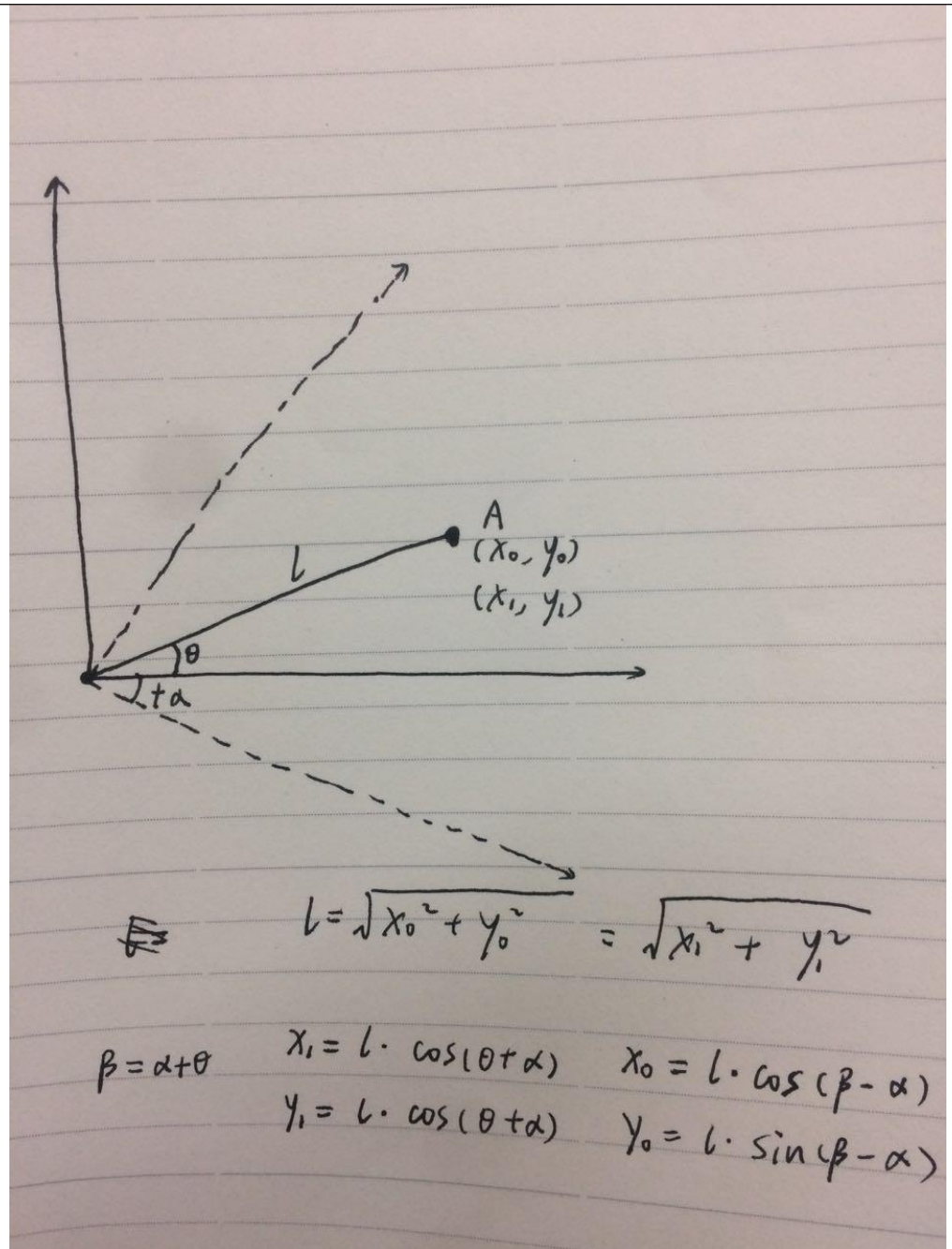


# 任意多边形影线填充设计报告

设计课题	任意多边形影线填充
数据结构设计	<pre>typedef struct Edge {     int ymax;     float deltax;     float x;     struct Edge *next; }Edge,*ptrtoEdge; //ET 结构体的节点结构  typedef struct edge    {     int highy;     int lowy;     int lowx;     float deltax; }edge,*ptrtoedge; //输入相邻两点生成的边的结构  typedef struct POINT {     int x;     int y; } POINT,*ptrtoPOINT; //点  typedef struct EThead {     int ylow;           //线段下端点的 y 值     Edge *next;         //指针指向 Edge 节点 }EThead; //ET 结构体的数组部分  typedef struct temppoint {</pre>

	<pre> double x; double y; }temppoint,*ptrtotemppoint;  //坐标为 double 型的点 </pre>
功能(函数)说明	<pre> temppoint changetonew(double x,double y,double alpha) //将标准坐标转换为倾斜坐标 temppoint changetoori(double x,double y,double alpha) //将倾斜坐标转换为标准坐标 bool isnull(); //功能：判断 AEL 表头是否为空 void createET(EThead head[],temppoint *new_p,int k); //功能：创建 ET 结构体 void print(); //功能：填充影线 void myInit(void) //界面初始化函数 void myDisplay() //界面展示函数 void myMouse(int button, int state,int x,int y) //鼠标 </pre>
程序思想	<p>1、坐标转换：</p> <p>因为本实验要求能够使用任意角度的影线填充，所以要有坐标转换。定义标准坐标系为，以界面左下角为原点，正右方向为 x 正方向；正上方向为 y 正方向。转移坐标系为，以界面左下角为原点，影线向右方向为 x 正方向，垂直影线向上方向为 y 正方向。坐标转移原理如下图：</p>



## 2、扫描线算法：

经过上述变换就得到了图形在以影线方向为 x 轴的坐标系的坐标，接下来选择平行投影线的线为扫描线与图形相交。并且将两端的交点转换至标准坐标系，从而得到一条投影线。这里需要使用两个数据结构来协助完成。ET 结构体和 AEL 结构体，其中 ET 结构体与书上介绍的略有不同。书上是以最低点 y 值为数组序号，但是本程序中，坐标变换后的 y 值往往可能是负值，所以这里的 ET 结构体数组部分定义如下：

```
typedef struct EThead
{
```

```

int ylow;           //线段下端点的 y 值
Edge *next;         //指针指向 Edge 节点
}Ethead;

```

在本程序中，ET 数组是一个“紧致”的数组结构，即：从 Ethead[0]到最后一个元素中间没有为空的情况。ET 后的节点是相同最低点的一系列边的参数。这些节点在插入后会对其进行排序，按照 x 值从小到大排列，在将节点插入 AEL 中时就可以保证 AEL 的顺序排列。

定义 smx 变量，初值为最小 y 值。然后以下循环至 ET 和 AEL 都空

One: 如果 smx==ET 表头的值则将该表头后的链表插入到 AEL。插入的过程为：两个指针 f、r，保持 f 在 r 前一位。

```

If(f 不为空 && f->data > ET 第一个节点 data)
    f=f->next;

```

只需要将 r 指向 ET 链表，ET 链表末端指向 f 即可完成插入操作。将 smx 与图形的交点坐标经过坐标转换后的点坐标送到打印函数打印出该条扫描线。

Two: 如果 smx>该边较高端点的 y 值，则在 AEL 中将该节点删除

Three: 将 AEL 中每个节点的 x 值加上 deltax, smx++

### 3、总体思路:

a、鼠标在界面上点击 k 个点，mymouse 函数记录点在标准坐标系的坐标值。New\_P 数组记录相应点在转换坐标系上的坐标值。

b、edge 结构体数组 pe 记录转换后的 k 个边（与投影线方向平行的边不记录）的情况

```

typedef struct edge
{
    int highy;
    int lowy;
    int lowx;
    float deltax;
}edge;

```

包括：边较高的端点的 y 值；较低端点的 x、y 值；斜率的倒数。

记录方式为：定义一个 count，初值为 0。每进一个点，count++，若 count==2，则将 pe[i]、pe[i+1]生成一条边。最后将 pe[k-1]、pe[0]生成一条边。

### c、建立 ET 结构体:

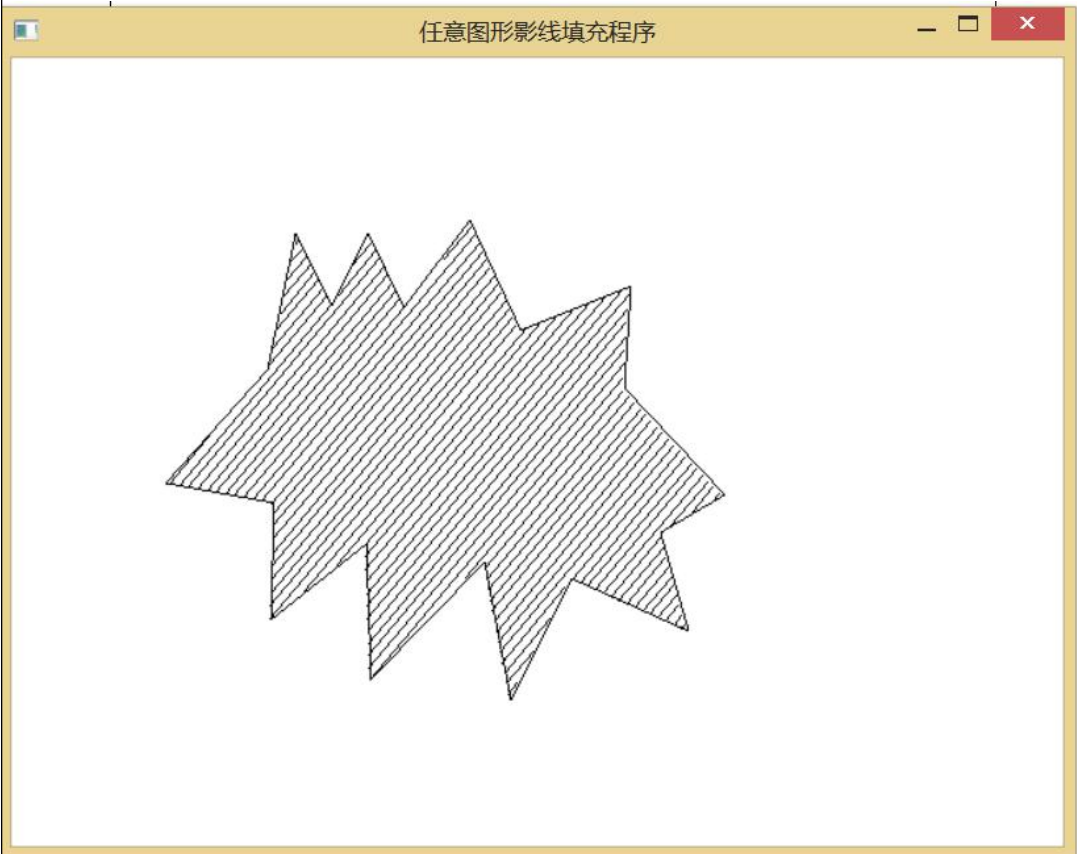
将 pe 数组中的 lowy 按照从小到大不重复排列，并且插入到 ET 的数

组部分 (Ethead), 设为 Ethead[0~n]。然后，重新扫描 pe 数组，若有 pe[i].lowy==Ethead[j]，则将 pe[i]的节点信息插入到 Ethead[j]后面。待

	<p>EThead[n]插入完毕后，对 EThead[0~n]，将其后面的链表按照 x 从小到大排列。</p> <p>d、扫描线算法填充图形 先将原图形打印。再按照 2 中的扫描线算法，结合 ET 结构体和 AEL 链表进行扫描。 只要 ET 和 AEL 不为空就持续进行。</p>
界面设计和使用说明	<p>按照提示进行操作，其中沿 x 轴向下为正（alpha），反之为负（alpha）。</p>
调试分析	<p>Case1:</p> <div data-bbox="359 907 758 1037"><p>请输入点数 30 请输入影线角度 -30</p></div> <p>（30 边形）</p> <div data-bbox="359 1108 1436 1960"><p>任意图形影线填充程序</p></div>

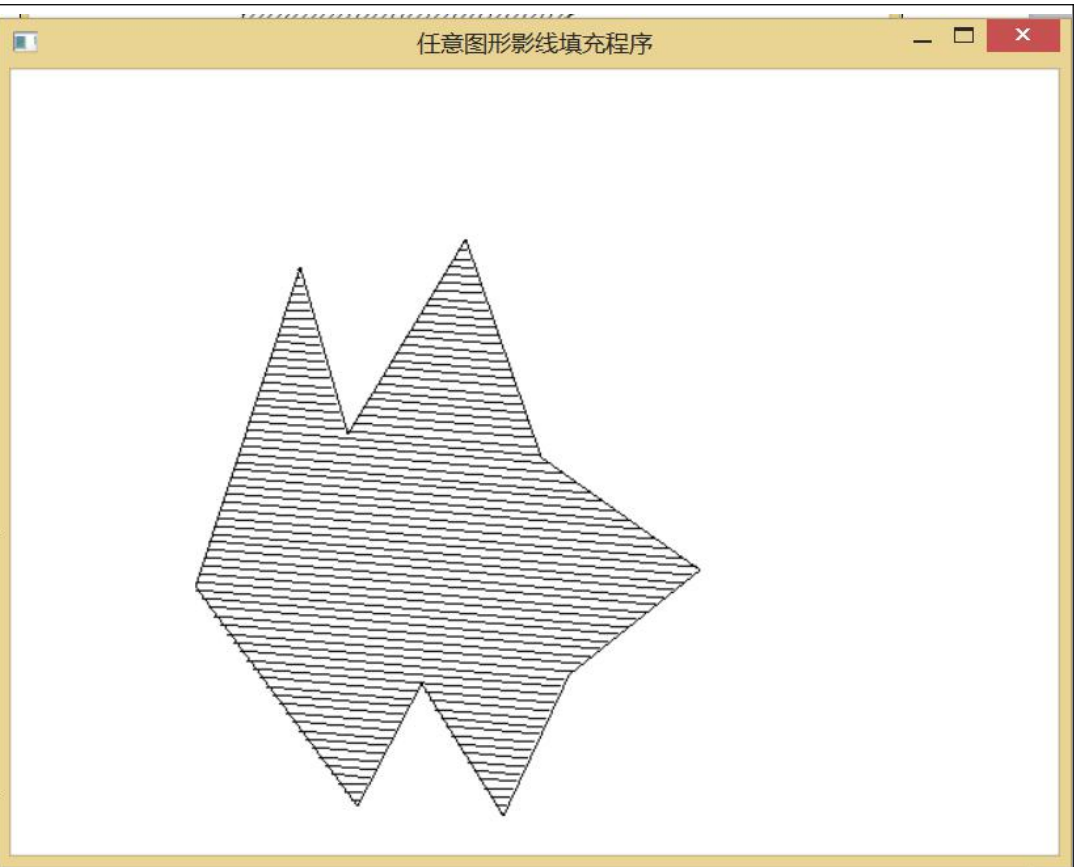
Case2:

请输入点数  
20  
请输入影线角度  
-50



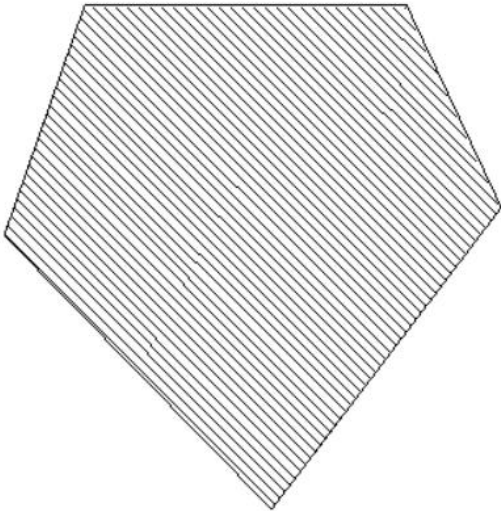
Case3:

请输入点数  
10  
请输入影线角度  
5

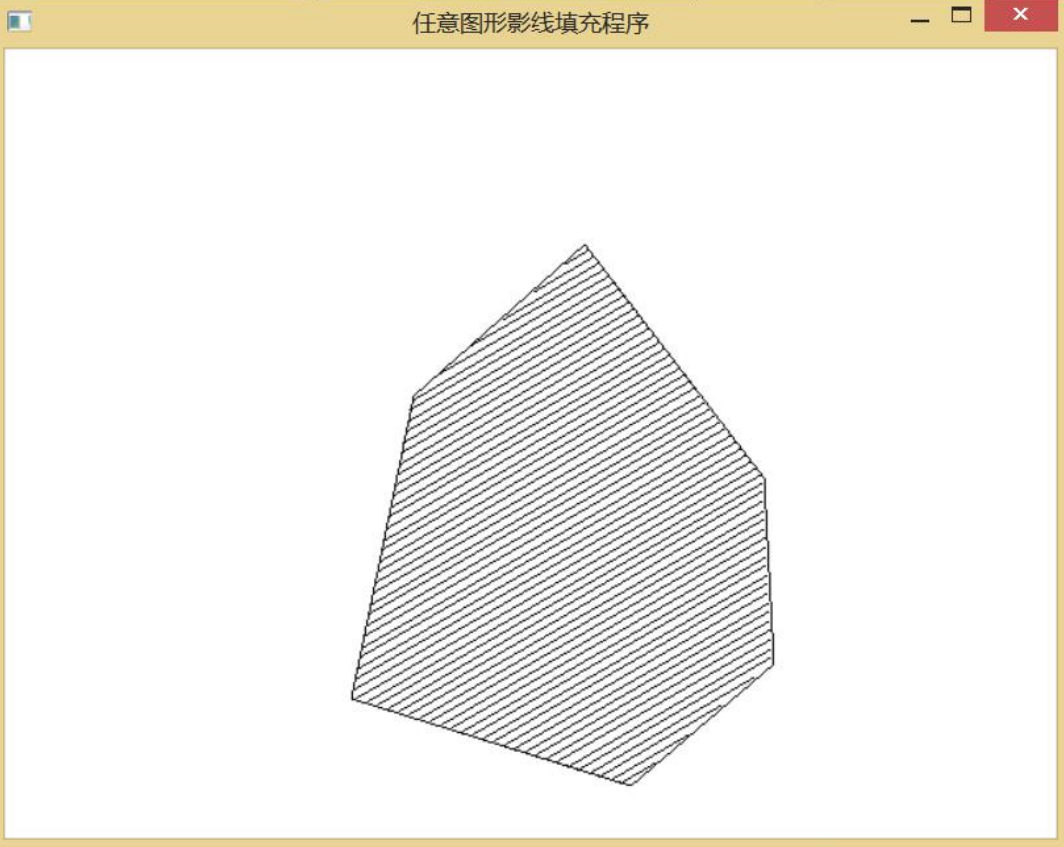


Case4:

请输入点数  
5  
请输入影线角度  
45

	<div data-bbox="363 197 1428 1052"><p>任意图形影线填充程序</p></div> <div data-bbox="354 1068 734 1270"><p>Case5:</p><p>请输入点数</p><p>6</p><p>请输入影线角度</p><p>-30</p></div>
--	---



	
<p>代码</p>	<pre> /*1552492 胡世杰 */ #include &lt;glut.h&gt; #include&lt;math.h&gt; #include&lt;iostream&gt; #define WINDOW_HEIGHT 1000 #define PI 3.1415926 using namespace std;  //扫描线算法: /* ET ymax: 边的上端点的 y 坐标值 x: 边的下端点的 x 坐标 dmax: 边的斜率的倒数  AET: ymax: 所交边的最大 y 值 x: 当前扫描线与边的交点的 x 坐标 dmax: 边的斜率的倒数 */  typedef struct Edge </pre>

```

{
    int ymax;
    float deltax;
    float x;
    struct Edge *next;
}Edge,*ptrtoEdge;

typedef struct edge    //输入过程中记录的边
{
    int highy;
    int lowy;
    int lowx;
    float deltax;
}edge,*ptrtoedge;

typedef struct POINT
{
    int x;
    int y;
} POINT,*ptrtoPOINT;

typedef struct EThead
{
    int ylow;           //线段下端点的 y 值
    Edge *next;         //指针指向 Edge 节点
}EThead;

typedef struct temppoint
{
    double x;
    double y;
}temppoint,*ptrtotemppoint;

double alpha;
int k;                //输入点数
POINT *p;
bool select=false;
EThead head[50];      //点数最大为 50
int headlen;
edge *pe;             //边（不包括平行边）
int pelen;            //边（不包括平行边）的数目
ptrtoEdge AEL;        //ael 表头

/*POINT changetonew(double x,double y,double alpha) //将标准坐标转换为倾斜坐标
{

```

```

POINT node;
if(alpha==0)
{
    node.x=x;
    node.y=y;
}

else
{
    double r=sqrt(x*x+y*y);
    double cita=acos(x/r);
    node.x=r*cos(alpha+cita);
    node.y=r*sin(alpha+cita);
}
return node;
} */
temppoint changetonew(double x,double y,double alpha) //将标准坐标转换为倾斜坐标
{
    temppoint node;
    if(alpha==0)
    {
        node.x=x;
        node.y=y;
    }

    else
    {
        double r=sqrt(x*x+y*y);
        double cita=acos(x/r);
        node.x=r*cos(alpha+cita);
        node.y=r*sin(alpha+cita);
    }
    return node;
}
/*POINT changetoori(double x,double y,double alpha) //将倾斜坐标转换为标准坐标
{
    POINT node;
    if(alpha==0)
    {
        node.x=x;
        node.y=y;
    }
    else
    {

```

```

        double r=sqrt(x*x+y*y);
        double belta=acos(x/r);
        if(y<0)
            belta=-fabs(belta);
        node.x=r*cos(belta-alpha);
        node.y=r*sin(belta-alpha);
    }
    return node;
} */

temppoint changetoori(double x,double y,double alpha) //将倾斜坐标转换为标准坐标
{
    temppoint node;
    if(alpha==0)
    {
        node.x=x;
        node.y=y;
    }
    else
    {
        double r=sqrt(x*x+y*y);
        double belta=acos(x/r);
        if(y<0)
            belta=-fabs(belta);
        node.x=r*cos(belta-alpha);
        node.y=r*sin(belta-alpha);
    }
    return node;
}

bool isnull();
//功能：判断 AEL 表头是否为空
//返回：空 false、非空 true

//void createET(EThead head[],POINT *new_p,int k);
void createET(EThead head[],temppoint *new_p,int k);
//功能：创建 ET 结构体
//参数：head, new_p: 点在新坐标系下的坐标，k: 点的数目
//返回：head 数组结构体

void print();
//功能：填充影线

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);
}

void myDisplay()
{

    glClear(GL_COLOR_BUFFER_BIT);
    //POINT *new_p=new POINT[k+1];
    temppoint *new_p=new temppoint[k+1];
    temppoint *ori_p=new temppoint[k+1];
    int isedge=0;
    if(select==true)          //输入了 k 个点
    {
        glBegin(GL_LINE_LOOP);
        for(int i=0;i<k;i++)
        {

            new_p[i]=changetonew(p[i].x,p[i].y,alpha);
            isedge++;
            if(isedge==2)
            {

                //else
                //{
                    if(new_p[i].y!=new_p[i-1].y)          //不是平行边，加入 pe 数组
                    {
                        isedge--;
                        if(new_p[i].y>new_p[i-1].y)
                        {
                            pe[pelen].highy=new_p[i].y;
                            pe[pelen].lowy=new_p[i-1].y;
                            pe[pelen].lowx=new_p[i-1].x;

                            pe[pelen].deltax=(new_p[i-1].x-new_p[i].x)/(new_p[i-1].y-new_p[i].y);
                            pelen++;
                        }
                    }
                    else
                    {

```

```

        pe[pelen].highy=new_p[i-1].y;
        pe[pelen].lowy=new_p[i].y;
        pe[pelen].lowx=new_p[i].x;

    pe[pelen].deltax=(new_p[i-1].x-new_p[i].x)/(new_p[i-1].y-new_p[i].y);
    pelen++;
}
} //if(new_p[i].y!=new_p[0].y)
//}

    if(i==k-1)        //k-1&0
    {
        if(new_p[i].y!=new_p[0].y)        //不是平行边，加入 pe 数组
        {
            if(new_p[i].y>new_p[0].y)
            {
                pe[pelen].highy=new_p[i].y;
                pe[pelen].lowy=new_p[0].y;
                pe[pelen].lowx=new_p[0].x;

                pe[pelen].deltax=(new_p[0].x-new_p[i].x)/(new_p[0].y-new_p[i].y);
                pelen++;
            }
            else
            {
                pe[pelen].highy=new_p[0].y;
                pe[pelen].lowy=new_p[i].y;
                pe[pelen].lowx=new_p[i].x;

                pe[pelen].deltax=(new_p[0].x-new_p[i].x)/(new_p[0].y-new_p[i].y);
                pelen++;
            }
        } //if(new_p[i].y!=new_p[0].y)
    }

} //if(isedge==2)
ori_p[i]=changetoori(new_p[i].x,new_p[i].y,alpha);
glVertex2i(ori_p[i].x,ori_p[i].y);
}

glEnd();
createET(head,new_p,k);
print();
} //if(select==true)
glutSwapBuffers();
glFlush();

```

```

}
void myMouse(int button, int state,int x,int y) //鼠标控制部分，通过点击来记录点
{
    static int number=0;

    if(state==GLUT_DOWN)
    {
        if(button==GLUT_LEFT_BUTTON)
        {
            p[number].x=x;
            p[number].y=480-y;

            //标准坐标系的坐标

            glBegin(GL_POINTS);
            glVertex2i(x,480-y);
            glEnd();          //鼠标点显示在 board 上

            glFlush();
            // POINT b;
            tempoint b;
            b=changetonew(p[number].x,p[number].y,alpha);
            cout<<"原始坐标为: "<<p[number].x<<" "<<p[number].y<<endl;
            cout<<"转换后坐标为: "<<b.x<<" "<<b.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[])
{
    for(int i=0;i<50;i++)
        head[i].next=NULL;
    cout<<"请输入点数"<<endl;
    cin>>k;
}

```

```

    pe=new edge[k+1];
    pelen=0;
    headlen=0;
    AEL=new Edge;
    AEL->next=NULL;
    AEL->x=-1<<30;
    cout<<"请输入影线角度"<<endl;
    cin>>alpha;
    alpha=alpha*PI/180.0;
    p=(POINT*)malloc(k*sizeof(POINT));
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(640, 480);
    glutCreateWindow("任意图形影线填充程序");
    glutMouseFunc(myMouse);
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 0;
}

//void createET(EThead head[],POINT *new_p,int k)
void createET(EThead head[],temppoint *new_p,int k)
{
    //将下端点 y 值顺序不重复排列
    int pecount=0;           //记录访问 pe 数组元素的个数
    int flag[50];           //记录值是否被访问过
    for(int i=0;i<50;i++)
    {
        flag[i]=0;
    }
    int mini=0;
    int min=10000;
    while(pecount<pelen) //访问数目小于记录边数
    {
        for(int i=0;i<pelen;i++)
        {
            if(flag[i]==0)
            {
                if(min>pe[i].lowy)
                {
                    min=pe[i].lowy;
                    mini=i;
                }
            }
        }
        //将下端点 y 值顺序不重复排列
        //记录访问 pe 数组元素的个数
        //记录值是否被访问过
        pecount++;
        new_p[mini]=pe[mini];
    }
}

```



```

        //cout<<"i="<<i<<endl;
        //cout<<"min="<<min<<endl;
        //cout<<"mini="<<mini<<endl;
    }
}
}
pecount++;
head[headlen].ylow=min;
headlen++;
flag[mini]=1;
for(int i=0;i<pelen;i++)    //把与上面找到的 ylow 相同的节点也置为访问状态
{
    if(flag[i]==0 && pe[i].lowy==min)
    {
        pecount++;
        flag[i]=1;
    }
}
min=10000;
} //while(pecount<pelen)
//建立 ET 结构体
for(int i=0;i<headlen;i++)
{
    for(int j=0;j<pelen;j++)
    {
        if(head[i].ylow==pe[j].lowy)    //将 pe[j]的参数节点插入到 head[i]后
        {
            ptrtoEdge info=new Edge;
            info->ymax=pe[j].highy;
            info->x=pe[j].lowx;
            info->deltax=pe[j].deltax;
            info->next=NULL;
            if(head[i].next==NULL)
            {
                head[i].next=info;
            }
            else
            {
                ptrtoEdge find=head[i].next;
                while(find->next!=NULL)
                {
                    find=find->next;
                } //while(find->next!=NULL)
                find->next=info;
            }
        }
    }
}

```

```

        }
        } //if(head[i].ylow==pe[j].lowy)
    }
}
//将 head 后挂节点按照 x 值顺序排列
for(int i=0;i<headlen;i++)
{
    if(head[i].next->next!=NULL)
    //不只挂一个节点
    {
        ptrtoEdge r=head[i].next;
        //ptrtoEdge f=head[i].next->next;
        while(r!=NULL)
        {
            ptrtoEdge f=head[i].next->next;
            while(f!=NULL)
            {
                if(r->x>f->x)
                {
                    int tempymax;
                    int tempx;
                    double tempdeltax;
                    tempymax=f->ymax;
                    tempx=f->x;
                    tempdeltax=f->deltax;
                    f->ymax=r->ymax;
                    f->x=r->x;
                    f->deltax=r->deltax;
                    r->ymax=tempymax;
                    r->x=tempx;
                    r->deltax=tempdeltax;
                }
                f=f->next;
            } //while(f!=NULL)
            r=r->next;
        }
    } //if(head[i].next!=NULL)
}
}
bool isnull()
{
    if(AEL->next==NULL)
    {
        return false;
    }
}

```

```

    }
    else
        return true;
}
void print()
{
    //ptrtoEdge tail=AEL;          //tail 指向 AEL 尾节点
    int smx=head[0].ylow;          //smx 定义为图形最低点 y 值
    int db=0;
    //cout<<"smx="<<smx<<endl;
    int count=0;                  //遍历 head[]记录，最大为 headlen
    while(isnull()==true || count<headlen)
    {

        if(smx==head[count].ylow)

            //扫描线与当前线段下端点 y 值相等，将 head 后的链表加入 AEL
            //插入的顺序为 x 顺序排列
            {
                if(AEL->next==NULL)
                    AEL->next=head[count].next;
                else
                {
                    ptrtoEdge f=AEL->next;
                    ptrtoEdge r=AEL;
                    while(f!=NULL && f->x<head[count].next->x)
                    {
                        r=f;
                        f=f->next;
                    }//while(f->next!=NULL && f->x<head[count].next->x)
                    if(f==NULL)
                        r->next=head[count].next;
                    else
                    {
                        r->next=head[count].next;
                        while(r->next!=NULL)
                            r=r->next;
                        r->next=f;
                    }
                }
                count++;
            }//if(smx==head[count].ylow)

        //将 ymax<=smx 的节点删除
    }
}

```

```

ptrtoEdge f=AEL->next;
ptrtoEdge r=AEL;
while(f!=NULL)
{
    while(f!=NULL && f->ymax<=smx)
    {
        r->next=f->next;
        f=f->next;
    }
    r=f;
    if(f!=NULL)
        f=f->next;
}
if(isnull()==false)    //为空
{
    break;
}

//坐标转换后投影
temppoint new_p1,new_p2,ori_p1,ori_p2;
r=AEL->next;
f=r->next;
new_p1.x=r->x;
new_p1.y=smx;
new_p2.x=f->x;
new_p2.y=smx;
ori_p1=changetoori(new_p1.x,new_p1.y,alpha);
ori_p2=changetoori(new_p2.x,new_p2.y,alpha);
while(1)
{
    if((db+1)%5==0)
    {
        glBegin(GL_LINES);
        glVertex2i(int(ori_p1.x),int(ori_p1.y));
        glVertex2i(int(ori_p2.x),int(ori_p2.y));
        glEnd();
    }

    r=f->next;
    if(r==NULL)
        break;
    f=r->next;
    if(f==NULL)
        break;
    new_p1.x=r->x;

```

```
        new_p1.y=smx;
        new_p2.x=f->x;
        new_p2.y=smx;
        ori_p1=changetoori(new_p1.x,new_p1.y,alpha);
        ori_p2=changetoori(new_p2.x,new_p2.y,alpha);

    }
    //更新节点和 smx
    smx++;
    db++;
    r=AEL->next;
    while(r!=NULL)
    {
        r->x=r->x+r->deltax;
        r=r->next;
    }
} //while(isnull()==true || count<headlen)
```