VoroComp: A Program for Voronoi Diagram Computations. Program Specification.

June 17, 1999

1 Introduction

The `VoroComp` program is designed to compute Voronoi diagrams. It consists of two parts linked together with RMI, namely, a graphical user interface (GUI) interacting with the user and computational kernel (CK) computing Voronoi Diagram.

The section **Computational Kernel** below consists of two subsections - **Basic Definitions** and **Specification of the Computational Kernel**. In the subsection **Basic Definitions** formal definitions of Voronoi diagram and related notions are given. They are used in the next section to define formal specification of CK.

In the section **Graphical User Interface** GUI is specified.

2 The Computational Kernel

2.1 Basic Definitions

In this chapter letters p, q, r denote points on the real plane, letter P denotes a set of points, line segment between p and q is denoted as pq.

Definition 1 (Distance between two points) *Distance between two points $p = \langle p_x, p_y \rangle$ and $q = \langle q_x, q_y \rangle$:*

$$\mathtt{dist}(p, q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

Definition 2 (Perpendicular bisector) *Let p, q be distinct points. Perpendicular bisector for pq is defined as follows*

$$\mathtt{PBisector}(pq) := \{r \mid \mathtt{dist}(p, r) = \mathtt{dist}(q, r)\}.$$

Definition 3 (Voronoi Diagram Cell) *Voronoi diagram cell for $p \in P$ is defined as follows:*

$$\mathtt{VDCell}(p, P) := \{q \mid \mathit{for\ every}\ r \in P\ \mathit{if}\ r \neq p\ \mathit{then}\ \mathtt{dist}(p, q) < \mathtt{dist}(r, q)\}$$

Definition 4 (Voronoi Diagram) *Voronoi diagram for P is defined as*

$$\mathit{VD}(P) := \{s \mid s = \mathit{VDCell}(p, P) \text{ for some } p \in P\}$$

Definition 5 (Voronoi half-plane) *For two distinct points p and q Voronoi half-plane containing p and the perpendicular bisector of pq is defined as*

$$h(p, q) := \{r \mid pr \cap \mathit{PBisector}(pq) = \emptyset \text{ or } pr \cap \mathit{PBisector}(pq) = \{r\}\}$$

Definition 6 (Set of bisectors) *A set of perpendicular bisectors for $p \in P$ is defined as*

$$B(p) := \{b \mid \text{there exists } q \in P \text{ such that } q \neq p \text{ and } b = \mathit{PBisector}(p, q)\}.$$

It is easy to see that `VDCell`$(p, P) = (\cap_{p,q \in P, p \neq q} h(p, q)) - B(p)$.

Definition 7 (Voronoi Diagram Branches) *Define Voronoi diagram branches for $p \in P$ as*

$$\mathit{VDBranch}(p, P) := (\cap_{p,q \in P, p \neq q} h(p, q)) \cap B(p).$$

Definition 8 (Set of all Voronoi Diagram Branches) *Set of all Voronoi diagram branches for P is defined as follows:*

$$\mathit{VDB}(P) := \cup_{p \in P} \mathit{VDBranch}(p, P)).$$

2.2 Specification of the Computational Kernel

In this subsection we give specification of the computational kernel written in-Java Modeling Language (JML, see http://www.cs.iastate.edu/̃leavens/JML.html). All the computations inside the CK are performed in floating point numbers, but at the end the result is converted to integers.

```
public class Point{

   int x,y;

   Point(int x,int y);
      normal_behavior:
        modifiable: this.x,this.y;
        ensures: this.x==x && this.y==y;

   public int x();
      normal_behavior:
        ensures: \result==this.x;

   public int y();
      normal_behavior:
        ensures: \result==this.y;
```

```
   public abstract boolean isEqual(Point p);
      normal_bahavior:
        requires: p!=null;
        ensures: \result== ((this.x==p.x) && (this.y==p.y));
}

public abstract class AbstractLine{
   Point p1,p2;

   AbstractLine(Point p1,Point p2);
      modifiable: this.p1,this.p2;
      requires: p1.x()!=p2.x() && p1.y()!=p2.y() &&
                p1!=null  && p2!=null;
      ensures: this.p1==p1 && this.p2==p2;

   public Point first();
      normal_behavior:
        ensures: \result==this.p1;

   public Point second();
      normal_behavior:
        ensures: \result==this.p2;

   public abstract boolean isEqual(AbstractLine s);
}

public class InfiniteRay extends AbstractLine{

   InfiniteRay(Point p1,Point p2);
      modifiable: this.p1,this.p2;
      requires: p1.x()!=p2.x() && p1.y()!=p2.y() &&
                p1!=null  && p2!=null;
      ensures: this.p1==p1 && this.p2==p2;

   public boolean isEqual(AbstractLine s);
      normal_behavior:
        requires: s isinstanceof InfiniteRay;
        ensures: \result=(this.p1.isEqual(s.p1) && this.p2.isEqual(s,p2));
}

public class InfiniteLine extends AbstractLine{

   InfiniteLine(Point p1,Point p2);
      modifiable: this.p1,this.p2;
      requires: p1.x()!=p2.x() && p1.y()!=p2.y() &&
                p1!=null  && p2!=null;
```

```
      ensures: this.p1==p1 && this.p2==p2;

   public boolean isEqual(AbstractLine s);
      normal_behavior:
        requires: s isinstanceof InfiniteLine;
        ensures: \result=(this.p1.isEqual(s.p1) && this.p2.isEqual(s,p2));
}

public class LineSegment extends AbstractLine{

   LineSegment(Point p1,Point p2);
      modifiable: this.p1,this.p2;
      requires: p1.x()!=p2.x() && p1.y()!=p2.y() && p1!=null && p2!=null;
      ensures: this.p1==p1 && this.p2==p2;


   public boolean isEqual(AbstractLine s);
      normal_behavior:
        requires: s isinstanceof LineSegment;
        ensures: \result=(this.p1.isEqual(s.p1) && this.p2.isEqual(s,p2));
}


public class Points{
   Vector points;

   public invariant: \forall (Point p) \forall(int i,j)
           (points.elementAt(i)==p1 && points.elementAt(j)==p2 &&
                p1.isEqual(p2) ==>
                i=j);

   public invariant: \forall (Object o)
        (points.contains(o) ==> \exists (Point p) (o==p));

   Points();
      normal_behavior:
         ensures: points.isEmpty();

   public void add(Point p);
      normal_behavior:
        requires: p!=null;
        ensures: !points.isEmpty() && points.contains(p);

   public Enumeration elements();
      normal_behavior:
        ensures: \result == points.elements();
}
```

```
public class VoronoiDiagram{
        Vector vd;

   public invariant: \forall (AbstractLine o)
        (vd.contains(o) ==> \exists (LineSegment l) (o==l) ||
        \exists (InfiniteRay r) (o==r) ||
        \exists (InfiniteLine l) (o==l));

   public invariant: \forall(int i,j)
           (vd.elementAt(i)==p1 && vd.elementAt(j)==p2
                && p1.isEqual(p2) ==> i=j);

   VoronoiDiagram();
      normal_behavior:
         ensures: vd.isEmpty();

   public void addLine(AbstractLine s);
      normal_behavior;
        requires: s!=null;
        ensures: !vd.isEmpty() &&
                vd.contains(s);

   public Enumeration elements();
      normal_behavior:
        ensures: \result == vd.elements();
}

public class ComputationalKernel{
   Points points;
   VoronoiDiagram vd;

   public ComputationalKernel(Points points);
      normal behavior:
        modifiable: this.points;
        requires: points!=null;
        ensures: points==this.points;

   public VoronoiDiagram Compute();
      normal behavior:
        modifiable: this.vd;
        ensures: this.vd==VDB(this.points);
}
```

3 Graphical User Interface

The graphical user interface (GUI for short) provides a nice frontend to the computational kernel. It allows the user to visualize graphically the input he wants to pass on to the computational kernel.

The GUI consists of two parts: menu system, where user can choose some action described below, and panel with vertical and horizontal scrollbars, where the user is able to place and remove points and see the Voronoi diagram.

After appearing GUI the user can perform the following operations:

- insert points on the panel by
  - (a) clicking on the left button of mouse or
  - (b) loading them from a file using File menu bar item Open
- delete inserted points using Edit menu bar item Delete
- compute Voronoi diagram for inserted points using Action menu bar item Compute
- save inserted points in a file using File menu bar item Save
- clear the panel (remove all points) using Action menu bar Reset
- get on-line help about using the program from Help menu bar item Contents

- get information about developers of the program from Help menu bar item About
- exit the program using File menu bar item Exit

We describe below menu bar items:
Menu *File* has entries:

Open - Prompts you for a filename, reads the file which name was specified and displays its contents (points) on the panel. If file with the specified name does not exist new file with this name will be created. If there are points already inserted on the panel, **Open** requires first to save them before opening a file provided that the points were not saved in a file.

Save - If you want to save points from the panel that were not saved before **Save** prompts you for a filename and writes the points in this file. If a file with the specified name already existed it will be overwritten. If you want to save points on which you operated and initially they were loaded from a file then they will be saved in that file. **Save** saves only points but not Voronoi diagrams.

Exit - Exits the program. Before exiting it requires to save points if they are not saved.

Menu *Edit* has entries:

Delete Deletes the last entered point. One can use **Delete** to delete points in reversed order of their appearance. If there are not points to delete then **Delete** performs no action.

Menu *Action* has entries:

Compute - Computes Voronoi Diagram for given points and shows it. If there is at most onepoint **Compute** performs no action. After computing Voronoi diagram adding/deleting a point leads to disappearing Voronoi diagram and only points will be shown. The user can use **Compute** again to compute Voronoi diagram for obtained set of points. **Compute** performs no action if there is already computed diagram on the panel.

Reset - removes all points from the panel. If there were no points on the panel it performs no action.

Menu *Help* has entries:

Contents Shows the contents of the help

About Shows information about developers.

The size of the area of the panel in which the user puts points is 1000x1000 pixels. Only certain part of the area is visible at the moment. Using vertical and horizontal scrollbars the user can scroll up-down and left-right to observe the other parts. Resizing the GUI window changes the size of visible part of the area.