Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

*Institute for Theoretical Computer Science* *25.9.2008*
*Dr. B. Gärtner, Dr. M. Hoffmann and Eva Schuberth*

# Computational Geometry Homework 1 HS08

URL: `http://www.ti.inf.ethz.ch/ew/courses/CG08/`

## Exercise 1

As you all know, real numbers in a computer are usually represented in a floating point arithmetic and therefore not exact. Thus, roundoff errors may cause unexpected results, especially when comparing two values, which are essentially equal.

Such a behaviour can cause trouble in many geometric applications. We will focus on the following setting: if three points $a, b, c$ are close to colinear, then the predicate `rightturn(a,b,c)` indicating whether $c$ lies to the right of the (oriented) line $ab$ can give either answer (but for a fixed ordered triple, it always gives the same answer) and it does not have to be consistent with `rightturn(b,c,a)` or many other permutations of the arguments.

Such a predicate can make the Jarvis wrap, which you have seen in the lecture, fail completely. To remind you, here is its pseudocode:

```
INPUT: points[1..n];
OUTPUT: convex_hull;

pt_start := pt_current := lexicographically_smallest_element(points);
pt_next := first_element_of_points_different_from(pt_start);

do {
  convex_hull.add(pt_current);
  for (i := 0; i < n; i++) {
    p := points[i];
    if (rightturn(pt_current, pt_next, p)) then pt_next := p;
  }
  pt_current := pt_next;
  pt_next := pt_start;
} while (pt_current != pt_start);
```

The predicate `rightturn(a,b,c)` is calculated as follows:

```
rightturn(a,b,c) := [(b.x - a.x)*(c.y - a.y) < (c.x - a.x)*(b.y - a.y)];
```

Input point sets are shown in the figures together with the respective outcomes of the Jarvis wrap. In the first case, the algorithm fails finding the convex hull and instead, returns a polygon which does not even contain all the points inside. In the second case, it manages to get into an infinite loop, adding the lower thick point followed by the upper thick point point again and again (the picture on the right side of the figure). Both sets consist of points lying close to a triangle (so that each of the points "belongs" to exactly one of its sides).

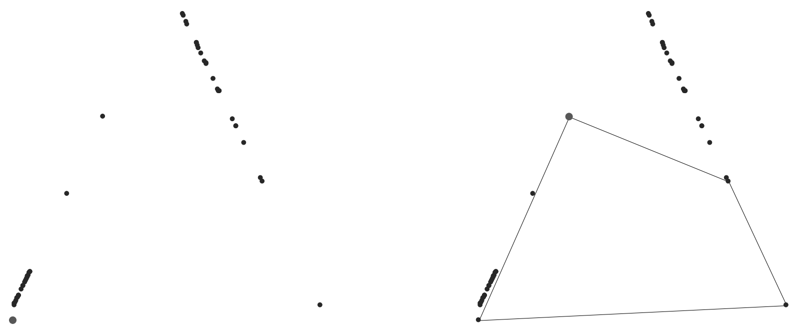

Figure 1: The first point set and its "convex hull"

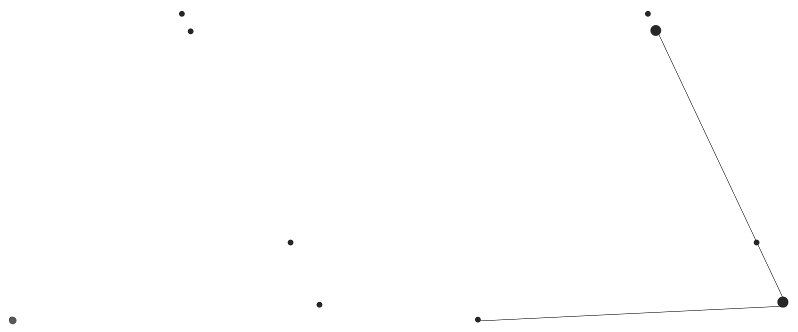

Figure 2: The second point set and its "convex hulls"

**Question**: How does it happen that the algorithm fails?

To specify the question more clearly, you are asked to describe the run of the algorithm (step by step) on each of the two inputs drawn on the figures, which leads to the result described on the right side of the figure. In order to do so, you are allowed to determine the order of (the relevant) points on the input (in the `points` array) and values of the predicate `rightturn` obeying the following rules:

$$\texttt{rightturn(a,b,c)} := \begin{cases} \texttt{!rightturn(a,c,b)} & \text{if all three points are distinct} \\ \texttt{false} & \text{if some two points are the same} \\ \texttt{false} & \text{if } a, b, c \text{ are all distinct and do not all lie on} \\ & \text{a same side of the triangle and } c \text{ is to the left} \\ & \text{of the oriented line } ab \\ \texttt{true} & \text{if } a, b, c \text{ are all distinct and do not all lie on} \\ & \text{a same side of the triangle and } c \text{ is to the right} \\ & \text{of the oriented line } ab \\ \texttt{any} & \text{if } a, b, c \text{ are distinct and all lie on a same side } s \\ & \text{of the triangle} \end{cases}$$

**Due date:** 9.10.2007, 13h15