Massachusetts Institute of Technology
6.838: Geometric Computation
Professors Piotr Indyk and Seth Teller

Handout 4
September 13, 2001

# Problem set 1

## Due **1pm, Thursday September** $27^{th}$.

## Required Exercises:

**Problem 1. Application of convex hulls: diameter**

Let $P$ be a set of points on the plane. The diameter of $P$ is defined as $\max_{p,q \in P} \|p-q\|_2$, where $\|\cdot\|$ is the Euclidean norm. Let CH be the convex hull of $P$.

- A point $p$ on the convex hull is called *unnecessary*, if it lies on the segment between its predecessor and its successor on CH. Show that removing unnecessary points from $P$ does not change the diameter of $P$. Also, give an $O(n)$ time algorithm which detects and removes from $P$ all unnecessary points.
- For each CH segment $s = p - p'$, define $anti(s)$ to be the set of points furthest from the line containing $s$. Show that if CH does not contain any unnecessary points, then the cardinality of $anti(s)$ is at most 2.
- Show that the diameter pair is a subset of $\{p, p'\} \cup anti(p-p')$ for some segment $p-p'$.
- Based on the above observations, give an $O(n \log n)$-time algoritm for finding the diameter of $P$.

**Problem 2. Vertical segment intersection.**

In class we saw an algorithm for reporting segment intersections when all segments are either horizontal or vertical. The description given in class focused on detecting intersections between horizontal and vertical segments, leaving open the problem of detecting the intersections between vertical and vertical (or horizontal and horizontal) segments. To fill this gap, construct an efficient algorithm, which given a set of vertical (only) segments, reports all pairs of segments having nonempty intersection.

**Problem 3. Textbook, exercise 4.16, p. 94**.

On $n$ parallel tracks $n$ trains are going with constant speeds $v_1 \ldots v_n$. At time $t = 0$ the trains are at positions $k_1 \ldots k_n$. Give an $O(n \log n)$ time algorithm that detects all trains that at some moment are leading. To this end, use the algorithm for computing the intersection of half-spaces. (Hint: draw all of the trains on a single space-time diagram.)

**Problem 4. Textbook, exercise 3.3, p. 60.**

A *rectilinear polygon* is a simple polygon of which all edges are either horizontal or vertical. Give an example to show that $\lfloor n/4 \rfloor$ cameras are sometimes necessary to guard a rectilinear polygon with $n$ vertices.

## Optional Exercises:

**Problem A. Textbook, exercise 2.12, p. 43**.

Let $S$ be a set of $n$ triangles in the plane. The boundaries of the triangles are disjoint, but it is possible that a triangle lies entirely inside another triangle. Let $P$ be a set of $n$ points in the plane. Give an $O(n \log n)$-time algorithm that reports each point in $P$ lying outside all triangles.

If you wish, you can assume that all $x$-coordinates of points in $P$ as well as of vertices of triangles in $T$ are different.

**Problem B. Degenerate monotone chains.**

Suppose you are given an input polygon with several consecutive vertices having the same $y$ coordinate. Suppose you use the algorithm given in the text (and in class) to decompose this polygon into $y$-monotone pieces. Will the text's algorithm triangulate these pieces correctly? If no, show an input that breaks the algorithm. If yes, argue the algorithm's correctness.

**Problem C. Well-feasible Linear Programs.**

Suppose you are given a set of $d$-dimensional halfspaces defining some (non-empty, full-dimensional) feasible region. For example, if $d = 2$ we have at least three half-planes whose common intersection is a polygon with positive area, not a line segment or a point. The LP algorithm we saw in class will produce an optimum tight on $d$ constraints in the generic case. Suppose we wish to find a feasible point that is "well inside" all of the positive halfspaces. Given the initial LP instance (constraints and objective function), show how to construct a new LP instance such that the optimum for this instance is well-feasible for the original instance.

## Optional Programming Exercises:

**Part 0.** Use your favorite Web search to find some computational geometry applets for convex hull, triangulation, etc. Note the various methods people use for interactive specification of input points and polygons.

**Part 1.** Adopt an existing Java interface (or write one from scratch) to allow addition, deletion, and editing (moving) points on the $xy$ plane. Support clearing (starting over with zero points). Support a restriction (say when pressing the shift key) of the entered point to a purely horizontal displacement, if the true mouse position is inside a 45-degree cone to the left or right of the point at which the shift key was pressed, or purely vertical displacement, if within the complement of this cone.

**Part 2.** Add a menu button to instigate a polyon-creation mode, in which points, as they are added, form a linear chain. Add a "close" button which creates an edge from the last point entered to the first point entered. Design it so that your point-editing code can be re-used, that is, after the chain is completed you can go back and drag its vertices around.

**Part 3.** Implement a linear-time algorithm to compute the convex hull of a simple polygon. Hint: use an appropriate Leftof() predicate. Does your solution handle the case of three or more (consecutive or non-consecutive) collinear vertices?

**Part 4.** Implement and animate Andrews' convex hull algorithm.

**Part 5.** Create a short web page that briefly describes your solution in plain language, links to a running applet that demonstrates your running solution, and gives instructions for using the applet. Make sure to tell us about any extra capabilities your solution has. The page should also link to your source code. Email the profs a link to the top-level web page.