity matrix (model will be at the origin)
    glm::mat4 Model = glm::mat4(1.0f);
    glm::mat4 rotate_model = glm::rotate(30.0f, glm::vec3(1.0f, 0.0f, 0.0f));
    glm::mat4 myMatrix = glm::translate(glm::vec3(2.5f, 0.0f, 0.0f));      //平移
矩阵

    // Our ModelViewProjection : multiplication of our 3 matrices
    glm::mat4 MVP = Projection * View * Model; // Remember, matrix
multiplication is the other way around
    glm::mat4 MVP1 = Projection * View * Model * myMatrix;

    unsigned int MatrixID = glGetUniformLocation(shaderProgramYellow, "MVP");

    //顶点数据
    float vertexs[] = {
        -1.0f,-1.0f,-1.0f,
        -1.0f,-1.0f, 1.0f,
        -1.0f, 1.0f, 1.0f,
         1.0f, 1.0f,-1.0f,
        -1.0f,-1.0f,-1.0f,
        -1.0f, 1.0f,-1.0f,
         1.0f,-1.0f, 1.0f,
        -1.0f,-1.0f,-1.0f,
         1.0f,-1.0f,-1.0f,
         1.0f, 1.0f,-1.0f,
         1.0f,-1.0f,-1.0f,
        -1.0f,-1.0f,-1.0f,
        -1.0f,-1.0f,-1.0f,
        -1.0f, 1.0f, 1.0f,
        -1.0f, 1.0f,-1.0f,
         1.0f,-1.0f, 1.0f,
        -1.0f,-1.0f, 1.0f,
        -1.0f,-1.0f,-1.0f,
        -1.0f, 1.0f, 1.0f,
        -1.0f,-1.0f, 1.0f,
         1.0f,-1.0f, 1.0f,
         1.0f, 1.0f, 1.0f,
         1.0f,-1.0f,-1.0f,
         1.0f, 1.0f,-1.0f,
         1.0f,-1.0f,-1.0f,
         1.0f, 1.0f, 1.0f,
         1.0f,-1.0f, 1.0f,
         1.0f, 1.0f, 1.0f,
         1.0f, 1.0f,-1.0f,
        -1.0f, 1.0f,-1.0f,
         1.0f, 1.0f, 1.0f,
        -1.0f, 1.0f,-1.0f,
        -1.0f, 1.0f, 1.0f,
         1.0f, 1.0f, 1.0f,
        -1.0f, 1.0f, 1.0f,
```

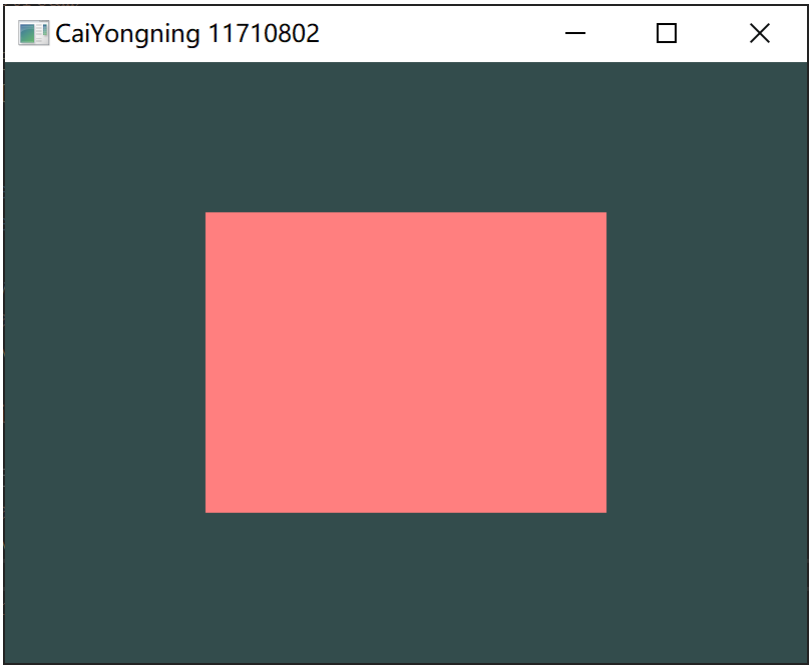
```
        1.0f,-1.0f, 1.0f
    };
    float color[] = {
        0.583f,  0.771f,  0.014f,
        0.609f,  0.115f,  0.436f,
        0.327f,  0.483f,  0.844f,
        0.822f,  0.569f,  0.201f,
        0.435f,  0.602f,  0.223f,
        0.310f,  0.747f,  0.185f,
        0.597f,  0.770f,  0.761f,
        0.559f,  0.436f,  0.730f,
        0.359f,  0.583f,  0.152f,
        0.483f,  0.596f,  0.789f,
        0.559f,  0.861f,  0.639f,
        0.195f,  0.548f,  0.859f,
        0.014f,  0.184f,  0.576f,
        0.771f,  0.328f,  0.970f,
        0.406f,  0.615f,  0.116f,
        0.676f,  0.977f,  0.133f,
        0.971f,  0.572f,  0.833f,
        0.140f,  0.616f,  0.489f,
        0.997f,  0.513f,  0.064f,
        0.945f,  0.719f,  0.592f,
        0.543f,  0.021f,  0.978f,
        0.279f,  0.317f,  0.505f,
        0.167f,  0.620f,  0.077f,
        0.347f,  0.857f,  0.137f,
        0.055f,  0.953f,  0.042f,
        0.714f,  0.505f,  0.345f,
        0.783f,  0.290f,  0.734f,
        0.722f,  0.645f,  0.174f,
        0.302f,  0.455f,  0.848f,
        0.225f,  0.587f,  0.040f,
        0.517f,  0.713f,  0.338f,
        0.053f,  0.959f,  0.120f,
        0.393f,  0.621f,  0.362f,
        0.673f,  0.211f,  0.457f,
        0.820f,  0.883f,  0.371f,
        0.982f,  0.099f,  0.879f
    };
    static GLfloat color_random[12 * 3 * 3];
    srand((unsigned)time(0));
    for (int i = 0; i < 12 * 3; i++)
    {
        color_random[3 * i + 0] = rand() / float(RAND_MAX);
        color_random[3 * i + 1] = rand() / float(RAND_MAX);
        color_random[3 * i + 2] = rand() / float(RAND_MAX);
    }

    //创建VBO和VAO
    unsigned int VBOs[2], VAO;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glGenBuffers(2, VBOs);
    //顶点 绑定
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexs), vertexs, GL_STATIC_DRAW);
```

```
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
    //颜色 绑定
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(color_random), color_random,
GL_STATIC_DRAW);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(1);

    //glBindBuffer(GL_ARRAY_BUFFER, VBOs[1]);
    //glBufferData(GL_ARRAY_BUFFER, sizeof(color), color, GL_STATIC_DRAW);
    //glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    //glEnableVertexAttribArray(1);

    //线框模式
    //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);

    //渲染
    while (!glfwWindowShouldClose(window))
    {


        //处理用户操作
        processInput(window);

        //渲染背景

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glUseProgram(shaderProgramYellow);
        glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);



        glDrawArrays(GL_TRIANGLES, 0, 12 * 3);



        //切换缓存
        //随机颜色
        srand((unsigned)time(0));
        for (int i = 0; i < 12 * 3; i++)
        {
            color_random[3 * i + 0] = rand() / float(RAND_MAX);
            color_random[3 * i + 1] = rand() / float(RAND_MAX);
            color_random[3 * i + 2] = rand() / float(RAND_MAX);
        }


        glfwSwapBuffers(window);
        glfwPollEvents();

        glDisableVertexAttribArray(0);
        glDisableVertexAttribArray(1);
```

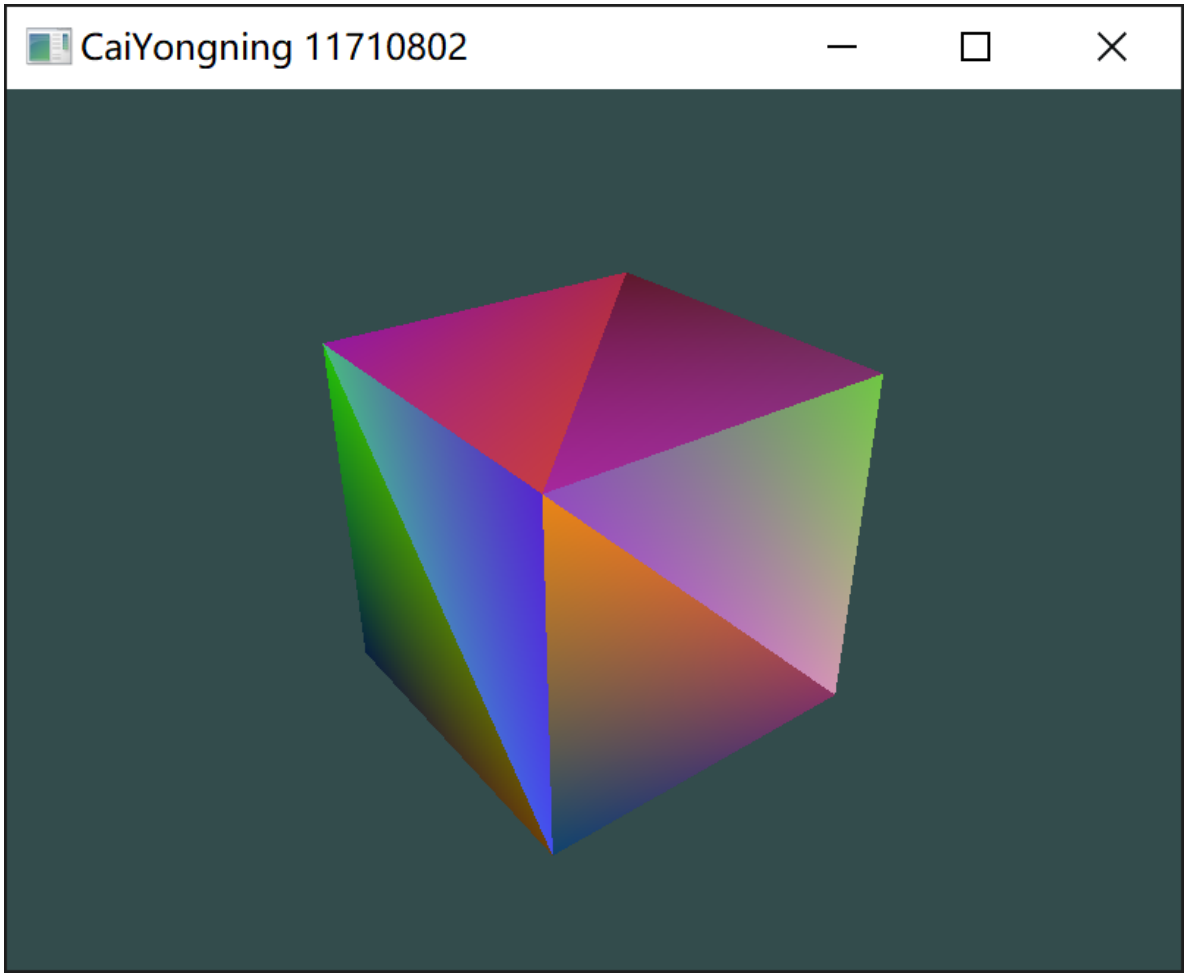
```
        glBindBuffer(GL_ARRAY_BUFFER, VBOs[0]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertexs), vertexs, GL_STATIC_DRAW);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, VBOs[1]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(color_random), color_random,
GL_STATIC_DRAW);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
        glEnableVertexAttribArray(1);
    }

    //释放内存
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(2, VBOs);
    glDeleteProgram(shaderProgramYellow);
    //终止程序
    glfwTerminate();
    return 0;
}

//处理用户操作
void processInput(GLFWwindow* window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

//适应窗口大小变化
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

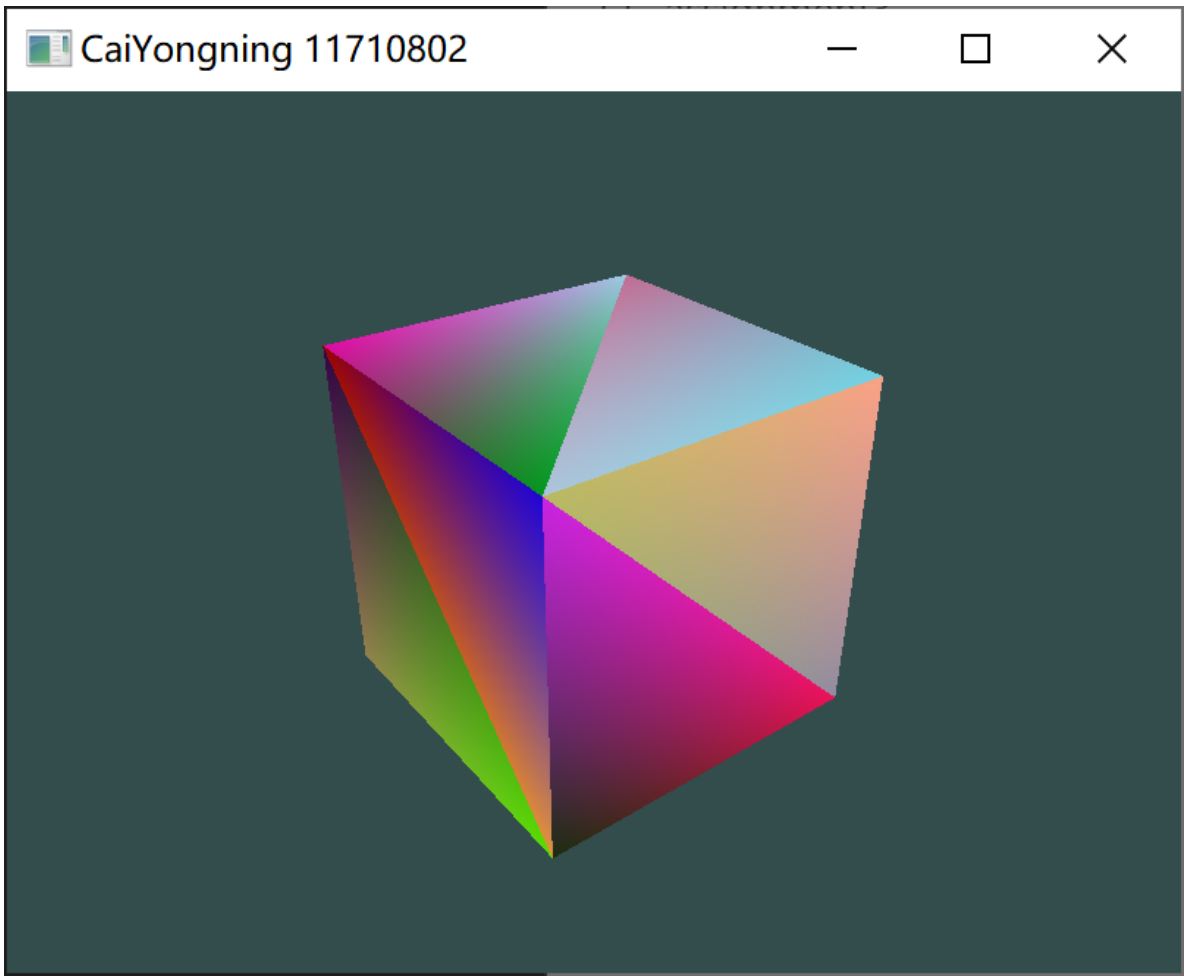
Experimental results

draw a rectangle with VBO and VAO



draw a cube




CaiYongning 11710802