计算机学院实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：实验二：多边形的扫描转换与填充 | | 学号：202000120101 |
| 日期：2022.9.27 | 班级：20.2 | 姓名：尹国泰 |
| **Email：1018693208@qq.com** | | |
| 实验目的：    了解多边形的表示方式，区域填充基本原理，掌握多边形的扫描转换算法，  实现x扫描线填充（使用活性边表结构） | | |
| 实验环境介绍：  操作系统：Window10  编译器环境：MinGW，VSCode  OpenGL环境：freeglut | | |
| 解决问题的主要思路：  1.实现边表结构，并初始化新边表NET和活性边表AET  2.通过使用数组存储顶点的方式来存储多边形  3.根据多边形创建新边表NET  4.借助NET和AET实现x扫描线填充算法 | | |
| 实验步骤：  1.NET和AET结构的设计以及初始化   |  | | --- | | typedef struct ET{      float x;      float dx;      float ymax;      ET\* next;  }AET,NET;//活性边表，新边表  AET\* headAET;  NET\* headNET[h+5];  //初始化活性边表和新边表  headAET=new AET;  headAET->next=NULL;  headNET[h+5];  for(int i=ymin;i<=ymax;i++){       headNET[i] = new NET;       headNET[i]->next = NULL;  } | | | |
| 2.多边形的存储方式，多边形顶点确定方式   |  | | --- | | struct point{      int x, y;      point(){}      point(int \_x, int \_y)          :x(\_x), y(\_y) {}  };  vector<point> vertex;//多边形顶点集合  void mymouse(int button, int state, int x, int y){      if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN){          if(!newgl){              glClear(GL\_COLOR\_BUFFER\_BIT);              newgl=1;          }          drawbigpixelf(x, h - y);          point p(x, h - y);          vertex.push\_back(p);      }//左键确定多边形的顶点      if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN){          PolyScan();          glFlush();          clear();      }//右键填充      if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN){          glClear(GL\_COLOR\_BUFFER\_BIT);          glFlush();          clear();      }//中键清空  } |   确定使用的数据结构后，下面进行算法的实现  3.首先确定扫描线范围，根据多边形顶点数组创建NET   |  | | --- | | //确定扫描线最低和最高值      int ymin=h,ymax=0;      for(auto i:vertex) ymin=min(ymin,i.y),ymax=max(ymax,i.y);    //建立新边表NET      for(int j=0;j<vertex.size();j++){//遍历多边形顶点(按顺序)          int pre=(j-1+vertex.size())%vertex.size();//前一个点在vertex中的下标          int aft=(j+1)%vertex.size();//后一个点在vertex中的下标          if (vertex[pre].y > vertex[j].y){              //与前一个点构成的边是一条新的边              NET\* cur=new NET;              cur->x = vertex[j].x;              cur->ymax = vertex[pre].y;              float DX = vertex[pre].x-vertex[j].x;              float DY = vertex[pre].y-vertex[j].y;              cur->dx = DX/DY;              cur->next = headNET[vertex[j].y]->next;              headNET[vertex[j].y]->next = cur;          }          if (vertex[aft].y > vertex[j].y){              //与后一个点构成的边是一条新的边              NET\* cur = new NET;              cur->x = vertex[j].x;              cur->ymax = vertex[aft].y;              float DX = vertex[aft].x-vertex[j].x;              float DY = vertex[aft].y-vertex[j].y;              cur->dx = DX/DY;              cur->next = headNET[vertex[j].y]->next;              headNET[vertex[j].y]->next = cur;          }      } |   4.根据前面创建的NET中的数据，使用AET实现x扫描线填充  算法分析：依次从低到高遍历每条扫描线，对于每一条扫描线，依次完成以下四步操作  （1）删除AET中到达ymax的边  （2）将NET中在y=i这一扫描线新的边用插入排序加入到AET中  （3）以AET中的点两两配对的形式来进行填充  （4）更新AET中边的x 值   |  | | --- | | //通过活性边表AET来进行区域填充      for(int i=ymin;i<=ymax;i++){          NET \*curNET;          AET \*curAET,\*preAET;          //删除AET中到达ymax的边          preAET=headAET;          curAET=headAET->next;          while (curAET){              if (curAET->ymax == i){                  preAET->next = curAET->next;                  delete curAET;                  curAET = preAET->next;              }else{                  preAET = preAET->next;                  curAET = curAET->next;              }          }          //将NET中在y=i这一扫描线新的边用插入排序加入到AET中          curNET=headNET[i]->next;          while(curNET){              curAET= headAET;              while (curAET->next != NULL && curNET->x > curAET->next->x)                  curAET = curAET->next;              if(curAET->next != NULL && curNET->x == curAET->next->x                &&curNET->dx > curAET->next->dx)                  curAET = curAET->next;              AET \*tmp=new AET;              tmp->dx=curNET->dx;              tmp->ymax=curNET->ymax;              tmp->x=curNET->x;              tmp->next=curAET->next;              curAET->next=tmp;              curNET = curNET->next;          }          //以AET中的点两两配对的形式来进行填充          curAET=headAET->next;          while (curAET!= NULL && curAET->next != NULL){              for (float j = curAET->x; j < curAET->next->x; j++){                  drawpixelf(j,i);              }              curAET=curAET->next->next;          }          //更新AET中边的x值          curAET=headAET->next;          while(curAET){              curAET->x+=curAET->dx;              curAET=curAET->next;          }          glFlush();      } |   5．边表以及多边形顶点的清空操作   |  | | --- | | void clearET(ET\* cur){      while(cur!=NULL){          ET\* nxt=cur->next;          delete cur;          cur=nxt;      }  }  void clear(){      newgl=0;      vertex.clear();      clearET(headAET);      headAET=NULL;      for(int i=0;i<=h;i++) clearET(headNET[i]),headNET[i]=NULL;  } |   实验结果展示及分析：  1.鼠标左键按顺序确定多边形顶点    2.鼠标右键进行多边形区域填充    3.完成一次绘图后，再单击鼠标左键会将上次绘图和选点清空并重新选点，也可以使用鼠标中键只进行清空操作    4.重新选点后右键填充    经过多次测试，多边形能够正常进行区域填充 | | |
| 实验中存在的问题及解决：  在最初的测试中发现会出现如下这种缺失一行的情况    进行观察，发现缺失的情况是在一条新的多边形边出现时发生的，分析代码，发现x扫描线算法是按照以下顺序执行的  （1）将NET中在y=i这一扫描线新的边用插入排序加入到AET中  （2）以AET中的点两两配对的形式来进行填充  （3）删除AET中到达ymax的边  （4）更新AET中边的x 值  这样在一条新的边从NET加入AET时，填充之前并没有删除AET中到达ymax的边，**导致新边的ymin和旧边的ymax同时存在，图中绿色箭头所指的顶点算做2个端点（本应算1个端点）**，那么在这一条扫描线上有3个端点，第3个端点没有与之匹配的端点，出现填充缺失。  将x扫描线算法执行顺序改为下面这样，先删除AET中到达ymax的边  （1）删除AET中到达ymax的边  （2）将NET中在y=i这一扫描线新的边用插入排序加入到AET中  （3）以AET中的点两两配对的形式来进行填充  （4）更新AET中边的x 值  如此可以保证在填充时像绿色箭头处的这种非极值顶点只计算1次，修改后不再出现缺失 | | |