实验五 圆的扫描转换和种子点填充实验

时间：2022年4月13日

地点：信息学院2202

1、实验内容

实验一：圆的扫描转换

实验二：种子点填充算法

2、实验目的

实验一：输入圆的半径，画出圆

实验二：输入多边形，种子点位置，填充多边形

3、实验代码

实验一：圆的扫描转换

#include <windows.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <math.h>

void init(void)

{

glClearColor (0.5, 1.0, 1.0, 0.0); // 指定清空颜色（背景色）为白色

gluOrtho2D (-150.0, 150.0, -150.0, 150.0); //指定二维坐标系中被显示的区域

}

void putpixel(int x,int y)

{

glPointSize(1.0f);

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

}

void circlePoint(int x, int y)

{

putpixel(x,y);

putpixel(y,x);

putpixel(-y,x);

putpixel(-x,y);

putpixel(-x,-y);

putpixel(-y,-x);

putpixel(y,-x);

putpixel(x,-y);

}

void MidBresenhamcircle(int r)

{

int x, y, d;

x=0;

y=r;

d=1.25-r; /\*赋初值\*/

while(x<y) /\*循环退出条件\*/

{

circlePoint(x,y); /\*绘制一个点\*/

if (d<0) d+=2\*x+3; /\*判别式更新\*/

else

{

d+=2\*(x-y)+5;

y--; /\*y方向上走一步\*/

}

x++; /\*x方向上走一步\*/

} /\*while循环结束\*/

}

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);// 清空显示窗口

glColor3f(0.0,0.0,0.0);// 指定前景色（当前绘制颜色）为黑色

MidBresenhamcircle(80);

glFlush();// 使绘制立即反映到屏幕上

}

int main (int argc, char\*\* argv)

{

glutInit (&argc, argv); // 初始 GLUT.

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB); //设定显示模式

glutInitWindowPosition (500, 200); // 设定窗口位置

glutInitWindowSize (300, 300); // 设定窗口大小

glutCreateWindow ("An Example OpenGL Program");

// 用前面指定参数创建窗口，并定义窗口名称

init( ); // 进行一些初始化工作

glutDisplayFunc(&myDisplay); // 指定绘制的回调函数

glutMainLoop ( ); // 进入无穷循环，等待事件处理

}

实验二：种子点填充算法

#define \_STDCALL\_SUPPORTED

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

#include <stdlib.h>

typedef float Color[3];

//获取像素点的颜色

void getpixel(GLint x, GLint y, Color color) {

glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, color);

}

//画点函数

void setpixel(GLint x, GLint y) {

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

}

//比较颜色是否相等

int compareColor(Color color1, Color color2) {

if (color1[0] != color2[0] || color1[1] != color2[1] || color1[2] != color2[2]) { return 0; }

else { return 1; }

}

void boundaryFill4(int x, int y, Color fillColor, Color boarderColor) {

Color interiorColor;

getpixel(x, y, interiorColor);

if (compareColor(interiorColor, fillColor) == 0 && compareColor(interiorColor, boarderColor) == 0) {

setpixel(x, y);

boundaryFill4(x + 1, y, fillColor, boarderColor);

boundaryFill4(x - 1, y, fillColor, boarderColor);

boundaryFill4(x, y + 1, fillColor, boarderColor);

boundaryFill4(x, y - 1, fillColor, boarderColor);

}

}

void boundaryFill8(int x, int y, Color fillColor, Color boarderColor) {

Color interiorColor, a, b, c, d;

getpixel(x, y, interiorColor);

getpixel(x + 1, y - 1, a);

getpixel(x, y - 1, b);

getpixel(x, y + 1, c);

getpixel(x - 1, y, d);

int i = 0;

if (compareColor(a, boarderColor) == 1) i++;

if (compareColor(b, boarderColor) == 1) i++;

if (compareColor(c, boarderColor) == 1) i++;

if (compareColor(d, boarderColor) == 1) i++;

if (i <= 1) {

if (compareColor(interiorColor, fillColor) == 0 && compareColor(interiorColor, boarderColor) == 0) {

setpixel(x, y);

boundaryFill8(x + 1, y, fillColor, boarderColor);

boundaryFill8(x, y - 1, fillColor, boarderColor);

boundaryFill8(x - 1, y, fillColor, boarderColor);

boundaryFill8(x, y + 1, fillColor, boarderColor);

boundaryFill8(x + 1, y - 1, fillColor, boarderColor);

boundaryFill8(x - 1, y - 1, fillColor, boarderColor);

boundaryFill8(x - 1, y + 1, fillColor, boarderColor);

boundaryFill8(x + 1, y + 1, fillColor, boarderColor);

}

}

}

void polygon() {

glBegin(GL\_LINE\_LOOP);

glLineWidth(5);

//此处修改坐标，绘制多边形

glVertex2f(100, 100);

glVertex2f(100, 200);

//glVertex2f(200, 200);

glVertex2f(200, 100);

glEnd();

}

void display(void) {

Color fillColor = {0.0, 1.0, 1.0};//填充颜色 蓝色

Color boarderColor = {0.0, 1.0, 0.0};//边界颜色 绿色

glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0, 0, 500, 500);

glColor3fv(boarderColor);

polygon();

glColor3fv(fillColor);

boundaryFill4(110, 110, fillColor, boarderColor);//设置起点坐标及颜色

//boundaryFill8(110, 110, fillColor, boarderColor);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RED);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("BoundaryFill1");

glClearColor(1, 1, 1, 0.0);

glMatrixMode(GL\_PROJECTION);//投影模型

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glutDisplayFunc(display);

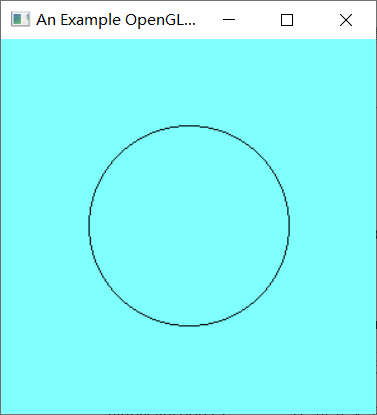
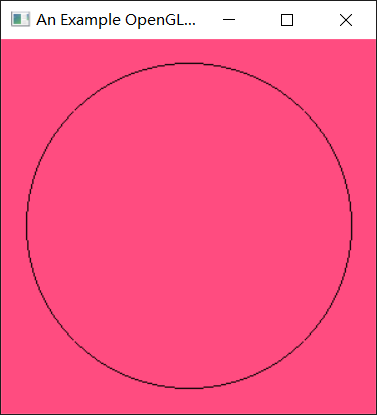
glutMainLoop();

return 0;

}

4、实验结果

实验一：圆的扫描转换



实验二：种子点填充算法

图片包含 图形用户界面

描述已自动生成

5、实验总结

Bresenham画圆算法与画线算法原理基本相同，都是利用渐趋逼近的方法去确定下一个需要画的点。在画圆时采用了八分法，用X轴、Y轴及y=x和y=-x四条直线将圆分为八份，画圆时只需画出一个部分，按照对称关系即可快速画出其他的部分。