//7.28 加入数据读取

//定位器按照规划点 move=放大倍数\*Δx 精度提高 平滑移动

//放大倍数\*Δx 走位量是精确放大的 精度满足

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <gl/glut.h>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include <learnopengl/filesystem.h>

#include <learnopengl/shader\_m.h>

#include <learnopengl/camera.h>

#include <learnopengl/model.h>

#include<iomanip>

#include <iostream>

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);

void judge\_move();

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode);

void jude\_viewlock();

//void do\_movement();

//void jude\_direction();

void jude\_direction\_change();

void do\_movement\_slider\_1();

void do\_movement\_slider\_2();

void do\_movement\_slider\_3();

void do\_movement\_slider\_4();

void read\_data\_of\_txt();

//------------------------------------ camera自定义

//void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

float yaw = -90.0f;

float pitch = 0.0f;

glm::vec3 cameraPos = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront = glm::vec3(7.5f, 0.0f, 7.5f) - glm::vec3(30.0f, 30.0f, 30.0f);//摄像机视线方向

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

float viewlocker = 0.0f;

glm::vec3 cameraFront\_now = cameraFront;

//glm::mat4 view = view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

//------------------------------------

// settings 设置窗口宽高

const unsigned int SCR\_WIDTH = 1000;

const unsigned int SCR\_HEIGHT = 800;

// camera

Camera camera(glm::vec3(0.0, 0.0, 20.0));

float lastX = SCR\_WIDTH / 2.0;

float lastY = SCR\_HEIGHT / 2.0;

float fov = 45.0;

//float lastX = SCR\_WIDTH;

//float lastY = SCR\_HEIGHT;

bool firstMouse = true;

// timing

float deltaTime = 0.0;

float lastFrame = 0.0;

bool keys[1024];

float angle = 0.0;

GLfloat movespeed = 0.3; //设置的移动速度

GLfloat velocity;

float scale\_proportion = 0.12; //模型大小设置的缩放比例

float smooth\_steps = 1/10; //平滑的步数

float arrive\_interval = 0.01;

double magnify\_slidermove = 1e3; //定位器移动量的放大倍数

struct Point {

double x;

double y;

double z;

};

const int lines\_txt=150; //txt文件中的数据行数

Point trans\_point[4\*lines\_txt];

//Point trans\_point[] = {

//

// { 11702.15617438560, 581.03223938796, 3261.15608958024},

// { 11643.29659054430, 1014.21458105330, 1978.42996869193},

// { 13168.57988926790, 1014.79342543805, 1911.76022939268},

// { 13216.88453858120, 575.78884948872, 3194.10601819121},

//

// { 11702.15617438660 ,581.03223938907, 3261.15608957777 },

// { 11643.29659054530 ,1014.21458105247,1978.42996868880},

// { 13168.57988926890 ,1014.79342543038,1911.76022938731},

// { 13216.88453858210 ,575.78884948302, 3194.10601818653},

//

// { 11702.15617439360 ,581.03223939682, 3261.15608956045},

// { 11643.29659055260 ,1014.21458104671,1978.42996866690 },

// { 13168.57988927550 ,1014.79342537664,1911.76022934978 },

// { 13216.88453858820 ,575.78884944312, 3194.10601815375 }

//};

double arrive\_group\_copy[4][3];

double arrive\_group[4][3];

int length;//length 为传过来的点的个数

//1-4号定位器 初始位置向量 --->注意现实坐标系和 仿真空间坐标系之间的 \*\*映射\*\*

glm::vec3 model\_pos = glm::vec3(0.0, 0.0, 0.0); //定位器底座

glm::vec3 model\_pos\_1 = glm::vec3(0.0, 0.0, 0.0);//X\_1

glm::vec3 model\_pos\_2 = glm::vec3(0.0, 0.0, 0.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_3 = glm::vec3(0.0, 0.0, 0.0);//Z\_1

glm::vec3 model\_pos\_4 = glm::vec3(15.0, 0.0, 0.0); //定位器底座

glm::vec3 model\_pos\_5 = glm::vec3(15.0, 0.0, 0.0);//X\_1

glm::vec3 model\_pos\_6 = glm::vec3(15.0, 0.0, 0.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_7 = glm::vec3(15.0, 0.0, 0.0);//Z\_1

glm::vec3 model\_pos\_8 = glm::vec3(0.0, 0.0, 15.0); //定位器底座

glm::vec3 model\_pos\_9 = glm::vec3(0.0, 0.0, 15.0);//X\_1

glm::vec3 model\_pos\_10 = glm::vec3(0.0, 0.0, 15.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_11 = glm::vec3(0.0, 0.0, 15.0);//Z\_1

glm::vec3 model\_pos\_12 = glm::vec3(15.0, 0.0, 15.0); //定位器底座

glm::vec3 model\_pos\_13 = glm::vec3(15.0, 0.0, 15.0);//X\_1

glm::vec3 model\_pos\_14 = glm::vec3(15.0, 0.0, 15.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_15 = glm::vec3(15.0, 0.0, 15.0);//Z\_1

//1号定位器各运动轴的运动方向

glm::vec3 slider\_1\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_1\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_1\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_2\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_2\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_2\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_3\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_3\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_3\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_4\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_4\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_4\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

double slider\_1\_x\_dir\_value\_change=0;

double slider\_1\_z\_dir\_value\_change = 0;

double slider\_1\_y\_dir\_value\_change = 0;

double slider\_2\_x\_dir\_value\_change = 0;

double slider\_2\_z\_dir\_value\_change = 0;

double slider\_2\_y\_dir\_value\_change = 0;

double slider\_3\_x\_dir\_value\_change = 0;

double slider\_3\_z\_dir\_value\_change = 0;

double slider\_3\_y\_dir\_value\_change = 0;

double slider\_4\_x\_dir\_value\_change = 0;

double slider\_4\_z\_dir\_value\_change = 0;

double slider\_4\_y\_dir\_value\_change = 0;

//标准的三方向参考向量

glm::vec3 x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 y\_direction = glm::vec3(0.0, 1.0, 0.0);

//定位器各方向运动是否到达指定位置 判断标志

bool have\_arrive\_1\_x = false;

bool have\_arrive\_1\_z = false;

bool have\_arrive\_1\_y = false;

bool have\_arrive\_2\_x = false;

bool have\_arrive\_2\_z = false;

bool have\_arrive\_2\_y = false;

bool have\_arrive\_3\_x = false;

bool have\_arrive\_3\_z = false;

bool have\_arrive\_3\_y = false;

bool have\_arrive\_4\_x = false;

bool have\_arrive\_4\_z = false;

bool have\_arrive\_4\_y = false;

bool have\_arrive\_1 = false;

bool have\_arrive\_2 = false;

bool have\_arrive\_3 = false;

bool have\_arrive\_4 = false;

bool have\_one\_loop = false;

int raw\_slider\_1 = 0;

int raw\_slider\_2 = 1;

int raw\_slider\_3 = 2;

int raw\_slider\_4 = 3;

int steps\_slider\_1 = 0;

int steps\_slider\_2 = 0;

int steps\_slider\_3 = 0;

int steps\_slider\_4 = 0;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

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

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

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); // uncomment this statement to fix compilation on OS X

#endif

// glfw window creation

// --------------------

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);

//-------------------------------

glfwSetCursorPosCallback(window, mouse\_callback);

//----------------------------

glfwSetScrollCallback(window, scroll\_callback);

// tell GLFW to capture our mouse

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

//键盘捕捉

glfwSetKeyCallback(window, key\_callback);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

// build and compile shaders

// -------------------------

Shader ourShader("1.model\_loading.vs", "1.model\_loading.fs");

// load models

// -----------

Model ourModel("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_1("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_2("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_3("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_4("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_5("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_6("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_7("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_8("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_9("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_10("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_11("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_12("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_13("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_14("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_15("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

//Model ourModel\_4(FileSystem::getPath("resources/objects/dingweiqi/dizuo.obj"));

// draw in wireframe

//glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

//----------------------------------------------------------------

//第一次走位之前 （进入循环之前）装入目标位置点

//arrive\_1 = glm::vec3(trans\_point[raw\_slider\_1].x, trans\_point[raw\_slider\_1].y, trans\_point[raw\_slider\_1].z);

//arrive\_2 = glm::vec3(trans\_point[raw\_slider\_2].x, trans\_point[raw\_slider\_2].y, trans\_point[raw\_slider\_2].z);

//arrive\_3 = glm::vec3(trans\_point[raw\_slider\_3].x, trans\_point[raw\_slider\_3].y, trans\_point[raw\_slider\_3].z);

//arrive\_4 = glm::vec3(trans\_point[raw\_slider\_4].x, trans\_point[raw\_slider\_4].y, trans\_point[raw\_slider\_4].z);

//arrive\_1\_copy = model\_pos\_3; //arrivecopy 装入起始位置点

//arrive\_2\_copy = model\_pos\_7;

//arrive\_3\_copy = model\_pos\_11;

//arrive\_4\_copy = model\_pos\_15;

read\_data\_of\_txt();//读取txt数据装入 结构体数组中

length = 4\*lines\_txt;//计算传过来的点的个数

//Point结构体中变量类型 float 3\*4 double 3\*4\*2 //改变结构体变量类型记得回来改这里

//-----------------------------------------

//注意现实坐标系和 仿真空间坐标系之间的 \*\*映射\*\*

arrive\_group\_copy[0][0] = trans\_point[raw\_slider\_1].x; arrive\_group\_copy[0][1] = trans\_point[raw\_slider\_1].y;

arrive\_group\_copy[0][2] = trans\_point[raw\_slider\_1].z;

arrive\_group\_copy[1][0] = trans\_point[raw\_slider\_2].x; arrive\_group\_copy[1][1] = trans\_point[raw\_slider\_2].y;

arrive\_group\_copy[1][2] = trans\_point[raw\_slider\_2].z;

arrive\_group\_copy[2][0] = trans\_point[raw\_slider\_3].x; arrive\_group\_copy[2][1] = trans\_point[raw\_slider\_3].y;

arrive\_group\_copy[2][2] = trans\_point[raw\_slider\_3].z;

arrive\_group\_copy[3][0] = trans\_point[raw\_slider\_4].x; arrive\_group\_copy[3][1] = trans\_point[raw\_slider\_4].y;

arrive\_group\_copy[3][2] = trans\_point[raw\_slider\_4].z;

raw\_slider\_1 += 4; raw\_slider\_2 += 4; raw\_slider\_3 += 4; raw\_slider\_4 += 4;

//[][0] [][1] 水平方向 [][2]竖直方向 arrive\_group,arrive\_group\_copy

arrive\_group[0][0] = trans\_point[raw\_slider\_1].x; arrive\_group[0][1] = trans\_point[raw\_slider\_1].y;

arrive\_group[0][2] = trans\_point[raw\_slider\_1].z; //1号定位器 目标点

arrive\_group[1][0] = trans\_point[raw\_slider\_2].x; arrive\_group[1][1] = trans\_point[raw\_slider\_2].y;

arrive\_group[1][2] = trans\_point[raw\_slider\_2].z; //2号定位器 目标点

arrive\_group[2][0] = trans\_point[raw\_slider\_3].x; arrive\_group[2][1] = trans\_point[raw\_slider\_3].y;

arrive\_group[2][2] = trans\_point[raw\_slider\_3].z; //3号定位器 目标点

arrive\_group[3][0] = trans\_point[raw\_slider\_4].x; arrive\_group[3][1] = trans\_point[raw\_slider\_4].y;

arrive\_group[3][2] = trans\_point[raw\_slider\_4].z; //4号定位器 目标点

//--------------------------------------------------------------------

//\*\*\*\*\*\*计算各个定位器三个轴的移动方向\*\*\*\*

//jude\_direction();//计算第一组位置轴的移动方向

jude\_direction\_change();

// render loop //循环开始

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = glfwGetTime();

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

velocity = movespeed \* deltaTime;

// render

// ------

glClearColor(0.3f, 0.5f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// don't forget to enable shader before setting uniforms

ourShader.use();

//

//给定光照

ourShader.setVec3("viewPos", camera.Position);

ourShader.setFloat("shininess", 64.0f);

ourShader.setVec3("light.position", glm::vec3(0.0f, 5.0f, 0.0f));

ourShader.setVec3("light.ambient", 0.5f, 0.5f, 0.5f);

ourShader.setVec3("light.diffuse", 0.8f, 0.8f, 0.8f);

ourShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);

ourShader.setFloat("light.constant", 1.0f);

ourShader.setFloat("light.linear", 0.045f);

ourShader.setFloat("light.quadratic", 0.0075f);

GLfloat sun\_light\_position[] = { 10.0f, 10.0f, 10.0f, 1.0f }; //光源的位置在世界坐标系圆心，齐次坐标形式

GLfloat sun\_light\_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的环境光，为0

GLfloat sun\_light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的漫反射光，全白光

GLfloat sun\_light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式下的镜面光 ，全白光

glLightfv(GL\_LIGHT0, GL\_POSITION, sun\_light\_position);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, sun\_light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, sun\_light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, sun\_light\_specular);

//开启灯光

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

//世界空间变换矩阵初始化

glm::mat4 model = glm::mat4(1.0);

glm::mat4 model\_1 = glm::mat4(1.0);

glm::mat4 model\_2 = glm::mat4(1.0);

glm::mat4 model\_3 = glm::mat4(1.0);

glm::mat4 model\_4 = glm::mat4(1.0);

glm::mat4 model\_5 = glm::mat4(1.0);

glm::mat4 model\_6 = glm::mat4(1.0);

glm::mat4 model\_7 = glm::mat4(1.0);

glm::mat4 model\_8 = glm::mat4(1.0);

glm::mat4 model\_9 = glm::mat4(1.0);

glm::mat4 model\_10 = glm::mat4(1.0);

glm::mat4 model\_11 = glm::mat4(1.0);

glm::mat4 model\_12 = glm::mat4(1.0);

glm::mat4 model\_13 = glm::mat4(1.0);

glm::mat4 model\_14 = glm::mat4(1.0);

glm::mat4 model\_15 = glm::mat4(1.0);

//视角锁定

jude\_viewlock();

// view/projection transformations 观察空间 +视角锁定

//------------------------------------------------------

//视角锁定 Version2.0

/\*if (viewlocker == 1.0)

{

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

}

if (viewlocker==0.0)

{

glm::vec3 cameraPos\_lock = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront\_lock = glm::vec3(-1.0f, -1.0f, -1.0f);//摄像机视线方向

glm::vec3 cameraUp\_lock = glm::vec3(0.0f, 1.0f, 0.0f);

glm::mat4 projection\_lock = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view\_lock = glm::lookAt(cameraPos\_lock, cameraPos\_lock + cameraFront\_lock, cameraUp\_lock);

ourShader.setMat4("projection", projection\_lock);

ourShader.setMat4("view", view\_lock);//view 观察矩阵

}\*/

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

// render the loaded model

//------------------------------------------------

//小车按键控制随意移动

judge\_move(); //执行前进，计算更新位置向量model\_pos\_

//judge\_turn(); //执行旋转，计算更新位置向量model\_pos\_

//---------------------------------------------------------------------

if (have\_one\_loop) //have\_one\_loop==true 则更新下一组 要达到的点

{

if (raw\_slider\_4 < length - 1)//如果不是最后一组位置点-->接着更新位置点组

{

raw\_slider\_1 += 4;

raw\_slider\_2 += 4;

raw\_slider\_3 += 4;

raw\_slider\_4 += 4;

arrive\_group\_copy[0][0] = arrive\_group[0][0]; arrive\_group\_copy[0][1] = arrive\_group[0][1];

arrive\_group\_copy[0][2] = arrive\_group[0][2];

arrive\_group\_copy[1][0] = arrive\_group[1][0]; arrive\_group\_copy[1][1] = arrive\_group[1][1];

arrive\_group\_copy[1][2] = arrive\_group[1][2];

arrive\_group\_copy[2][0] = arrive\_group[2][0]; arrive\_group\_copy[2][1] = arrive\_group[2][1];

arrive\_group\_copy[2][2] = arrive\_group[2][2];

arrive\_group\_copy[3][0] = arrive\_group[3][0]; arrive\_group\_copy[3][1] = arrive\_group[3][1];

arrive\_group\_copy[3][2] = arrive\_group[3][2];

//[][0] [][1] 水平方向 [][2]竖直方向 arrive\_group,arrive\_group\_copy

arrive\_group[0][0] = trans\_point[raw\_slider\_1].x; arrive\_group[0][1] = trans\_point[raw\_slider\_1].y;

arrive\_group[0][2] = trans\_point[raw\_slider\_1].z;

arrive\_group[1][0] = trans\_point[raw\_slider\_2].x; arrive\_group[1][1] = trans\_point[raw\_slider\_2].y;

arrive\_group[1][2] = trans\_point[raw\_slider\_2].z;

arrive\_group[2][0] = trans\_point[raw\_slider\_3].x; arrive\_group[2][1] = trans\_point[raw\_slider\_3].y;

arrive\_group[2][2] = trans\_point[raw\_slider\_3].z;

arrive\_group[3][0] = trans\_point[raw\_slider\_4].x; arrive\_group[3][1] = trans\_point[raw\_slider\_4].y;

arrive\_group[3][2] = trans\_point[raw\_slider\_4].z;

jude\_direction\_change();//重新确定方向

have\_arrive\_1 = false; have\_arrive\_1\_x = false; have\_arrive\_1\_z = false; have\_arrive\_1\_y = false;

have\_arrive\_2 = false; have\_arrive\_2\_x = false; have\_arrive\_2\_z = false; have\_arrive\_2\_y = false;

have\_arrive\_3 = false; have\_arrive\_3\_x = false; have\_arrive\_3\_z = false; have\_arrive\_3\_y = false;

have\_arrive\_4 = false; have\_arrive\_4\_x = false; have\_arrive\_4\_z = false; have\_arrive\_4\_y = false;

have\_one\_loop = false; //重新置位到达状态

}

else

{

have\_arrive\_1 = true;

have\_arrive\_2 = true;

have\_arrive\_3 = true;

have\_arrive\_4 = true;

have\_one\_loop = true; //重新置位到达状态

}

}

//定位器运送物体到指定点

if (!have\_one\_loop)

{

do\_movement\_slider\_1();

do\_movement\_slider\_2();

do\_movement\_slider\_3();

do\_movement\_slider\_4();

if (have\_arrive\_1 && have\_arrive\_2 && have\_arrive\_3 && have\_arrive\_4)

{

have\_one\_loop = true;

}

}

//---------------------------------------------

//定位器底座

model = glm::translate(model, model\_pos); //定义模型每个循环出现的位置

model = glm::scale(model, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model);

ourModel.Draw(ourShader);

//X\_1

// render the loaded model

model\_1 = glm::translate(model\_1, model\_pos\_1); //定义模型每个循环出现的位置

model\_1 = glm::scale(model\_1, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_1);

ourModel\_1.Draw(ourShader);

// Y\_1

// render the loaded model

model\_2 = glm::translate(model\_2, model\_pos\_2); //定义模型每个循环出现的位置

model\_2 = glm::scale(model\_2, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_2);

ourModel\_2.Draw(ourShader);

// Z\_1

// render the loaded model

model\_3 = glm::translate(model\_3, model\_pos\_3); //定义模型每个循环出现的位置

model\_3 = glm::scale(model\_3, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_3);

ourModel\_3.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_4 = glm::translate(model\_4, model\_pos\_4); //定义模型每个循环出现的位置

model\_4 = glm::scale(model\_4, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_4);

ourModel\_4.Draw(ourShader);

//X\_1

// render the loaded model

model\_5 = glm::translate(model\_5, model\_pos\_5); //定义模型每个循环出现的位置

model\_5 = glm::scale(model\_5, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_5);

ourModel\_5.Draw(ourShader);

// Y\_1

// render the loaded model

model\_6 = glm::translate(model\_6, model\_pos\_6); //定义模型每个循环出现的位置

model\_6 = glm::scale(model\_6, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_6);

ourModel\_6.Draw(ourShader);

// Z\_1

// render the loaded model

model\_7 = glm::translate(model\_7, model\_pos\_7); //定义模型每个循环出现的位置

model\_7 = glm::scale(model\_7, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_7);

ourModel\_7.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_8 = glm::translate(model\_8, model\_pos\_8); //定义模型每个循环出现的位置

model\_8 = glm::scale(model\_8, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_8);

ourModel\_8.Draw(ourShader);

//X\_1

// render the loaded model

model\_9 = glm::translate(model\_9, model\_pos\_9); //定义模型每个循环出现的位置

model\_9 = glm::scale(model\_9, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_9);

ourModel\_9.Draw(ourShader);

// Y\_1

// render the loaded model

model\_10 = glm::translate(model\_10, model\_pos\_10); //定义模型每个循环出现的位置

model\_10 = glm::scale(model\_10, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_10);

ourModel\_10.Draw(ourShader);

// Z\_1

// render the loaded model

model\_11 = glm::translate(model\_11, model\_pos\_11); //定义模型每个循环出现的位置

model\_11 = glm::scale(model\_11, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_11);

ourModel\_11.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_12 = glm::translate(model\_12, model\_pos\_12); //定义模型每个循环出现的位置

model\_12 = glm::scale(model\_12, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_12);

ourModel\_12.Draw(ourShader);

//X\_1

// render the loaded model

model\_13 = glm::translate(model\_13, model\_pos\_13); //定义模型每个循环出现的位置

model\_13 = glm::scale(model\_13, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_13);

ourModel\_13.Draw(ourShader);

// Y\_1

// render the loaded model

model\_14 = glm::translate(model\_14, model\_pos\_14); //定义模型每个循环出现的位置

model\_14 = glm::scale(model\_14, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_14);

ourModel\_14.Draw(ourShader);

// Z\_1

// render the loaded model

model\_15 = glm::translate(model\_15, model\_pos\_15); //定义模型每个循环出现的位置

model\_15 = glm::scale(model\_15, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_15);

ourModel\_15.Draw(ourShader);

//--------------------------------------------------------------------------------------------------------------------

// 按键input

// -----------------------------------------------------------------------

processInput(window);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// glfw: terminate, clearing all previously allocated GLFW resources.

// ------------------------------------------------------------------

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

// ---------------------------------------------------------------------------------------------------------

void do\_movement\_slider\_1()

{

//-------------------------------------------------

if (!have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走x方向

{

model\_pos\_1.x += smooth\_steps\*slider\_1\_x\_dir\_value\_change;

model\_pos\_2.x += smooth\_steps\*slider\_1\_x\_dir\_value\_change;

model\_pos\_3.x += smooth\_steps\*slider\_1\_x\_dir\_value\_change;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_x = true;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完Z向

{

model\_pos\_2.z += smooth\_steps\*slider\_1\_z\_dir\_value\_change;

model\_pos\_3.z += smooth\_steps\*slider\_1\_z\_dir\_value\_change;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_z = true;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完y向

{

model\_pos\_3.y += smooth\_steps\*slider\_1\_y\_dir\_value\_change;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_y = true;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && have\_arrive\_1\_y)

{

have\_arrive\_1 = true;

steps\_slider\_1 = 0;

}

}

void do\_movement\_slider\_2()

{

//---------------------------------------------

if (!have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y) //走完x方向

{

model\_pos\_5.x += smooth\_steps\*slider\_2\_x\_dir\_value\_change;

model\_pos\_6.x += smooth\_steps\*slider\_2\_x\_dir\_value\_change;

model\_pos\_7.x += smooth\_steps\*slider\_2\_x\_dir\_value\_change;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_x = true;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_6.z += smooth\_steps\*slider\_2\_z\_dir\_value\_change;

model\_pos\_7.z += smooth\_steps\*slider\_2\_z\_dir\_value\_change;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_z = true;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_7.y += smooth\_steps\*slider\_2\_y\_dir\_value\_change;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_y = true;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && have\_arrive\_2\_y)

{

have\_arrive\_2 = true;

}

}

void do\_movement\_slider\_3()

{

//------------------------------------------------

if (!have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y) //走完x方向

{

model\_pos\_9.x += smooth\_steps\*slider\_3\_x\_dir\_value\_change;

model\_pos\_10.x += smooth\_steps\*slider\_3\_x\_dir\_value\_change;

model\_pos\_11.x += smooth\_steps\*slider\_3\_x\_dir\_value\_change;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_x = true;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_10.z += smooth\_steps\*slider\_3\_z\_dir\_value\_change;

model\_pos\_11.z += smooth\_steps\*slider\_3\_z\_dir\_value\_change;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_z = true;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_11.y += smooth\_steps\*slider\_3\_y\_dir\_value\_change;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_y = true;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && have\_arrive\_3\_y)

{

have\_arrive\_3 = true;

}

}

void do\_movement\_slider\_4()

{

//---------------------------------

if (!have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y) //走完x方向

{

model\_pos\_13.x += smooth\_steps\*slider\_4\_x\_dir\_value\_change;

model\_pos\_14.x += smooth\_steps\*slider\_4\_x\_dir\_value\_change;

model\_pos\_15.x += smooth\_steps\*slider\_4\_x\_dir\_value\_change;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_x = true;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_14.z += smooth\_steps\*slider\_4\_z\_dir\_value\_change;

model\_pos\_15.z += smooth\_steps\*slider\_4\_z\_dir\_value\_change;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_z = true;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_15.y += smooth\_steps\*slider\_4\_y\_dir\_value\_change;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_y = true;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && have\_arrive\_4\_y)

{

have\_arrive\_4 = true;

}

}

void jude\_direction\_change()

{

//注意现实坐标系和 仿真空间坐标系之间的 映射

slider\_1\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[0][0]-arrive\_group\_copy[0][0]);

slider\_1\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[0][1] - arrive\_group\_copy[0][1]);

slider\_1\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[0][2] - arrive\_group\_copy[0][2]);

slider\_2\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[1][0] - arrive\_group\_copy[1][0]);

slider\_2\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[1][1] - arrive\_group\_copy[1][1]);

slider\_2\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[1][2] - arrive\_group\_copy[1][2]);

slider\_3\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[2][0] - arrive\_group\_copy[2][0]);

slider\_3\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[2][1] - arrive\_group\_copy[2][1]);

slider\_3\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[2][2] - arrive\_group\_copy[2][2]);

slider\_4\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[3][0] - arrive\_group\_copy[3][0]);

slider\_4\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[3][1] - arrive\_group\_copy[3][1]);

slider\_4\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[3][2] - arrive\_group\_copy[3][2]);

}

void read\_data\_of\_txt()

{

int i = 0, j = 0;

ifstream fin("D:/5-VScode/test\_15/data\_AGV\_1.txt");

double\*\* p = new double\*[16];

for (int i = 0; i < 16; i++)

{

p[i] = new double[13];

}

while (!fin.eof())//判断是否输入结束

{

for (i = 0; i < 16; i++)

{ for (j = 0; j < 13; j++)

{

fin >> p[i][j];}

}

}

for (i = 0; i < 16; i++)

{

trans\_point[4 \* i + 0] = { p[i][1],p[i][2],p[i][3] };

trans\_point[4 \* i + 1] = { p[i][4],p[i][5],p[i][6] };

trans\_point[4 \* i + 2] = { p[i][7],p[i][8],p[i][9] };

trans\_point[4 \* i + 3] = { p[i][10],p[i][11],p[i][12] };

}

for (int i = 0; i < 16; i++)

{

delete[]p[i];

}

delete[]p;

fin.close();

}

/\* void do\_movement\_slider\_1()

{

//-------------------------------------------------

if (!have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走x方向

{

model\_pos\_1 += smooth\_steps\*slider\_1\_x\_direction;

model\_pos\_2 += smooth\_steps\*slider\_1\_x\_direction;

model\_pos\_3 += smooth\_steps\*slider\_1\_x\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_x = true;

steps\_slider\_1 = 0;

model\_pos\_1.x += slider\_1\_x\_direction.x;

model\_pos\_2.x += slider\_1\_x\_direction.x;

model\_pos\_3.x += slider\_1\_x\_direction.x;

}

}

if (have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完Z向

{

model\_pos\_2 += smooth\_steps\*slider\_1\_z\_direction;

model\_pos\_3 += smooth\_steps\*slider\_1\_z\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_z = true;

model\_pos\_2.z += slider\_1\_z\_direction.z;

model\_pos\_3.z += slider\_1\_z\_direction.z;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完y向

{

model\_pos\_3 += smooth\_steps\*slider\_1\_y\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_y = true;

model\_pos\_3.y += slider\_1\_y\_direction.y;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && have\_arrive\_1\_y)

{

have\_arrive\_1 = true;

steps\_slider\_1 = 0;

}

}

\*/

//void jude\_direction()

//{

//

// //--------------------------------------------------------------

// //1号定位器各轴移动方向

// //通过对glm::vec3(arrive\_1.x - arrive\_1\_copy.x, 0, 0) 的检测，勉强能够达到七位有效数字的精度

//

// slider\_1\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_1.x - arrive\_1\_copy.x, 0, 0);

// slider\_1\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_1.z - arrive\_1\_copy.z);

// slider\_1\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_1.y - arrive\_1\_copy.y, 0);

//

// //--------------------------------------------------------------

// //2号定位器各轴移动方向

//

// slider\_2\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_2.x - arrive\_2\_copy.x, 0, 0);

// slider\_2\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_2.z - arrive\_2\_copy.z);

// slider\_2\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_2.y - arrive\_2\_copy.y, 0);

//

//

// //--------------------------------------------------------------

// //3号定位器各轴移动方向

//

// slider\_3\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_3.x - arrive\_3\_copy.x, 0, 0);

// slider\_3\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_3.z - arrive\_3\_copy.z);

// slider\_3\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_3.y - arrive\_3\_copy.y, 0);

//

// //--------------------------------------------------------------

// //4号定位器各轴移动方向

//

// slider\_4\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_4.x - arrive\_4\_copy.x, 0, 0);

// slider\_4\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_4.z - arrive\_4\_copy.z);

// slider\_4\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_4.y - arrive\_4\_copy.y, 0);

//

//}

/\*

void jude\_direction()

{

//--------------------------------------------------------------

//1号定位器各轴移动方向

if (arrive\_1.x - model\_pos\_3.x > 0)

{

slider\_1\_x\_direction = 1.0f\*x\_direction;

}

else

{

slider\_1\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_1.z - model\_pos\_3.z > 0)

{

slider\_1\_z\_direction = 1.0f\*z\_direction;

}

else

{

slider\_1\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_1.y - model\_pos\_3.y > 0)

{

slider\_1\_y\_direction = 1.0f\*y\_direction;

}

else

{

slider\_1\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//2号定位器各轴移动方向

if (arrive\_2.x - model\_pos\_7.x > 0)

{

slider\_2\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_2\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_2.z - model\_pos\_7.z > 0) {

slider\_2\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_2\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_2.y - model\_pos\_7.y > 0) {

slider\_2\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_2\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//3号定位器各轴移动方向

if (arrive\_3.x - model\_pos\_11.x > 0) {

slider\_3\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_3\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_3.z - model\_pos\_11.z > 0) {

slider\_3\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_3\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_3.y - model\_pos\_11.y > 0) {

slider\_3\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_3\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//四号定位器各轴移动方向

if (arrive\_4.x - model\_pos\_15.x > 0) {

slider\_4\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_4\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_4.z - model\_pos\_15.z > 0) {

slider\_4\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_4\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_4.y - model\_pos\_15.y > 0) {

slider\_4\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_4\_y\_direction = -1.0f\*y\_direction;

}

}\*/

void processInput(GLFWwindow\* window)

{

//float angle = 0.0f;

glm::vec3 front = glm::vec3(0.0f, 0.0f, -1.0f);

float cameraSpeed = 2.5f \* deltaTime;

//float cameraSpeed = 0.05f; // adjust accordingly

if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS)

cameraPos += cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS)

cameraPos -= cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS)

cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS)

cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

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)

{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

if (fov >= 1.0f && fov <= 45.0f)

fov -= yoffset;

if (fov <= 1.0f)

fov = 1.0f;

if (fov >= 45.0f)

fov = 45.0f;

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GL\_TRUE);

if (key >= 0 && key < 1024)

{

//设置按下/释放键为true或false

if (action == GLFW\_PRESS)

keys[key] = true;

else if (action == GLFW\_RELEASE)

keys[key] = false;

}

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos)

{

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos;

lastX = xpos;

lastY = ypos;

float sensitivity = 0.1;

xoffset \*= sensitivity;

yoffset \*= sensitivity;

yaw += xoffset;

pitch += yoffset;

if (pitch > 89.0f)

pitch = 89.0f;

if (pitch < -89.0f)

pitch = -89.0f;

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

//--------------------------------------------

//视角锁定 Version1.0

if (viewlocker == 1.0)

{

cameraFront = glm::normalize(front);

cameraFront\_now = cameraFront;

}

if (viewlocker == 0.0)

{

cameraFront = cameraFront\_now;//摄像机视线方向

//cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

}

}

void jude\_viewlock()

{

if (keys[GLFW\_KEY\_X])//X 视角解锁

{

viewlocker = 1.0;

}

if (keys[GLFW\_KEY\_C])//C 视角锁定

{

viewlocker = 0.0;

}

}

void judge\_move()

{

//前进后退计算更新位置向量

if (keys[GLFW\_KEY\_A])

{

model\_pos\_1 += velocity\*x\_direction;

model\_pos\_2 += velocity\*x\_direction;

model\_pos\_3 += velocity\*x\_direction;

model\_pos\_5 += velocity\*x\_direction;

model\_pos\_6 += velocity\*x\_direction;

model\_pos\_7 += velocity\*x\_direction;

model\_pos\_9 += velocity\*x\_direction;

model\_pos\_10 += velocity\*x\_direction;

model\_pos\_11 += velocity\*x\_direction;

model\_pos\_13 += velocity\*x\_direction;

model\_pos\_14 += velocity\*x\_direction;

model\_pos\_15 += velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_D])

{

model\_pos\_1 -= velocity\*x\_direction;

model\_pos\_2 -= velocity\*x\_direction;

model\_pos\_3 -= velocity\*x\_direction;

model\_pos\_5 -= velocity\*x\_direction;

model\_pos\_6 -= velocity\*x\_direction;

model\_pos\_7 -= velocity\*x\_direction;

model\_pos\_9 -= velocity\*x\_direction;

model\_pos\_10 -= velocity\*x\_direction;

model\_pos\_11 -= velocity\*x\_direction;

model\_pos\_13 -= velocity\*x\_direction;

model\_pos\_14 -= velocity\*x\_direction;

model\_pos\_15 -= velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_W])

{

model\_pos\_2 += velocity\*z\_direction;

model\_pos\_3 += velocity\*z\_direction;

model\_pos\_6 += velocity\*z\_direction;

model\_pos\_7 += velocity\*z\_direction;

model\_pos\_10 += velocity\*z\_direction;

model\_pos\_11 += velocity\*z\_direction;

model\_pos\_14 += velocity\*z\_direction;

model\_pos\_15 += velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_S])

{

model\_pos\_2 -= velocity\*z\_direction;

model\_pos\_3 -= velocity\*z\_direction;

model\_pos\_6 -= velocity\*z\_direction;

model\_pos\_7 -= velocity\*z\_direction;

model\_pos\_10 -= velocity\*z\_direction;

model\_pos\_11 -= velocity\*z\_direction;

model\_pos\_14 -= velocity\*z\_direction;

model\_pos\_15 -= velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_R])

{

model\_pos\_3 += velocity\*y\_direction;

model\_pos\_7 += velocity\*y\_direction;

model\_pos\_11 += velocity\*y\_direction;

model\_pos\_15 += velocity\*y\_direction;

}

if (keys[GLFW\_KEY\_F])

{

model\_pos\_3 -= velocity\*y\_direction;

model\_pos\_7 -= velocity\*y\_direction;

model\_pos\_11 -= velocity\*y\_direction;

model\_pos\_15 -= velocity\*y\_direction;

}

}

//小车按照规划路线自移动 实现 ---新办法

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <gl/glut.h>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include <learnopengl/filesystem.h>

#include <learnopengl/shader\_m.h>

#include <learnopengl/camera.h>

#include <learnopengl/model.h>

#include <iostream>

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);

void judge\_move();

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode);

void jude\_viewlock();

//void do\_movement();

void jude\_direction();

void do\_movement\_slider\_1();

void do\_movement\_slider\_2();

void do\_movement\_slider\_3();

void do\_movement\_slider\_4();

//------------------------------------ camera自定义

//void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

float yaw = -90.0f;

float pitch = 0.0f;

glm::vec3 cameraPos = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront = glm::vec3(15.0f, 0.0f, 15.0f) - glm::vec3(30.0f, 30.0f, 30.0f);//摄像机视线方向

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

float viewlocker = 0.0f;

glm::vec3 cameraFront\_now = cameraFront;

//glm::mat4 view = view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

//------------------------------------

// settings 设置窗口宽高

const unsigned int SCR\_WIDTH = 1000;

const unsigned int SCR\_HEIGHT = 800;

// camera

Camera camera(glm::vec3(0.0f, 0.0f, 20.0f));

float lastX = SCR\_WIDTH / 2.0f;

float lastY = SCR\_HEIGHT / 2.0f;

float fov = 45.0f;

//float lastX = SCR\_WIDTH;

//float lastY = SCR\_HEIGHT;

bool firstMouse = true;

// timing

float deltaTime = 0.0f;

float lastFrame = 0.0f;

//1-4号定位器 初始位置向量

glm::vec3 model\_pos = glm::vec3(10.0f, 0.0f, 10.0f); //定位器底座

glm::vec3 model\_pos\_1 = glm::vec3(10.0f, 0.0f, 10.0f);//X\_1

glm::vec3 model\_pos\_2 = glm::vec3(10.0f, 0.0f, 10.0f);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_3 = glm::vec3(10.0f, 0.0f, 10.0f);//Z\_1

glm::vec3 model\_pos\_4 = glm::vec3(20.0f, 0.0f, 10.0f); //定位器底座

glm::vec3 model\_pos\_5 = glm::vec3(20.0f, 0.0f, 10.0f);//X\_1

glm::vec3 model\_pos\_6 = glm::vec3(20.0f, 0.0f, 10.0f);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_7 = glm::vec3(20.0f, 0.0f, 10.0f);//Z\_1

glm::vec3 model\_pos\_8 = glm::vec3(10.0f, 0.0f, 20.0f); //定位器底座

glm::vec3 model\_pos\_9 = glm::vec3(10.0f, 0.0f, 20.0f);//X\_1

glm::vec3 model\_pos\_10 = glm::vec3(10.0f, 0.0f, 20.0f);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_11 = glm::vec3(10.0f, 0.0f, 20.0f);//Z\_1

glm::vec3 model\_pos\_12 = glm::vec3(20.0f, 0.0f, 20.0f); //定位器底座

glm::vec3 model\_pos\_13 = glm::vec3(20.0f, 0.0f, 20.0f);//X\_1

glm::vec3 model\_pos\_14 = glm::vec3(20.0f, 0.0f, 20.0f);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_15 = glm::vec3(20.0f, 0.0f, 20.0f);//Z\_1

bool keys[1024];

float angle = 0.0f;

GLfloat movespeed = 0.3f; //设置的移动速度

GLfloat velocity;

float scale\_proportion = 0.1f; //设置的缩放比例

struct Point {

float x;

float y;

float z;

};

Point trans\_point[] = {

{ 11.5f,2.0f,9.0f },

{ 18.5f, 1.0f, 11.0f },

{ 9.0f, 1.5f, 20.5f },

{ 21.0f, 1.0f, 20.0f },

{ 10.0f, 0.0f, 10.0f },

{ 20.0f, 0.0f, 10.0f },

{ 10.0f, 0.0f, 20.0f },

{ 20.0f, 0.0f, 20.0f }

};

int length;//length 为传过来的点的个数

//1-4号定位器要到达的点

float arrive[][3] = {

{ 11.5f,2.0f,9.0f },

{ 18.5f, 1.0f, 11.0f },

{ 9.0f, 1.5f, 20.5f },

{ 21.0f, 1.0f, 20.0f },

{ 10.0f, 0.0f, 10.0f },

{ 20.0f, 0.0f, 10.0f },

{ 10.0f, 0.0f, 20.0f },

{ 20.0f, 0.0f, 20.0f }

};

glm::vec3 arrive\_1 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_2 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_3 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_4 = glm::vec3(0, 0, 0);

//1号定位器各运动轴的运动方向

glm::vec3 slider\_1\_x\_direction = glm::vec3(1.0f, 0.0f, 0.0f);

glm::vec3 slider\_1\_z\_direction = glm::vec3(0.0f, 0.0f, 1.0f);

glm::vec3 slider\_1\_y\_direction = glm::vec3(0.0f, 1.0f, 0.0f);

glm::vec3 slider\_2\_x\_direction = glm::vec3(1.0f, 0.0f, 0.0f);

glm::vec3 slider\_2\_z\_direction = glm::vec3(0.0f, 0.0f, 1.0f);

glm::vec3 slider\_2\_y\_direction = glm::vec3(0.0f, 1.0f, 0.0f);

glm::vec3 slider\_3\_x\_direction = glm::vec3(1.0f, 0.0f, 0.0f);

glm::vec3 slider\_3\_z\_direction = glm::vec3(0.0f, 0.0f, 1.0f);

glm::vec3 slider\_3\_y\_direction = glm::vec3(0.0f, 1.0f, 0.0f);

glm::vec3 slider\_4\_x\_direction = glm::vec3(1.0f, 0.0f, 0.0f);

glm::vec3 slider\_4\_z\_direction = glm::vec3(0.0f, 0.0f, 1.0f);

glm::vec3 slider\_4\_y\_direction = glm::vec3(0.0f, 1.0f, 0.0f);

//标准的三方向参考向量

glm::vec3 x\_direction = glm::vec3(1.0f, 0.0f, 0.0f);

glm::vec3 z\_direction = glm::vec3(0.0f, 0.0f, 1.0f);

glm::vec3 y\_direction = glm::vec3(0.0f, 1.0f, 0.0f);

//定位器各方向运动是否到达指定位置 判断标志

bool have\_arrive\_1\_x = false;

bool have\_arrive\_1\_z = false;

bool have\_arrive\_1\_y = false;

bool have\_arrive\_2\_x = false;

bool have\_arrive\_2\_z = false;

bool have\_arrive\_2\_y = false;

bool have\_arrive\_3\_x = false;

bool have\_arrive\_3\_z = false;

bool have\_arrive\_3\_y = false;

bool have\_arrive\_4\_x = false;

bool have\_arrive\_4\_z = false;

bool have\_arrive\_4\_y = false;

bool have\_arrive\_1 = false;

bool have\_arrive\_2 = false;

bool have\_arrive\_3 = false;

bool have\_arrive\_4 = false;

bool have\_one\_loop = false;

int raw\_slider\_1 = 0;

int raw\_slider\_2 = 1;

int raw\_slider\_3 = 2;

int raw\_slider\_4 = 3;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

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

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

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); // uncomment this statement to fix compilation on OS X

#endif

// glfw window creation

// --------------------

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);

//-------------------------------

glfwSetCursorPosCallback(window, mouse\_callback);

//----------------------------

glfwSetScrollCallback(window, scroll\_callback);

// tell GLFW to capture our mouse

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

//键盘捕捉

glfwSetKeyCallback(window, key\_callback);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

// build and compile shaders

// -------------------------

Shader ourShader("1.model\_loading.vs", "1.model\_loading.fs");

// load models

// -----------

Model ourModel("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_1("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_2("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_3("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_4("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_5("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_6("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_7("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_8("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_9("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_10("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_11("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_12("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_13("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_14("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_15("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

//Model ourModel\_4(FileSystem::getPath("resources/objects/dingweiqi/dizuo.obj"));

// draw in wireframe

//glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

//----------------------------------------------------------------

arrive\_1 = glm::vec3(arrive[0][0], arrive[0][1], arrive[0][2]);

arrive\_2 = glm::vec3(arrive[1][0], arrive[1][1], arrive[1][2]);

arrive\_3 = glm::vec3(arrive[2][0], arrive[2][1], arrive[2][2]);

arrive\_4 = glm::vec3(arrive[3][0], arrive[3][1], arrive[3][2]);

length = sizeof(trans\_point) / 12;//计算传过来的点的个数

//--------------------------------------------------------------------

//\*\*\*\*\*\*计算各个定位器三个轴的移动方向\*\*\*\*

jude\_direction();//计算第一组位置轴的移动方向

// render loop

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = glfwGetTime();

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

velocity = movespeed \* deltaTime;

// render

// ------

glClearColor(0.3f, 0.5f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// don't forget to enable shader before setting uniforms

ourShader.use();

//

//给定光照

ourShader.setVec3("viewPos", camera.Position);

ourShader.setFloat("shininess", 64.0f);

ourShader.setVec3("light.position", glm::vec3(0.0f, 5.0f, 0.0f));

ourShader.setVec3("light.ambient", 0.5f, 0.5f, 0.5f);

ourShader.setVec3("light.diffuse", 0.8f, 0.8f, 0.8f);

ourShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);

ourShader.setFloat("light.constant", 1.0f);

ourShader.setFloat("light.linear", 0.045f);

ourShader.setFloat("light.quadratic", 0.0075f);

GLfloat sun\_light\_position[] = { 10.0f, 10.0f, 10.0f, 1.0f }; //光源的位置在世界坐标系圆心，齐次坐标形式

GLfloat sun\_light\_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的环境光，为0

GLfloat sun\_light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的漫反射光，全白光

GLfloat sun\_light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式下的镜面光 ，全白光

glLightfv(GL\_LIGHT0, GL\_POSITION, sun\_light\_position);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, sun\_light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, sun\_light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, sun\_light\_specular);

//开启灯光

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

//世界空间变换矩阵初始化

glm::mat4 model = glm::mat4(1.0f);

glm::mat4 model\_1 = glm::mat4(1.0f);

glm::mat4 model\_2 = glm::mat4(1.0f);

glm::mat4 model\_3 = glm::mat4(1.0f);

glm::mat4 model\_4 = glm::mat4(1.0f);

glm::mat4 model\_5 = glm::mat4(1.0f);

glm::mat4 model\_6 = glm::mat4(1.0f);

glm::mat4 model\_7 = glm::mat4(1.0f);

glm::mat4 model\_8 = glm::mat4(1.0f);

glm::mat4 model\_9 = glm::mat4(1.0f);

glm::mat4 model\_10 = glm::mat4(1.0f);

glm::mat4 model\_11 = glm::mat4(1.0f);

glm::mat4 model\_12 = glm::mat4(1.0f);

glm::mat4 model\_13 = glm::mat4(1.0f);

glm::mat4 model\_14 = glm::mat4(1.0f);

glm::mat4 model\_15 = glm::mat4(1.0f);

//视角锁定

jude\_viewlock();

// view/projection transformations 观察空间 +视角锁定

//------------------------------------------------------

//视角锁定 Version2.0

/\*if (viewlocker == 1.0)

{

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

}

if (viewlocker==0.0)

{

glm::vec3 cameraPos\_lock = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront\_lock = glm::vec3(-1.0f, -1.0f, -1.0f);//摄像机视线方向

glm::vec3 cameraUp\_lock = glm::vec3(0.0f, 1.0f, 0.0f);

glm::mat4 projection\_lock = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view\_lock = glm::lookAt(cameraPos\_lock, cameraPos\_lock + cameraFront\_lock, cameraUp\_lock);

ourShader.setMat4("projection", projection\_lock);

ourShader.setMat4("view", view\_lock);//view 观察矩阵

}\*/

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

// render the loaded model

//------------------------------------------------

//小车按键控制随意移动

judge\_move(); //执行前进，计算更新位置向量model\_pos\_

//judge\_turn(); //执行旋转，计算更新位置向量model\_pos\_

//---------------------------------------------------------------------

if (have\_one\_loop) //have\_one\_loop==true 则更新下一组 要达到的点

{

if (raw\_slider\_4 < length - 1)//8 换成i 数一下传过来的数组有几行

{

raw\_slider\_1 += 4;

raw\_slider\_2 += 4;

raw\_slider\_3 += 4;

raw\_slider\_4 += 4;

}

arrive\_1 = glm::vec3(arrive[raw\_slider\_1][0], arrive[raw\_slider\_1][1], arrive[raw\_slider\_1][2]);

arrive\_2 = glm::vec3(arrive[raw\_slider\_2][0], arrive[raw\_slider\_2][1], arrive[raw\_slider\_2][2]);

arrive\_3 = glm::vec3(arrive[raw\_slider\_3][0], arrive[raw\_slider\_3][1], arrive[raw\_slider\_3][2]);

arrive\_4 = glm::vec3(arrive[raw\_slider\_4][0], arrive[raw\_slider\_4][1], arrive[raw\_slider\_4][2]);

jude\_direction();

have\_arrive\_1 = false; have\_arrive\_1\_x = false; have\_arrive\_1\_z = false; have\_arrive\_1\_y = false;

have\_arrive\_2 = false; have\_arrive\_2\_x = false; have\_arrive\_2\_z = false; have\_arrive\_2\_y = false;

have\_arrive\_3 = false; have\_arrive\_3\_x = false; have\_arrive\_3\_z = false; have\_arrive\_3\_y = false;

have\_arrive\_4 = false; have\_arrive\_4\_x = false; have\_arrive\_4\_z = false; have\_arrive\_4\_y = false;

have\_one\_loop = false;

}

//定位器运送物体到指定点

do\_movement\_slider\_1();

do\_movement\_slider\_2();

do\_movement\_slider\_3();

do\_movement\_slider\_4();

if ( have\_arrive\_1 && have\_arrive\_2 && have\_arrive\_3 && have\_arrive\_4)

{

have\_one\_loop = true;

}

//---------------------------------------------

//定位器底座

model = glm::translate(model, model\_pos); //定义模型每个循环出现的位置

model = glm::scale(model, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model);

ourModel.Draw(ourShader);

//X\_1

// render the loaded model

model\_1 = glm::translate(model\_1, model\_pos\_1); //定义模型每个循环出现的位置

model\_1 = glm::scale(model\_1, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_1);

ourModel\_1.Draw(ourShader);

// Y\_1

// render the loaded model

model\_2 = glm::translate(model\_2, model\_pos\_2); //定义模型每个循环出现的位置

model\_2 = glm::scale(model\_2, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_2);

ourModel\_2.Draw(ourShader);

// Z\_1

// render the loaded model

model\_3 = glm::translate(model\_3, model\_pos\_3); //定义模型每个循环出现的位置

model\_3 = glm::scale(model\_3, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_3);

ourModel\_3.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_4 = glm::translate(model\_4, model\_pos\_4); //定义模型每个循环出现的位置

model\_4 = glm::scale(model\_4, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_4);

ourModel\_4.Draw(ourShader);

//X\_1

// render the loaded model

model\_5 = glm::translate(model\_5, model\_pos\_5); //定义模型每个循环出现的位置

model\_5 = glm::scale(model\_5, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_5);

ourModel\_5.Draw(ourShader);

// Y\_1

// render the loaded model

model\_6 = glm::translate(model\_6, model\_pos\_6); //定义模型每个循环出现的位置

model\_6 = glm::scale(model\_6, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_6);

ourModel\_6.Draw(ourShader);

// Z\_1

// render the loaded model

model\_7 = glm::translate(model\_7, model\_pos\_7); //定义模型每个循环出现的位置

model\_7 = glm::scale(model\_7, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_7);

ourModel\_7.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_8 = glm::translate(model\_8, model\_pos\_8); //定义模型每个循环出现的位置

model\_8 = glm::scale(model\_8, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_8);

ourModel\_8.Draw(ourShader);

//X\_1

// render the loaded model

model\_9 = glm::translate(model\_9, model\_pos\_9); //定义模型每个循环出现的位置

model\_9 = glm::scale(model\_9, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_9);

ourModel\_9.Draw(ourShader);

// Y\_1

// render the loaded model

model\_10 = glm::translate(model\_10, model\_pos\_10); //定义模型每个循环出现的位置

model\_10 = glm::scale(model\_10, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_10);

ourModel\_10.Draw(ourShader);

// Z\_1

// render the loaded model

model\_11 = glm::translate(model\_11, model\_pos\_11); //定义模型每个循环出现的位置

model\_11 = glm::scale(model\_11, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_11);

ourModel\_11.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_12 = glm::translate(model\_12, model\_pos\_12); //定义模型每个循环出现的位置

model\_12 = glm::scale(model\_12, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_12);

ourModel\_12.Draw(ourShader);

//X\_1

// render the loaded model

model\_13 = glm::translate(model\_13, model\_pos\_13); //定义模型每个循环出现的位置

model\_13 = glm::scale(model\_13, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_13);

ourModel\_13.Draw(ourShader);

// Y\_1

// render the loaded model

model\_14 = glm::translate(model\_14, model\_pos\_14); //定义模型每个循环出现的位置

model\_14 = glm::scale(model\_14, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_14);

ourModel\_14.Draw(ourShader);

// Z\_1

// render the loaded model

model\_15 = glm::translate(model\_15, model\_pos\_15); //定义模型每个循环出现的位置

model\_15 = glm::scale(model\_15, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_15);

ourModel\_15.Draw(ourShader);

//--------------------------------------------------------------------------------------------------------------------

// 按键input

// -----------------------------------------------------------------------

processInput(window);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// glfw: terminate, clearing all previously allocated GLFW resources.

// ------------------------------------------------------------------

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

// ---------------------------------------------------------------------------------------------------------

void do\_movement\_slider\_1()

{

//------------------------------------------

if (abs(model\_pos\_3.x - arrive\_1.x)<0.01f)

{

have\_arrive\_1\_x = true;

}

if (have\_arrive\_1\_x)

{

if (abs(model\_pos\_3.z - arrive\_1.z)< 0.01f)

{

have\_arrive\_1\_z = true;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z)

{

if (abs(model\_pos\_3.y - arrive\_1.y)<0.01f)

{

have\_arrive\_1\_y = true;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && have\_arrive\_1\_y)

{

have\_arrive\_1 = true;

}

//-------------------------------------------------

if (!have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走x方向

{

model\_pos\_1 += velocity\*slider\_1\_x\_direction;

model\_pos\_2 += velocity\*slider\_1\_x\_direction;

model\_pos\_3 += velocity\*slider\_1\_x\_direction;

}

if (have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完Z向

{

model\_pos\_2 += velocity\*slider\_1\_z\_direction;

model\_pos\_3 += velocity\*slider\_1\_z\_direction;

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完y向

{

model\_pos\_3 += velocity\*slider\_1\_y\_direction;

}

}

void do\_movement\_slider\_2()

{

if (abs(model\_pos\_7.x - arrive\_2.x)<0.01f)

{

have\_arrive\_2\_x = true;

}

if (have\_arrive\_2\_x)

{

if (abs(model\_pos\_7.z - arrive\_2.z)<0.01f)

{

have\_arrive\_2\_z = true;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z)

{

if (abs(model\_pos\_7.y - arrive\_2.y)<0.01f)

{

have\_arrive\_2\_y = true;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && have\_arrive\_2\_y)

{

have\_arrive\_2 = true;

}

//---------------------------------------------

if (!have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y) //走完x方向

{

model\_pos\_5 += velocity\*slider\_2\_x\_direction;

model\_pos\_6 += velocity\*slider\_2\_x\_direction;

model\_pos\_7 += velocity\*slider\_2\_x\_direction;

}

if (have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_6 += velocity\*slider\_2\_z\_direction;

model\_pos\_7 += velocity\*slider\_2\_z\_direction;

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_7 += velocity\*slider\_2\_y\_direction;

}

}

void do\_movement\_slider\_3()

{

if (abs(model\_pos\_11.x - arrive\_3.x)<0.01f)

{

have\_arrive\_3\_x = true;

}

if (have\_arrive\_3\_x)

{

if (abs(model\_pos\_11.z - arrive\_3.z)<0.01f)

{

have\_arrive\_3\_z = true;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z)

{

if (abs(model\_pos\_11.y - arrive\_3.y)<0.01f)

{

have\_arrive\_3\_y = true;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && have\_arrive\_3\_y)

{

have\_arrive\_3 = true;

}

//------------------------------------------------

if (!have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y) //走完x方向

{

model\_pos\_9 += velocity\*slider\_3\_x\_direction;

model\_pos\_10 += velocity\*slider\_3\_x\_direction;

model\_pos\_11 += velocity\*slider\_3\_x\_direction;

}

if (have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_10 += velocity\*slider\_3\_z\_direction;

model\_pos\_11 += velocity\*slider\_3\_z\_direction;

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_11 += velocity\*slider\_3\_y\_direction;

}

}

void do\_movement\_slider\_4()

{

if (abs(model\_pos\_15.x - arrive\_4.x)<0.01f)

{

have\_arrive\_4\_x = true;

}

if (have\_arrive\_4\_x)

{

if (abs(model\_pos\_15.z - arrive\_4.z)<0.01f)

{

have\_arrive\_4\_z = true;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z)

{

if (abs(model\_pos\_15.y - arrive\_4.y)<0.01f)

{

have\_arrive\_4\_y = true;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && have\_arrive\_4\_y)

{

have\_arrive\_4 = true;

}

//---------------------------------

if (!have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y) //走完x方向

{

model\_pos\_13 += velocity\*slider\_4\_x\_direction;

model\_pos\_14 += velocity\*slider\_4\_x\_direction;

model\_pos\_15 += velocity\*slider\_4\_x\_direction;

}

if (have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_14 += velocity\*slider\_4\_z\_direction;

model\_pos\_15 += velocity\*slider\_4\_z\_direction;

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_15 += velocity\*slider\_4\_y\_direction;

}

}

void jude\_direction()

{

//--------------------------------------------------------------

//1号定位器各轴移动方向

if (arrive\_1.x - model\_pos\_3.x > 0)

{

slider\_1\_x\_direction = 1.0f\*x\_direction;

}

else

{

slider\_1\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_1.z - model\_pos\_3.z > 0)

{

slider\_1\_z\_direction = 1.0f\*z\_direction;

}

else

{

slider\_1\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_1.y - model\_pos\_3.y > 0)

{

slider\_1\_y\_direction = 1.0f\*y\_direction;

}

else

{

slider\_1\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//2号定位器各轴移动方向

if (arrive\_2.x - model\_pos\_7.x > 0)

{

slider\_2\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_2\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_2.z - model\_pos\_7.z > 0) {

slider\_2\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_2\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_2.y - model\_pos\_7.y > 0) {

slider\_2\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_2\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//3号定位器各轴移动方向

if (arrive\_3.x - model\_pos\_11.x > 0) {

slider\_3\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_3\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_3.z - model\_pos\_11.z > 0) {

slider\_3\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_3\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_3.y - model\_pos\_11.y > 0) {

slider\_3\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_3\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//四号定位器各轴移动方向

if (arrive\_4.x - model\_pos\_15.x > 0) {

slider\_4\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_4\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_4.z - model\_pos\_15.z > 0) {

slider\_4\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_4\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_4.y - model\_pos\_15.y > 0) {

slider\_4\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_4\_y\_direction = -1.0f\*y\_direction;

}

}

void processInput(GLFWwindow\* window)

{

//float angle = 0.0f;

glm::vec3 front = glm::vec3(0.0f, 0.0f, -1.0f);

float cameraSpeed = 2.5f \* deltaTime;

//float cameraSpeed = 0.05f; // adjust accordingly

if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS)

cameraPos += cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS)

cameraPos -= cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS)

cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS)

cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

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)

{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

if (fov >= 1.0f && fov <= 45.0f)

fov -= yoffset;

if (fov <= 1.0f)

fov = 1.0f;

if (fov >= 45.0f)

fov = 45.0f;

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GL\_TRUE);

if (key >= 0 && key < 1024)

{

//设置按下/释放键为true或false

if (action == GLFW\_PRESS)

keys[key] = true;

else if (action == GLFW\_RELEASE)

keys[key] = false;

}

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos)

{

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos;

lastX = xpos;

lastY = ypos;

float sensitivity = 0.1;

xoffset \*= sensitivity;

yoffset \*= sensitivity;

yaw += xoffset;

pitch += yoffset;

if (pitch > 89.0f)

pitch = 89.0f;

if (pitch < -89.0f)

pitch = -89.0f;

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

//--------------------------------------------

//视角锁定 Version1.0

if (viewlocker == 1.0)

{

cameraFront = glm::normalize(front);

cameraFront\_now = cameraFront;

}

if (viewlocker == 0.0)

{

cameraFront = cameraFront\_now;//摄像机视线方向

//cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

}

}

void jude\_viewlock()

{

if (keys[GLFW\_KEY\_X])//X 视角解锁

{

viewlocker = 1.0;

}

if (keys[GLFW\_KEY\_C])//C 视角锁定

{

viewlocker = 0.0;

}

}

void judge\_move()

{

//前进后退计算更新位置向量

if (keys[GLFW\_KEY\_A])

{

model\_pos\_1 += velocity\*x\_direction;

model\_pos\_2 += velocity\*x\_direction;

model\_pos\_3 += velocity\*x\_direction;

model\_pos\_5 += velocity\*x\_direction;

model\_pos\_6 += velocity\*x\_direction;

model\_pos\_7 += velocity\*x\_direction;

model\_pos\_9 += velocity\*x\_direction;

model\_pos\_10 += velocity\*x\_direction;

model\_pos\_11 += velocity\*x\_direction;

model\_pos\_13 += velocity\*x\_direction;

model\_pos\_14 += velocity\*x\_direction;

model\_pos\_15 += velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_D])

{

model\_pos\_1 -= velocity\*x\_direction;

model\_pos\_2 -= velocity\*x\_direction;

model\_pos\_3 -= velocity\*x\_direction;

model\_pos\_5 -= velocity\*x\_direction;

model\_pos\_6 -= velocity\*x\_direction;

model\_pos\_7 -= velocity\*x\_direction;

model\_pos\_9 -= velocity\*x\_direction;

model\_pos\_10 -= velocity\*x\_direction;

model\_pos\_11 -= velocity\*x\_direction;

model\_pos\_13 -= velocity\*x\_direction;

model\_pos\_14 -= velocity\*x\_direction;

model\_pos\_15 -= velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_W])

{

model\_pos\_2 += velocity\*z\_direction;

model\_pos\_3 += velocity\*z\_direction;

model\_pos\_6 += velocity\*z\_direction;

model\_pos\_7 += velocity\*z\_direction;

model\_pos\_10 += velocity\*z\_direction;

model\_pos\_11 += velocity\*z\_direction;

model\_pos\_14 += velocity\*z\_direction;

model\_pos\_15 += velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_S])

{

model\_pos\_2 -= velocity\*z\_direction;

model\_pos\_3 -= velocity\*z\_direction;

model\_pos\_6 -= velocity\*z\_direction;

model\_pos\_7 -= velocity\*z\_direction;

model\_pos\_10 -= velocity\*z\_direction;

model\_pos\_11 -= velocity\*z\_direction;

model\_pos\_14 -= velocity\*z\_direction;

model\_pos\_15 -= velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_R])

{

model\_pos\_3 += velocity\*y\_direction;

model\_pos\_7 += velocity\*y\_direction;

model\_pos\_11 += velocity\*y\_direction;

model\_pos\_15 += velocity\*y\_direction;

}

if (keys[GLFW\_KEY\_F])

{

model\_pos\_3 -= velocity\*y\_direction;

model\_pos\_7 -= velocity\*y\_direction;

model\_pos\_11 -= velocity\*y\_direction;

model\_pos\_15 -= velocity\*y\_direction;

}

}

//7.23 定位器按照规划点 平滑移动 精度不够

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <gl/glut.h>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include <learnopengl/filesystem.h>

#include <learnopengl/shader\_m.h>

#include <learnopengl/camera.h>

#include <learnopengl/model.h>

#include <iostream>

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);

void judge\_move();

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode);

void jude\_viewlock();

//void do\_movement();

void jude\_direction();

void do\_movement\_slider\_1();

void do\_movement\_slider\_2();

void do\_movement\_slider\_3();

void do\_movement\_slider\_4();

//------------------------------------ camera自定义

//void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

float yaw = -90.0f;

float pitch = 0.0f;

glm::vec3 cameraPos = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront = glm::vec3(15.0f, 0.0f, 15.0f) - glm::vec3(30.0f, 30.0f, 30.0f);//摄像机视线方向

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

float viewlocker = 0.0f;

glm::vec3 cameraFront\_now = cameraFront;

//glm::mat4 view = view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

//------------------------------------

// settings 设置窗口宽高

const unsigned int SCR\_WIDTH = 1000;

const unsigned int SCR\_HEIGHT = 800;

// camera

Camera camera(glm::vec3(0.0, 0.0, 20.0));

float lastX = SCR\_WIDTH / 2.0;

float lastY = SCR\_HEIGHT / 2.0;

float fov = 45.0;

//float lastX = SCR\_WIDTH;

//float lastY = SCR\_HEIGHT;

bool firstMouse = true;

// timing

float deltaTime = 0.0;

float lastFrame = 0.0;

//1-4号定位器 初始位置向量

glm::vec3 model\_pos = glm::vec3(10.0, 0.0, 10.0); //定位器底座

glm::vec3 model\_pos\_1 = glm::vec3(10.0, 0.0, 10.0);//X\_1

glm::vec3 model\_pos\_2 = glm::vec3(10.0, 0.0, 10.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_3 = glm::vec3(10.0, 0.0, 10.0);//Z\_1

glm::vec3 model\_pos\_4 = glm::vec3(20.0, 0.0, 10.0); //定位器底座

glm::vec3 model\_pos\_5 = glm::vec3(20.0, 0.0, 10.0);//X\_1

glm::vec3 model\_pos\_6 = glm::vec3(20.0, 0.0, 10.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_7 = glm::vec3(20.0, 0.0, 10.0);//Z\_1

glm::vec3 model\_pos\_8 = glm::vec3(10.0, 0.0, 20.0); //定位器底座

glm::vec3 model\_pos\_9 = glm::vec3(10.0, 0.0, 20.0);//X\_1

glm::vec3 model\_pos\_10 = glm::vec3(10.0, 0.0, 20.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_11 = glm::vec3(10.0, 0.0, 20.0);//Z\_1

glm::vec3 model\_pos\_12 = glm::vec3(20.0, 0.0, 20.0); //定位器底座

glm::vec3 model\_pos\_13 = glm::vec3(20.0, 0.0, 20.0);//X\_1

glm::vec3 model\_pos\_14 = glm::vec3(20.0, 0.0, 20.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_15 = glm::vec3(20.0, 0.0, 20.0);//Z\_1

bool keys[1024];

float angle = 0.0;

GLfloat movespeed = 0.3; //设置的移动速度

GLfloat velocity;

float scale\_proportion = 0.1; //模型大小设置的缩放比例

float smooth\_steps = 1 / 600.0; //平滑的步数

float arrive\_interval = 0.01;

float magnify\_slidermove = 1.0e4; //定位器移动量的放大倍数

struct Point {

double x;

double y;

double z;

};

Point trans\_point[] = {

{ 10.00001, 0.00001, 10.00003 },

{ 20.0, 0.0, 10.0 },

{ 10.0, 0.0, 20.0 },

{ 20.0, 0.0, 20.0 },

{ 10.0, 0.0, 10.0 },

{ 20.0, 0.0, 10.0 },

{ 10.0, 0.0, 20.0 },

{ 20.0, 0.0, 20.0 }

};

int length;//length 为传过来的点的个数

/\*//1-4号定位器要到达的点

float arrive[][3] = {

{ 11.5f,2.0f,9.0f },

{ 18.5f, 1.0f, 11.0f },

{ 9.0f, 1.5f, 20.5f },

{ 21.0f, 1.0f, 20.0f },

{ 10.0f, 0.0f, 10.0f },

{ 20.0f, 0.0f, 10.0f },

{ 10.0f, 0.0f, 20.0f },

{ 20.0f, 0.0f, 20.0f }

};\*/

glm::vec3 arrive\_1 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_2 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_3 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_4 = glm::vec3(0, 0, 0);

glm::vec3 arrive\_1\_copy = glm::vec3(0, 0, 0);

glm::vec3 arrive\_2\_copy = glm::vec3(0, 0, 0);

glm::vec3 arrive\_3\_copy = glm::vec3(0, 0, 0);

glm::vec3 arrive\_4\_copy = glm::vec3(0, 0, 0);

//1号定位器各运动轴的运动方向

glm::vec3 slider\_1\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_1\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_1\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_2\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_2\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_2\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_3\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_3\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_3\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_4\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_4\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_4\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

//标准的三方向参考向量

glm::vec3 x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 y\_direction = glm::vec3(0.0, 1.0, 0.0);

//定位器各方向运动是否到达指定位置 判断标志

bool have\_arrive\_1\_x = false;

bool have\_arrive\_1\_z = false;

bool have\_arrive\_1\_y = false;

bool have\_arrive\_2\_x = false;

bool have\_arrive\_2\_z = false;

bool have\_arrive\_2\_y = false;

bool have\_arrive\_3\_x = false;

bool have\_arrive\_3\_z = false;

bool have\_arrive\_3\_y = false;

bool have\_arrive\_4\_x = false;

bool have\_arrive\_4\_z = false;

bool have\_arrive\_4\_y = false;

bool have\_arrive\_1 = false;

bool have\_arrive\_2 = false;

bool have\_arrive\_3 = false;

bool have\_arrive\_4 = false;

bool have\_one\_loop = false;

int raw\_slider\_1 = 0;

int raw\_slider\_2 = 1;

int raw\_slider\_3 = 2;

int raw\_slider\_4 = 3;

int steps\_slider\_1 = 0;

int steps\_slider\_2 = 0;

int steps\_slider\_3 = 0;

int steps\_slider\_4 = 0;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

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

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

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); // uncomment this statement to fix compilation on OS X

#endif

// glfw window creation

// --------------------

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);

//-------------------------------

glfwSetCursorPosCallback(window, mouse\_callback);

//----------------------------

glfwSetScrollCallback(window, scroll\_callback);

// tell GLFW to capture our mouse

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

//键盘捕捉

glfwSetKeyCallback(window, key\_callback);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

// build and compile shaders

// -------------------------

Shader ourShader("1.model\_loading.vs", "1.model\_loading.fs");

// load models

// -----------

Model ourModel("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_1("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_2("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_3("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_4("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_5("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_6("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_7("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_8("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_9("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_10("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_11("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_12("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_13("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_14("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_15("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

//Model ourModel\_4(FileSystem::getPath("resources/objects/dingweiqi/dizuo.obj"));

// draw in wireframe

//glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

//----------------------------------------------------------------

//第一次走位之前 装入目标位置点

arrive\_1 = glm::vec3(trans\_point[raw\_slider\_1].x, trans\_point[raw\_slider\_1].y, trans\_point[raw\_slider\_1].z);

arrive\_2 = glm::vec3(trans\_point[raw\_slider\_2].x, trans\_point[raw\_slider\_2].y, trans\_point[raw\_slider\_2].z);

arrive\_3 = glm::vec3(trans\_point[raw\_slider\_3].x, trans\_point[raw\_slider\_3].y, trans\_point[raw\_slider\_3].z);

arrive\_4 = glm::vec3(trans\_point[raw\_slider\_4].x, trans\_point[raw\_slider\_4].y, trans\_point[raw\_slider\_4].z);

arrive\_1\_copy = model\_pos\_3;

arrive\_2\_copy = model\_pos\_7;

arrive\_3\_copy = model\_pos\_11;

arrive\_4\_copy = model\_pos\_15;

length = sizeof(trans\_point) / (3\*4\*2);//计算传过来的点的个数

//Point结构体中变量类型 float 3\*4 double 3\*4\*2 //改变结构体变量类型记得回来改这里

//--------------------------------------------------------------------

//\*\*\*\*\*\*计算各个定位器三个轴的移动方向\*\*\*\*

jude\_direction();//计算第一组位置轴的移动方向

// render loop //循环开始

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = glfwGetTime();

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

velocity = movespeed \* deltaTime;

// render

// ------

glClearColor(0.3f, 0.5f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// don't forget to enable shader before setting uniforms

ourShader.use();

//

//给定光照

ourShader.setVec3("viewPos", camera.Position);

ourShader.setFloat("shininess", 64.0f);

ourShader.setVec3("light.position", glm::vec3(0.0f, 5.0f, 0.0f));

ourShader.setVec3("light.ambient", 0.5f, 0.5f, 0.5f);

ourShader.setVec3("light.diffuse", 0.8f, 0.8f, 0.8f);

ourShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);

ourShader.setFloat("light.constant", 1.0f);

ourShader.setFloat("light.linear", 0.045f);

ourShader.setFloat("light.quadratic", 0.0075f);

GLfloat sun\_light\_position[] = { 10.0f, 10.0f, 10.0f, 1.0f }; //光源的位置在世界坐标系圆心，齐次坐标形式

GLfloat sun\_light\_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的环境光，为0

GLfloat sun\_light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的漫反射光，全白光

GLfloat sun\_light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式下的镜面光 ，全白光

glLightfv(GL\_LIGHT0, GL\_POSITION, sun\_light\_position);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, sun\_light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, sun\_light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, sun\_light\_specular);

//开启灯光

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

//世界空间变换矩阵初始化

glm::mat4 model = glm::mat4(1.0);

glm::mat4 model\_1 = glm::mat4(1.0);

glm::mat4 model\_2 = glm::mat4(1.0);

glm::mat4 model\_3 = glm::mat4(1.0);

glm::mat4 model\_4 = glm::mat4(1.0);

glm::mat4 model\_5 = glm::mat4(1.0);

glm::mat4 model\_6 = glm::mat4(1.0);

glm::mat4 model\_7 = glm::mat4(1.0);

glm::mat4 model\_8 = glm::mat4(1.0);

glm::mat4 model\_9 = glm::mat4(1.0);

glm::mat4 model\_10 = glm::mat4(1.0);

glm::mat4 model\_11 = glm::mat4(1.0);

glm::mat4 model\_12 = glm::mat4(1.0);

glm::mat4 model\_13 = glm::mat4(1.0);

glm::mat4 model\_14 = glm::mat4(1.0);

glm::mat4 model\_15 = glm::mat4(1.0);

//视角锁定

jude\_viewlock();

// view/projection transformations 观察空间 +视角锁定

//------------------------------------------------------

//视角锁定 Version2.0

/\*if (viewlocker == 1.0)

{

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

}

if (viewlocker==0.0)

{

glm::vec3 cameraPos\_lock = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront\_lock = glm::vec3(-1.0f, -1.0f, -1.0f);//摄像机视线方向

glm::vec3 cameraUp\_lock = glm::vec3(0.0f, 1.0f, 0.0f);

glm::mat4 projection\_lock = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view\_lock = glm::lookAt(cameraPos\_lock, cameraPos\_lock + cameraFront\_lock, cameraUp\_lock);

ourShader.setMat4("projection", projection\_lock);

ourShader.setMat4("view", view\_lock);//view 观察矩阵

}\*/

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

// render the loaded model

//------------------------------------------------

//小车按键控制随意移动

judge\_move(); //执行前进，计算更新位置向量model\_pos\_

//judge\_turn(); //执行旋转，计算更新位置向量model\_pos\_

//---------------------------------------------------------------------

if (have\_one\_loop) //have\_one\_loop==true 则更新下一组 要达到的点

{

if (raw\_slider\_4 < length - 1)//如果不是最后一组位置点-->接着更新位置点组

{

raw\_slider\_1 += 4;

raw\_slider\_2 += 4;

raw\_slider\_3 += 4;

raw\_slider\_4 += 4;

arrive\_1\_copy = arrive\_1;

arrive\_2\_copy = arrive\_2;

arrive\_3\_copy = arrive\_3;

arrive\_4\_copy = arrive\_4;

arrive\_1 = glm::vec3(trans\_point[raw\_slider\_1].x, trans\_point[raw\_slider\_1].y, trans\_point[raw\_slider\_1].z);

arrive\_2 = glm::vec3(trans\_point[raw\_slider\_2].x, trans\_point[raw\_slider\_2].y, trans\_point[raw\_slider\_2].z);

arrive\_3 = glm::vec3(trans\_point[raw\_slider\_3].x, trans\_point[raw\_slider\_3].y, trans\_point[raw\_slider\_3].z);

arrive\_4 = glm::vec3(trans\_point[raw\_slider\_4].x, trans\_point[raw\_slider\_4].y, trans\_point[raw\_slider\_4].z);

jude\_direction();//重新确定方向

have\_arrive\_1 = false; have\_arrive\_1\_x = false; have\_arrive\_1\_z = false; have\_arrive\_1\_y = false;

have\_arrive\_2 = false; have\_arrive\_2\_x = false; have\_arrive\_2\_z = false; have\_arrive\_2\_y = false;

have\_arrive\_3 = false; have\_arrive\_3\_x = false; have\_arrive\_3\_z = false; have\_arrive\_3\_y = false;

have\_arrive\_4 = false; have\_arrive\_4\_x = false; have\_arrive\_4\_z = false; have\_arrive\_4\_y = false;

have\_one\_loop = false; //重新置位到达状态

}

else

{

have\_arrive\_1 = true;

have\_arrive\_2 = true;

have\_arrive\_3 = true;

have\_arrive\_4 = true;

have\_one\_loop = true; //重新置位到达状态

}

}

//定位器运送物体到指定点

if (!have\_one\_loop)

{

do\_movement\_slider\_1();

do\_movement\_slider\_2();

do\_movement\_slider\_3();

do\_movement\_slider\_4();

if (have\_arrive\_1 && have\_arrive\_2 && have\_arrive\_3 && have\_arrive\_4)

{

have\_one\_loop = true;

}

}

//---------------------------------------------

//定位器底座

model = glm::translate(model, model\_pos); //定义模型每个循环出现的位置

model = glm::scale(model, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model);

ourModel.Draw(ourShader);

//X\_1

// render the loaded model

model\_1 = glm::translate(model\_1, model\_pos\_1); //定义模型每个循环出现的位置

model\_1 = glm::scale(model\_1, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_1);

ourModel\_1.Draw(ourShader);

// Y\_1

// render the loaded model

model\_2 = glm::translate(model\_2, model\_pos\_2); //定义模型每个循环出现的位置

model\_2 = glm::scale(model\_2, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_2);

ourModel\_2.Draw(ourShader);

// Z\_1

// render the loaded model

model\_3 = glm::translate(model\_3, model\_pos\_3); //定义模型每个循环出现的位置

model\_3 = glm::scale(model\_3, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_3);

ourModel\_3.Draw(ourShader);

////\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

////定位器底座

//model\_4 = glm::translate(model\_4, model\_pos\_4); //定义模型每个循环出现的位置

//model\_4 = glm::scale(model\_4, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_4);

//ourModel\_4.Draw(ourShader);

////X\_1

//// render the loaded model

//model\_5 = glm::translate(model\_5, model\_pos\_5); //定义模型每个循环出现的位置

//model\_5 = glm::scale(model\_5, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_5);

//ourModel\_5.Draw(ourShader);

//// Y\_1

//// render the loaded model

//model\_6 = glm::translate(model\_6, model\_pos\_6); //定义模型每个循环出现的位置

//model\_6 = glm::scale(model\_6, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_6);

//ourModel\_6.Draw(ourShader);

//// Z\_1

//// render the loaded model

//model\_7 = glm::translate(model\_7, model\_pos\_7); //定义模型每个循环出现的位置

//model\_7 = glm::scale(model\_7, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_7);

//ourModel\_7.Draw(ourShader);

////\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

////定位器底座

//model\_8 = glm::translate(model\_8, model\_pos\_8); //定义模型每个循环出现的位置

//model\_8 = glm::scale(model\_8, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_8);

//ourModel\_8.Draw(ourShader);

////X\_1

//// render the loaded model

//model\_9 = glm::translate(model\_9, model\_pos\_9); //定义模型每个循环出现的位置

//model\_9 = glm::scale(model\_9, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_9);

//ourModel\_9.Draw(ourShader);

//// Y\_1

//// render the loaded model

//model\_10 = glm::translate(model\_10, model\_pos\_10); //定义模型每个循环出现的位置

//model\_10 = glm::scale(model\_10, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_10);

//ourModel\_10.Draw(ourShader);

//// Z\_1

//// render the loaded model

//model\_11 = glm::translate(model\_11, model\_pos\_11); //定义模型每个循环出现的位置

//model\_11 = glm::scale(model\_11, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_11);

//ourModel\_11.Draw(ourShader);

////\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

////定位器底座

//model\_12 = glm::translate(model\_12, model\_pos\_12); //定义模型每个循环出现的位置

//model\_12 = glm::scale(model\_12, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_12);

//ourModel\_12.Draw(ourShader);

////X\_1

//// render the loaded model

//model\_13 = glm::translate(model\_13, model\_pos\_13); //定义模型每个循环出现的位置

//model\_13 = glm::scale(model\_13, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_13);

//ourModel\_13.Draw(ourShader);

//// Y\_1

//// render the loaded model

//model\_14 = glm::translate(model\_14, model\_pos\_14); //定义模型每个循环出现的位置

//model\_14 = glm::scale(model\_14, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_14);

//ourModel\_14.Draw(ourShader);

//// Z\_1

//// render the loaded model

//model\_15 = glm::translate(model\_15, model\_pos\_15); //定义模型每个循环出现的位置

//model\_15 = glm::scale(model\_15, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

//ourShader.setMat4("model", model\_15);

//ourModel\_15.Draw(ourShader);

////--------------------------------------------------------------------------------------------------------------------

// 按键input

// -----------------------------------------------------------------------

processInput(window);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// glfw: terminate, clearing all previously allocated GLFW resources.

// ------------------------------------------------------------------

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

// ---------------------------------------------------------------------------------------------------------

void do\_movement\_slider\_1()

{

//-------------------------------------------------

if (!have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走x方向

{

model\_pos\_1 += smooth\_steps\*slider\_1\_x\_direction;

model\_pos\_2 += smooth\_steps\*slider\_1\_x\_direction;

model\_pos\_3 += smooth\_steps\*slider\_1\_x\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_x = true;

steps\_slider\_1 = 0;

model\_pos\_1.x += slider\_1\_x\_direction.x;

model\_pos\_2.x += slider\_1\_x\_direction.x;

model\_pos\_3.x += slider\_1\_x\_direction.x;

}

}

if (have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完Z向

{

model\_pos\_2 += smooth\_steps\*slider\_1\_z\_direction;

model\_pos\_3 += smooth\_steps\*slider\_1\_z\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_z = true;

model\_pos\_2.z += slider\_1\_z\_direction.z;

model\_pos\_3.z += slider\_1\_z\_direction.z;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完y向

{

model\_pos\_3 += smooth\_steps\*slider\_1\_y\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_y = true;

model\_pos\_3.y += slider\_1\_y\_direction.y;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && have\_arrive\_1\_y)

{

have\_arrive\_1 = true;

steps\_slider\_1 = 0;

}

}

void do\_movement\_slider\_2()

{

//---------------------------------------------

if (!have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y) //走完x方向

{

model\_pos\_5 += smooth\_steps\*slider\_2\_x\_direction;

model\_pos\_6 += smooth\_steps\*slider\_2\_x\_direction;

model\_pos\_7 += smooth\_steps\*slider\_2\_x\_direction;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_x = true;

model\_pos\_5.x = arrive\_2.x;

model\_pos\_6.x = arrive\_2.x;

model\_pos\_7.x = arrive\_2.x;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_6 += smooth\_steps\*slider\_2\_z\_direction;

model\_pos\_7 += smooth\_steps\*slider\_2\_z\_direction;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_z = true;

model\_pos\_6.z = arrive\_2.z;

model\_pos\_7.z = arrive\_2.z;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_7 += smooth\_steps\*slider\_2\_y\_direction;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_y = true;

model\_pos\_7.y = arrive\_2.y;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && have\_arrive\_2\_y)

{

have\_arrive\_2 = true;

}

}

void do\_movement\_slider\_3()

{

//------------------------------------------------

if (!have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y) //走完x方向

{

model\_pos\_9 += smooth\_steps\*slider\_3\_x\_direction;

model\_pos\_10 += smooth\_steps\*slider\_3\_x\_direction;

model\_pos\_11 += smooth\_steps\*slider\_3\_x\_direction;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_x = true;

model\_pos\_9.x = arrive\_3.x;

model\_pos\_10.x = arrive\_3.x;

model\_pos\_11.x = arrive\_3.x;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_10 += smooth\_steps\*slider\_3\_z\_direction;

model\_pos\_11 += smooth\_steps\*slider\_3\_z\_direction;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_z = true;

model\_pos\_10.z = arrive\_3.z;

model\_pos\_11.z = arrive\_3.z;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_11 += smooth\_steps\*slider\_3\_y\_direction;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_y = true;

model\_pos\_11.y = arrive\_3.y;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && have\_arrive\_3\_y)

{

have\_arrive\_3 = true;

}

}

void do\_movement\_slider\_4()

{

//---------------------------------

if (!have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y) //走完x方向

{

model\_pos\_13 += smooth\_steps\*slider\_4\_x\_direction;

model\_pos\_14 += smooth\_steps\*slider\_4\_x\_direction;

model\_pos\_15 += smooth\_steps\*slider\_4\_x\_direction;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_x = true;

model\_pos\_13.x = arrive\_4.x;

model\_pos\_14.x = arrive\_4.x;

model\_pos\_15.x = arrive\_4.x;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_14 += smooth\_steps\*slider\_4\_z\_direction;

model\_pos\_15 += smooth\_steps\*slider\_4\_z\_direction;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_z = true;

model\_pos\_14.z = arrive\_4.z;

model\_pos\_15.z = arrive\_4.z;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_15 += smooth\_steps\*slider\_4\_y\_direction;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_y = true;

model\_pos\_15.y = arrive\_4.y;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && have\_arrive\_4\_y)

{

have\_arrive\_4 = true;

}

}

void jude\_direction()

{

//--------------------------------------------------------------

//1号定位器各轴移动方向

slider\_1\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_1.x - arrive\_1\_copy.x, 0, 0);

slider\_1\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_1.z - arrive\_1\_copy.z);

slider\_1\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_1.y - arrive\_1\_copy.y, 0);

//--------------------------------------------------------------

//2号定位器各轴移动方向

slider\_2\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_2.x - arrive\_2\_copy.x, 0, 0);

slider\_2\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_2.z - arrive\_2\_copy.z);

slider\_2\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_2.y - arrive\_2\_copy.y, 0);

//--------------------------------------------------------------

//3号定位器各轴移动方向

slider\_3\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_3.x - arrive\_3\_copy.x, 0, 0);

slider\_3\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_3.z - arrive\_3\_copy.z);

slider\_3\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_3.y - arrive\_3\_copy.y, 0);

//--------------------------------------------------------------

//4号定位器各轴移动方向

slider\_4\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_4.x - arrive\_4\_copy.x, 0, 0);

slider\_4\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_4.z - arrive\_4\_copy.z);

slider\_4\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_4.y - arrive\_4\_copy.y, 0);

}

/\*

void jude\_direction()

{

//--------------------------------------------------------------

//1号定位器各轴移动方向

if (arrive\_1.x - model\_pos\_3.x > 0)

{

slider\_1\_x\_direction = 1.0f\*x\_direction;

}

else

{

slider\_1\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_1.z - model\_pos\_3.z > 0)

{

slider\_1\_z\_direction = 1.0f\*z\_direction;

}

else

{

slider\_1\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_1.y - model\_pos\_3.y > 0)

{

slider\_1\_y\_direction = 1.0f\*y\_direction;

}

else

{

slider\_1\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//2号定位器各轴移动方向

if (arrive\_2.x - model\_pos\_7.x > 0)

{

slider\_2\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_2\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_2.z - model\_pos\_7.z > 0) {

slider\_2\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_2\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_2.y - model\_pos\_7.y > 0) {

slider\_2\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_2\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//3号定位器各轴移动方向

if (arrive\_3.x - model\_pos\_11.x > 0) {

slider\_3\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_3\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_3.z - model\_pos\_11.z > 0) {

slider\_3\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_3\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_3.y - model\_pos\_11.y > 0) {

slider\_3\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_3\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//四号定位器各轴移动方向

if (arrive\_4.x - model\_pos\_15.x > 0) {

slider\_4\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_4\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_4.z - model\_pos\_15.z > 0) {

slider\_4\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_4\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_4.y - model\_pos\_15.y > 0) {

slider\_4\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_4\_y\_direction = -1.0f\*y\_direction;

}

}\*/

void processInput(GLFWwindow\* window)

{

//float angle = 0.0f;

glm::vec3 front = glm::vec3(0.0f, 0.0f, -1.0f);

float cameraSpeed = 2.5f \* deltaTime;

//float cameraSpeed = 0.05f; // adjust accordingly

if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS)

cameraPos += cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS)

cameraPos -= cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS)

cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS)

cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

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)

{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

if (fov >= 1.0f && fov <= 45.0f)

fov -= yoffset;

if (fov <= 1.0f)

fov = 1.0f;

if (fov >= 45.0f)

fov = 45.0f;

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GL\_TRUE);

if (key >= 0 && key < 1024)

{

//设置按下/释放键为true或false

if (action == GLFW\_PRESS)

keys[key] = true;

else if (action == GLFW\_RELEASE)

keys[key] = false;

}

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos)

{

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos;

lastX = xpos;

lastY = ypos;

float sensitivity = 0.1;

xoffset \*= sensitivity;

yoffset \*= sensitivity;

yaw += xoffset;

pitch += yoffset;

if (pitch > 89.0f)

pitch = 89.0f;

if (pitch < -89.0f)

pitch = -89.0f;

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

//--------------------------------------------

//视角锁定 Version1.0

if (viewlocker == 1.0)

{

cameraFront = glm::normalize(front);

cameraFront\_now = cameraFront;

}

if (viewlocker == 0.0)

{

cameraFront = cameraFront\_now;//摄像机视线方向

//cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

}

}

void jude\_viewlock()

{

if (keys[GLFW\_KEY\_X])//X 视角解锁

{

viewlocker = 1.0;

}

if (keys[GLFW\_KEY\_C])//C 视角锁定

{

viewlocker = 0.0;

}

}

void judge\_move()

{

//前进后退计算更新位置向量

if (keys[GLFW\_KEY\_A])

{

model\_pos\_1 += velocity\*x\_direction;

model\_pos\_2 += velocity\*x\_direction;

model\_pos\_3 += velocity\*x\_direction;

model\_pos\_5 += velocity\*x\_direction;

model\_pos\_6 += velocity\*x\_direction;

model\_pos\_7 += velocity\*x\_direction;

model\_pos\_9 += velocity\*x\_direction;

model\_pos\_10 += velocity\*x\_direction;

model\_pos\_11 += velocity\*x\_direction;

model\_pos\_13 += velocity\*x\_direction;

model\_pos\_14 += velocity\*x\_direction;

model\_pos\_15 += velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_D])

{

model\_pos\_1 -= velocity\*x\_direction;

model\_pos\_2 -= velocity\*x\_direction;

model\_pos\_3 -= velocity\*x\_direction;

model\_pos\_5 -= velocity\*x\_direction;

model\_pos\_6 -= velocity\*x\_direction;

model\_pos\_7 -= velocity\*x\_direction;

model\_pos\_9 -= velocity\*x\_direction;

model\_pos\_10 -= velocity\*x\_direction;

model\_pos\_11 -= velocity\*x\_direction;

model\_pos\_13 -= velocity\*x\_direction;

model\_pos\_14 -= velocity\*x\_direction;

model\_pos\_15 -= velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_W])

{

model\_pos\_2 += velocity\*z\_direction;

model\_pos\_3 += velocity\*z\_direction;

model\_pos\_6 += velocity\*z\_direction;

model\_pos\_7 += velocity\*z\_direction;

model\_pos\_10 += velocity\*z\_direction;

model\_pos\_11 += velocity\*z\_direction;

model\_pos\_14 += velocity\*z\_direction;

model\_pos\_15 += velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_S])

{

model\_pos\_2 -= velocity\*z\_direction;

model\_pos\_3 -= velocity\*z\_direction;

model\_pos\_6 -= velocity\*z\_direction;

model\_pos\_7 -= velocity\*z\_direction;

model\_pos\_10 -= velocity\*z\_direction;

model\_pos\_11 -= velocity\*z\_direction;

model\_pos\_14 -= velocity\*z\_direction;

model\_pos\_15 -= velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_R])

{

model\_pos\_3 += velocity\*y\_direction;

model\_pos\_7 += velocity\*y\_direction;

model\_pos\_11 += velocity\*y\_direction;

model\_pos\_15 += velocity\*y\_direction;

}

if (keys[GLFW\_KEY\_F])

{

model\_pos\_3 -= velocity\*y\_direction;

model\_pos\_7 -= velocity\*y\_direction;

model\_pos\_11 -= velocity\*y\_direction;

model\_pos\_15 -= velocity\*y\_direction;

}

}

//7.27 定位器按照规划点 move=放大倍数\*Δx 精度提高 平滑移动

//放大倍数\*Δx 走位量是精确放大的 精度满足

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <gl/glut.h>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include <learnopengl/filesystem.h>

#include <learnopengl/shader\_m.h>

#include <learnopengl/camera.h>

#include <learnopengl/model.h>

#include <iostream>

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);

void judge\_move();

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode);

void jude\_viewlock();

//void do\_movement();

//void jude\_direction();

void jude\_direction\_change();

void do\_movement\_slider\_1();

void do\_movement\_slider\_2();

void do\_movement\_slider\_3();

void do\_movement\_slider\_4();

//------------------------------------ camera自定义

//void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

float yaw = -90.0f;

float pitch = 0.0f;

glm::vec3 cameraPos = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront = glm::vec3(15.0f, 0.0f, 15.0f) - glm::vec3(30.0f, 30.0f, 30.0f);//摄像机视线方向

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

float viewlocker = 0.0f;

glm::vec3 cameraFront\_now = cameraFront;

//glm::mat4 view = view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

//------------------------------------

// settings 设置窗口宽高

const unsigned int SCR\_WIDTH = 1000;

const unsigned int SCR\_HEIGHT = 800;

// camera

Camera camera(glm::vec3(0.0, 0.0, 20.0));

float lastX = SCR\_WIDTH / 2.0;

float lastY = SCR\_HEIGHT / 2.0;

float fov = 45.0;

//float lastX = SCR\_WIDTH;

//float lastY = SCR\_HEIGHT;

bool firstMouse = true;

// timing

float deltaTime = 0.0;

float lastFrame = 0.0;

bool keys[1024];

float angle = 0.0;

GLfloat movespeed = 0.3; //设置的移动速度

GLfloat velocity;

float scale\_proportion = 0.1; //模型大小设置的缩放比例

float smooth\_steps = 1 / 600.0; //平滑的步数

float arrive\_interval = 0.01;

double magnify\_slidermove = 0.25e8; //定位器移动量的放大倍数

struct Point {

double x;

double y;

double z;

};

Point trans\_point[] = {

{ 11702.15617438560, 581.03223938796, 3261.15608958024},

{ 11643.29659054430, 1014.21458105330, 1978.42996869193},

{ 13168.57988926790, 1014.79342543805, 1911.76022939268},

{ 13216.88453858120, 575.78884948872, 3194.10601819121},

{ 11702.15617438660 ,581.03223938907, 3261.15608957777 },

{ 11643.29659054530 ,1014.21458105247,1978.42996868880},

{ 13168.57988926890 ,1014.79342543038,1911.76022938731},

{ 13216.88453858210 ,575.78884948302, 3194.10601818653},

{ 11702.15617439360 ,581.03223939682, 3261.15608956045},

{ 11643.29659055260 ,1014.21458104671,1978.42996866690 },

{ 13168.57988927550 ,1014.79342537664,1911.76022934978 },

{ 13216.88453858820 ,575.78884944312, 3194.10601815375 }

};

double arrive\_group\_copy[4][3];

double arrive\_group[4][3];

int length;//length 为传过来的点的个数

//1-4号定位器 初始位置向量 --->注意现实坐标系和 仿真空间坐标系之间的 \*\*映射\*\*

glm::vec3 model\_pos = glm::vec3(10.0, 0.0, 10.0); //定位器底座

glm::vec3 model\_pos\_1 = glm::vec3(10.0, 0.0, 10.0);//X\_1

glm::vec3 model\_pos\_2 = glm::vec3(10.0, 0.0, 10.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_3 = glm::vec3(10.0, 0.0, 10.0);//Z\_1

glm::vec3 model\_pos\_4 = glm::vec3(20.0, 0.0, 10.0); //定位器底座

glm::vec3 model\_pos\_5 = glm::vec3(20.0, 0.0, 10.0);//X\_1

glm::vec3 model\_pos\_6 = glm::vec3(20.0, 0.0, 10.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_7 = glm::vec3(20.0, 0.0, 10.0);//Z\_1

glm::vec3 model\_pos\_8 = glm::vec3(10.0, 0.0, 20.0); //定位器底座

glm::vec3 model\_pos\_9 = glm::vec3(10.0, 0.0, 20.0);//X\_1

glm::vec3 model\_pos\_10 = glm::vec3(10.0, 0.0, 20.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_11 = glm::vec3(10.0, 0.0, 20.0);//Z\_1

glm::vec3 model\_pos\_12 = glm::vec3(20.0, 0.0, 20.0); //定位器底座

glm::vec3 model\_pos\_13 = glm::vec3(20.0, 0.0, 20.0);//X\_1

glm::vec3 model\_pos\_14 = glm::vec3(20.0, 0.0, 20.0);//Y\_1 -->正常世界的空间坐标系的三个 方向

glm::vec3 model\_pos\_15 = glm::vec3(20.0, 0.0, 20.0);//Z\_1

//1号定位器各运动轴的运动方向

glm::vec3 slider\_1\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_1\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_1\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_2\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_2\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_2\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_3\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_3\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_3\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

glm::vec3 slider\_4\_x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 slider\_4\_z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 slider\_4\_y\_direction = glm::vec3(0.0, 1.0, 0.0);

double slider\_1\_x\_dir\_value\_change=0;

double slider\_1\_z\_dir\_value\_change = 0;

double slider\_1\_y\_dir\_value\_change = 0;

double slider\_2\_x\_dir\_value\_change = 0;

double slider\_2\_z\_dir\_value\_change = 0;

double slider\_2\_y\_dir\_value\_change = 0;

double slider\_3\_x\_dir\_value\_change = 0;

double slider\_3\_z\_dir\_value\_change = 0;

double slider\_3\_y\_dir\_value\_change = 0;

double slider\_4\_x\_dir\_value\_change = 0;

double slider\_4\_z\_dir\_value\_change = 0;

double slider\_4\_y\_dir\_value\_change = 0;

//标准的三方向参考向量

glm::vec3 x\_direction = glm::vec3(1.0, 0.0, 0.0);

glm::vec3 z\_direction = glm::vec3(0.0, 0.0, 1.0);

glm::vec3 y\_direction = glm::vec3(0.0, 1.0, 0.0);

//定位器各方向运动是否到达指定位置 判断标志

bool have\_arrive\_1\_x = false;

bool have\_arrive\_1\_z = false;

bool have\_arrive\_1\_y = false;

bool have\_arrive\_2\_x = false;

bool have\_arrive\_2\_z = false;

bool have\_arrive\_2\_y = false;

bool have\_arrive\_3\_x = false;

bool have\_arrive\_3\_z = false;

bool have\_arrive\_3\_y = false;

bool have\_arrive\_4\_x = false;

bool have\_arrive\_4\_z = false;

bool have\_arrive\_4\_y = false;

bool have\_arrive\_1 = false;

bool have\_arrive\_2 = false;

bool have\_arrive\_3 = false;

bool have\_arrive\_4 = false;

bool have\_one\_loop = false;

int raw\_slider\_1 = 0;

int raw\_slider\_2 = 1;

int raw\_slider\_3 = 2;

int raw\_slider\_4 = 3;

int steps\_slider\_1 = 0;

int steps\_slider\_2 = 0;

int steps\_slider\_3 = 0;

int steps\_slider\_4 = 0;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

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

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

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); // uncomment this statement to fix compilation on OS X

#endif

// glfw window creation

// --------------------

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);

//-------------------------------

glfwSetCursorPosCallback(window, mouse\_callback);

//----------------------------

glfwSetScrollCallback(window, scroll\_callback);

// tell GLFW to capture our mouse

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

//键盘捕捉

glfwSetKeyCallback(window, key\_callback);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

// build and compile shaders

// -------------------------

Shader ourShader("1.model\_loading.vs", "1.model\_loading.fs");

// load models

// -----------

Model ourModel("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_1("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_2("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_3("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_4("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_5("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_6("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_7("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_8("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_9("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_10("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_11("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

Model ourModel\_12("D:/5-VScode/test\_16/resources/objects/dingweiqi/dizuo.obj");

Model ourModel\_13("D:/5-VScode/test\_16/resources/objects/dingweiqi/x\_1.obj");

Model ourModel\_14("D:/5-VScode/test\_16/resources/objects/dingweiqi/y\_1.obj");

Model ourModel\_15("D:/5-VScode/test\_16/resources/objects/dingweiqi/z\_1.obj");

//Model ourModel\_4(FileSystem::getPath("resources/objects/dingweiqi/dizuo.obj"));

// draw in wireframe

//glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

//----------------------------------------------------------------

//第一次走位之前 （进入循环之前）装入目标位置点

//arrive\_1 = glm::vec3(trans\_point[raw\_slider\_1].x, trans\_point[raw\_slider\_1].y, trans\_point[raw\_slider\_1].z);

//arrive\_2 = glm::vec3(trans\_point[raw\_slider\_2].x, trans\_point[raw\_slider\_2].y, trans\_point[raw\_slider\_2].z);

//arrive\_3 = glm::vec3(trans\_point[raw\_slider\_3].x, trans\_point[raw\_slider\_3].y, trans\_point[raw\_slider\_3].z);

//arrive\_4 = glm::vec3(trans\_point[raw\_slider\_4].x, trans\_point[raw\_slider\_4].y, trans\_point[raw\_slider\_4].z);

//arrive\_1\_copy = model\_pos\_3; //arrivecopy 装入起始位置点

//arrive\_2\_copy = model\_pos\_7;

//arrive\_3\_copy = model\_pos\_11;

//arrive\_4\_copy = model\_pos\_15;

length = sizeof(trans\_point) / (3 \* 4 \* 2);//计算传过来的点的个数

//Point结构体中变量类型 float 3\*4 double 3\*4\*2 //改变结构体变量类型记得回来改这里

//-----------------------------------------

//注意现实坐标系和 仿真空间坐标系之间的 \*\*映射\*\*

arrive\_group\_copy[0][0] = trans\_point[raw\_slider\_1].x; arrive\_group\_copy[0][1] = trans\_point[raw\_slider\_1].y;

arrive\_group\_copy[0][2] = trans\_point[raw\_slider\_1].z;

arrive\_group\_copy[1][0] = trans\_point[raw\_slider\_2].x; arrive\_group\_copy[1][1] = trans\_point[raw\_slider\_2].y;

arrive\_group\_copy[1][2] = trans\_point[raw\_slider\_2].z;

arrive\_group\_copy[2][0] = trans\_point[raw\_slider\_3].x; arrive\_group\_copy[2][1] = trans\_point[raw\_slider\_3].y;

arrive\_group\_copy[2][2] = trans\_point[raw\_slider\_3].z;

arrive\_group\_copy[3][0] = trans\_point[raw\_slider\_4].x; arrive\_group\_copy[3][1] = trans\_point[raw\_slider\_4].y;

arrive\_group\_copy[3][2] = trans\_point[raw\_slider\_4].z;

raw\_slider\_1 += 4; raw\_slider\_2 += 4; raw\_slider\_3 += 4; raw\_slider\_4 += 4;

//[][0] [][1] 水平方向 [][2]竖直方向 arrive\_group,arrive\_group\_copy

arrive\_group[0][0] = trans\_point[raw\_slider\_1].x; arrive\_group[0][1] = trans\_point[raw\_slider\_1].y;

arrive\_group[0][2] = trans\_point[raw\_slider\_1].z; //1号定位器 目标点

arrive\_group[1][0] = trans\_point[raw\_slider\_2].x; arrive\_group[1][1] = trans\_point[raw\_slider\_2].y;

arrive\_group[1][2] = trans\_point[raw\_slider\_2].z; //2号定位器 目标点

arrive\_group[2][0] = trans\_point[raw\_slider\_3].x; arrive\_group[2][1] = trans\_point[raw\_slider\_3].y;

arrive\_group[2][2] = trans\_point[raw\_slider\_3].z; //3号定位器 目标点

arrive\_group[3][0] = trans\_point[raw\_slider\_4].x; arrive\_group[3][1] = trans\_point[raw\_slider\_4].y;

arrive\_group[3][2] = trans\_point[raw\_slider\_4].z; //4号定位器 目标点

//--------------------------------------------------------------------

//\*\*\*\*\*\*计算各个定位器三个轴的移动方向\*\*\*\*

//jude\_direction();//计算第一组位置轴的移动方向

jude\_direction\_change();

// render loop //循环开始

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = glfwGetTime();

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

velocity = movespeed \* deltaTime;

// render

// ------

glClearColor(0.3f, 0.5f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// don't forget to enable shader before setting uniforms

ourShader.use();

//

//给定光照

ourShader.setVec3("viewPos", camera.Position);

ourShader.setFloat("shininess", 64.0f);

ourShader.setVec3("light.position", glm::vec3(0.0f, 5.0f, 0.0f));

ourShader.setVec3("light.ambient", 0.5f, 0.5f, 0.5f);

ourShader.setVec3("light.diffuse", 0.8f, 0.8f, 0.8f);

ourShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);

ourShader.setFloat("light.constant", 1.0f);

ourShader.setFloat("light.linear", 0.045f);

ourShader.setFloat("light.quadratic", 0.0075f);

GLfloat sun\_light\_position[] = { 10.0f, 10.0f, 10.0f, 1.0f }; //光源的位置在世界坐标系圆心，齐次坐标形式

GLfloat sun\_light\_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的环境光，为0

GLfloat sun\_light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式的漫反射光，全白光

GLfloat sun\_light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //RGBA模式下的镜面光 ，全白光

glLightfv(GL\_LIGHT0, GL\_POSITION, sun\_light\_position);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, sun\_light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, sun\_light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, sun\_light\_specular);

//开启灯光

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

//世界空间变换矩阵初始化

glm::mat4 model = glm::mat4(1.0);

glm::mat4 model\_1 = glm::mat4(1.0);

glm::mat4 model\_2 = glm::mat4(1.0);

glm::mat4 model\_3 = glm::mat4(1.0);

glm::mat4 model\_4 = glm::mat4(1.0);

glm::mat4 model\_5 = glm::mat4(1.0);

glm::mat4 model\_6 = glm::mat4(1.0);

glm::mat4 model\_7 = glm::mat4(1.0);

glm::mat4 model\_8 = glm::mat4(1.0);

glm::mat4 model\_9 = glm::mat4(1.0);

glm::mat4 model\_10 = glm::mat4(1.0);

glm::mat4 model\_11 = glm::mat4(1.0);

glm::mat4 model\_12 = glm::mat4(1.0);

glm::mat4 model\_13 = glm::mat4(1.0);

glm::mat4 model\_14 = glm::mat4(1.0);

glm::mat4 model\_15 = glm::mat4(1.0);

//视角锁定

jude\_viewlock();

// view/projection transformations 观察空间 +视角锁定

//------------------------------------------------------

//视角锁定 Version2.0

/\*if (viewlocker == 1.0)

{

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

}

if (viewlocker==0.0)

{

glm::vec3 cameraPos\_lock = glm::vec3(30.0f, 30.0f, 30.0f); //摄像机位置

glm::vec3 cameraFront\_lock = glm::vec3(-1.0f, -1.0f, -1.0f);//摄像机视线方向

glm::vec3 cameraUp\_lock = glm::vec3(0.0f, 1.0f, 0.0f);

glm::mat4 projection\_lock = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view\_lock = glm::lookAt(cameraPos\_lock, cameraPos\_lock + cameraFront\_lock, cameraUp\_lock);

ourShader.setMat4("projection", projection\_lock);

ourShader.setMat4("view", view\_lock);//view 观察矩阵

}\*/

glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

ourShader.setMat4("projection", projection);

ourShader.setMat4("view", view);//view 观察矩阵

// render the loaded model

//------------------------------------------------

//小车按键控制随意移动

judge\_move(); //执行前进，计算更新位置向量model\_pos\_

//judge\_turn(); //执行旋转，计算更新位置向量model\_pos\_

//---------------------------------------------------------------------

if (have\_one\_loop) //have\_one\_loop==true 则更新下一组 要达到的点

{

if (raw\_slider\_4 < length - 1)//如果不是最后一组位置点-->接着更新位置点组

{

raw\_slider\_1 += 4;

raw\_slider\_2 += 4;

raw\_slider\_3 += 4;

raw\_slider\_4 += 4;

arrive\_group\_copy[0][0] = arrive\_group[0][0]; arrive\_group\_copy[0][1] = arrive\_group[0][1];

arrive\_group\_copy[0][2] = arrive\_group[0][2];

arrive\_group\_copy[1][0] = arrive\_group[1][0]; arrive\_group\_copy[1][1] = arrive\_group[1][1];

arrive\_group\_copy[1][2] = arrive\_group[1][2];

arrive\_group\_copy[2][0] = arrive\_group[2][0]; arrive\_group\_copy[2][1] = arrive\_group[2][1];

arrive\_group\_copy[2][2] = arrive\_group[2][2];

arrive\_group\_copy[3][0] = arrive\_group[3][0]; arrive\_group\_copy[3][1] = arrive\_group[3][1];

arrive\_group\_copy[3][2] = arrive\_group[3][2];

//[][0] [][1] 水平方向 [][2]竖直方向 arrive\_group,arrive\_group\_copy

arrive\_group[0][0] = trans\_point[raw\_slider\_1].x; arrive\_group[0][1] = trans\_point[raw\_slider\_1].y;

arrive\_group[0][2] = trans\_point[raw\_slider\_1].z;

arrive\_group[1][0] = trans\_point[raw\_slider\_2].x; arrive\_group[1][1] = trans\_point[raw\_slider\_2].y;

arrive\_group[1][2] = trans\_point[raw\_slider\_2].z;

arrive\_group[2][0] = trans\_point[raw\_slider\_3].x; arrive\_group[2][1] = trans\_point[raw\_slider\_3].y;

arrive\_group[2][2] = trans\_point[raw\_slider\_3].z;

arrive\_group[3][0] = trans\_point[raw\_slider\_4].x; arrive\_group[3][1] = trans\_point[raw\_slider\_4].y;

arrive\_group[3][2] = trans\_point[raw\_slider\_4].z;

jude\_direction\_change();//重新确定方向

have\_arrive\_1 = false; have\_arrive\_1\_x = false; have\_arrive\_1\_z = false; have\_arrive\_1\_y = false;

have\_arrive\_2 = false; have\_arrive\_2\_x = false; have\_arrive\_2\_z = false; have\_arrive\_2\_y = false;

have\_arrive\_3 = false; have\_arrive\_3\_x = false; have\_arrive\_3\_z = false; have\_arrive\_3\_y = false;

have\_arrive\_4 = false; have\_arrive\_4\_x = false; have\_arrive\_4\_z = false; have\_arrive\_4\_y = false;

have\_one\_loop = false; //重新置位到达状态

}

else

{

have\_arrive\_1 = true;

have\_arrive\_2 = true;

have\_arrive\_3 = true;

have\_arrive\_4 = true;

have\_one\_loop = true; //重新置位到达状态

}

}

//定位器运送物体到指定点

if (!have\_one\_loop)

{

do\_movement\_slider\_1();

do\_movement\_slider\_2();

do\_movement\_slider\_3();

do\_movement\_slider\_4();

if (have\_arrive\_1 && have\_arrive\_2 && have\_arrive\_3 && have\_arrive\_4)

{

have\_one\_loop = true;

}

}

//---------------------------------------------

//定位器底座

model = glm::translate(model, model\_pos); //定义模型每个循环出现的位置

model = glm::scale(model, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model);

ourModel.Draw(ourShader);

//X\_1

// render the loaded model

model\_1 = glm::translate(model\_1, model\_pos\_1); //定义模型每个循环出现的位置

model\_1 = glm::scale(model\_1, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_1);

ourModel\_1.Draw(ourShader);

// Y\_1

// render the loaded model

model\_2 = glm::translate(model\_2, model\_pos\_2); //定义模型每个循环出现的位置

model\_2 = glm::scale(model\_2, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_2);

ourModel\_2.Draw(ourShader);

// Z\_1

// render the loaded model

model\_3 = glm::translate(model\_3, model\_pos\_3); //定义模型每个循环出现的位置

model\_3 = glm::scale(model\_3, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_3);

ourModel\_3.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_4 = glm::translate(model\_4, model\_pos\_4); //定义模型每个循环出现的位置

model\_4 = glm::scale(model\_4, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_4);

ourModel\_4.Draw(ourShader);

//X\_1

// render the loaded model

model\_5 = glm::translate(model\_5, model\_pos\_5); //定义模型每个循环出现的位置

model\_5 = glm::scale(model\_5, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_5);

ourModel\_5.Draw(ourShader);

// Y\_1

// render the loaded model

model\_6 = glm::translate(model\_6, model\_pos\_6); //定义模型每个循环出现的位置

model\_6 = glm::scale(model\_6, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_6);

ourModel\_6.Draw(ourShader);

// Z\_1

// render the loaded model

model\_7 = glm::translate(model\_7, model\_pos\_7); //定义模型每个循环出现的位置

model\_7 = glm::scale(model\_7, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_7);

ourModel\_7.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_8 = glm::translate(model\_8, model\_pos\_8); //定义模型每个循环出现的位置

model\_8 = glm::scale(model\_8, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_8);

ourModel\_8.Draw(ourShader);

//X\_1

// render the loaded model

model\_9 = glm::translate(model\_9, model\_pos\_9); //定义模型每个循环出现的位置

model\_9 = glm::scale(model\_9, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_9);

ourModel\_9.Draw(ourShader);

// Y\_1

// render the loaded model

model\_10 = glm::translate(model\_10, model\_pos\_10); //定义模型每个循环出现的位置

model\_10 = glm::scale(model\_10, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_10);

ourModel\_10.Draw(ourShader);

// Z\_1

// render the loaded model

model\_11 = glm::translate(model\_11, model\_pos\_11); //定义模型每个循环出现的位置

model\_11 = glm::scale(model\_11, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_11);

ourModel\_11.Draw(ourShader);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//定位器底座

model\_12 = glm::translate(model\_12, model\_pos\_12); //定义模型每个循环出现的位置

model\_12 = glm::scale(model\_12, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_12);

ourModel\_12.Draw(ourShader);

//X\_1

// render the loaded model

model\_13 = glm::translate(model\_13, model\_pos\_13); //定义模型每个循环出现的位置

model\_13 = glm::scale(model\_13, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_13);

ourModel\_13.Draw(ourShader);

// Y\_1

// render the loaded model

model\_14 = glm::translate(model\_14, model\_pos\_14); //定义模型每个循环出现的位置

model\_14 = glm::scale(model\_14, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_14);

ourModel\_14.Draw(ourShader);

// Z\_1

// render the loaded model

model\_15 = glm::translate(model\_15, model\_pos\_15); //定义模型每个循环出现的位置

model\_15 = glm::scale(model\_15, glm::vec3(scale\_proportion, scale\_proportion, scale\_proportion)); // it's a bit too big for our scene, so scale it down

ourShader.setMat4("model", model\_15);

ourModel\_15.Draw(ourShader);

//--------------------------------------------------------------------------------------------------------------------

// 按键input

// -----------------------------------------------------------------------

processInput(window);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// glfw: terminate, clearing all previously allocated GLFW resources.

// ------------------------------------------------------------------

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

// ---------------------------------------------------------------------------------------------------------

void do\_movement\_slider\_1()

{

//-------------------------------------------------

if (!have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走x方向

{

model\_pos\_1.x += smooth\_steps\*slider\_1\_x\_dir\_value\_change;

model\_pos\_2.x += smooth\_steps\*slider\_1\_x\_dir\_value\_change;

model\_pos\_3.x += smooth\_steps\*slider\_1\_x\_dir\_value\_change;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_x = true;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完Z向

{

model\_pos\_2.z += smooth\_steps\*slider\_1\_z\_dir\_value\_change;

model\_pos\_3.z += smooth\_steps\*slider\_1\_z\_dir\_value\_change;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_z = true;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完y向

{

model\_pos\_3.y += smooth\_steps\*slider\_1\_y\_dir\_value\_change;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_y = true;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && have\_arrive\_1\_y)

{

have\_arrive\_1 = true;

steps\_slider\_1 = 0;

}

}

void do\_movement\_slider\_2()

{

//---------------------------------------------

if (!have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y) //走完x方向

{

model\_pos\_5.x += smooth\_steps\*slider\_2\_x\_dir\_value\_change;

model\_pos\_6.x += smooth\_steps\*slider\_2\_x\_dir\_value\_change;

model\_pos\_7.x += smooth\_steps\*slider\_2\_x\_dir\_value\_change;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_x = true;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && !have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_6.z += smooth\_steps\*slider\_2\_z\_dir\_value\_change;

model\_pos\_7.z += smooth\_steps\*slider\_2\_z\_dir\_value\_change;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_z = true;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && !have\_arrive\_2\_y)

{

model\_pos\_7.y += smooth\_steps\*slider\_2\_y\_dir\_value\_change;

steps\_slider\_2++;

if (steps\_slider\_2 >= 1 / smooth\_steps)

{

have\_arrive\_2\_y = true;

steps\_slider\_2 = 0;

}

}

if (have\_arrive\_2\_x && have\_arrive\_2\_z && have\_arrive\_2\_y)

{

have\_arrive\_2 = true;

}

}

void do\_movement\_slider\_3()

{

//------------------------------------------------

if (!have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y) //走完x方向

{

model\_pos\_9.x += smooth\_steps\*slider\_3\_x\_dir\_value\_change;

model\_pos\_10.x += smooth\_steps\*slider\_3\_x\_dir\_value\_change;

model\_pos\_11.x += smooth\_steps\*slider\_3\_x\_dir\_value\_change;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_x = true;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && !have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_10.z += smooth\_steps\*slider\_3\_z\_dir\_value\_change;

model\_pos\_11.z += smooth\_steps\*slider\_3\_z\_dir\_value\_change;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_z = true;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && !have\_arrive\_3\_y)

{

model\_pos\_11.y += smooth\_steps\*slider\_3\_y\_dir\_value\_change;

steps\_slider\_3++;

if (steps\_slider\_3 >= 1 / smooth\_steps)

{

have\_arrive\_3\_y = true;

steps\_slider\_3 = 0;

}

}

if (have\_arrive\_3\_x && have\_arrive\_3\_z && have\_arrive\_3\_y)

{

have\_arrive\_3 = true;

}

}

void do\_movement\_slider\_4()

{

//---------------------------------

if (!have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y) //走完x方向

{

model\_pos\_13.x += smooth\_steps\*slider\_4\_x\_dir\_value\_change;

model\_pos\_14.x += smooth\_steps\*slider\_4\_x\_dir\_value\_change;

model\_pos\_15.x += smooth\_steps\*slider\_4\_x\_dir\_value\_change;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_x = true;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && !have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_14.z += smooth\_steps\*slider\_4\_z\_dir\_value\_change;

model\_pos\_15.z += smooth\_steps\*slider\_4\_z\_dir\_value\_change;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_z = true;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && !have\_arrive\_4\_y)

{

model\_pos\_15.y += smooth\_steps\*slider\_4\_y\_dir\_value\_change;

steps\_slider\_4++;

if (steps\_slider\_4 >= 1 / smooth\_steps)

{

have\_arrive\_4\_y = true;

steps\_slider\_4 = 0;

}

}

if (have\_arrive\_4\_x && have\_arrive\_4\_z && have\_arrive\_4\_y)

{

have\_arrive\_4 = true;

}

}

void jude\_direction\_change()

{

//注意现实坐标系和 仿真空间坐标系之间的 映射

slider\_1\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[0][0]-arrive\_group\_copy[0][0]);

slider\_1\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[0][1] - arrive\_group\_copy[0][1]);

slider\_1\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[0][2] - arrive\_group\_copy[0][2]);

slider\_2\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[1][0] - arrive\_group\_copy[1][0]);

slider\_2\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[1][1] - arrive\_group\_copy[1][1]);

slider\_2\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[1][2] - arrive\_group\_copy[1][2]);

slider\_3\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[2][0] - arrive\_group\_copy[2][0]);

slider\_3\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[2][1] - arrive\_group\_copy[2][1]);

slider\_3\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[2][2] - arrive\_group\_copy[2][2]);

slider\_4\_x\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[3][0] - arrive\_group\_copy[3][0]);

slider\_4\_z\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[3][1] - arrive\_group\_copy[3][1]);

slider\_4\_y\_dir\_value\_change = magnify\_slidermove\*(arrive\_group[3][2] - arrive\_group\_copy[3][2]);

}

/\* void do\_movement\_slider\_1()

{

//-------------------------------------------------

if (!have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走x方向

{

model\_pos\_1 += smooth\_steps\*slider\_1\_x\_direction;

model\_pos\_2 += smooth\_steps\*slider\_1\_x\_direction;

model\_pos\_3 += smooth\_steps\*slider\_1\_x\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_x = true;

steps\_slider\_1 = 0;

model\_pos\_1.x += slider\_1\_x\_direction.x;

model\_pos\_2.x += slider\_1\_x\_direction.x;

model\_pos\_3.x += slider\_1\_x\_direction.x;

}

}

if (have\_arrive\_1\_x && !have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完Z向

{

model\_pos\_2 += smooth\_steps\*slider\_1\_z\_direction;

model\_pos\_3 += smooth\_steps\*slider\_1\_z\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_z = true;

model\_pos\_2.z += slider\_1\_z\_direction.z;

model\_pos\_3.z += slider\_1\_z\_direction.z;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && !have\_arrive\_1\_y) //走完y向

{

model\_pos\_3 += smooth\_steps\*slider\_1\_y\_direction;

steps\_slider\_1++;

if (steps\_slider\_1 >= 1 / smooth\_steps)

{

have\_arrive\_1\_y = true;

model\_pos\_3.y += slider\_1\_y\_direction.y;

steps\_slider\_1 = 0;

}

}

if (have\_arrive\_1\_x && have\_arrive\_1\_z && have\_arrive\_1\_y)

{

have\_arrive\_1 = true;

steps\_slider\_1 = 0;

}

}

\*/

//void jude\_direction()

//{

//

// //--------------------------------------------------------------

// //1号定位器各轴移动方向

// //通过对glm::vec3(arrive\_1.x - arrive\_1\_copy.x, 0, 0) 的检测，勉强能够达到七位有效数字的精度

//

// slider\_1\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_1.x - arrive\_1\_copy.x, 0, 0);

// slider\_1\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_1.z - arrive\_1\_copy.z);

// slider\_1\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_1.y - arrive\_1\_copy.y, 0);

//

// //--------------------------------------------------------------

// //2号定位器各轴移动方向

//

// slider\_2\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_2.x - arrive\_2\_copy.x, 0, 0);

// slider\_2\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_2.z - arrive\_2\_copy.z);

// slider\_2\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_2.y - arrive\_2\_copy.y, 0);

//

//

// //--------------------------------------------------------------

// //3号定位器各轴移动方向

//

// slider\_3\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_3.x - arrive\_3\_copy.x, 0, 0);

// slider\_3\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_3.z - arrive\_3\_copy.z);

// slider\_3\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_3.y - arrive\_3\_copy.y, 0);

//

// //--------------------------------------------------------------

// //4号定位器各轴移动方向

//

// slider\_4\_x\_direction = magnify\_slidermove \*glm::vec3(arrive\_4.x - arrive\_4\_copy.x, 0, 0);

// slider\_4\_z\_direction = magnify\_slidermove \*glm::vec3(0, 0, arrive\_4.z - arrive\_4\_copy.z);

// slider\_4\_y\_direction = magnify\_slidermove \*glm::vec3(0, arrive\_4.y - arrive\_4\_copy.y, 0);

//

//}

/\*

void jude\_direction()

{

//--------------------------------------------------------------

//1号定位器各轴移动方向

if (arrive\_1.x - model\_pos\_3.x > 0)

{

slider\_1\_x\_direction = 1.0f\*x\_direction;

}

else

{

slider\_1\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_1.z - model\_pos\_3.z > 0)

{

slider\_1\_z\_direction = 1.0f\*z\_direction;

}

else

{

slider\_1\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_1.y - model\_pos\_3.y > 0)

{

slider\_1\_y\_direction = 1.0f\*y\_direction;

}

else

{

slider\_1\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//2号定位器各轴移动方向

if (arrive\_2.x - model\_pos\_7.x > 0)

{

slider\_2\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_2\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_2.z - model\_pos\_7.z > 0) {

slider\_2\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_2\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_2.y - model\_pos\_7.y > 0) {

slider\_2\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_2\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//3号定位器各轴移动方向

if (arrive\_3.x - model\_pos\_11.x > 0) {

slider\_3\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_3\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_3.z - model\_pos\_11.z > 0) {

slider\_3\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_3\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_3.y - model\_pos\_11.y > 0) {

slider\_3\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_3\_y\_direction = -1.0f\*y\_direction;

}

//--------------------------------------------------------------

//四号定位器各轴移动方向

if (arrive\_4.x - model\_pos\_15.x > 0) {

slider\_4\_x\_direction = 1.0f\*x\_direction;

}

else {

slider\_4\_x\_direction = -1.0f\*x\_direction;

}

if (arrive\_4.z - model\_pos\_15.z > 0) {

slider\_4\_z\_direction = 1.0f\*z\_direction;

}

else {

slider\_4\_z\_direction = -1.0f\*z\_direction;

}

if (arrive\_4.y - model\_pos\_15.y > 0) {

slider\_4\_y\_direction = 1.0f\*y\_direction;

}

else {

slider\_4\_y\_direction = -1.0f\*y\_direction;

}

}\*/

void processInput(GLFWwindow\* window)

{

//float angle = 0.0f;

glm::vec3 front = glm::vec3(0.0f, 0.0f, -1.0f);

float cameraSpeed = 2.5f \* deltaTime;

//float cameraSpeed = 0.05f; // adjust accordingly

if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS)

cameraPos += cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS)

cameraPos -= cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS)

cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS)

cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

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)

{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

if (fov >= 1.0f && fov <= 45.0f)

fov -= yoffset;

if (fov <= 1.0f)

fov = 1.0f;

if (fov >= 45.0f)

fov = 45.0f;

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GL\_TRUE);

if (key >= 0 && key < 1024)

{

//设置按下/释放键为true或false

if (action == GLFW\_PRESS)

keys[key] = true;

else if (action == GLFW\_RELEASE)

keys[key] = false;

}

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos)

{

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos;

lastX = xpos;

lastY = ypos;

float sensitivity = 0.1;

xoffset \*= sensitivity;

yoffset \*= sensitivity;

yaw += xoffset;

pitch += yoffset;

if (pitch > 89.0f)

pitch = 89.0f;

if (pitch < -89.0f)

pitch = -89.0f;

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

//--------------------------------------------

//视角锁定 Version1.0

if (viewlocker == 1.0)

{

cameraFront = glm::normalize(front);

cameraFront\_now = cameraFront;

}

if (viewlocker == 0.0)

{

cameraFront = cameraFront\_now;//摄像机视线方向

//cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

}

}

void jude\_viewlock()

{

if (keys[GLFW\_KEY\_X])//X 视角解锁

{

viewlocker = 1.0;

}

if (keys[GLFW\_KEY\_C])//C 视角锁定

{

viewlocker = 0.0;

}

}

void judge\_move()

{

//前进后退计算更新位置向量

if (keys[GLFW\_KEY\_A])

{

model\_pos\_1 += velocity\*x\_direction;

model\_pos\_2 += velocity\*x\_direction;

model\_pos\_3 += velocity\*x\_direction;

model\_pos\_5 += velocity\*x\_direction;

model\_pos\_6 += velocity\*x\_direction;

model\_pos\_7 += velocity\*x\_direction;

model\_pos\_9 += velocity\*x\_direction;

model\_pos\_10 += velocity\*x\_direction;

model\_pos\_11 += velocity\*x\_direction;

model\_pos\_13 += velocity\*x\_direction;

model\_pos\_14 += velocity\*x\_direction;

model\_pos\_15 += velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_D])

{

model\_pos\_1 -= velocity\*x\_direction;

model\_pos\_2 -= velocity\*x\_direction;

model\_pos\_3 -= velocity\*x\_direction;

model\_pos\_5 -= velocity\*x\_direction;

model\_pos\_6 -= velocity\*x\_direction;

model\_pos\_7 -= velocity\*x\_direction;

model\_pos\_9 -= velocity\*x\_direction;

model\_pos\_10 -= velocity\*x\_direction;

model\_pos\_11 -= velocity\*x\_direction;

model\_pos\_13 -= velocity\*x\_direction;

model\_pos\_14 -= velocity\*x\_direction;

model\_pos\_15 -= velocity\*x\_direction;

}

if (keys[GLFW\_KEY\_W])

{

model\_pos\_2 += velocity\*z\_direction;

model\_pos\_3 += velocity\*z\_direction;

model\_pos\_6 += velocity\*z\_direction;

model\_pos\_7 += velocity\*z\_direction;

model\_pos\_10 += velocity\*z\_direction;

model\_pos\_11 += velocity\*z\_direction;

model\_pos\_14 += velocity\*z\_direction;

model\_pos\_15 += velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_S])

{

model\_pos\_2 -= velocity\*z\_direction;

model\_pos\_3 -= velocity\*z\_direction;

model\_pos\_6 -= velocity\*z\_direction;

model\_pos\_7 -= velocity\*z\_direction;

model\_pos\_10 -= velocity\*z\_direction;

model\_pos\_11 -= velocity\*z\_direction;

model\_pos\_14 -= velocity\*z\_direction;

model\_pos\_15 -= velocity\*z\_direction;

}

if (keys[GLFW\_KEY\_R])

{

model\_pos\_3 += velocity\*y\_direction;

model\_pos\_7 += velocity\*y\_direction;

model\_pos\_11 += velocity\*y\_direction;

model\_pos\_15 += velocity\*y\_direction;

}

if (keys[GLFW\_KEY\_F])

{

model\_pos\_3 -= velocity\*y\_direction;

model\_pos\_7 -= velocity\*y\_direction;

model\_pos\_11 -= velocity\*y\_direction;

model\_pos\_15 -= velocity\*y\_direction;

}

}