COMPUTER GRAPHICS

NAME : SHRIRANG. R. MHALGI

CLASS : S.E.

DIV : B

ROLL NO : 222006

PROBLEM STATEMENT :

Write a java program to implement scanline algorithm.

CODE :

package cgg;

import java.awt.\*;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.util.\*;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import javax.swing.JFrame;

public class Assignment6Scanline extends JFrame implements MouseListener

{

static boolean intial=true;

static boolean choice=false;

int noOfVertices,count;

ArrayList<Edge> edgeTable; // to store edges

ArrayList<Edge> activeEdges;

ArrayList<Integer> xcord=new ArrayList<Integer>();

int previousX,previousY,startX,startY;

public Assignment6Scanline()

{

this.noOfVertices=0;

this.count=0;

this.previousX=0;

this.previousY=0;

this.startX=0;

this.startY=0;

edgeTable=new ArrayList<Edge>();

activeEdges=new ArrayList<Edge>();

}

public static void main(String args[])

{

Assignment6Scanline sc=new Assignment6Scanline();

sc.setSize(1500,1500);

sc.setVisible(true);

sc.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

sc.getEdges();

//sc.draw();

}

public void printEdgeTable()

{

for(Edge edge:edgeTable)

{

System.out.println(" Edge No: "+edge.edgeNo);

System.out.println(" X1: "+edge.x1);

System.out.println(" Y1: "+edge.y1);

System.out.println(" X2: "+edge.x2);

System.out.println(" Y2: "+edge.y2);

System.out.println(" Slope: "+edge.slope);

}

}

public int[] minNMax()

{

int arr[]= {edgeTable.get(0).y1,edgeTable.get(0).y1,edgeTable.get(0).x1,edgeTable.get(0).x1}; // ymin,ymax,xmix,xmax

for(Edge temp:edgeTable)

{

if(arr[0]>temp.y1)//y min

arr[0]=temp.y1;

if(arr[1]<temp.y1)//y max

arr[1]=temp.y1;

if(arr[2]>temp.x1)//x min

arr[2]=temp.x1;

if(arr[3]<temp.x1)//y max

arr[3]=temp.x1;

}

return arr;

}

public boolean checkEdgeActivity(Edge temp,int curY) // check if current value of y lies on the edge

{

boolean flag=false;

if ((temp.y1<curY && curY<=temp.y2 )||(temp.y1>=curY && curY>temp.y2))

flag=true;

return flag;

}

public void getActiveEdges(int curY) // needs bit of changes

{

for(Edge edge : edgeTable)

{

if(checkEdgeActivity(edge, curY) && !activeEdges.contains(edge))// add edge if not present and is active

activeEdges.add(edge);

else

if(activeEdges.contains(edge))

activeEdges.remove(edge);

}

}

public void scanline()

{

Graphics g=getGraphics();

printEdgeTable();

int minMax[]=minNMax();

int yMin=minMax[0],yMax=minMax[1],tempX=0;

System.out.println("Y min :"+yMin+" yMAx: "+yMax);

for(int i=yMin;i<=yMax;i++)

{

g.setColor(Color.GREEN);

if(i%3==0)

g.setColor(Color.blue);

else if(i%2==0)

g.setColor(Color.red);

activeEdges.clear();

getActiveEdges(i);// check active edges for current y

xcord.clear();// find xcords and store in list

for(Edge edge : activeEdges)

{

tempX=(((i-edge.y1)\*(edge.dx))/edge.dy)+edge.x1; // Finding intersection . Now add to list of xcordinates

xcord.add(tempX);

}

xcord.sort(new Comparator<Integer>()

{

public int compare(Integer x1,Integer x2)

{

int a=0;

if(x1>x2)

a=1;

else if(x1==x2)

a=0;

else

a=-1;

return a;

}

});

// now after sorting the points draw line

for(int j=0;j<xcord.size();j+=2)

{

int x1 = xcord.get(j);

if (j+1>=xcord.size())

break;

int x2 =xcord.get(j+1);

try

{

Thread.sleep(50);

g.drawLine(x1, i, x2, i);

} catch (Exception e)

{

System.out.println(e.toString());

}

}

}

}

public void draw()

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

try

{

System.out.println("Enter True to scan line fill");

Assignment6Scanline.choice=Boolean.parseBoolean(br.readLine());

}

catch (Exception e)

{

System.out.println(e.toString());

}

Assignment6Scanline.intial=false;

}

// for mouse events

@Override

public void mouseReleased(MouseEvent m) {}

@Override

public void mousePressed(MouseEvent m) {}

@Override

public void mouseExited(MouseEvent m) {}

@Override

public void mouseEntered(MouseEvent m) {}

@Override

public void mouseClicked(MouseEvent m)

{

// add edges to edges table

Graphics g=getGraphics();

g.setColor(Color.BLUE);

int x,y;

x=m.getXOnScreen();

y=m.getYOnScreen();

System.out.println("count "+count);

if(count==0)

{

previousX=x;

previousY=y;

startX=x;

startY=y;

count++;

}

else

{

if(count<noOfVertices)

{

Edge temp=new Edge(count,previousX,previousY,x,y); // x1,y1,x2,y2

edgeTable.add(temp);

g.drawLine(temp.x1, temp.y1, temp.x2, temp.y2);

previousX=x;

previousY=y;

}

if(count==(noOfVertices-1))

{

System.out.println("For last edge");

Edge temp=new Edge(count+1,x,y,startX,startY); // x1,y1,x2,y2

edgeTable.add(temp);

g.drawLine(temp.x1, temp.y1, temp.x2, temp.y2);

removeList();

}

count++;

}

}

private void addList()

{

addMouseListener(this);

}

private void removeList()

{

removeMouseListener(this);

Assignment6Scanline.intial=false;

System.out.println(Assignment6Scanline.intial);

scanline();

}

public void getEdges()

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

try

{

System.out.println("Enter No of vertices");

this.noOfVertices=Integer.parseInt(br.readLine());

System.out.println("Drawing edges "+noOfVertices);

addList();

}

catch (Exception e)

{

System.out.println(e.toString());

}

}

}

class Edge

{

int x1,x2,y1,y2,dx,dy;

int edgeNo; // yC y coordinate and ind :Index

double slope;

Edge(int edgeNo,int x1,int y1,int x2,int y2)

{

this.x1=x1;

this.y1=y1;

this.x2=x2;

this.y2=y2;

this.edgeNo=edgeNo;

this.dx=x2-x1;

this.dy=y2-y1;

this.slope=(double)(dy)/(double)(dx);

}

}

OUTPUT :

