Homework #3: Point location, polygon triangulation [60 points]
Due Date: in class, on Wednesday, 14 March 2007 – ** no late days for this homework **

- **The Common 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 will be shown in class, 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 will discuss 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*]. 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.)

- **The Additional Theory Problems**

Problem 3. [10 points]

Read Section B1 of the "Ruler, Compass, and Computer" paper (included in the course reader). Show how the interval stabbing structure presented there can be adapted to report all the intervals containing the query point, not just count them. The time for the reporting operation should be $O(\log n + k)$, where k is the number of intervals reported. Then show that counting and reporting can also be done (within the same time bounds) when the query object is not a point but another interval, and we are interested in the original intervals intersecting the query interval. The more ambitious of you can now try to use this method plus a sweep line idea to give an $O(n \log n + k)$ algorithm for reporting all intersecting pairs among n axis-aligned rectangles in the plane (again k is the output size) [5 extra points]. This is a very useful operation in design rule checking for VLSI circuits.

Problem 4. [10 points]

Let S be the set of vertices of a simple polygon P in the plane and call a diagonal AB of P *extreme* if both A and B are vertices of the convex hull of S. Show that, unless P is convex, P can be triangulated without using any extreme diagonals.

Problem 5. [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 will cover the other direction in class).

Problem 6. [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 7. [40 points]

There are two options on this problem. The first option is to do to the following project on cutting triangulated surfaces. The second is to do a project of your own choosing.

Option 1: Handling surface self-intersections. You will be given as input a possibly self-intersecting triangulated surface in 3D. Self-intersecting surfaces can arise in computer graphics as the result of errors in 3D scanning, undetected collisions in physical simulations, and so on. The goal is to discover all self-intersections between different triangles in the surface, and to cut the surface along all of these intersections to produce a new mesh that is now devoid of self intersections. The resulting cut surface will have boundaries along all the intersections of the previous surface, and may have additional connected components as well. Away from the self-intersections, however, it should have the same topology as the original surface.

More concretely, each triangle in the original surface may intersect with other triangles along one or more segments. These intersections will cut the triangle into one or more subpolygons. Two examples of intersection segments in triangles are shown below. The resulting set of polygons will have coincident vertices at the intersection points, and these split vertices must be represented correctly in order to compute the correct final topology. Moreover, each polygonal piece must be joined correctly to adjacent pieces of neighboring triangles to produce the correct global topology.

You should compute the set of intersecting triangles efficiently using bounding volume hierarchies, spatial partitions, or other range searching techniques. Additional credit will

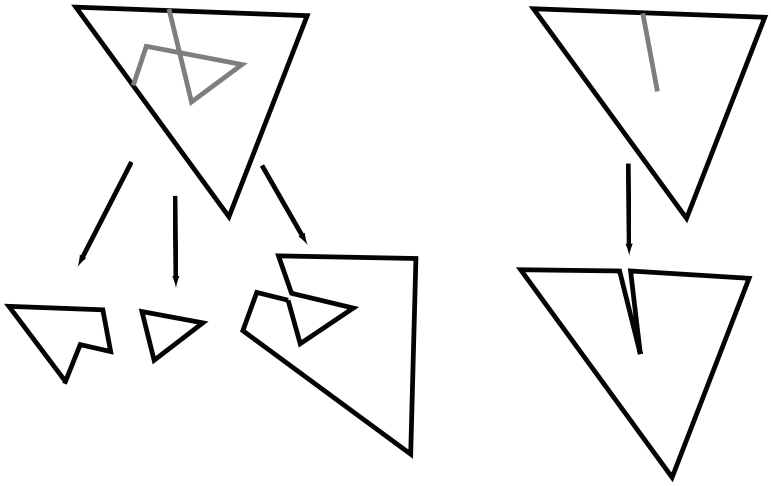

Figure 1: Triangle intersections.

be given for algorithms that exploit the geometry of the surface to accelerate the queries, such as those given in reference [2] below. CGAL provides routines for computing segment intersections between triangles (described in [1]).

Once the intersection segments are computed, there are several possible algorithms for splitting the mesh along the intersection curves. For example, you can implement the above description directly by breaking each triangle separately into polygons, and then joining edges and vertices between adjacent polygons. Alternatively, you can walk along the intersection curve topologically, breaking triangles apart incrementally. Whichever approach you choose, you should be careful to correctly handle self-intersections of the intersection curve itself (i.e., the intersection curve will not necessarily be a 1-manifold).

Your program should take as input a single triangulated surface in .obj file format, and produce a visualization of the intersection curve and any separate components resulting from cutting the surface. For example, you can provide a key that steps through each component hiding all of the others. You should also write out each connected component as a separate .obj file.

CGAL provides data structures for storing and modifying polygonal surfaces, which you can use for most of the operations (including reading .obj files) [3]. You should detect degeneracies, such as two intersecting triangle edges, and quit with an appropriate message if they are found (or handle them for extra credit). Note that even for non-degenerate surfaces it is possible for cuts to start at a mesh vertex as shown in the third case below (figure taken from [4]).

We will provide parameterized examples of intersecting surfaces with and without non-manifold intersection curves.

1. Devillers, O. and Guigue, P., *Faster Triangle-Triangle Intersection Tests*, INRIA Research Report 4488, June 2002 (`http://hal.inria.fr/docs/00/07/21/00/PDF/RR-4488.pdf`)

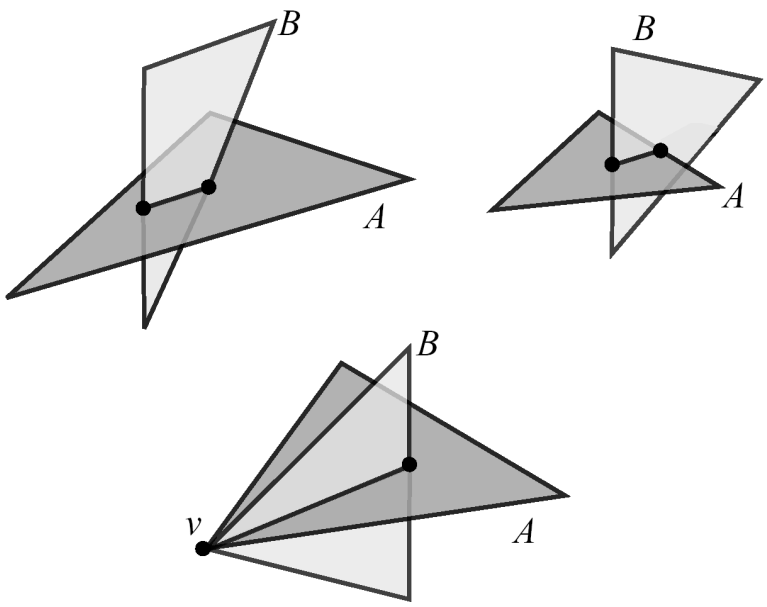

Figure 2: Intersecting triangle edges (from [4]).

2. Park, Sang C., *Triangle Mesh Intersection*, The Visual Computer, Volume 20, Number 7, pages 448–456, 2004.

3. `http://www.cgal.org/Manual/3.2/doc_html/cgal_manual/Polyhedron/Chapter_main.html`

4. Baraff, D., Witkin, A., and Kass, M., *Untangling Cloth*, SIGGRAPH 2003, pages 862–870. (`http://graphics.pixar.com/UntanglingCloth/paper.pdf`)

Option 2: An individually designed project. A second option on this final programming problem is for you to propose a project that integrates techniques from this course with another area where you are currently working on. Such a project should clearly demonstrate the use of computational geometry methods and provide an assessment of their effectiveness. In your write-up you will need to provide enough background about the other area to allows us to evaluate the appropriateness of the methods you have chosen to implement. *This option has to be approved by the instructor and a one page project proposal about it will be due in class on Wednesday, 28 February 2007.*