Homework #1: Arrangements, zones, straight and topological sweeps [70 points]
Due Date: Wednesday, 11 February 2009

Doing problems is a very important part of this course. Although you have two weeks to do this assignment, do not delay starting to work on it — several of these problems are not routine exercises. If you cannot solve the problem fully, please write up whatever you can do and document any partial results you have obtained in the process. We intend to be generous with partial credit.

You are encouraged to collaborate in study groups of up to three students on the solution of the homeworks. If you do collaborate on theory problems, you must write up solutions on your own and acknowledge your collaborators by name in the write-up for each problem. If you obtain a solution with outside help (e.g., through literature search, another student not in the class, etc.), acknowledge your source, and write up the solution on your own. For programming problems a single write-up per group is acceptable.

Each problem set will consist of three parts, to accommodate the different needs of the students in the theory and applied tracks of the course. The common theory problems are to be worked out by everyone. The additional theory problems should be done by those in the theory track. The programming problem(s) should be implemented by those in the applied track.

- **The Common Theory Problems**

Problem 1. [10 points]

In an arrangement $\mathcal{A}$ of n lines in the plane, a single face can have at most n sides. Prove that any m distinct faces can have at most $n + 4\binom{m}{2}$ sides altogether. [[This bound is best possible if $4\binom{m}{2} \leq n$ and is known as *Canham's Lemma*; it implies, for instance, that any $\sqrt{n}$ faces can have a total of only $O(n)$ sides altogether.]]

Problem 2. [10 points]

We saw in class that, in an arrangement $\mathcal{A}$ of n lines in the plane, the zone of another line ℓ has combinatorial complexity $O(n)$ (zone theorem). Given as input only the n lines of the arrangement and ℓ (say by their line equations), show how to compute all the faces of $\mathcal{A}$ comprising the zone of ℓ, in linear space and $O(n \log n)$ time.

Problem 3. [10 points]

Show that the topological sweep that computes the arrangement of n lines in the plane can be carried through within the same time and space bounds, even if the search for an intersection when propagating a line into a bay of the upper horizon tree is done

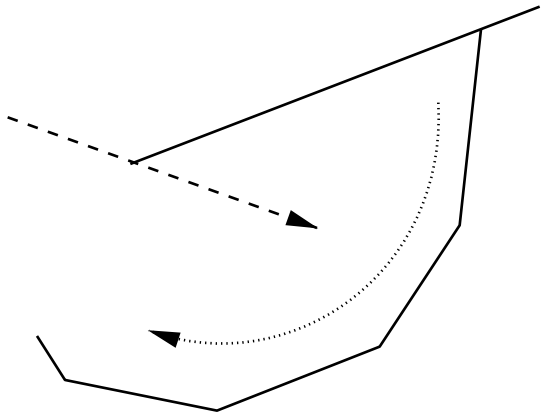

Figure 1: Searching for an intersection clockwise

by traversing the bay *clockwise* instead, starting from the other (shortened) edge of the elementary step. See figure 1.

- **The Additional Theory Problems**

Problem 4. [20 points]

Suppose that we modify the Bentley-Ottmann straight-line sweep method for computing a line arrangement so that, when processing an event, the future events corresponding to intersections for newly created adjacencies between active segments are added to the priority queue, but the events corresponding to the adjacencies just destroyed are *not* removed. This will still give a correct algorithm, but now the priority queue size may increase, as each event adds possibly two new adjacencies but removes only one. Prove that, given any four lines a, b, x, y in descending slope order, not all three intersections ax, ay, by can be events present in the priority queue at once. Use this fact to argue that maximum size of the priority queue now is at most $O(n \log n)$. Give an example showing that this bound is in fact attainable. (Partial credit will be given for any subquadratic upper bound.)

Problem 5. [5 points]

Show that the topological sweep that computes the arrangement of n lines in the plane can be carried through within the same time and space bounds, even when the topological line is required to proceed *vertically* through each region and can only move horizontally by following arrangement edges. In other words, the topological line should consist entirely of vertical segments crossing faces and portions of arrangement edges. This variant can be used to compute the vertical threads needed for a trapezoidal decomposition of the arrangement.

Problem 6. [15 points]

Show how to implement the topological sweep we discussed in class using only a *single* horizon tree, say the upper horizon tree. (*Hint:* The crucial step is to discover efficiently a vertex of the tree where an elementary step can be carried out.)

- **The Programming Problem**

Problem 7. [40 points]

The goal of this problem is provide familiarity with implementing geometric computations and with some of the algorithms packages we are going to be using throughout the course.

Implement your algorithm for solving the earlier Problem 2 (or another algorithm, if you think it is preferable in practice) in C++, using the libraries specified below. More specifically, the input of the program is a list $\mathcal{L}$ of n lines $\ell_1, \ell_2, \cdots, \ell_n$ and a query line ℓ which is specified interactively by the user. The output is the zone of ℓ in the arrangement $\mathcal{A}(\mathcal{L})$, i.e. the collection of all the convex faces of $\mathcal{A}(\mathcal{L})$ crossed by ℓ. The input lines can be specified interactively, or their coefficients may be read from a file. The input lines, the query line, and the zone of the query line are to be displayed in a graphics window.

The Programming Environment

Throughout this course, we will use existing C++ libraries to facilitate geometric programming. We use CGAL (Computational Geometry Algorithms Library) for geometric algorithms and Qt (a multi-platform GUI) for the graphical user interface.

CGAL implements geometric primitives such as points, vectors, lines, and predicates acting on these primitives, as well as many standard data structures and geometric algorithms. Qt is a C++ application framework that lets developers to write graphical applications that run on many different operating systems including Windows, Linux/Unix, and Mac OS X. While Qt is sophisticated, starting to write programs using it is actually fairly easy. The *Support Library* in CGAL provides a simple interface between CGAL and Qt, allowing us to concentrate on the algorithmic aspect of the assignment. You are also welcome to use STL (Standard Template Library) for many basic data structures. More information about these software packages can be found at:

```
http://www.cgal.org
http://www.qtsoftware.com/products
http://www.sgi.com/tech/stl/
```

The corresponding manuals can be accessed from the `Links` part of the class web page. CGAL makes heavy use of C++ and templates. If you are not yet comfortable with templates, the tutorials at `http://www.cplusplus.com/doc/tutorial/` may be useful.

For this first project, you will only need to read the CGAL manual introduction and the relevant sections about straight lines and the graphical interface.

Getting Started

The Myth machines in Gates B08 can be used for this assignment, if you do not wish to use your own computer — we provide the necessary infrastructure. A sample project (in `/usr/class/cs268/hw1-template`) demonstrates basic usage of CGAL and Qt, and provides a makefile with all the necessary variables set. We also provide a number of demos and examples from CGAL, some of which might be useful to inspect. They require that a `CGAL_DIR` environment variable be defined, before being built. Do `source /usr/class/cs268/setup.csh` in order to set this properly. For more information, see the programming part of the course web page.

Deliverables

To get full credit for this problem, you need to hand in a two-to-three page write-up of your algorithm along with your implementation code. In the write-up, you should provide a complexity analysis of your algorithm and a justification for the key implementation decisions you made, including how you handled degeneracies and possible numerical errors. Please submit the write-up as pdf or text.

In your program you should allow for at least two forms of input. One is the standard input (`std::cin`) and the other is interactive input directly in the graphics window. The number of lines to specify interactively should be passed as a command line argument.

CGAL geometric object classes (including the line class) provide operators for IO streams (<<, >>). The input is interpreted as a sequence of space delimited numbers (end-of-line acts a whitespace character) and split into subsequences of length 3. Each subsequence is then taken to be a triple of line coefficients (a, b, c), corresponding to the line $ax + by + c = 0$. When using a predefined CGAL kernel that supports exact (without roundoff errors) geometric constructions, each number in the input must be of the form p/q, where p and q are integers. Notice that even the integer coefficients must be written as fractions (e.g. $5/1$ instead of 5). With the inexact kernel, the standard floating point notation is used, fractions do not work, and the / character is not valid. Please refer to the sample program for how input should work, and to the CGAL manual for how different kernels are used.

The output should also be produced in two ways. One is to paint the zone in the window; the other is to list the convex faces in the zone in the ordering they appear along the query line. For each face, you need to list the indices of the lines in counterclockwise order around the boundary of the face, starting from the leftmost intersection of the query line and the face.

Your program must handle any non-degenerate input and report error conditions (not crash or hang) when it cannot handle the input presented. More points will be given for solutions that correctly report the degeneracies in the input, as well as for those that encapsulate all access to the input data in a few atomic predicates with dis-

crete output. For full credit, your solution has to produce the correct output, even when presented with degenerate input. One of the ways to handle degeneracies is by using the functions provided by CGAL's exact kernel to implement your tests. Generally, credit will be given based on the correctness, efficiency, and clarity of your implementation and write-up.

An additional piece of code is provided to process the polygons (zone faces) before displaying them in the graphics window, to prevent the visualization problems that CGAL's Qt interface has when the polygon size is large compared to the currently visible area. Please refer to the source code (`/usr/class/cs268/hw1-template/dice_polygon.*`) for additional details.

To submit your project, email it to the TA. Further instructions will be given in the class web page.