计算机图形学-OpenGL 环境的配置

前言

这文档的初始目的是记录本人在学习计算机图形学初期时配置环境的困难以及解决方式。经过思考后，我觉得应该把这记录升级为教程，希望这个文档可以帮助更多的后人，快速地渡过配置环境的困难。不要没苦硬吃。

正文

综合网上的资源，我发现了一个比较好的懒人包（这包已经放在我的文件夹中了）。接下来将介绍本人的配置流程。注意我用的 vs 是 2022 版本。可能大多数同学是 2019？我觉得应该没什么区别。

Step1)
把这个资源包下载下来(=_=+)。打开这个文件夹，找到子文件夹“openGL”。

Step2)
你现在应该可以看到 openGL 文件夹下方有三个子文件夹以及一个.c 文件。它们分别是：dll、include、lib 和 glad.c.

Step3)
其中 include 文件夹中的内容放入文件夹路径：
"C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.41.34120\include"
其中 lib 文件夹中的内容放入文件夹路径：
"C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.41.34120\lib\x86"
其中 dll 文件夹中的内容放入文件夹路径：
"C:\Windows\SysWOW64"
注意去掉双引号。部分人的 Microsoft Visual Studio 可能在 Program Files (x86)里面？我的不是这样。

Step4)
打开 VS2022，随便新建一个项目，接下来点击“视图(V)*(如果你之前没接触过 VS，这个按钮在右上角)*→解决方案资源管理器(P)→源文件”，将资源包里给出的 glad.c 文件拖到“源文件”处。
注意：在把 glad.c 拖过去之后，如果改变了 glad.c 的路径，你需要重新拖动一次，否则将无法定位路径。

Step5)
右键-源文件→添加→新建项→建立一个 cpp 文件，名字随便。导入头文件

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

接下来即可编写代码。

如果你想看一看自己是否成功，可以复制附录代码到你的 cpp 文件中，如果运行成功，则打印一个红色黑底的三角形。

附录

Algorithm1 OpenGL_Learning

```
#include <iostream>
#include <openGL/glad.h>
#include <GLFW/glfw3.h>
#define GLEW_STATIC
using namespace std;

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

const char *vertex_shader_source =
"#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"
"}\n\0";

const char *fragment_shader_source =
"#version 330 core\n"
"out vec4 FragColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(1.0f, 0.0f, 0.0f, 1.0f);\n"
"}\n\0";

// 定义顶点数组对象和顶点缓冲对象
    unsigned int vertex_array_object;
    unsigned int vertex_buffer_object;

int initial(void) {
    // 定义三角形的顶点数据
    const float triangle[] = {
        -0.5f, -0.5f, 0.0f,
         0.5f, -0.5f, 0.0f,
         0.0f,  0.5f, 0.0f
    };

    // 创建顶点数组对象
    glGenVertexArrays(1, &vertex_array_object);
    // 绑定顶点数组对象
    glBindVertexArray(vertex_array_object);

    // 创建顶点缓冲对象
    glGenBuffers(1, &vertex_buffer_object);
```

```
    // 绑定顶点缓冲对象
    glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_object);
    // 向缓冲对象中写入数据
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    int success;
    char infoLog[512];

    int vertex_shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertex_shader, 1, &vertex_shader_source, NULL);
    glCompileShader(vertex_shader);

    glGetShaderiv(vertex_shader, GL_COMPILE_STATUS, &success);
    if (!success) {
        glGetShaderInfoLog(vertex_shader, 512, NULL, infoLog);
        std::cout  <<  "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n"  <<  infoLog  <<
std::endl;
    }

    int fragment_shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragment_shader, 1, &fragment_shader_source, NULL);
    glCompileShader(fragment_shader);

    glGetShaderiv(fragment_shader, GL_COMPILE_STATUS, &success);
    if (!success) {
        glGetShaderInfoLog(fragment_shader, 512, NULL, infoLog);
        std::cout  <<  "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n"  <<  infoLog  <<
std::endl;
    }

    int shader_program = glCreateProgram();
    glAttachShader(shader_program, vertex_shader);
    glAttachShader(shader_program, fragment_shader);
    glLinkProgram(shader_program);

    glGetProgramiv(shader_program, GL_LINK_STATUS, &success);
    if (!success) {
        glGetProgramInfoLog(shader_program, 512, NULL, infoLog);
        std::cout   <<   "ERROR::SHADER::PROGRAM::LINKING_FAILED\n"   <<   infoLog   <<
std::endl;
    }

    glUseProgram(shader_program);

    return 0;
```

```
}

void Draw(void) {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glBindVertexArray(vertex_array_object);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    glBindVertexArray(0);
}

void reshape(GLFWwindow* window, int width, int height) {
    glViewport(0, 0, width, height);
}

int main() {
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
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);

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

    initial();

    glfwSetFramebufferSizeCallback(window, reshape);

    while (!glfwWindowShouldClose(window)) {
        Draw();
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwDestroyWindow(window);
    glfwTerminate();

}
```