计算机图形学

封闭多边形影线填充实验报告

庞一统 1552286 计算机二班

2017

## 实验要求与目的

实现任意封闭多边形影线填充。所谓影线是指，一组等间距的平行斜线，间距和斜线倾角自定义。工程设计图中剖面图即为此种影线填充。使用OpenGL和C++编程，画线程序自行设计，并测试一些典型的例子。

## 实验环境

IDE :Visual Studio Community2015版

编程语言：C++

编程依赖包：OpenGL glut

硬件： CPU Intel Core i7-5500U @2.4GHz 内存：4GB

## 数据结构设计

边结构

typedef struct Edge

{

float x; /\* 当前扫描线与边的交点的x值 \*/

float dx; /\* 从当前扫描线到下一扫描线之间的x的增量 1/m\*/

int dmax; /\* 点到y=kx/y=kx+b的距离 \*/

struct Edge \*next;

}Edge;

点结构

typedef struct POINT

{

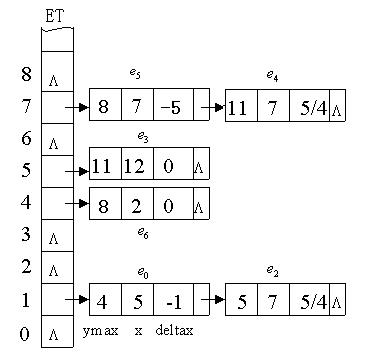
int x;

int y;

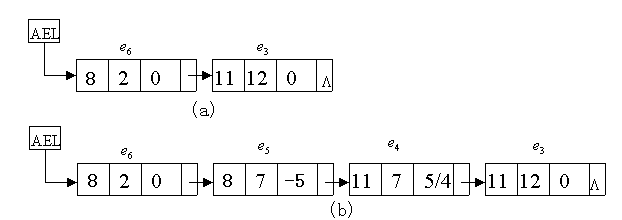
int d;

} POINT;

有序边表（ET）:Edge[MAX] 示意图



活性边表（AET）示意图

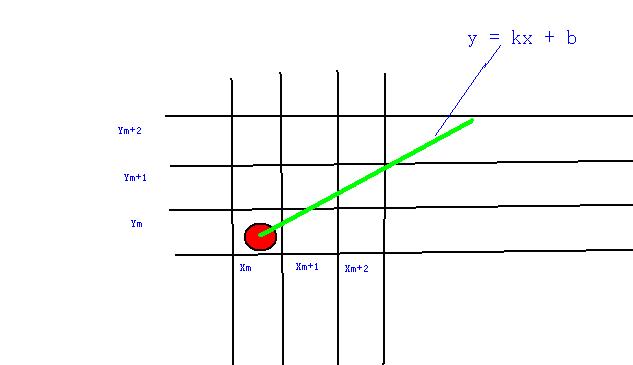


## 算法设计

1. **画直线算法**

Bresenham算法是DDA算法画线算法的一种改进算法。本质上它也是采取了步进的思想。不过它比DDA算法作了优化，避免了步进时浮点数运算，同时为选取符合直线方程的点提供了一个好思路。首先通过直线的斜率确定了在x方向进行单位步进还是y方向进行单位步进：当斜率k的绝对值|k|<1时，在x方向进行单位步进；当斜率k的绝对值|k|>1时，在y方向进行单位步进。

下面以|k|<1时推导Bresenham算法的数学依据：



Dupper = Ym+1 - Yreal = Ym+1 - k\*(Xm+1)+b）; 表示Ym+1和方程真实值的差

Ddown = Yreal - Ym = k\*(Xm+1)+b）- Ym; 表示Ym和方程真实值的差

那就是我们要比较Dupper和Ddown的大小。假设

**Diff = Dupper - Ddown = （Ym+1 - k\*(Xm+1)+b）） - （k\*(Xm+1)+b）- Ym）**

令△X 为线段x方向的间距，△Y 为线段y方向的间距。

Pm = △X\* Diff = 2\*△X\* Ym-2\*△Y\* Xm-2\*△Y-△X\*（2b-1）；

那么Pm+1 = Pm+2\*△X\*（Ym+1- Ym）-2\*△Y；其中Ym+1- Ym取0还是1，取决于Pm的符号。

根据等式Diff = Dupper - Ddown = （Ym+1 - k\*(Xm+1)+b）） - （k\*(Xm+1)+b）- Ym）以及k = △Y/△X，我们可以得出起始像素（x0,y0）的参数p0的值：

P0 =△X-2\*△Y；

同理我们推出|k|>1的情况，Qm = 2\*Xm\*△Y-2\*Ym\*△X+（2b-2）\*△X+△Y；

Qm+1 = Qm+2\*（Xm+1-Xm）\*△Y-2\*△X；

其中Xm+1-Xm等于0还是1，取决于Qm的符号

其中第一个参数Q0 = △Y-2\*△X；

**算法具体步骤即为：**

1. 输入线段的起点和终点。

2. 判断线段的斜率是否存在（即起点和终点的x坐标是否相同），若相同，即斜率不存在，只需计算y方向的单位步进（△Y+1次），x方向的坐标保持不变即可绘制直线。

3. 计算线段的斜率k，分为下面几种情况处理

a. k等于0，即线段平行于x轴，即程序只需计算x方向的单位步进，y方向的值不变

b. |k|等于1，即线段的x方向的单位步进和y方向的单位步进一样，皆为1。直接循环△X次计算x和y坐标。

4. 根据输入的起点和终点的x、y坐标值的大小决定x方向和y方向的单位步进是1还是-1

5. 画出第一个点。

6. 若|k| <1，设m =0,计算P0，如果Pm>0，下一个要绘制的点为（Xm+单位步进，Ym），

Pm+1 = Pm -2\*△Y;

否则要绘制的点为（Xm+单位步进，Ym+单位步进）

Pm+1 = Pm+2\*△X-2\*△Y；

7. 重复执行第七步△X-1次；

8. 若|k| >1，设m =0,计算Q0，如果Qm>0，下一个要绘制的点为（Xm，Ym+单位步进），

Pm+1 = Pm -2\*△X;

否则要绘制的点为（Xm+单位步进，Ym+单位步进）

Pm+1 = Pm+2\*△Y-2\*△X；

9. 重复执行第9步△Y-1次；

1. **影线扫描算法**

扫描线填充算法的基本思想是：用水平扫描线从上到下（或从下到上）扫描由多条首尾相连的线段构成的多边形，每根扫描线与多边形的某些边产生一系列交点。将这些交点按照x坐标排序，将排序后的点两两成对，作为线段的两个端点，以所填的颜色画水平直线。多边形被扫描完毕后，颜色填充也就完成了。扫描线填充算法也可以归纳为以下4个步骤：

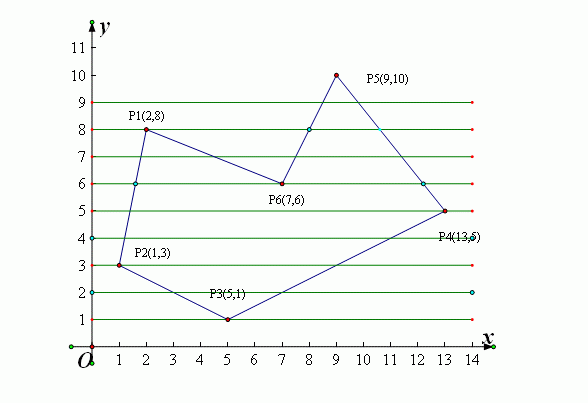
（1）求交点。即扫描线和多边形的交点。

（2）交点排序。

（3）对排序后的点两两匹配。

（4）更新扫描线，判断是否完成多边形扫描。

算法的关键是第一步，求交点 。



为了减少计算量，扫描线算法需要维护一张“活动边Active Edge组成的表，称为“活化边表Active Edge Table（AET）”。如上文所提。

假设当前扫描线与多边形的某一条边的交点已经通过直线段求交算法计算出来，得到交点的坐标为（x, y），则下一条扫描线与这条边的交点不需要再求交计算，通过步进关系可以直接得到新交点坐标为（x + △x, y + 1）。△x为y增加1个单位时，x增加的单位。即多边形边界的斜率的倒数。

言而总之，“活动边表”是扫描线填充算法的核心，保证了填充时不需太多的求交点运算。为了方便活性边表的建立与更新，还需要建立一个边表ET（New Edge Table）来存放所有的边.

具体算法步骤如下：

1、建立ET；

2、将扫描线纵坐标y的初值置为ET中非空

元素的最小序号，如图中，y=1；

3、置AEL为空；

4、执行下列步骤直至ET和AEL都为空．

4.1、如ET中的第y类非空，则将其中的所有

边取出并插入AEL中；

4.2、如果有新边插入AEL，则对AEL中各边排序；

4.3、对AEL中的边两两配对，（1和2为一对，3和4为一对，…），将每对边中x坐标按规则取整，获得有效的填充区段，再填充．

4.4、将当前扫描线纵坐标 y 值递值1；

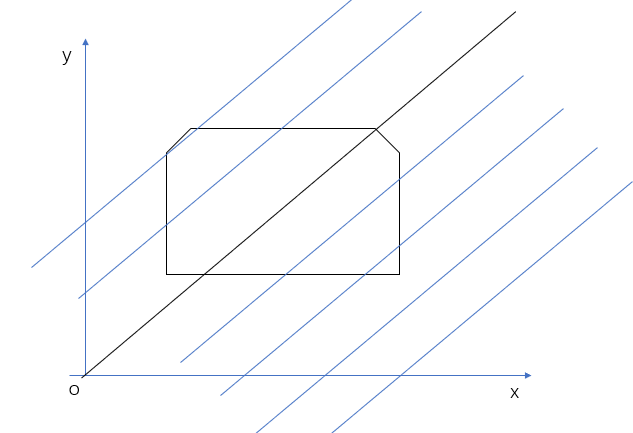
4.5、将AEL中满足y = ymax边删去（因为每条边被看作下闭上开的）；

4.6、对AEL中剩下的每一条边的x递增deltax，即x = x+deltax．

注意事项：

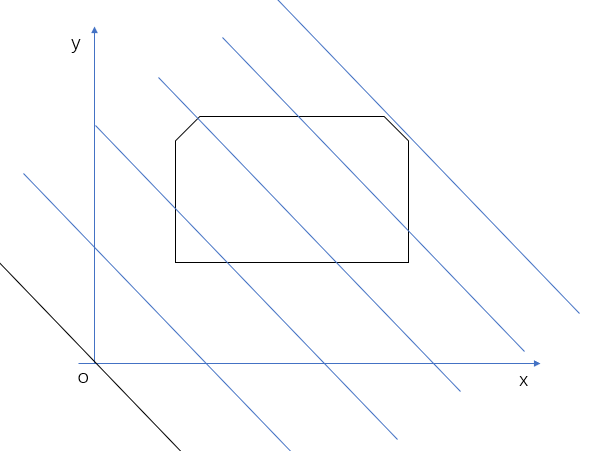
由于在本实验中，扫描线的斜率可任意输入，在建立边表时，利用点到直线的距离公式来进行计算，得到边的两点到参考扫描线的距离y=kx，完成建表；

扫描线族的范围则在y=kx+dmin\* 到y=kx+(dmax-1)\*之间，同时采取左闭右开，下闭上开的填充原则。其中，k为扫描线斜率，d为多边形某边端点到扫描线y=kx的距离。



当k>0时，由于坐标系的关系，存在y=kx穿过多边形的情况，此时再计算dmax,建表结果将正确，需要使得多边形在扫描参考线的同一侧；这种情况下，我们需要先计算出各端点到y=kx的最大值，即最大距离，然后对y=kx进行平移变换，变成y=kx-d\*

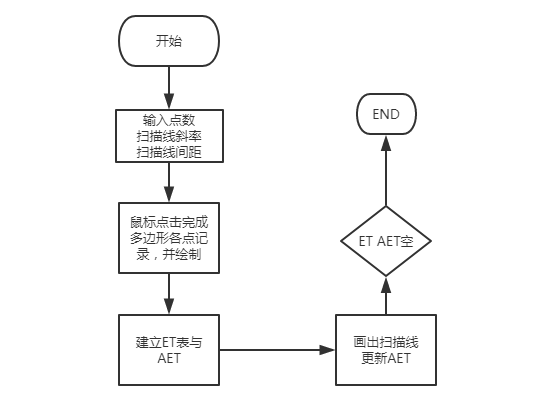
之后，以变换后的参考线进行d计算,完成计算；

  
当k<0时如上图所示，无特殊情况，正常计算即可；

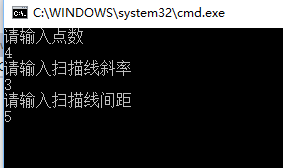
简言之，在计算过程中，当扫描线为非平行于坐标轴的直线时，即要完成坐标系的变换，变为相对于y=kx的坐标系；

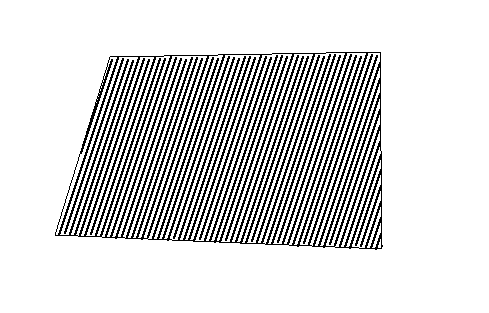
在填充扫描线时，根据直线方程y=kx+ds\* ，计算出y，再利用直线算法，画出扫描线即可；为了实现等间距填充，只需在程序中加入计数器，每隔一定数量再画出扫描线。

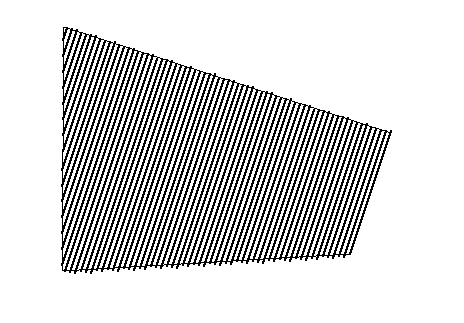
## 程序逻辑流程

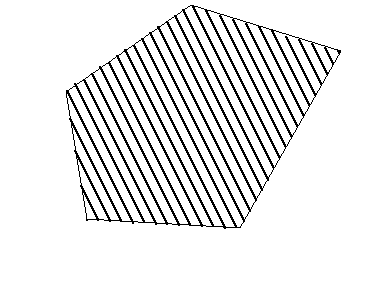


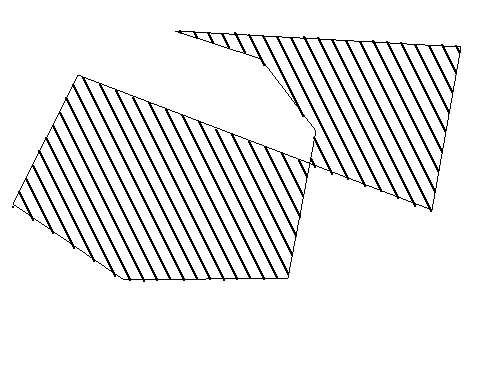
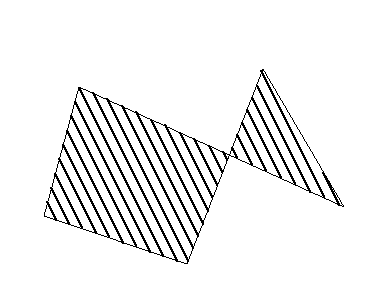
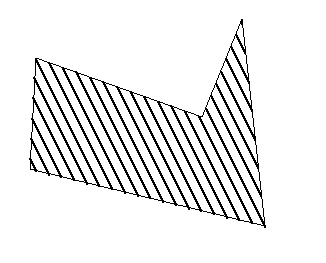
## 程序截图

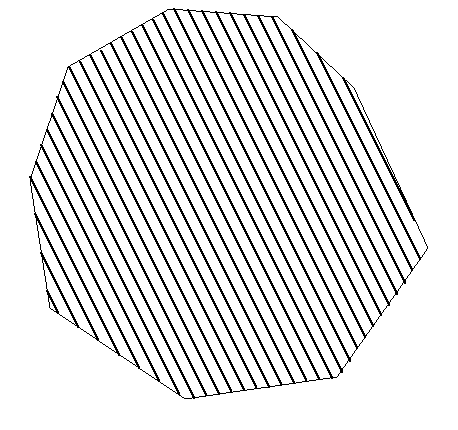












## 实验总结

本次实验实现了封闭多边形影线填充的基本功能，并能够自定义影线斜率与间距。

通过实验，实践了直线算法与多边形扫描填充算法，对于算法有了较深的理解；同时，也结合实际实验，学习了OpenGL的编程应用。

## 代码附录

#pragma once

#define WINDOW\_HEIGHT 1000

#include<iostream>

#include<math.h>

using namespace std;

#define NULL 0

typedef struct Edge

{

float x; /\* 当前扫描线与边的交点的x值 \*/

float dx; /\* 从当前扫描线到下一扫描线之间的x的增量 \*/

int dmax; /\* 计算点到y=kx的距离 \*/

struct Edge \*next;

}Edge;

typedef struct POINT

{

int x;

int y;

int d;

} POINT;

/\*

直线Bresonham算法

\*/

void Bre\_Line(int x0, int y0, int x1, int y1)

{

int x, y, dx, dy, e;

int i, temp;

int change = 0;

int Step\_X = 1, Step\_Y = 1;

dx = x1 - x0, dy = y1 - y0;

glBegin(GL\_POINTS);

if (dx<0) {

dx = -dx;

Step\_X = -1;

}

if (dy<0) {

dy = -dy;

Step\_Y = -1;

}

if (dy>dx) {

temp = dx;

dx = dy;

dy = temp;

change = 1;

}

e = -dx, x = x0, y = y0;

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

{

//画点

glVertex3i(x, y, 0);

if (change) y = y + Step\_Y;

else x = x + Step\_X;

e = e + 2 \* dy;

if (e >= 0)

{

if (change) x = x + Step\_X;

else y = y + Step\_Y;

e = e - 2 \* dx;

}

}

glEnd();

}

/\*

计算所有点中到y=kx距离的最大值

\*/

int MaxD(POINT \*pts, int cnt)

{

float max = pts[0].d;

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

{

if (max<pts[i].d)

max = pts[i].d;

}

return max;

}

/\*

计算每一点到参考线y=kx+-b的间距

利用点到直线的距离公式

\*/

int CalcutaleD(int k, POINT \*pts, int cnt,bool dir)

{

double n = 1 + k\*k;

float m = sqrt(n);

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

{

pts[i].d = abs((pts[i].y) - k\*(pts[i].x)) / m;

}

if (dir)

{

int max = MaxD(pts, cnt);

int lens = max\*sqrt(float(1 + k\*k));

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

pts[i].d = abs((pts[i].y) - k\*(pts[i].x)+lens) / m;

return lens;

}

return 0;

//return lens;

}

/\*\*\*\*\*

计算所有点到y=kx距离的最小值

int MinD(POINT \*pts, int cnt)

{

float min = pts[0].d;

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

{

if (min>pts[i].d)

min = pts[i].d;

}

return min;

}

\*\*\*\*\*/

/\*

按交点的x升序对边进行插入排序

用于建立边表

\*/

void InsertEdge(Edge \*list, Edge \*edge)

{

Edge \*p, \*q = list;

p = q->next;

while (p)

{

if (edge->x<p->x)

p = NULL;

else

{

q = p;

p = p->next;

}

}

edge->next = q->next;

q->next = edge;

}

/\*

计算下一条非水平线的d值

到参考扫描线的距离

\*/

int dNext(int k, int cnt, POINT \*pts)

{

int j;

if ((k + 1)>(cnt - 1))

j = 0;

else

j = k + 1;

while (pts[k].d == pts[j].d)

{

if ((j + 1)>(cnt - 1))

j = 0;

else

j++;

}

return(pts[j].d);

}

/\*

生成有序边表的一条边

lower（低点） upper（高点）

构建边表

\*/

void MakeEdgeRec(POINT lower, POINT upper, int dComp, Edge \*edge, Edge \*edges[],float k)

{

float k1;

//float k =0;

int db = 1;

k1 = (float)(upper.y - lower.y) / (float)(upper.x - lower.x);

cout << upper.y << endl << endl;

k1 = k1 - k;

edge->dx =k1? (db)\*sqrt(1 + k\*k) / k1 :0;

edge->x = lower.x;

if (upper.d < dComp)

edge->dmax = upper.d - 1;//该点既不是局部高点也不是局部低点

else

edge->dmax = upper.d;//局部高点

InsertEdge(edges[lower.d], edge);

}

/\*

建立有序边表

\*/

void BuildEdgeList(int cnt, POINT \*pts, Edge \*edges[],float k)

{

Edge \*edge;

POINT v1, v2;

int i, dPrev = pts[cnt - 2].d;

v1 = pts[cnt - 1];

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

{

v2 = pts[i];

if (v1.d != v2.d)

{

edge = (Edge \*)malloc(sizeof(Edge));

if ((v1.d < v2.d))

MakeEdgeRec(v1, v2, dNext(i, cnt, pts), edge, edges,k); //高点的下一个点

else

MakeEdgeRec(v2, v1, dPrev, edge, edges,k);

}

dPrev = v1.d;

v1 = v2;

}

}

/\*

建立活性边表

scan 对应有序边表扫描线

\*/

void BuildActiveList(int scan, Edge \* active, Edge \*edges[])

{

Edge \*p, \*q;

p = edges[scan]->next;

while (p)

{

q = p->next;

InsertEdge(active, p);

p = q;

}

}

/\*

对扫描线填充

计算各点y坐标

利用直线算法计算

\*/

void FillScan(int scan, Edge \* active,int k,bool dir, int lens)

{

Edge \*p1, \*p2;

p1 = active->next;

while (p1)

{

p2 = p1->next;

int x1 = p1->x;

int x2 = p2->x;

int y1, y2;

if (dir)

{

y1 = k \* x1 + scan\*sqrt((float)(1 + k\*k)) - lens;

y2 = k \* x2 + scan\*sqrt((float)(1 + k\*k)) - lens;

}

else

{

y1 = k \* x1 + scan\*sqrt((float)(1 + k\*k));

y2 = k \* x2 + scan\*sqrt((float)(1 + k\*k));

}

//cout<<x1<<" "<<x2<<" "<<y1<<" "<<y2<<endl;

Bre\_Line(x1, y1, x2, y2);

p1 = p2->next;

}

}

/\*

删除不再相交的边

\*/

void DeleteAfter(Edge \*q)

{

Edge \*p = q->next;

q->next = p->next;

free(p);

}

/\*、

更新下一条扫描线

\*/

void UpdateActiveList(int scan, Edge \*active)

{

Edge \*q = active, \*p = active->next;

while (p)

if (scan >= p->dmax)

{

p = p->next;

DeleteAfter(q);

}

else

{

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

q = p;

p = p->next;

}

}

int db = 0;

/\*

活性边表重新排序

\*/

void ResortActiveList(Edge \*active)

{

Edge \*q;

Edge \*p = active->next;

active->next = NULL;

while (p)

{

q = p->next;

InsertEdge(active, p);

p = q;

}

}

/\*

对图形进行扫描填充

\*/

void ScanFill(int cnt, POINT \*pts,int k,int \_db)

{

bool dir;

//int k=1;

dir = k > 0 ? true : false;

int lens=CalcutaleD(k, pts, cnt,dir);

Edge \*edges[WINDOW\_HEIGHT], \*active;

int i, scan;

for (i = 0; i<WINDOW\_HEIGHT; i++)

{

edges[i] = (Edge \*)malloc(sizeof(Edge));

edges[i]->next = NULL;

}

BuildEdgeList(cnt, pts, edges,k);

/\*for(int i=0;i<WINDOW\_HEIGHT;i++)

{p=edges[i]->next;

while(p!=NULL)

{

p=p->next;

}}\*/

active = (Edge \*)malloc(sizeof(Edge));

active->next = NULL;

for (scan = 0; scan<WINDOW\_HEIGHT; scan++)

{

BuildActiveList(scan, active, edges);

if (active->next)

{

if (db % \_db == 0)//每4个间距画一条线。

{

FillScan(scan, active,k,dir,lens);

}

UpdateActiveList(scan, active);

ResortActiveList(active);

}

db++;

}

}

#include <GL/glut.h>

#include"draw.h"

#include<iostream>

using namespace std;

int k;

int dr;

int pos;

POINT \*p;

bool select = false;//图形点输入完成标志

/\*

OpenGL初始化

\*/

void myInit(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0f, 0.0f, 0.0f);

glPointSize(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 640.0, 0.0, 480.0);

}

/\*

图形扫描与显示

\*/

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

if (select == true)

{

//多边形各点

glBegin(GL\_LINE\_LOOP);

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

glVertex2i(p[i].x, p[i].y);

glEnd();

//扫描填充

ScanFill(k, p,dr,pos);

}

//双缓存技术

glutSwapBuffers();

glFlush();

}

/\*

鼠标控制部分

点击来记录点

\*/

void myMouse(int button, int state, int x, int y)

{

static int number = 0;

if (state == GLUT\_DOWN)

{

if (button == GLUT\_LEFT\_BUTTON)

{

p[number].x = x;

p[number].y = 480 - y;

glBegin(GL\_POINTS);

glVertex2i(x, 480 - y);

glEnd();

glFlush();

cout << p[number].x << " " << p[number].y << endl;

number++;

if (number == k)

{

number = 0;

select = true;

glutPostRedisplay();

}

}

else if (button == GLUT\_RIGHT\_BUTTON)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

}

}

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

{

cout << "请输入点数" << endl;

cin >> k;

cout << "请输入扫描线斜率" << endl;

cin >> dr;

cout << "请输入扫描线间距" << endl;

cin >> pos;

p = (POINT\*)malloc(k\*sizeof(POINT));

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(640, 480);

glutCreateWindow("1552286-影线填充实验");

glutMouseFunc(myMouse);

glutDisplayFunc(myDisplay);

myInit();

glutMainLoop();

return 0;

}