《计算机图形学原理实践》

实验报告

|  |  |  |
| --- | --- | --- |
| 实验名称 | ： | 多边形有效边表填充 |
|  |  |  |
| 姓 名 | ： |  |
| 学 号 | ： |  |
| 专业班级 | ： |  |
| 实验时间 | ： |  |

西南科技大学计算机科学与技术学院

**一、实验目的**

1.掌握有效边表和桶表的数据结构；

2.了解颜色对话框的调用方法；

3.掌握动态链表的排序算法；

**二、实验步骤**

1.调用颜色对话框读取填充色；

2.设置多边形顶点坐标值，计算扫描线的最大值和最小值；

3.用多边形覆盖的扫描线动态建立桶节点；

4.循环访问多边形的所有顶点。根据边的终点y值比起点y值高或变得终点y值比起点y值低两种情况，计算每条边的ymin。在桶表中寻找该ymin相对应的桶节点，计算该边的x|ymin,ymax,△x,并以此链接该表结点到桶结点；

5.对每个桶结点链接的边表，根据x|ymin值得大小进行排序，若x|ymin相等，则按照△x由小到大排序；

6.循环访问每个桶结点，将桶内每个结点的边表合并为有效边表，并循环访问有效边表；

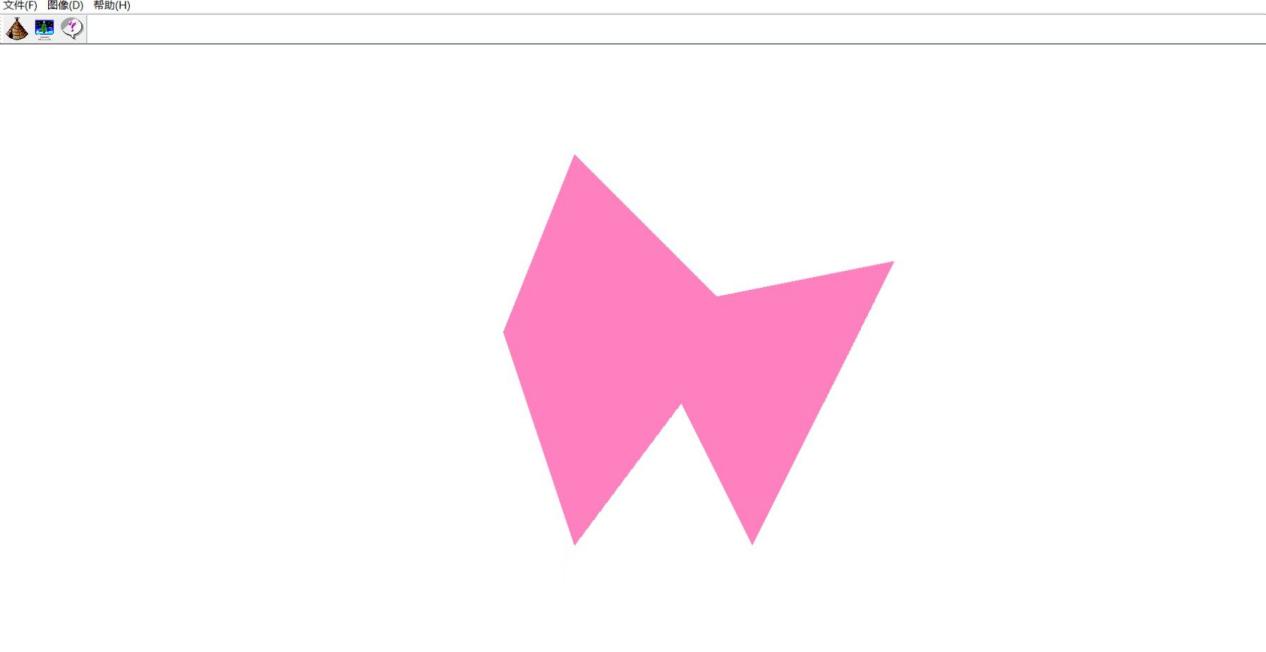
7.从有效边表中取出扫描线上相邻两条边的结点对进行配对。填充时设置一个逻辑变量blnFlag（初始值为0），每访问一个结点，把blnFlag值求反一次，若blnFlag为真值，则把从当前结点的x值开始到下一个结点的x值结束的区间用指定颜色填充；

8.循环下一桶结点，按照xi+1=xi+△x修改有效边表，同事合并桶结点内的新边表，形成新的有效边表；

9.如果桶结点的扫描线值大于或等于有效边表中某个结点的ymax值，则该边为无效边；

10.当桶结点不为空则转step6,否则删除桶表和边表的头结点，算法结束。

**三、实验结果**



**四、实验总结**

实验代码参考了老师上课时所提供的关键代码，结合所学知识，利用VS软件实现多边形的绘制，填充颜色时，选择自己喜欢的颜色。利用有效边表和桶表的数据结构，创建有效边表和桶表类，实现多边形的有效边角填充。这是第二次实验，两次实验让我逐渐掌握了如何利用理论知识书写代码，完成目标。

**附录：源代码（关键代码）**

#include "stdafx.h"

#include "Test.h"

#include "Fill.h"

#include "math.h"

#define Round(f) int(floor(f+0.5))//四舍五入宏定义

#ifdef \_DEBUG

#undef THIS\_FILE

static char THIS\_FILE[]=\_\_FILE\_\_;

#define new DEBUG\_NEW

#endif

//////////////////////////////////////////////////////////////////////

// Construction/Destruction

//////////////////////////////////////////////////////////////////////

CFill::CFill()

{

PNum=0;

P=NULL;

pEdge=NULL;

pHeadB=NULL;

pHeadE=NULL;

}

CFill::~CFill()

{

if(P!=NULL)

{

delete[] P;

P=NULL;

}

ClearMemory();

}

void CFill::SetPoint(CPi2 \*p,int m)

{

P=new CPi2[m];//创建一维动态数组

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

{

P[i]=p[i];

}

PNum=m;

}

void CFill::CreateBucket()//创建桶表

{

}

void CFill::CreateEdge()//创建边表

{

}

void CFill::Gouraud(CDC \*pDC)//填充多边形

{

}

void CFill::AddET(CAET \*pNewEdge)//合并ET表

{

CAET \*pCE=pHeadE;

if(pCE==NULL)

{

pHeadE=pNewEdge;

pCE=pHeadE;

}

else

{

while(pCE->pNext!=NULL)

{

pCE=pCE->pNext;

}

pCE->pNext=pNewEdge;

}

}

void CFill::ETOrder()//边表的冒泡排序算法

{

CAET \*pT1,\*pT2;

int Count=1;

pT1=pHeadE;

if(pT1==NULL)

{

return;

}

if(pT1->pNext==NULL)//如果该ET表没有再连ET表

{

return;//桶结点只有一条边，不需要排序

}

while(pT1->pNext!=NULL)//统计边结点的个数

{

Count++;

pT1=pT1->pNext;

}

for(int i=0;i<Count-1;i++)//冒泡排序

{

CAET \*\*pPre=&pHeadE;

pT1=pHeadE;

for (int j=0;j<Count-1-i;j++)

{

pT2=pT1->pNext;

if ((pT1->x>pT2->x)||((pT1->x==pT2->x)&&(pT1->k>pT2->k)))

{

pT1->pNext=pT2->pNext;

pT2->pNext=pT1;

\*pPre=pT2;

pPre=&(pT2->pNext);//调整位置为下次遍历准备

}

else

{

pPre=&(pT1->pNext);

pT1=pT1->pNext;

}

}

}

}

void CFill::ClearMemory()//安全删除所有桶与桶上连接的边

{

DeleteAETChain(pHeadE);

CBucket \*pBucket=pHeadB;

while (pBucket!=NULL)//针对每一个桶

{

CBucket \*pBucketTemp=pBucket->pNext;

DeleteAETChain(pBucket->pET);

delete pBucket;

pBucket=pBucketTemp;

}

pHeadB=NULL;

pHeadE=NULL;

}

void CFill::DeleteAETChain(CAET \*pAET)

{

while (pAET!=NULL)

{

CAET \*pAETTemp=pAET->pNext;

delete pAET;

pAET=pAETTemp;

}

}