# 一、GLFW+GLEW+SOIL+GLM

## 窗口及初始化设置

glfwInit();// Initialize GLFW

//\*\*\*设置GLFW，第一变量设置选项类型，第二变量设置数值

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);//设定OpenGL使用版本3.3

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);//设定使用core profile方式编程

glfwWindowHint(GLFW\_RESIZABLE, GL\_FALSE);//设定不能改变窗口大小

//\*\*\*创建窗口对象，第一二变量设置窗口尺寸，第三窗口名称，第四设置哪个显示器使用全屏模式，或NULL为窗口模式，第五设置哪个窗口共享资源，或NULL不共享

GLFWwindow \*window= glfwCreateWindow(WIDTH,HEIGHT, “NAME”,NULL,NULL);//建立失败的话，window值为NULL

glfwMakeContextCurrent(window);//设置为当前使用context

//\*\*\*设置事件回调函数，第一窗口，第二回调函数名（自定义）

glfwSetKeyCallback(window,key\_callback);//键盘按键

glfwSetCursorPosCallback(window,mouse\_callback);//鼠标移动

glfwSetScrollCallback(window,scroll\_callback);//鼠标滚动

//\*\*\*设置输入模式，第一窗口，第二模式，第三设置值

glfwSetInputMode(window,GLFW\_CURSOR,GLFW\_CURSOR\_DISABLED);//设置隐藏光标和窗口封闭住光标

glewExperimental= GL\_TRUE;//使用glew的高阶功能

glewInit();//初始化glew，成功返回值GLEW\_OK

//\*\*\*设置视口

glfwGetFramebufferSize(window,&width,&height);//获取窗口的帧缓存大小

glViewport(0,0,width,height);//设置视口的原点坐标和尺寸（在创建的窗口屏幕坐标上）

//\*\*\*设置OpenGL的功能

glEnable(GL\_DEPTH\_TEST);//启动深度测试

glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_LINE);//设置用线条方式绘制多边形，默认填充方式绘制多边形

glEnable(GL\_CULL\_FACE);//设置face culling

glCullFace(GL\_BACK);

glFrontFace(GL\_CW);//定义决定正面的方式，GL\_CCW为逆时针

## 顶点缓存和顶点信息读取

//\*\*\*设置顶点缓存，顶点数组，单元缓存对象

GLuint VAO,VBO, EBO;//创建存储缓存ID的变量

glGenVertexArrays(1,&VAO);//生成缓存对象，第一生成对象的数量，第二生成对象的ID数组的首地址传送到哪

glGenBuffers(1,&VBO);

glGenBuffers(1,&EBO);

//\*\*\*绑定顶点数组对象，该对象存储所有的和顶点属性有关的设置和缓存设置（VBO与EBO）

glBindVertexArray(VAO);

glBindBuffer(GL\_ARRAY\_BUFFER,VBO);//绑定缓存对象为当前使用的array buffer对象

glBufferData(GL\_ARRAY\_BUFFER,sizeof(vertices),vertices,GL\_STATIC\_DRAW);//将储存顶点信息的数组，传输给缓存对象，第一缓存类型，第二数组大小，第三数组地址，第四绘制方式

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER,EBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER,sizeof(indices),indices,GL\_STATIC\_DRAW);//将储存顶点索引的数组传输给缓存对象

glVertexAttribPointer(0,3,GL\_FLOAT,GL\_FALSE,5\*sizeof(GLfloat),(GLvoid\*)0);//将VBO中数据按照设定格式读入VAO，第一是属性位置（attribute location），第二是属性所含数值个数，第三数值类型，第四是否归一化数值，第五步长（stride），第六数值起始位置

glEnableVertexAttribArray(0); //激活顶点属性，默认不激活，"0"就是顶点着色器中定义的属性位置

glVertexAttribPointer(1,3,GL\_FLOAT,GL\_FALSE,8\*sizeof(float),(GLvoid\*)(3\*sizeof(float)));

glEnableVertexAttribArray(1);

glVertexAttribPointer(2,2,GL\_FLOAT,GL\_FALSE,5\*sizeof(float),(GLvoid\*)(3\*sizeof(float)));

glEnableVertexAttribArray(2);

glBindVertexArray(0);//反绑定 VAO

glBindBuffer(GL\_ARRAY\_BUFFER,0);//反绑定VBO

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER,0); //反绑定EBO一定要在VAO之后

## 纹理读取与设置

## 着色器配置

GLuint program = glCreateProgram(); //创建着色器程序的对象

GLuint shader = glCreateShader(shaderType); //shaderType为枚举，如GL\_COMPUTE\_SHADER, GL\_VERTEX\_SHADER, GL\_TESS\_CONTROL\_SHADER, GL\_TESS\_EVALUATION\_SHADER, GL\_GEOMETRY\_SHADER, or GL\_FRAGMENT\_SHADER

glShaderSource(shader, count, (const GLchar\*\*)source, (const Glint\*)length); //为shader配置内容。source和length分别为着色器程序的内容字符串，及其长度；这两个量都可以是数组，如果length为null，则source须为空字符结尾（实际上内存内容为全0，注意不是0字符）

glCompileShader(shader);

glGetShaderiv(shader, parameterName, (GLint\*)params) //获取shader的信息，parameterName指定信息种类，为枚举，如：GL\_SHADER\_TYPE, GL\_DELETE\_STATUS, GL\_COMPILE\_STATUS, GL\_INFO\_LOG\_LENGTH, GL\_SHADER\_SOURCE\_LENGTH. params是用于存储返回查询内容的变量地址

glGetShaderInfoLog(shader, maxLength, (GLsizei\*)length, (GLchar\*)infoLog) //获取shader日志，maxLength表示用于存储日志的buffer大小，length用于获取日志的实际大小，infoLog是buffer地址

glAttachShader(program, shader);

glLinkProgram(program);

glGetProgramiv(program, parameterName, (GLint\*)params) //获取程序信息，parameterName取值可以是：GL\_DELETE\_STATUS, GL\_LINK\_STATUS, GL\_VALIDATE\_STATUS, GL\_INFO\_LOG\_LENGTH, GL\_ATTACHED\_SHADERS, GL\_ACTIVE\_ATTRIBUTES, GL\_ACTIVE\_ATTRIBUTE\_MAX\_LENGTH, GL\_ACTIVE\_UNIFORMS, GL\_ACTIVE\_UNIFORM\_MAX\_LENGTH.

glGetProgramInfoLog(shader, maxLenth, length, infoLog) //类似shaderInfoLog

glDeleteShader(shaderType);

glUseProgram((GLuint)program);

//\*\*\*设置着色器参数

GLuint mvpID = glGetUniformLocation((GLuint)program, “MVP”); //获取uniformID

glUniformMatrix4fv(mvpID, 1, GL\_FALSE, &mvp[0][0]); //传递uniform值到着色器程序，此类函数有诸多变种，用于传递不同数据和设置

## 渲染步骤

glClearColor(r, g, b, a);

glClear(GL\_COLOR\_BUFFER\_BIT); //也可以位或GL\_DEPTH\_BUFFER\_BIT

glLineWidth(4.0f); //调整GL\_LINE相关类型的绘制线条粗细

glBindVertexArray(vao);

glDrawArrays(GL\_TRIANGLES, 0, 3);

或

glDrawElements(GL\_LINE\_LOOP, 36, GL\_UNSIGNED\_SHORT, 0); //drawarrays按照顺序绘制array buffer的顶点。drawelements按照element array buffer的索引绘制顶点。第一是图元种类，第二是EAB的大小，第三是EAB的数据类型，

glBindVertexArray(0);

//\*\*\*输出内置字体的字符串

glColor3f(1.0, 1.0, 1.0);

glRasterPos3f(x, y, z); //设置输出位置

glutBitmapCharacter((void\*)font, int character); //输出font字体的单个字符，字符串输出需要利用循环，font是枚举，有诸多可选值

//\*\*\*最后输出

glFlush();

或

glutSwapBuffers();

# 二、GLUT方式

## 创建和初始化窗口

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_MULTISAMPLE); //注意不要和glEnable的选项混淆

glutSetOption(GLUT\_MULTISAMPLE, 8);

glutInitWindowSize(1024, 1024);

glutInitWindowPosition(0, 0);

glutCreateWindow(“Hello World”);

## 渲染回调函数注册和主循环

glutDisplayFunc(xxx); //在设置完顶点信息后进行渲染，xxx为执行渲染步骤的函数名

glutPostRedisplay(); //将当前窗口设置为需要重新渲染的窗口，在下一次迭代中会调用注册的displayfunc

glutMainLoop(); //开启glut事件处理主循环，此函数不会直接返回会持续处理各种事件回调

## 输入事件回调函数注册

glutKeyboardFunc(func); //函数原型void func(unsigned char key, int x, int y), x和y是事件触发时鼠标位置，key是ASCII码字符类型

glutKeyboardUpFunc(func); //函数原型一致，放开按键的回调

glutSpecialFunc(func); //函数原型void func(int key, int x, int y)，此函数是针对键盘上的非ASCII类的特殊按键而设计的回调函数注册，key参数需要使用GLUT\_KEY\_\* 系列的枚举

glutSpecialUpFunc(func);

glutMouseFunc(mouseDownFunc); //鼠标按键回调函数，原型为void func(int button, int state, int x, int y)，button是GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON, or GLUT\_RIGHT\_BUTTON，state是GLUT\_UP or GLUT\_DOWN

glutPassiveMotionFunc(mouseMoveFunc); //鼠标移动回调函数，原型void func(int x, int y)，当鼠标在窗口中移动时就会调用，且无点击动作

glutMotionFunc(mouseMoveFunc) //类似上一个，但是是当鼠标移动并有点击动作时调用（并不在乎什么键或多少键）

## 其他GLUT函数

glutGet(GLUT\_ELAPSED\_TIME);

glutTimerFunc(33, func, 0); //每隔一段时间调用func回调函数

# 三、GLM其他常用语句和函数功能

glm::perspective((glm::radians)fieldOfView, aspectRatio, (float)nearPlane, (float)farPlane) //返回透视投影矩阵

glm::frustum(left, right, bottom, top, near, far) //返回透视投影矩阵，视景体定义法

glm::ortho(left, right, bottom, top, zNear, zFar) //返回正交投影矩阵

glm::lookAt((glm::vec3)cameraPos, (glm::vec3)lookAtTarget, (glm::vec3)upDir) //返回视点观察矩阵view

# 四、GLSL常用语句和函数功能

# 附录

## OpenGL Primitives图元

GL\_POINTS 点

GL\_LINES 独立线段

GL\_LINE\_STRIP 折线/连续线段

GL\_LINE\_LOOP 闭合折线

GL\_TRIANGLE 独立三角形

GL\_TRIANGLE\_STRIP 三角形条带

GL\_TRIANGLE\_FAN 所有三角形有一共同顶点的三角形条带