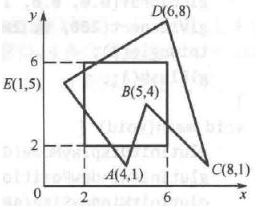
第三次上机报告

1. 实验目的

试用Sutherland-Hodgeman多边形裁剪算法对如图所示的多边形进行裁剪，要求画出每次裁剪对应的图形，并标明输入和输出的顶点。用自己编写的Sutherland-Hodgeman多边形裁剪程序予以验证。



1. 实验原理

1.输入顶点ABCDE

2.使用窗口左边界进行剪裁，与边DE和边AE交于点v1、v2，将v1、v2插入输出链表，检测到点E在左边界左侧，从链表中删除该点，得到输出链表 ABCDv1v2

3.输入链表ABCDv1v2，使用窗口下边界进行剪裁，与边EA和边AB交于点v3、v4，将v3、v4插入输出链表，检测到点A在下边界下侧，从链表中 删除该点，得到输出链表v3v4BCDv1v2

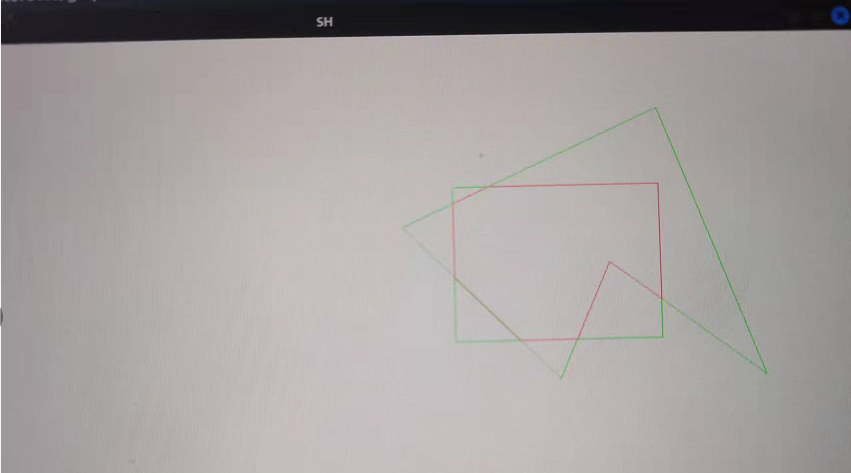
4.输入链表v3v4BCDv1v2，使用窗口右边界进行剪裁，与边BC交于点v5，将v5插入输出链表v3v4Bv5CDv1v2，检测到点C在右边界右侧，从链表中删除该店，得到输出链表v3v4Bv5Dv1v2，检测到右边界上只有一个交点，将窗口右上角顶点加入输出链表，得到输出链表v3v4Bv5RTDv1v2

5.输入链表v3v4Bv5RTDv1v2，使用窗口上边界进行剪裁，与边DE交于点v6，将v6插入输出链表v3v4Bv5RTDv6v1v2，检测到点D在上边界上侧，从链表中删除该店得到输出链表v3v4Bv5RTv6v1v2

6.使用openGL画线函数依次根据链表中的节点画线，得到剪裁后的图形

1. 实验步骤
2. 定义结构体structed，结构体中定义了点point，并且每个point储存四个值，分别为点的横纵坐标以及指向下一个和上一个点的指针nxt和lst
3. 在display函数中定义图中所有的点，包括给点的横纵坐标赋值以及确定点在链表中的数据结构
4. 定义初始化窗口以及初始化图形函数，其中使用到画线函数。
5. 定义确定交点函数bool interaction，求出窗口边界与图形的交点。
6. 定义判断函数bool ifinside，用于判断图形上的点是否在指定边的一侧
7. 定义SH函数，用于结合ifinside函数和interaction函数使用四条窗口边界进行剪裁，该函数的一个参数是链表的头节点，用于输入链表，每次运行完该函数回得到一个输出链表，在使用递归将输出链表的头节点赋给该函数进行下一次剪裁，并得到新的输出链表
8. 运行完递归函数SH后得到最终的链表，通过画线函数将该链表中的点依次连接画成直线，最后得到剪裁后的图形

四、实验结果分析



结果正确

五、源代码（由于使用VIM编辑，不好打注释）

#include<iostream>

#include <GL/glut.h>

#include <cmath>

using namespace std;

// 初始化OpenGL场景

struct point{

float x;

float y;

//int key;

//int key;

//int out;

point\*nxt;

point\*lst;

};

void window(){

glBegin(GL\_LINE\_STRIP);

glVertex2i(200,200);

glVertex2i(600,200);

glVertex2i(600,600);

glVertex2i(200,600);

glVertex2i(200,200);

glEnd();

}

void image(point\*A,point\*B,point\*C,point\*D,point\*E)

{

glBegin(GL\_LINE\_STRIP);

glVertex2i(400, 100);

glVertex2i(500, 400);

glVertex2i(C->x, C->y);

glVertex2i(D->x, D->y);

glVertex2i(E->x, E->y);

glVertex2i(A->x, A->y);

glEnd();

}

bool intersection(point\*a, point\*b, point\*c, point\*d, point\*i){ //line ab and bc interacted at point i

//i=new point();

if(c->x==200&&c->y==600){

i->x=200;

i->y=(b->y-a->y)/(b->x-a->x)\*200+a->y-(b->y-a->y)/(b->x-a->x)\*a->x;

cout<<"lt"<<endl;

//cout<<"i "<<i->x<<" "<<i->y<<endl;

}

if(c->x==600&&c->y==200){

i->x=600;

i->y=(b->y-a->y)/(b->x-a->x)\*600+a->y-(b->y-a->y)/(b->x-a->x)\*a->x;

cout<<"rb"<<endl;

//cout<<"i "<<i->x<<" "<<i->y<<endl;

}

if(c->x==200&&c->y==200){

i->y=200;

i->x=(200-a->y+(b->y-a->y)/(b->x-a->x)\*a->x)\*(b->x-a->x)/(b->y-a->y);

//cout<<"i "<<i->x<<" "<<i->y<<endl;

cout<<"lb"<<endl;

}

if(c->x==600&&c->y==600){

i->y=600;

i->x=(600-a->y+(b->y-a->y)/(b->x-a->x)\*a->x)\*(b->x-a->x)/(b->y-a->y);

//cout<<"i "<<i->x<<" "<<i->y<<endl;

cout<<"rt"<<endl;

}

if((i->x<c->x&&i->x<d->x)||(i->x>c->x&&i->x>d->x)||(i->y<c->y&&i->y<d->y)||(i->y>c->y&&i->y>d->y)||(i->x<a->x&&i->x<b->x)||(i->x>a->x&&i->x>b->x)||(i->y<a->y&&i->y<b->y)||(i->y>a->y&&i->y>b->y)){

//return false;

delete i;

return false;

}

else{

cout<<"true"<<endl;

cout<<"i:"<<i->x<<" "<<i->y<<endl;

return true;

}

}

bool ifinside(point\*a,point\*E){

if(E->x==200&&E->y==600){

if(a->x>=E->x)return true;

else return false;

}

else if(E->x==200&&E->y==200){

if(a->y>=E->y)return true;

else return false;

}

else if(E->x==600&&E->y==200){

if(a->x<=E->x)return true;

else return false;

}

else if(E->x==600&&E->y==600){

if(a->y<=E->y)return true;

else return false;

}

}

float SH(point\*a,point\*A,point\*B,point\*C,point\*D,point\*E,int key){

key++;

cout<<"start SH"<<endl;

cout<<"key="<<key<<endl;

if(key==5)return 0;

point\*i1;

i1=new point();

if(intersection(A,B,a,a->nxt,i1)==true){

if(A->lst==NULL){

if(A->nxt->nxt!=NULL){

i1->nxt=A->nxt->nxt;

A->nxt->nxt->lst=i1;}

A->nxt->nxt=i1;

i1->lst=A->nxt;

}

else{

if(A->nxt!=NULL){

i1->nxt=A->nxt;

A->nxt->lst=i1;

}

A->nxt=i1;

i1->lst=A;}

cout<<"i1->x:"<<i1->x<<endl;

}

point\*i2;

i2=new point();

if(intersection(B,C,a,a->nxt,i2)==true){

if(B->lst==NULL){

if(B->nxt->nxt!=NULL){

i2->nxt=B->nxt->nxt;

B->nxt->nxt->lst=i2;}

B->nxt->nxt=i2;

i2->lst=B->nxt;

}

else{

if(B->nxt!=NULL){

i2->nxt=B->nxt;

B->nxt->lst=i2;

}

B->nxt=i2;

i2->lst=B;}

cout<<"i2->x:"<<i2->x<<endl;

}

point\*i3;

i3=new point();

if(intersection(C,D,a,a->nxt,i3)==true){

if(C->lst==NULL){

if(C->nxt->nxt!=NULL){

i3->nxt=C->nxt->nxt;

C->nxt->nxt->lst=i3;}

C->nxt->nxt=i3;

i3->lst=C->nxt;

}

else{

if(C->nxt!=NULL){

i3->nxt=C->nxt;

C->nxt->lst=i3;

}

C->nxt=i3;

i3->lst=C;}

cout<<"i3->x:"<<i3->x<<endl;

}

point\*i4;

i4=new point();

if(intersection(D,E,a,a->nxt,i4)==true){

cout<<"i4->x:"<<i4->x<<endl;

if(D->lst==NULL){

if(D->nxt->nxt!=NULL){

i4->nxt=D->nxt->nxt;

D->nxt->nxt->lst=i4;}

D->nxt->nxt=i4;

i4->lst=D->nxt;

}

else{

if(D->nxt!=NULL){

i4->nxt=D->nxt;

D->nxt->lst=i4;

}

D->nxt=i4;

i4->lst=D;}

cout<<"i4->x:"<<(float)i4->x<<endl;

}

point\*i5;

i5=new point();

if(intersection(E,A,a,a->nxt,i5)==true){cout<<"i5->x"<<i5->x<<endl;

if(E->lst==NULL){

if(E->nxt->nxt!=NULL){

i5->nxt=E->nxt->nxt;

E->nxt->nxt->lst=i5;}

E->nxt->nxt=i5;

i5->lst=E->nxt;

}

else{

if(E->nxt!=NULL){

i5->nxt=E->nxt;

E->nxt->lst=i5;

}

E->nxt=i5;

i5->lst=E;}

cout<<"i5->x"<<i5->x<<endl;

}

if(A->lst!=NULL)

if(ifinside(A,a)==false){

A->lst->nxt=A->nxt;

A->nxt->lst=A->lst;

//A=A->nxt

A->lst=NULL;

//A->nxt=NULL;}

}

if(B->lst!=NULL)

if(ifinside(B,a)==false){

B->lst->nxt=B->nxt;

B->nxt->lst=B->lst;

//B=B->nxt

B->lst==NULL;

//B->nxt=NULL;

}

if(C->lst!=NULL)

if(ifinside(C,a)==false){

C->lst->nxt=C->nxt;

C->nxt->lst=C->lst;

//C=C->nxt;

C->lst=NULL;

//C->nxt=NULL;

}

if(D->lst!=NULL)

if(ifinside(D,a)==false){

D->lst->nxt=D->nxt;

D->nxt->lst=D->lst;

//D=D->nxt

D->lst=NULL;

//D->nxt=NULL;

}

if(E->lst!=NULL)

if(ifinside(E,a)==false){

E->lst->nxt=E->nxt;

E->nxt->lst=E->lst;

//E=E->nxt;

E->lst=NULL;

//E->nxt=NULL;

}

SH(a->nxt,A,B,C,D,E,key);

}

void out(/\*point\*A,point\*B,point\*C,point\*D,point\*E,\*/point\*h,point\*rt){

cout<<"start out"<<endl;

/\*point\*p;

p=new point[5];

//point\*h;

//h=head;

h=h->nxt;

int i=0;

/\*while(h!=NULL){

cout<<h->x<<endl;

\*p[i]=h;

i++;

h=h->nxt;

}\*/

/\*cout<<h->nxt->x<<endl;

cout<<h->nxt->nxt->x<<endl;

cout<<h->nxt->nxt->nxt->x<<endl;\*/

glBegin(GL\_LINE\_STRIP);

glVertex2i(h->nxt->x, h->nxt->y);

glVertex2i(h->nxt->nxt->x, h->nxt->nxt->y);

glVertex2i(h->nxt->nxt->nxt->x, h->nxt->nxt->nxt->y);

glVertex2i(rt->x,rt->y);

glVertex2i(h->nxt->nxt->nxt->nxt->x, h->nxt->nxt->nxt->nxt->y);

glVertex2i(h->nxt->nxt->nxt->nxt->nxt->x,h->nxt->nxt->nxt->nxt->nxt->y);

glVertex2i(h->nxt->nxt->nxt->nxt->nxt->nxt->x,h->nxt->nxt->nxt->nxt->nxt->nxt->y);

glVertex2i(h->nxt->nxt->nxt->nxt->nxt->nxt->nxt->x,h->nxt->nxt->nxt->nxt->nxt->nxt->nxt->y);

glVertex2i(h->nxt->x, h->nxt->y);

glEnd();

}

void Initial( ){

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //设置窗口背景颜色为白色

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

gluOrtho2D(-1000.0, 1000.0, -1000, 1000.0); // 投影面上的模型坐标范围

}

// 显示图形

void Display(void){

point\*lb;

point\*lt;

point\*rb;

point\*rt;

lb=new point();

lt=new point();

rb=new point();

rt=new point();

lt->nxt=lb;

lb->nxt=rb;

rb->nxt=rt;

rt->nxt=lt;

lt->x=200;lt->y=600;

lb->x=200;lb->y=200;

rb->x=600;rb->y=200;

rt->x=600;rt->y=600;

int key=0;

point\*head;

point\*A;

point\*B;

point\*C;

point\*D;

point\*E;

head=new point();

A=new point();

B=new point();

C=new point();

D=new point();

E=new point();

head->nxt=A;

A->nxt=B;

B->nxt=C;

C->nxt=D;

D->nxt=E;

E->lst=D;

D->lst=C;

C->lst=B;

B->lst=A;

A->lst=head;

A->x=400;A->y=100;

B->x=500;B->y=400;

C->x=800;C->y=100;

D->x=600;D->y=800;

E->x=100;E->y=500;

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

// 此处需增加调用基本图形生成函数

// 例如:

glColor3f(1.0f, 0.0f, 0.0f); //设置当前的绘图颜色为红色

//structed();

window();

//glFlush();

image(A,B,C,D,E);

SH(lt,A,B,C,D,E,key);

glFlush();

/\*glRectf(50.0f, 100.0f, 150.0f, 50.0f); //绘制一个矩形

glFlush();

int a;

cin>>a;

if(a==1) {

glColor3f(0.0f,1.0f,0.0f);

glRectf(50.0f,100.0f,150.0f,50.0f);

glFlush();

}

glFlush(); //处理所有的OpenGL程序\*/

glColor3f(0.0f,1.0f,0.0f);

out(head,rt);

glFlush();

}

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

glutInit(&argc, argv); // glut初始化

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);//初始窗口显示模式

glutInitWindowSize(1000,750 ); //设置窗口的尺寸

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

glutCreateWindow("SH"); //创建一个窗口

glutDisplayFunc(Display); //设置当前窗口的显示回调函数

Initial(); //完成窗口初始化

glutMainLoop(); //启动主GLUT事件处理循环

return 0;

}