Homework #2: Voronoi and Delaunay diagrams [60 points]
Due Date: Tuesday, 11 May 1999

- **The Common Theory Problems**

Problem 1. [5 points]

The edges of both the Delaunay and Voronoi diagrams are line segments. Give a simple necessary and sufficient condition on a pair of sites A and B so that AB is a Delaunay edge *and* AB intersects its dual Voronoi edge.

Problem 2. [10 points]

We mentioned in class that a triangulation of a set S on n sites in the plane is a Delaunay triangulation if and only if every edge passes the *InCircle* test with respect to its two adjacent triangles. This gives a linear-time algorithm to verify that a triangulation is Delaunay, and it also suggests the following algorithm to fix it up (if it's not). Start with any triangulation of the n sites. If an edge fails the *InCircle* test, then swap it with the other diagonal of the quadrilateral formed by the two adjacent triangles (this edge must pass the test). Make this idea into a rigorous algorithm and prove its correctness. Prove that your algorithm always terminates in $O(n^2)$ steps. (*Open Problem:* Can this method be parallelized in an interesting way? What processor/time bounds can you get?)

Problem 3. [10 points]

Second-order Voronoi diagram:
Given n sites in the plane, suppose we partition the plane according the nearest *and* the second-nearest site. Thus all points in the same region have the same nearest and second-nearest neighbor. This is sometimes called the *second-order* Voronoi diagram. Show that it is also of size $O(n)$ and can be computed in time $O(n \log n)$.

- **The Additional Theory Problems**

Problem 4. [10 points]

Davenport-Schinzel sequences of order 2 and triangulations:
Let P be any convex polygon with n vertices. A triangulation of P is a collection of $n-3$ non-intersecting chords connecting pairs of vertices of P and partitioning P into $n-2$ triangles. Set up a correspondence between such triangulations and $DS(n-1,2)$ sequences, as follows. Number the vertices $1, 2, \ldots, n$ in their order along ∂P. Let T be a given triangulation. Include in T the edges of P too. For each vertex i, let $T(i)$ be the sequence of vertices $j < i$ connected to i in T and arranged in *decreasing* order, and let U_T be the concatenation of $T(2), T(3), \ldots, T(n)$.

(a) Show that U_T is a $DS(n-1,2)$ sequence of maximum length.

(b) Show that any $DS(n-1,2)$-sequence of maximum length can be realized in this manner, perhaps with an appropriate renumbering of its symbols.

(c) Use (a) and (b) to show that the number of different $DS(n,2)$ sequences of maximum length is $\frac{1}{n-1}\binom{2n-4}{n-2}$ (where two sequences are different if one cannot obtain one sequence from the other by renumbering its symbols).

Problem 5. [10 points]

We have discussed in class the lifting map $\lambda(x,y) : (x,y) \mapsto (x,y,x^2+y^2)$ from points in the xy-plane to points on the paraboloid of revolution $z = x^2 + y^2$. As we will also see, the downwards-looking faces of the convex hull of the lifted images of a collection of sites in the xy-plane correspond to the Delaunay diagram of the sites. What do the upward-looking faces correspond to? Is there an analogous Voronoi diagram? How fast can it be computed?

Problem 6. [15 points]

We want to do an analysis of the randomized incremental algorithm for Delaunay triangulations discussed in class, but based on the appearance and disappearance of Delaunay edges during the process, rather than that of triangles.

We defined the *weight* of a triangle Δ to be the number of sites inside the circumcircle of Δ and related in the class analysis this weight with the probability that Δ would ever appear as a Delaunay triangle during the random process. How should we define the corresponding notion of the weight of an edge $e = AB$? Below we define one possibility, but feel free to explore your own definition, as long as it leads to the same eventual result.

One way is to let the weight of an edge AB be w if there is a circle through A and B which contains exactly w other sites to the left of AB and w sites to the right. Show that this notion of weight is well defined. Prove an edge of weight w arises as Delaunay at some point of the random process with probability at most $4/(w+1)(w+2)$.

By emulating the argument given in class for triangles, show the combinatorial-geometric result that in any collection of n sites, the number of edges of weight at most w is $O(n(k+1))$.

Briefly outline how combining these results (using the summation-by-parts trick shown in class) allows us to conclude that the expected number of edges that arise during the random process is $O(n)$ (of course, this also follows immediately from the result for triangles ...).

Given an edge AB in a group of n sites, how fast can you calculate its weight? How fast can you calculate the *minimum* number of other sites whose deletion would make AB a Delaunay edge? Are those two quantities related?

- **The Programming Problem**

Problem 7. [35 points]

Curve reconstruction from unordered points:

In 1997, Nina Amenta, Marshall Bern, and David Eppstein proposed a new method for reconstructing smooth closed curves from unordered sample points sampled along the curve. Their beautifully simple method extensively uses the concepts of the Voronoi and Delaunay diagrams of a set of sites that we have studied. Their paper *The Crust and the β-Skeleton: Combinatorial Curve Reconstruction* appeared in Graphical Models & Image Processing, **60/2**, 125–135, March 1998. The tech. report can be accessed on-line from

`http://www.ics.uci.edu/~eppstein/pubs/a-amenta.html`

For this problem you need to implement their crust *algorithm (*crust *is what they call the reconstructed curve). Please use the Voronoi/Delaunay libraries from* LEDA *or* CGAL *— this should make the task very straightforward.*

Their paper proves that if a smooth closed curve is sampled at a sufficient high local density (depending on what they call the 'local feature size'), then the crust algorithm will always perfectly recover the sequence of samples along the original shape.
Experiment with the crust algorithm to determine if the theoretical bounds on the sampling densities needed to make the algorithm work are reasonably tight. Use a class of 'non-pathological shapes' (your definition) for your experiments and determine empirically the sampling density at which the crust algorithm starts to break down.

The crust method also has several limitations:

- it does not properly handle open curves, or curves that contain sharp corners,
- though it can reconstruct multiple closed contours that are well separated, in general it does not handle the reconstruction of planar subdivisions sampled along their boundaries (for similar reasons as above), and
- it is sensitive to noise — it may go astray if the sampled points do not lie exactly on the given curve.

Focus on at least one of these limitations (open curves, sharp corners, T-junctions, noise in the data) and try to improve the algorithm so as to extend the range of data it can handle.

You may also want to look at a very recently proposed alternate definition of the crust given by Chris Gold and Jack Snoeyink (*Crust and Anti-Crust: A One-Step Boundary and Skeleton Extraction Algorithm, 1999 ACM Symp. Comp. Geometry, to appear.*) The on-line version can be found at

`http://www.cs.ubc.ca/spider/snoeyink/demos/crust/Crust.pdf`

There has also been a 3-d extension of the crust proposed, in Nina Amenta, Marshall Bern and Manolis Kamvysselis: *A New Voronoi-Based Surface Reconstruction Algorithm, Siggraph '98, pages 415-421, 1998.*

The Programming Environment

There is a Java implementation of the algorithm written by Jack Snoeyink. It can be found at

`http://www.cs.ubc.ca/spider/snoeyink/demos/crust/home.html`

You may want to play with it and see what your program is expected to do.

To test your program and facilitate your experiments, we provide a library and an example program to interactively input, edit, and sample cubic spline curves. The detailed description of the library and program can be found at

`http://graphics.stanford.edu/courses/cs368/spline.html`

Deliverables

Please hand in a write-up of your implementation of the original crust algorithm and the extensions you have chosen to implement. Can you *prove* that (under certain conditions) the extended algorithm works where the original one does not? Print a copy of the data set(s) and reconstructions that you used for the empirical evaluation of the original and extended algorithms. Offer an analysis of the experimental results and draw whatever conclusions you think would be the most helpful to the next person to attack this problem.

Again, your program is expected to allow two types of data entry. It should allow the user to read from a data file or to enter the sample points interactively. The data file format is the number of points followed by the (x, y) coordinates for each point. For output, it should be able to draw the crusts and to list all the edges in a crust.

Problem 8. [0 points]

The programming problem of the next and last homework assignment (#3) will be an *open ended project assignment*. Please hand in on the date homework #2 is due a one-to-two page description of your proposed project in *implementing one or more geometric algorithms to solve a problem of interest to you.* Describe the problem you want to solve, why it is interesting, and outline some of the approaches that you plan to investigate. We will give you feedback based on this proposal. The topic is itself completely open, except for one condition: your problem and algorithms must operate on data whose dimensionality at least 3.

There will be a special class meeting to offer project suggestions and discuss ideas that need further refinement.