CS268: Geometric Algorithms Handout # 7
Design and Analysis
Stanford University Monday, 7 November 2016

Homework #3: Point location, polygon triangulation, visibility, convex hulls [80 points]
Due Date: in class, on Monday, 5 December 2016 – ** no late days for this homework **

- **The Theory Problems**

Problem 1. [10 points]

Consider a subdivision of the plane consisting entirely of rectangles aligned with the x- and y-axes. Such a subdivision is clearly monotone for any direction θ, $0 < \theta < \pi/2$ (in other words, a line in the direction θ always cuts each region along at most a single segment). As was shown in class in the monotone subdivision point location lectures, for each fixed direction θ, the corresponding "above" relation between rectangles is acyclic. Prove that there is actually a linear ordering of the regions that is consistent with the "above" relations *for all θ, $0 < \theta < \pi/2$, simultaneously*.

Problem 2. [10 points]

We discussed briefly in class the point location method due to David Kirpatrick which constructs a hierarchy of coarser and coarser triangular subdivisions over the original triangulation [*SIAM J. Comp., 12 (1983), 28–35*] by repeatedly removing large subsets of independent vertices and re-triangulating the resulting holes. Adapt his method to perform point location on subdivisions consisting entirely of rectangles aligned with the axes. Your method should build a hierarchy of subdivisions that are all of the same type as the original: edges must be vertical or horizontal and regions rectangular. Your asymptotic preprocessing, space, and query bounds should be the same as Kirkpatrick's. (*Hint:* You may want to consider removing elements other that vertices in the coarsening process.)

Problem 3. [10 points]

Let P be a simple polygon on n sides. Show how to compute the vertical trapezoidalization of P in linear time, starting from an arbitrary triangulation of P (we covered the other direction in class).

Problem 4. [10 points]

Let P be a simple polygon of n sides. Consider a particular side e of P as a luminous neon tube, casting light towards the interior of P. The illuminated area of P is another polygon V, called the *weak-visibility* polygon of P from e. Every point of V has the property that "it can see" some point of e. Show how to compute V in linear time, starting from an arbitrary triangulation of P.

- **The Programming Problem**

Problem 5. [40 points]

QuickHull in 3D

For this problem you will implement the randomized incremental algorithm known as *QuickHull* for computing the convex hull of a 3D point set. You should choose your initial points, as well as the next point to add to the hull, uniformly at random within a cube of side length 2 centered at the origin and aligned with the axes.

In the previous programming assignments, we have provided you with the actual geometric primitives (both data and computational) you needed to accomplish your goal. This time around, we are leaving it up to you to figure out which primitives you need, as well as the most convenient way to represent them in code. The starter code does enforces one constraint, however: you must use the provided half-edge data structure to build and store your convex hull. (Technically, it is missing one of the pointers that the real half-edge data structure contains, but this pointer is unnecessary for this application and turns out to be a pain to maintain it correctly under all the operations you will be performing here.) In particular, you are going to need to take advantage of the half-edge data structure in order to efficiently discover and delete the triangles visible from the point currently being processed. Refer to the previous assignment for more information on half-edges.

Once you have computed the convex hull, you will need to output it. Your output should be in the object file format (OFF), which is a standard plain text format for storing 2D meshes in 3D. Using this format will enable you to use existing software to view and manipulate your mesh. For this project, we recommend viewing your file in *MeshLab*. You will need to output the entire point set for this project, not just the points on the convex hull. Doing so will allow you to render all the points along with the mesh in MeshLab, which will be useful in checking your work. You can find the OFF specification here: `http://people.sc.fsu.edu/~jburkardt/data/off/off.html`. You can find MeshLab here: `http://meshlab.sourceforge.net/`.

Finally, to illustrate how we can leverage the convex hull computation to solve other problems, we have included a very short implementation of the Delaunay triangulation that offloads all the work to your convex hull implementation, using the lifting map described in class. While you don't need to make any modifications to this code, you can use it to further verify the correctness of your code.

For up to 5 points extra credit, find example 2D point sets (for the Delaunay triangulation part) and 3D point sets (for the convex hull part) that illustrate how precision and near-degeneracy issues can dramatically affect the output of these algorithms. Careful: Make sure that these really are examples of precision issues and not logic errors in your code!

Submission instructions: Pack all of your source files into a single zip file and email the file to the CA (mhsung@cs.stanford.edu) with the subject line starting with [cs268-hw3]. Include a README file with the names of all team members, and an explanation of what examples of precision issues (if any) you included.