云南大学

本科实验报告

课程名称： 计算机图形学实验

实验名称： 实验四 OpenGL下图形的交互控制

学院（系）： 信息学院

专 业： 计算机科学与技术

年 级： 2020级

姓 名： 胡诚皓

学 号： 20201060330

指导教师： 吴昊

成 绩：

2022年 4 月 20 日

**一. 实验目的**

1. 熟悉OpenGL Glut中常用的回调函数；

2. 掌握用鼠标对图形进行交互控制的方法；

3. 掌握用键盘对图形进行交互控制的方法；

**二. 实验内容**

仔细阅读实验指导书，基于OpenGL完成以下内容：

（1）编程练习1

编制一个OpenGL的程序，生成一个600\*600大小的窗口，上面绘制一个200\*300大小的矩形，用键盘的四个方向键控制矩形朝上下左右进行移动，按c键退出程序。

（2）编程练习2

编制一个OpenGL的程序，生成一个600\*600大小的窗口，上面用鼠标画线。点击左键生成第一点，再点击左键生成第二点，用这两点画一条直线，然后再点击左键两次，再画另外一条直线。点击右键把屏幕清空。

**三. 实验代码**

**1. 编程练习1**

#include <GL/glut.h>

#include <iostream>

int WINDOW\_HEIGHT = 600;

int WINDOW\_WIDTH = 600;

// 移动时每一步的距离

const int move\_step = 5;

struct rectangle {

int x, y;// 左下角坐标

int width, height;

};

// 初始化矩形

rectangle rect1 = {10, 10, 200, 300};

// 标记是否需要重绘

bool need\_redraw = false;

// 普通键位的回调函数

void normal\_keyboard(unsigned char key, int x, int y) {

if (key == 'c' || key == 'C') {// c键退出

exit(0);

}

}

// 特殊键位的回调函数

void spec\_keyboard(int key, int x, int y) {

// std::cout << rect1.x << " " << rect1.y << std::endl;

if (key == GLUT\_KEY\_UP && rect1.y + rect1.height + move\_step <= WINDOW\_HEIGHT) {// 向上移动，但上边界不能超过屏幕高度

rect1.y += move\_step;

} else if (key == GLUT\_KEY\_DOWN && rect1.y - move\_step >= 0) {// 向下移动，但下边界不能小于0

rect1.y -= move\_step;

} else if (key == GLUT\_KEY\_LEFT && rect1.x - move\_step >= 0) {// 向左移动，但左边界不能小于0

rect1.x -= move\_step;

} else if (key == GLUT\_KEY\_RIGHT && rect1.x + rect1.width + move\_step <= WINDOW\_WIDTH) {// 向右移动，但右边界不能超过屏幕宽度

rect1.x += move\_step;

}

need\_redraw = true;

}

// 绘制矩形rect

void drawRect(rectangle rect) {

glRecti(rect.x, rect.y, rect.x + rect.width, rect.y + rect.height);

}

// 设置画笔颜色，并绘制一开始的矩形

void draw() {

glColor3f(0.4f, 0.8f, 1.0f);

drawRect(rect1);

glFlush();

}

void refresh() {

// 不需要重绘

if (!need\_redraw) {

return;

}

// 需要重绘，先清屏，再绘制矩形

glClear(GL\_COLOR\_BUFFER\_BIT);

drawRect(rect1);

glFlush();

need\_redraw = false;

}

void reshape(int width, int height) {

// 根据当前窗口大小改变全局变量

WINDOW\_HEIGHT = height;

WINDOW\_WIDTH = width;

// 重新设置裁剪窗口和视口

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();// 重新投影

gluOrtho2D(0.0, WINDOW\_WIDTH, 0.0, WINDOW\_HEIGHT);

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

// 窗口大小改变后需要进行重绘

need\_redraw = true;

}

int main(int argc, char \*argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutCreateWindow("interactive window");

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, WINDOW\_WIDTH, 0.0, WINDOW\_HEIGHT);

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glutDisplayFunc(draw);

glutIdleFunc(refresh);

glutSpecialFunc(spec\_keyboard);

glutKeyboardFunc(normal\_keyboard);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

**2. 编程练习2**

#include <GL/glut.h>

#include <iostream>

#include <vector>

int WINDOW\_HEIGHT = 600;

int WINDOW\_WIDTH = 600;

struct dcPt {

int x;

int y;

};

// 记录线的集合

std::vector<std::pair<dcPt, dcPt>> lines = {};

// 记录第一个点

dcPt pre;

// 是否已标记第一个点

bool pre\_flag = false;

// 需要重绘

bool need\_redraw = false;

// 将清屏操作独立出来

void clear\_screen() {

lines.clear();

glClearColor(1.0, 1.0, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

glEnd();

glFlush();

}

// 鼠标事件回调函数

void mouse(int button, int state, int x, int y) {

if (!pre\_flag) {// 鼠标左键点击第一个点

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

pre\_flag = true;

pre.x = x;

pre.y = y;

}

} else {// 鼠标左键点击第二个点

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

std::cout << "draw: " << pre.x << " " << pre.y << " to " << x << " " << y << std::endl;

lines.emplace\_back(pre, dcPt{x, y});

pre\_flag = false;

need\_redraw = true;

}

}

// 鼠标右键清屏

if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) {

std::cout << "clear" << std::endl;

clear\_screen();

}

}

// 设置画笔颜色为66CCFF、线宽为2

void draw() {

glColor3f(0.4f, 0.8f, 1.0f);

glLineWidth(2.0f);

}

// 根据lines中的更新重绘

void refresh() {

// 不需要重绘

if (!need\_redraw) {

return;

}

// 需要重绘，先清屏，再绘制lines中的线

glClear(GL\_COLOR\_BUFFER\_BIT);

for (std::pair<dcPt, dcPt> & line : lines) {

glBegin(GL\_LINES);

glVertex2i(line.first.x, WINDOW\_HEIGHT-line.first.y);

glVertex2i(line.second.x, WINDOW\_HEIGHT-line.second.y);

glEnd();

}

glFlush();

need\_redraw = false;

}

// 窗口大小改变时的回调函数

void reshape(int width, int height) {

// 根据当前窗口大小改变全局变量

WINDOW\_HEIGHT = height;

WINDOW\_WIDTH = width;

// 重新设置裁剪窗口和视口

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();// 重新投影

gluOrtho2D(0.0, WINDOW\_WIDTH, 0.0, WINDOW\_HEIGHT);

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

// 窗口大小改变后需要进行重绘

need\_redraw = true;

}

int main(int argc, char \*argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutCreateWindow("interactive window");

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, WINDOW\_WIDTH, 0.0, WINDOW\_HEIGHT);

clear\_screen();

glutDisplayFunc(draw);

glutMouseFunc(mouse);

glutReshapeFunc(reshape);

glutIdleFunc(refresh);

glutMainLoop();

return 0;

}

**四. 实验结果**

**1. 编程练习1**



双击图标查看演示视频

**2. 编程练习2**



双击图标查看演示视频

**五. 实验分析**

1. 非静态画面显示

对非静态画面，此处采用把画面变化统一在编写的空闲回调函数refresh中，并使用need\_redraw来标识是否需要重绘。若need\_redraw在Glut内部调用refresh时为true，就按照refresh中的代码先进行清屏，再重绘矩形或所有线段。这样可以使得整体的画面处理逻辑更加简洁。

若在后续要添加更多需要绘制的图形，可以直接添加代码，而不用重新修改之前已有的绘制代码。

2. 窗口大小变化时的处理

在窗口大小变化时，需要考虑是要将已绘制的图形形状进行拉伸还是保持原有的比例，位置是否要根据窗口缩放的比例进行移动还是单纯地保持原有位置不动。

此处的两个题目中，均采用了“保持位置、大小和形状都不变”的策略，通过gluOrtho2D调整投影坐标到调整后的大小达到保持坐标点是绝对的，同时使用glViewport来调整视口的大小，使得图形不会被拉伸与缩放。

3. 考虑的其他细节

①通过上下左右方向键调整矩形位置时，显然需要让矩形不能跑到看不到的裁剪窗口之外；

②按“c”键退出，需要考虑用户可能按的可能是大写的“C”，也可能是小写的。