

CS133 Computational Geometry, Winter 2003

Assignments

Part 1 Due: Friday, February 14

Part 2 Due: Friday, February 28

Part 3 Due: Friday, March 14

1. For the first assignment you will implement an algorithm that triangulates a monotone polygon (monotone along the y -axis).
 - The first step is to implement code that checks if the polygon is monotone in linear time.
Assume that the polygon is given as a sequence of n coordinate pairs, that all the coordinates are integers and distinct.
Prove that your checking routine runs in linear time.
 - Implement code that assumes that the input polygon is monotone, and triangulates the polygon in linear time.
The output should be the list of $n - 2$ triangles.
2. For this assignment, you will have to implement an algorithm that creates a random simple polygon.
 - The program should get the desired number of vertices of the polygon as input.
 - You should output the polygon as a sequence of two dimensional points, as well as provide a graphical description. This can be for example a Java applet that draws the polygon, or a postscript file, or an OpenGL window.
 - Your program will be judged on how efficient your polygon generation routine is, and how random your polygons are. You have to:
 - (a) Give a description of the algorithm you are using.
 - (b) Show that the algorithm can generate any rectangle of n vertices with non-zero probability.
 - (c) Give an analysis of the running time of the algorithm.
3. You have also to give a routine that checks if the polygon you are building is indeed a simple polygon. In other words, you have to check that all the intersections between edges are consecutive edges that have exactly one common vertice.
 - Assume that the input to this routine is the number of vertices, followed by the x and y coordinate of each vertex. Assume integer coordinates.
 - Start by implementing a simple $O(n^2)$ technique that checks every pair of edges.
 - Show how you can check if a polygon of n vertices is simple in $O(n \log n)$ time. You can modify the algorithm of section 7.7 to do so.
 - Finally, implement the $O(n \log n)$ algorithm.