**计算机图形学实验**

**姓 名：杨媛**

**学 号：20201050475**

**专 业：计算机科学与技术**

**教 师：钱文华**

实验一 直线段生成算法

时间：2022.3.16

地点：信息学院2202

1. 实验内容：

熟悉OPENGL，通过示例程序生成直线段。

1. 实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

1. 实验代码：

#include <windows.h>

#include <GL/glut.h>

//示例程序生成直线段

void init (void)

{

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

gluOrtho2D (0.0, 400.0, 0.0, 300.0); //指定二维坐标系中被显示的区域

}

void display (void)

{

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

glColor3f (0.0, 0.0, 1.0); // 指定前景色（当前绘制颜色）为蓝色

glBegin (GL\_LINES);

glVertex2i (180, 15); // 指定顶点

glVertex2i (10, 145);

glEnd ( );

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 (50, 100); // 设定窗口位置

glutInitWindowSize (400, 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

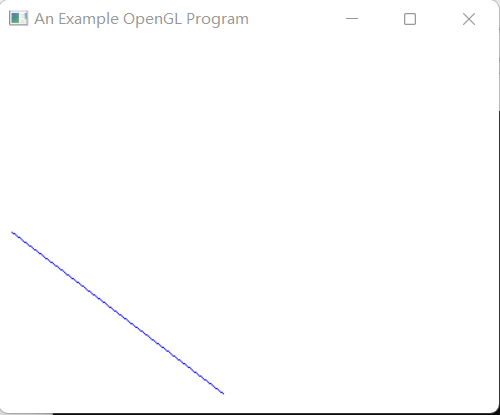
init ( ); // 进行一些初始化工作

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

glutMainLoop ( ); // 进入无穷循环，等待事件处理

return 0;

}

4、实验结果：

实验二 DDA直线生成算法

时间：2022.3.23

地点：信息学院2202

1、实验内容：

熟悉OPENGL，通过DDA、中点算法生成直线段。

2、实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

3、实验代码：

#include <windows.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <math.h>

//DDA算法生成直线段

void DDA\_Line(int x1, int y1, int x2, int y2) {

double dx, dy, e, x, y;

dx = x2 - x1;

dy = y2 - y1;

e = (fabs(dx) > fabs(dy)) ? fabs(dx) : fabs(dy);

dx /= e;

dy /= e;

x = x1;

y = y1;

for (int i = 0; i < e; i++) {

glPointSize(5.0);

glBegin(GL\_POINTS);

glVertex2i(int(x + 0.5), int(y + 0.5));

glEnd();

glFlush();

x += dx;

y += dy;

}

}

void display(void) {

glClearColor(1.0, 1.0, 1.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0, 0, 500, 500);

DDA\_Line(100, 400, 250, 300);

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RED);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0,0);

glutCreateWindow("DDA\_line");

glutDisplayFunc(display);

glColor3f(0.0, 0.0, 0.0);//颜色

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glutMainLoop();

return 0;

}

4、实验结果：



实验三 Bresenham、 改进Bresenham算法生成直线段实验

时间：2022.3.30

地点：信息学院2202

1、实验内容：

熟悉OPENGL，通过Bresenham中点、改进Bresenham算法生成直线段。

2、实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

3、实验代码：

//中点Bresenham算法生成直线

#include <windows.h>

#include <GL/glut.h>

#include <math.h>

#include<iostream>

#define WIDTH 500 //窗口宽度

#define HEIGHT 500 //窗口高度

void Init(){//初始化

glClearColor(1.0f,1.0f,1.0f,1.0f); //设置背景颜色，完全不透明

glColor3f(1.0f,0.0f,0.0f); //设置画笔颜色

glMatrixMode(GL\_PROJECTION); //设置投影

gluOrtho2D(0.0, WIDTH, 0.0, HEIGHT); //设置投影区域

else d+=UpIncre;

}

}

else if(dy<(-dx)) //k<-1

{

d=-dy-2\*dx;

UpIncre=2\*dx+2\*dy;

DownIncre=2\*dx;

while(y>=y1)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

y--;

if(d<0)

{

x++;

d-=UpIncre;

}

else d-=DownIncre;

}

}

else //k>1和k不存在

{

d=dy-2\*dx;

UpIncre=2\*dy-2\*dx;

DownIncre=-2\*dx;

while(y<=y1)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

y++;

if(d<0)

{

x++;

d+=UpIncre;

}

else d+=DownIncre;

}

}

}

void Display() //显示函数

}

void MidpointBresenham(int x0,int y0,int x1,int y1) { //中点Bresenham算法画线

int dx,dy,d,UpIncre,DownIncre,x,y;

if(x0>x1){

x=x1;x1=x0;x0=x;

y=y1;y1=y0;y0=y;

}

x = x0,y = y0;

dx = x1-x0;

dy = y1-y0;

if(dy>0&&dy<=dx){ //0<k<=1

d = dx-2\*dy;

UpIncre = 2\*dx-2\*dy;

DownIncre = -2\*dy;

while(x<=x1){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

x++;

if(d<0){

y++;

d+=UpIncre;

}

else

d+=DownIncre;

}

}

else if((dy>=(-dx))&&dy<=0) //-1<=k<=0

{

d=dx-2\*dy;

UpIncre=-2\*dy;

DownIncre=-2\*dx-2\*dy;

while(x<=x1)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

x++;

if(d>0)

{

y--;

d+=DownIncre;

}

//改进的Bresenham算法

#include <windows.h>

#include <GL/glut.h>

#include <math.h>

#include<iostream>

using namespace std;

void ProBresenham(int x0, int y0, int x1, int y1){

int x, y, dx, dy, e;

dx = x1 - x0;

dy = y1 - y0;

e = -dx;

x = x0;

y = y0;

while (x <= x1) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

x++;

e = e + 2 \* dy;

if (e > 0) {

y++;

e = e - 2 \* dx;

}

}

}

void Display(){//显示函数

{

glClear(GL\_COLOR\_BUFFER\_BIT); //清空颜色堆栈

MidpointBresenham(100,200,200,100); //画直线

glFlush(); //清空缓冲区指令

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //初始化显示模式

glutInitWindowSize(WIDTH,HEIGHT); //设置窗口尺寸

glutInitWindowPosition(200,100); //设置窗口位置

glutCreateWindow("Bresenham画直线"); //创建窗口

glutDisplayFunc(Display); //注册显示函数

Init(); //初始化

glutMainLoop(); //进入程序循环

return 0;

}

glClear(GL\_COLOR\_BUFFER\_BIT); //清空颜色堆栈

ProBresenham(100,100,400,400);//画直线

glFlush(); //清空缓冲区指令

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //初始化显示模式

glutInitWindowSize(600, 600); //设置窗口尺寸

glutInitWindowPosition(200, 100); //设置窗口位置

glutCreateWindow("改进Bresenham算法"); //创建窗口

glutDisplayFunc(Display); //注册显示函数

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //设置背景颜色，完全不透明

glColor3f(1.0f, 0.0f, 0.0f); //设置画笔颜色

glMatrixMode(GL\_PROJECTION); //设置投影

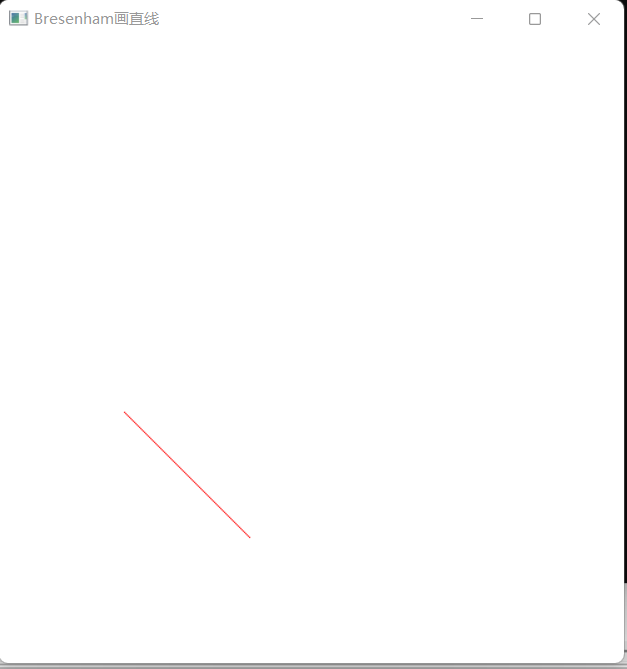
gluOrtho2D(0.0, 600, 0.0, 600); //设置投影区域

glutMainLoop(); //进入程序循环

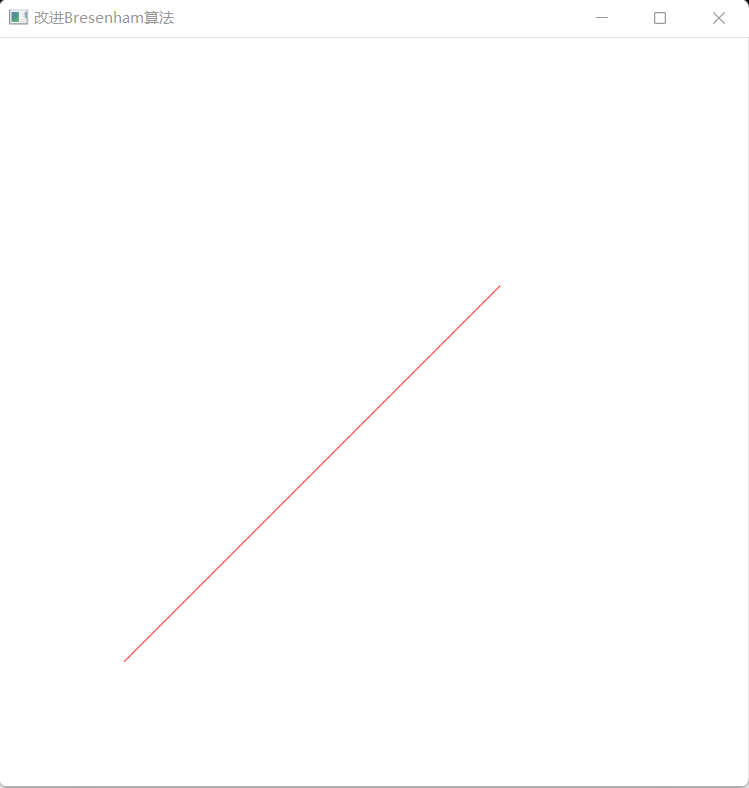
return 0;

}

4、实验结果：

中点Bresenham算法生成直线：

改进的Bresenham算法：



实验四 填充算法实验

时间：2022.4.6

地点：信息学院2202

1、实验内容：

使用opengl，用扫描线填充算法填充多边形。

2、实验目的：

验证扫描线填充算法，指定任意的多边形边数，填充多边形。

#include <windows.h>

#include <GL/glut.h>

//扫描线填充算法填充多边形

const int POINTNUM=4;//多边形点数

//定义结构体用于活性边表AET和新边表NET

typedef struct XET

{

float x;

float dx,ymax;

XET\* next;

} AET,NET;

3、实验代码：

p->dx=(polypoint[(j-1+POINTNUM)%POINTNUM].x-polypoint[j].x)/(polypoint[(j-1+POINTNUM)%POINTNUM].y-polypoint[j].y);

p->next=pNET[i]->next;

pNET[i]->next=p;

}

if(polypoint[(j+1+POINTNUM)%POINTNUM].y>polypoint[j].y)

{

NET \*p=new NET;

p->x=polypoint[j].x;

p->ymax=polypoint[(j+1+POINTNUM)%POINTNUM].y;

p->dx=(polypoint[(j+1+POINTNUM)%POINTNUM].x-polypoint[j].x)/(polypoint[(j+1+POINTNUM)%POINTNUM].y-polypoint[j].y);

p->next=pNET[i]->next;

pNET[i]->next=p;

}

}

}

//建立并更新活性边表AET

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

{

//计算新的交点x,更新AET

NET \*p=pAET->next;

while(p)

{

p->x=p->x + p->dx;

p=p->next;

}

//更新后新AET先排序

//断表排序,不再开辟空间

AET \*tq=pAET;

p=pAET->next;

tq->next=NULL;

//定义点结构体point

struct point

{

float x;

float y;

} polypoint[POINTNUM]= { {250,50},{550,150},{550,400},{250,250}};

void PolyScan()

{

//计算最高点的y坐标(扫描到此结束)

int MaxY=0;

int i;

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

if(polypoint[i].y>MaxY)

MaxY=polypoint[i].y;

//初始化AET表

AET \*pAET=new AET;

pAET->next=NULL;

//初始化NET表

NET \*pNET[1024];

for(i=0; i<=MaxY; i++){

pNET[i]=new NET;

pNET[i]->next=NULL;

}

glClear(GL\_COLOR\_BUFFER\_BIT);//赋值的窗口显示

glColor3f(1.0,0.0,0.0);//设置直线的颜色红色

glBegin(GL\_POINTS);

//扫描并建立NET表

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

{

for(int j=0; j<POINTNUM; j++)

if(polypoint[j].y==i)

{

//一个点跟前面的一个点形成一条线段，跟后面的点也形成线段 if(polypoint[(j-1+POINTNUM)%POINTNUM].y>polypoint[j].y) {

NET \*p=new NET;

p->x=polypoint[j].x; p->ymax=polypoint[(j-1+POINTNUM)%POINTNUM].y;

p=p->next->next;//考虑端点情况

}

}

glEnd();

glFlush();

}

void init(int argc,char\*\* argv)

{

glutInit(&argc,argv); //初始化 GLUT

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //设置显示模式：单个缓存和使用RGB模型

glutInitWindowPosition(50,100); //设置窗口的顶部和左边位置

glutInitWindowSize(400,300); //设置窗口的高度和宽度

glutCreateWindow("Scan Program");

glClearColor(1.0,1.0,1.0,0); //窗口背景颜色为白色

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0,600,0,450);

}

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,0.4,0.2);

glPointSize(1);

glBegin(GL\_POINTS);

PolyScan();

glEnd();

glFlush();

}

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

{

init(argc,argv);

glutDisplayFunc(myDisplay);

glutMainLoop();

return 0;

}

while(p) {

while(tq->next && p->x >= tq->next->x)

tq=tq->next;

NET \*s=p->next;

p->next=tq->next;

tq->next=p;

p=s;

tq=pAET;

}

//(改进算法)先从AET表中删除ymax==i的结点

AET \*q=pAET;

p=q->next;

while(p){

if(p->ymax==i){

q->next=p->next;

delete p;

p=q->next;

}

else{

q=q->next;

p=q->next;

}

}

//将NET中的新点加入AET,并用插入法按X值递增排序

p=pNET[i]->next;

q=pAET;

while(p){

while(q->next && p->x >= q->next->x)

q=q->next;

NET \*s=p->next;

p->next=q->next;

q->next=p;

p=s;

q=pAET;

}

//配对填充颜色

p=pAET->next;

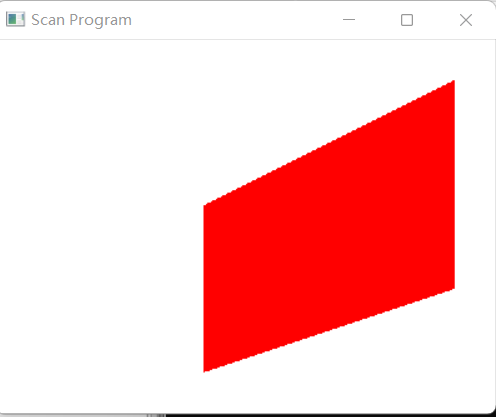
while(p && p->next)

{

for(float j=p->x; j<=p->next->x; j++)

glVertex2i(static\_cast<int>(j),i);

4、实验结果：



实验五 圆扫描转换和种子点填充实验

时间：2022.4.13

地点：信息学院2202

1、实验内容：

（1）圆扫描转换；

（2）种子点填充算法。

2、实验目的：

（1）输入圆的半径，画出圆。

（2）输入多边形，种子点位置，填充多边形。

3、实验代码：

#include <windows.h>//圆扫描转换

#include <GL/glut.h>

#include <iostream>

using namespace std;

float r;

void BresenhamCircle(void){

float d, x, y;

glClear(GL\_COLOR\_BUFFER\_BIT);

x = 0;

y = r;

d = 1 - r;

while (x <= y){

glBegin(GL\_POINTS);

glColor3f(1.0, 0.0, 0.0);//线的颜色为红色

glVertex3f( x, y, 0);

glVertex3f( -x, -y, 0);

glVertex3f( -x, y, 0);

glVertex3f( x, -y, 0);

glVertex3f( y, -x, 0);

glVertex3f( -y, -x, 0);

glVertex3f( -y, x, 0);

glVertex3f( y, x, 0);

glEnd();

if (d < 0)

d += 2 \* x + 3;

else{

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

y--;

}

x++;

glFlush();

}

}

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

glutInit(&argc, argv); //初始化glut

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_SINGLE | GLUT\_RGBA);

//设置窗口的模式－深度缓存，单缓存，颜色模型

glutInitWindowPosition(200, 200); //设置窗口的位置

glutInitWindowSize(500, 500); //设置窗口的大小

glutCreateWindow("circle"); //创建窗口并赋予title

cout << "请输入r：" << endl;

cin >> r;

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-500.0, 500.0, -500.0, 500.0);

glMatrixMode(GL\_MODELVIEW);

glutDisplayFunc(BresenhamCircle);//调用renderScene把绘制传送到窗口

glutMainLoop(); //进入循环等待

return 0;

}

//种子点填充算法

#include <windows.h>

#include <GL/glut.h>

#include <fstream>

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,y+1, fillColor, boarderColor);

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

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

boundaryFill4(x+1,y, 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, 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

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RED);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("BoundaryFill");

glClearColor(1, 1, 1, 0.0);

glMatrixMode(GL\_PROJECTION);//投影模型

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glutDisplayFunc(display);

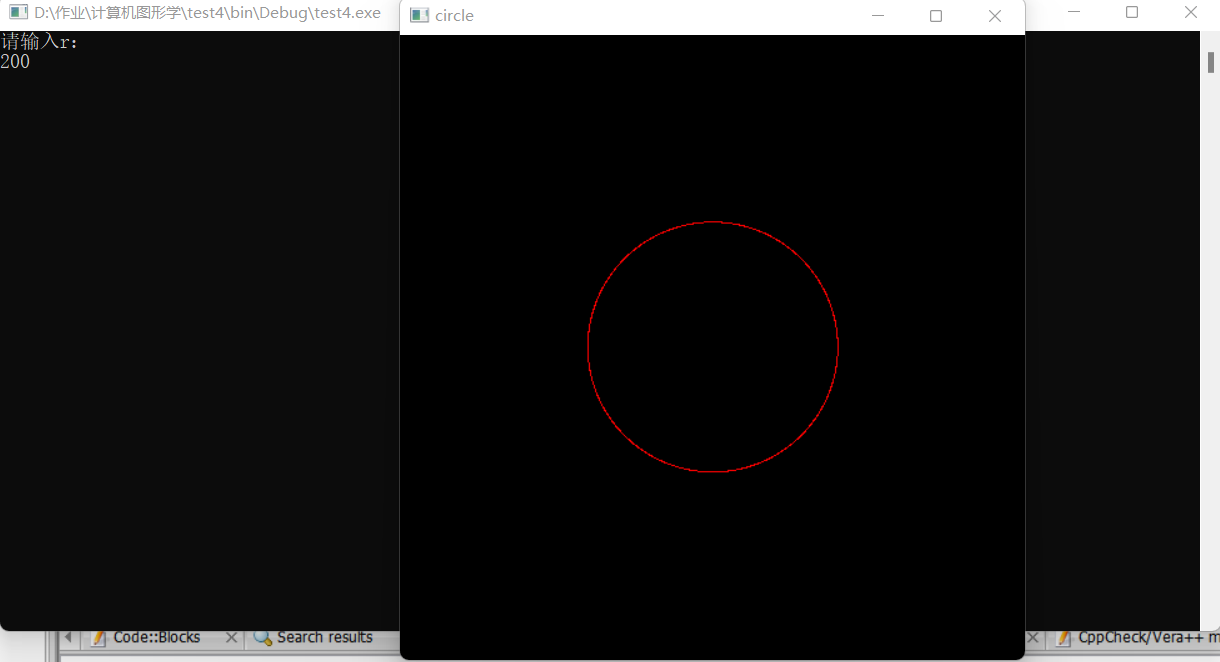
glutMainLoop();

return 0;

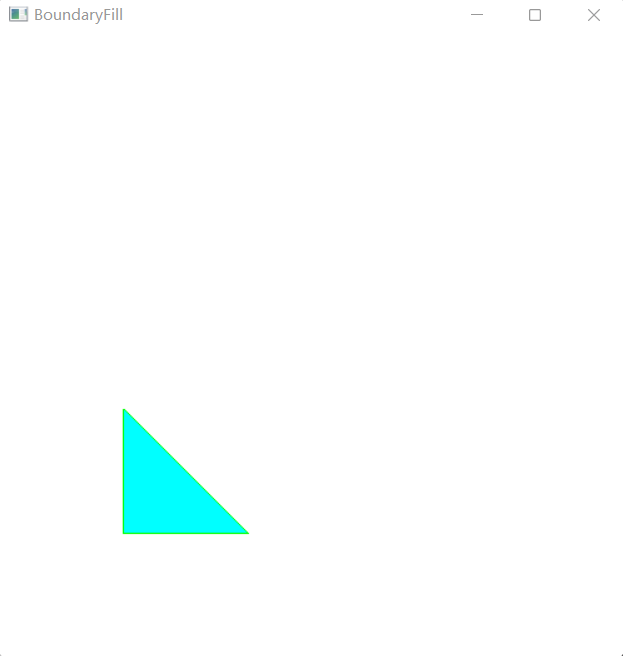
}

4、实验结果：

圆扫描转换：



种子点填充算法：



实验六 二维几何变换实验

时间：2022.4.20

地点：信息学院2202

1、实验内容：

教材P161，二维几何变换算法（平移、比例、旋转、对称）。

2、实验目的：

验证二维几何变换，熟悉变换矩阵。

3、实验代码：

#include <windows.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <math.h>

GLsizei winWidth = 600, winHeight = 600;

GLfloat xwcMin = 0.0, xwcMax = 225.0;

GLfloat ywcMin = 0.0, ywcMax = 225.0;

class wcPt2D

{

public:

GLfloat x, y;

};

typedef GLfloat Matrix3x3[3][3];

Matrix3x3 matComposite;

const GLdouble pi = 3.14159;

void Init()

{

glClearColor(1.0, 1.0, 1.0, 0.0);

}

void Matrix3x3SetIdentity(Matrix3x3 matIdent3x3)

{

GLint row, col;

for (row = 0; row < 3; row++)

{

for (col = 0; col < 3; col++)

{

matIdent3x3[row][col] = (row == col);

}

}

}

void Matrix3x3PreMultiply(Matrix3x3 m1, Matrix3x3 m2)

{

GLint row, col;

Matrix3x3 matTemp;

for (row = 0; row < 3; row++)

{

for (col = 0; col < 3; col++)

{

matTemp[row][col] =

m1[row][0]\* m2[0][col] +

m1[row][1]\* m2[1][col]+

m1[row][2]\* m2[2][col];

}

}

for (row = 0; row < 3; row++)

{

for (col = 0; col < 3; col++)

{

m2[row][col] = matTemp[row][col];

}

}

}

}

void Translate2D(GLfloat tx, GLfloat ty)

{

Matrix3x3 matTransl;

Matrix3x3SetIdentity(matTransl);

matTransl[0][2] = tx;

matTransl[1][2] = ty;

Matrix3x3PreMultiply(matTransl, matComposite);

}

void Rotate2D(wcPt2D pivotPt, GLfloat theta)

{

Matrix3x3 matRot;

Matrix3x3SetIdentity(matRot);

matRot[0][0] = cos(theta);

matRot[0][1] = -sin(theta);

matRot[0][2] = pivotPt.x \* (1-cos(theta)) +pivotPt.y \* sin(theta);

matRot[1][0] = sin(theta);

matRot[1][1] = cos(theta);

matRot[1][2] = pivotPt.y \* (1 - cos(theta)) -pivotPt.x \* sin(theta);

Matrix3x3PreMultiply(matRot, matComposite);

}

void Scale2D(GLfloat sx, GLfloat sy, wcPt2D fixedPt)

{

Matrix3x3 matScale;

Matrix3x3SetIdentity(matScale);

matScale[0][0] = sx;

matScale[0][2] = (1 - sx) \* fixedPt.x;

matScale[1][1] = sy;

matScale[1][2] = (1 - sy) \* fixedPt.y;

Matrix3x3PreMultiply(matScale, matComposite);

}

void TransformVerts2D(GLint nVerts, wcPt2D \* verts)

{

GLint k;

GLfloat temp;

for (k = 0; k < nVerts; k++)

{

temp = matComposite[0][0] \* verts[k].x +

matComposite[0][1] \* verts[k].y +

matComposite[0][2];

verts[k].y = matComposite[1][0] \* verts[k].x +

matComposite[1][1] \* verts[k].y +

matComposite[1][2];

verts[k].x = temp;

glColor3f(0, 0, 1);

Trangle(verts);

Matrix3x3SetIdentity(matComposite);

Scale2D(sx, sy, fixedPt);

Rotate2D(pivPt, theta);

Translate2D(tx, ty);

TransformVerts2D(nVerts, verts);

glColor3f(1, 0, 0);

Trangle(verts);

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(50, 50);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Geometric Transformation Sequence");

Init();

glutDisplayFunc(DisplayFcn);

glutReshapeFunc(WinReshapFcn);

glutMainLoop();

return 0;

}

}

void Trangle(wcPt2D \* verts)

{

GLint k;

glBegin(GL\_TRIANGLES);

for (k = 0; k < 3; k++)

{

glVertex2f(verts[k].x, verts[k].y);

}

glEnd();

}

void WinReshapFcn(GLint newWidth, GLint newHeight)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

void DisplayFcn()

{

GLint nVerts = 3;

wcPt2D verts[3] = { {50,25},{150,25 },{100,100} };

wcPt2D centroidPt;

GLint k, xSum = 0, ySum = 0;

for (size\_t k = 0; k < nVerts; k++)

{

xSum += verts[k].x;

ySum += verts[k].y;

}

centroidPt.x = GLfloat(xSum) / GLfloat(nVerts);

centroidPt.y = GLfloat(ySum) / GLfloat(nVerts);

wcPt2D pivPt, fixedPt;

pivPt = centroidPt;

fixedPt = centroidPt;

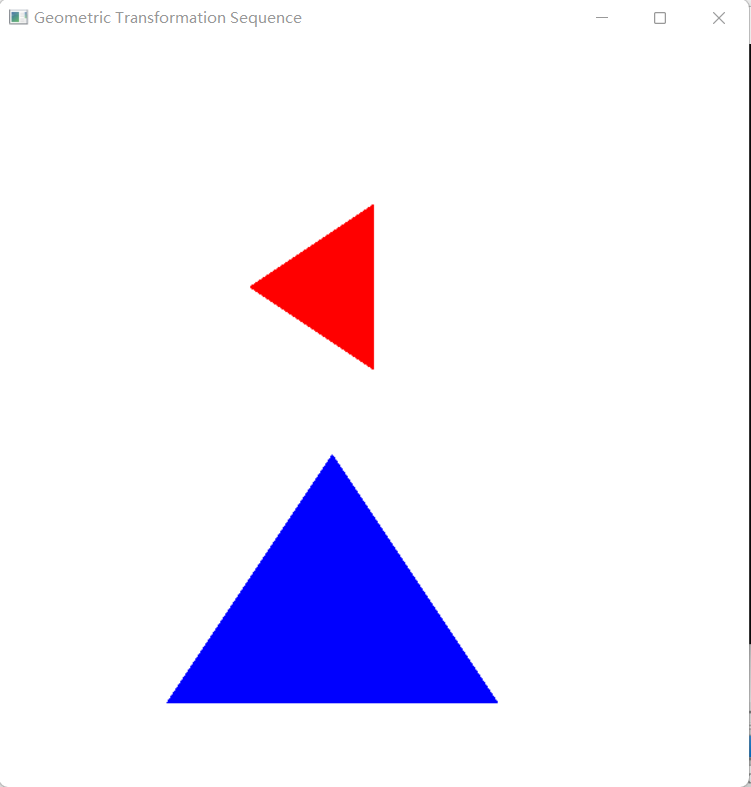
GLfloat tx = 0, ty = 100;

GLfloat sx = 0.5, sy = 0.5;

GLdouble theta = pi / 2;

glClear(GL\_COLOR\_BUFFER\_BIT);

4、实验结果：



实验七 GLUT鼠标函数实验和反走样技术

时间：2022.4.27

地点：信息学院2202

1、实验内容：

（1）教材P458，GLUT鼠标函数。

（2）使用opengl,实现任一反走样技术。

2、实验目的：

调用鼠标函数完成相应功能,2-3个程序。

3、实验代码：

//鼠标函数

GLsizei winWidth=400,winHeight=300;

void init (void){

glClearColor (1.0,1.0,1.0,1.0);

glMatrixMode (GL\_PROJECTION);

gluOrtho2D (0.0,200.0,0.0,150.0);

}

void displayFcn (void){

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.0);

glPointSize (3.0);

}

void winReshapeFcn (GLint newWidth,GLint newHeight){

glViewport (0,0,newWidth,newHeight);

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

gluOrtho2D(0.0,GLdouble (newWidth),0.0,GLdouble (newHeight));

winWidth=newWidth;

winHeight=newHeight;

}

void plotPoint (GLint x,GLint y){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

void mousePtP1ot (GLint button,GLint action, GLint xMouse,GLint yMouse){

if (button==GLUT\_LEFT\_BUTTON && action==GLUT\_DOWN)

plotPoint(xMouse, winHeight- yMouse);

glFlush();

}

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

glutInit (&argc, argv);

glutInitDisplayMode (GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowPosition(100,100);

glutInitWindowSize(winWidth,winHeight);

glutCreateWindow("Mouse Plot Points");

init( );

glutDisplayFunc(displayFcn);

glutReshapeFunc(winReshapeFcn);

glutMouseFunc(mousePtP1ot);

glutMainLoop();

return 0;

}

//反走样

#include <windows.h>

#include <gl/glut.h>

GLuint lineList; //指定显示列表ID

void Initial()

{

glClearColor(1.0f, 1.0f, 1.0f, 0.0f);//设置窗口背景颜色（清空当前颜色缓冲）

glLineWidth(12.0f); //设置线的宽度

glColor4f (0.0, 0.0, 1.0, 1.0); //设置为颜色

lineList = glGenLists(1); //创建显示列表ID

glNewList(lineList, GL\_COMPILE); //定义显示列表

glBegin(GL\_LINE\_LOOP); //绘制线段

glVertex2f(1,1); //第一个点

glVertex2f(4,3); //第二个点

glEnd(); //绘制结束

glEndList(); //显示列表结束

}

void ChangeSize(GLsizei w, GLsizei h) //typedef int GLsizei

{

if(h == 0) h = 1;

glViewport(0, 0, w, h); //指定窗口位置

glMatrixMode(GL\_PROJECTION); //接下来进行投影操作

glLoadIdentity(); //重置当前指定的矩阵为单位矩阵.

if(w<=h)

gluOrtho2D(0.0, 5.0, 0.0, 6.0\*(GLfloat)h/(GLfloat)w);//更改模型坐标范围

else

gluOrtho2D(0.0, 5.0\*(GLfloat)w/(GLfloat)h, 0.0, 6.0);//更改模型坐标范围

glMatrixMode(GL\_MODELVIEW); //模型视景的操作

glLoadIdentity(); //重置当前指定的矩阵为单位矩阵

}

void Displayt(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);//用当前背景色填充窗口

glCallList(lineList); //函数执行显示列表

glFlush(); //强制刷新缓冲，保证绘图命令将被执行

}

void Displayw(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT); //用当前背景色填充窗口

glEnable(GL\_LINE\_SMOOTH); //使用反走样

glEnable (GL\_BLEND); //启用混合函数

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA); //指定混合函数

glCallList(lineList); //函数执行显示列表

glFlush(); //强制刷新缓冲，保证绘图命令将被执行

}

int main(void)

{

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //使用双缓存及RGB模型

glutInitWindowSize(300, 300); //窗口大小

glutCreateWindow("原始图形"); //窗口名称

glutDisplayFunc(Displayt); //执行函数

glutReshapeFunc(ChangeSize);

Initial();

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //使用双缓存及RGB模型

glutInitWindowPosition(300, 300);

glutInitWindowSize(300, 300);

glutCreateWindow("反走样图形");

glutDisplayFunc(Displayw);

glutReshapeFunc(ChangeSize);

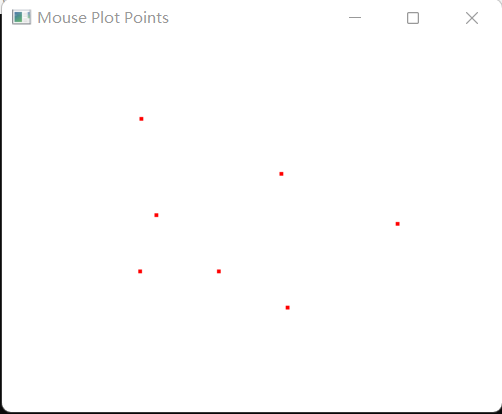
Initial();

glutMainLoop();

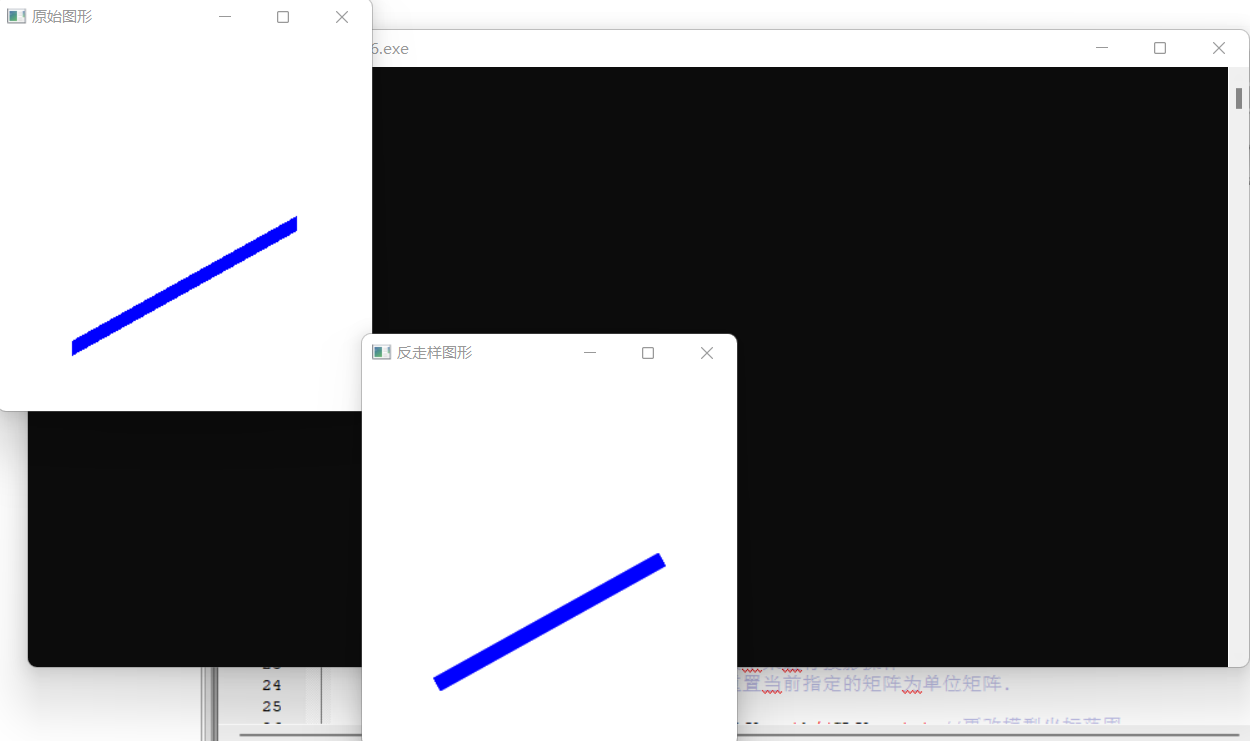
return 0;

}

4、实验结果：

鼠标函数（每次按下鼠标左键时，就在鼠标光标所在位置画一个红点）：

反走样技术：



实验八 二维图像裁剪实验

时间：2022.5.4

地点：信息学院2202

1、实验内容：

使用opengl，用Cohen-Sutherland线段裁剪算法对直线段进行裁剪。

2、实验目的：

验证Cohen-Sutherland裁剪算法,从键盘输入任意的直线段,用指定的裁剪窗口裁剪直线段。

3、实验代码：

#include <windows.h>

#include <GL/glut.h>

#define LEFT 1

#define RIGHT 2

#define BOTTOM 4

#define TOP 8

int x1 = 50, y1 = 150, x2 = 200, y2 = 250, XL = 100, XR = 300, YB = 100, YT = 200;

//(x1,y1)、(x2,y2)为直线段的端点，XL为左边界，XR为右边界，YB为下边界，YT为上边界

int x1\_init =x1, y1\_init =y1, x2\_init =x2, y2\_init =y2; //将直线段端点备份，以便画出裁剪前的直线段

char encode(float x, float y){

char c = 0;

if (x < XL) c |= LEFT;

if (x > XR) c |= RIGHT;

if (y < YB) c |= BOTTOM;

if (y > YT) c |= TOP;

return c;

}

void CS\_LineClip(int &x1, int &y1, int &x2, int &y2, int XL, int XR, int YB, int YT) {

char code1 = encode(x1, y1);

char code2 = encode(x2, y2);

char code;

int x, y;

while (code1 || code2) {

if ((code1 & code2) != 0)return; //在外同侧

if (code1 != 0) code = code1;

else code = code2;

if ((LEFT & code) != 0) {

x = XL; y = y1 + (y2 - y1) \* (XL - x1) / (x2 - x1);

}

else if ((RIGHT & code) != 0) {

x = XR; y = y1 + (y2 - y1) \* (XR - x1) / (x2 - x1);

}

else if ((BOTTOM & code) != 0) {

y = YB; x = x1 + (x2 - x1) \* (YB - y1) / (y2 - y1);

}

else if ((TOP & code) != 0) {

y = YT; x = x1 + (x2 - x1) \* (YT - y1) / (y2 - y1);

}

if (code == code1) {

x1 = x; y1 = y; code1 = encode(x, y);

}

else { x2 = x; y2 = y; code2 = encode(x, y); }

}

}

void init(void){

glClearColor(1.0, 1.0, 1.0, 0.0); // 设置背景颜色

glMatrixMode(GL\_PROJECTION); // 设置投影参数

gluOrtho2D(0.0, 600.0, 0.0, 400.0); // 设置场景的大小

}

void draw(void){

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 1.0, 0.0);// 设置画图颜色

glLineWidth(2);// 设置边框宽度

glPointSize(2);

glPushMatrix();

glBegin(GL\_LINE\_LOOP);

glVertex2i(XL, YT);

glVertex2i(XL, YB);

glVertex2i(XR, YB);

glVertex2i(XR, YT);

glEnd();

//绘制未裁剪前的线段

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2i(x1\_init, y1\_init);

glVertex2i(x2\_init, y2\_init);

glEnd();

//裁剪线段

CS\_LineClip(x1, y1, x2, y2, XL, XR, YB, YT);

//绘制裁剪后的线段

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2i(x1, y1);

glVertex2i(x2, y2);

glEnd();

glPopMatrix();

glFlush();// 处理绘图pipeLine

}

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

glutInit(&argc, argv);// 初始化GLUT环境

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);// GLUT显示模式：单缓冲区、RGB颜色模型

glutInitWindowSize(800, 600);// 显示窗口的大小

glutCreateWindow("Cohen-Sutherland裁剪算法");//创建显示窗口，加上标题

init();

glutDisplayFunc(draw);// 调用绘图函数

glutMainLoop();// 进入事件处理循环

return 0;

}

4、实验结果：

