Ubuntu下Clion配置、使用glfw,glad开发OpenGL项目

发布于 2018-10-27 Clion (/tags/Clion/) OpenGL (/tags/OpenGL/) CMake (/tags/CMake/)

Ubuntu下Clion配置、使用glfw,glad开发OpenGL项目

本文主要解决以下问题:

> 1. Ubuntu下glfw、glad、opengl等依赖库的安装
> 2. Ubuntu下Clion　OpenGL项目中CMakeLists.txt文件的配置

本学期计算机图形学要求使用OpenGL库，老师给的OpenGL教程 (https://learnopengl-cn.github.io/)中并没有说明Linux下使用Clion开发OpenGL项目的环境配置，而网上相关的教程很零散，所以答主在这里统一整理一下。

编译安装GLFW

1. 查看是否安装CMake

```
$ cmake -version
cmake version 3.12.3
```

如果没有安装，用以下命令安装

```
$ sudo apt update
$ sudo apt install cmake
```

2. 去官网下载 (www.glfw.org/download.html)最新的GLFW

3. 解压刚刚下载的文件，在同级目录下新建glfw-build文件夹,并切换目录到glfw-build中

4. 在glfw-build目录下打开终端，输入下列命令

```
$ cmake ../glfw-3.2.1 # 这里换成你解压出来的glfw文件夹名
$ make
$ sudo make install
```

5. 如果有类似这样的输出信息，即表示GLFW已配置好

```
Install the project...
-- Install configuration: ""
-- Up-to-date: /usr/local/include/GLFW
-- Up-to-date: /usr/local/include/GLFW/glfw3native.h
-- Up-to-date: /usr/local/include/GLFW/glfw3.h
-- Installing: /usr/local/lib/cmake/glfw3/glfw3Config.cmake
-- Installing: /usr/local/lib/cmake/glfw3/glfw3ConfigVersion.cmake
-- Installing: /usr/local/lib/cmake/glfw3/glfw3Targets.cmake
-- Installing: /usr/local/lib/cmake/glfw3/glfw3Targets-noconfig.cmake
-- Installing: /usr/local/lib/pkgconfig/glfw3.pc
-- Installing: /usr/local/lib/libglfw3.a
```

安装glad库

1. 官网在线生成 (https://glad.dav1d.de/)，选择好需要的版本和模式

点击页面最下方右下角的GENERATE按钮,在随后弹出的页面中，点击glad.zip进行下载

2. 将压缩包中的include文件夹下的glad和KHR拷贝至/usr/local/include中

在Clion中创建项目

1. 将glad压缩包src文件夹中的glad.c移动至新建项目目录中

2. 修改CMakeLists.txt文件内容

```
cmake_minimum_required(VERSION 3.12)
project(YOUR_PROJECT_NAME)

set(CMAKE_CXX_STANDARD 14)

#   添加源文件
set(SOURCE_FILES main.cpp glad.c)
add_executable(YOUR_PROJECT_NAME ${SOURCE_FILES})
target_link_libraries(YOUR_PROJECT_NAME glfw3 GL m Xrandr Xi X11 Xxf86vm pthread dl Xineram
```

测试

在新建项目main.cpp中写入

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

using namespace std;

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 GLchar *vertexShaderSource = "#version 330 core\n"
                                   "layout (location = 0) in vec3 position;\n"
                                   "layout (location = 1) in vec3 color;\n"
                                   "out vec3 ourcolor;\n"
                                   "void main()\n"
                                   "{\n"
                                   " gl_Position = vec4(position,1.0);\n"
                                   " ourcolor = color;\n"
                                   "}\0";

const GLchar *fragmentShaderSource = "#version 330 core\n"
                                     "in vec3 ourcolor;\n"
                                     "out vec4 color;\n"
                                     "void main()\n"
                                     "{\n"
                                     " color = vec4(ourcolor,1.0f);\n"
                                     "}\n\0";

int main() {
    // glfw 初始化和配置
    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,"计算机图形学-homeworl
    if(window == NULL){
        cout<<"Failed to create glfw window"<<endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window,framebuffer_size_callback);

    // 初始化GLAD load all opengl function pointers
    if(!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)){
        cout<<"Failed to initialize GLAD"<<endl;
```

```
    return -1;
}

// 构建编译着色器程序
// vertex shader
int vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader,1,&vertexShaderSource,NULL);
glCompileShader(vertexShader);
// 检查着色器编译是否正确
int success;
char infoLog[512];
glGetShaderiv(vertexShader,GL_COMPILE_STATUS,&success);
if(!success){
    glGetShaderInfoLog(vertexShader,512,NULL,infoLog);
    cout<<"ERROR::SHADER::VERTEX::COMPILATION_FAILED\n"<<infoLog<<endl;
}

// fragment shader
int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader,1,&fragmentShaderSource,NULL);
glCompileShader(fragmentShader);
//检查是否有编译错误
glGetShaderiv(fragmentShader,GL_COMPILE_STATUS,&success);
if(!success){
    glGetShaderInfoLog(fragmentShader,512,NULL,infoLog);
    cout<<"ERROE::SHADER::FRAGMENT::COMPILATION_FAILED\n"<<infoLog<<endl;
}
// link shaders
int shaderProgram = glCreateProgram();
glAttachShader(shaderProgram,vertexShader);
glAttachShader(shaderProgram,fragmentShader);
glLinkProgram(shaderProgram);
// 检查是否有链接错误
glGetProgramiv(shaderProgram,GL_LINK_STATUS,&success);
if(!success){
    glGetProgramInfoLog(shaderProgram,512,NULL,infoLog);
    cout<<"ERROR::SHADER::PROGRAM::LINKING_FAILED\n"<<infoLog<<endl;
}
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);

// 设置顶点数据(buffer)并且配置顶点属性
float vertices[] ={
        // Positions        // Colors
        0.0f,-0.5f,0.0f, 1.0f,0.0f,0.0f,  // 1 Bottom Right
        -0.5f,-0.5f,0.0f, 1.0f,0.0f,0.0f,  // 1 Bottom Left
        0.0f,0.0f,0.0f,  1.0f,0.0f,0.0f,   // 1 Top Right
        -0.5f,0.0f,0.0f,  1.0f,0.0f,0.0f,   // 1 Top Left
        0.0f,0.0f,0.0f, 0.0f,0.0f,1.0f,  // 2 Bottom Right
        -0.5f,0.0f,0.0f, 0.0f,0.0f,1.0f,  // 2 Bottom Left
        0.3f,0.3f,0.0f, 0.0f,0.0f,1.0f,   // 2 Top Right
        -0.2f,0.3f,0.0f, 0.0f,0.0f,1.0f,  // 2 Top Left
        0.3f,-0.2f,0.0f, 0.0f,1.0f,0.0f,  // 3 Bottom Right
        0.0f,-0.5f,0.0f, 0.0f,1.0f,0.0f,  // 3 Bottom Left
        0.3f,0.3f,0.0f,  0.0f,1.0f,0.0f,   // 3 Top Right
        0.0f,0.0f,0.0f,  0.0f,1.0f,0.0f   // 3 Top Left
};

```

```
109     // 绘制矩形的顺序
110     unsigned int indices[] = {  // note that we start from 0!
111         0, 1, 3,  // 1 Triangle
112         0, 2, 3,  // 2 Triangle
113         4, 5, 7,  // 3 Triangle
114         4, 6, 7,   // 4 Triangle
115         8, 9, 11,  // 5 Triangle
116         8, 10, 11   // 6 Triangle
117     };
118
119     unsigned int VBO,VAO,EBO;
120     glGenVertexArrays(1,&VAO);
121     glGenBuffers(1,&VBO);
122     glGenBuffers(1,&EBO);
123     // 首先绑定VAO对象　再绑定VBO对象　然后配置顶点属性
124     glBindVertexArray(VAO);
125
126     glBindBuffer(GL_ARRAY_BUFFER,VBO);
127     glBufferData(GL_ARRAY_BUFFER,sizeof(vertices),vertices,GL_STATIC_DRAW);
128
129     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,EBO);
130     glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(indices),indices,GL_STATIC_DRAW);
131
132     // 配置顶点属性　0
133     glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE,6*sizeof(float),(GLvoid*)0);
134     glEnableVertexAttribArray(0);
135     // 配置顶点属性　1
136     glVertexAttribPointer(1,3,GL_FLOAT,GL_FALSE,6*sizeof(float),(GLvoid*)(3 * sizeof(GLfloat)));
137     glEnableVertexAttribArray(1);
138
139     // unbind VBO
140     glBindBuffer(GL_ARRAY_BUFFER,0);
141     // unbind VAO
142     glBindVertexArray(0);
143
144
145     // render loop
146     while(!glfwWindowShouldClose(window)){
147         // 输入
148         processInput(window);
149
150         // 渲染指令
151         glClearColor(0.2f,0.3f,0.3f,1.0f);
152         glClear(GL_COLOR_BUFFER_BIT);
153
154         // 画出一个三角形
155         glUseProgram(shaderProgram);
156         glBindVertexArray(VAO);
157         //glDrawArrays(GL_TRIANGLES,0,6);
158         glDrawElements(GL_TRIANGLES,18,GL_UNSIGNED_INT,0);
159         // 检查并调用事件　交换缓冲
160         glfwSwapBuffers(window);
161         glfwPollEvents();
162     }
163
164     // 释放资源
165     glDeleteVertexArrays(1,&VAO);
166     glDeleteBuffers(1,&VBO);
```

```
    glDeleteBuffers(1,&EBO);

    // glfw terminate
    glfwTerminate();
    return 0;
}

// 处理键盘输入
void processInput(GLFWwindow *window)
{
    if(glfwGetKey(window,GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window,true);
}

// glfw: whenever the window size changed (by OS or user resize) this callback function executes
void framebuffer_size_callback(GLFWwindow* window,int width,int height)
{
    glViewport(0,0,width,height);
}
```

成功运行!

注:

1. Ubuntu下默认已安装OpenGL库，如未安装可自行安装
2. Clion中CMakeLists.txt文件配置使用教程 (https://juejin.im/post/5a6f32e86fb9a01ca6031230)

如有问题请联系QQ:1020072294

分享到

(http://twitter.com/home?status=http://yoursite.com/2018/10/27/Ubuntu下Clion配置、使用Glfw-Glad开发OpenGL项目/%20Ginger's blog%20Ubuntu下Clion配置、使用glfw,glad开发OpenGL项目)

« 上一篇: Linux终端代理设置 (/2018/12/04/Linux终端代理设置/) 下一篇: Node入门-1 » (/2018/10/16/Node入门-1/)