Spring 2024, Computational Geometry Projects

DUE: before midnight of May 3rd (Friday), by email to instructor (taoju@wust.edu)

Grading:

All projects turned in by the due date will get a perfect (100%) score. What will vary is how much the project as a whole is worth. A solid project on one of the below topics is expected to be worth about 20% of your total grade. Projects with a significant "wow" factor may count for as much as 30% of the total grade. You can choose to not to do a final project, in which case your grade will depend solely on the homeworks (50%) and exams (50%). No late submissions are allowed; they will be treated as no submission. Projects can be done by 1 or 2 person groups.

Project types:

These project ideas are broken up into different categories:

Category One: Pedagogy (Graphical Applets)

The applets are meant to illustrate concepts learned in class --- it is *not* sufficient to just implement the algorithm, your implementation must also highlight (visually) the important steps of the algorithm and allow a user to fully appreciate HOW it works. Applets that illustrate extension of algorithms in class to more general scenario (e.g., 3-dimensions) will earn extra points.

COLLABORATION POLICY: There is a great deal of code that is publicly available on the web. You are welcome to use this (e.g. classes that draw graphs, and triangles and nodes, that support balanced binary trees, etc), BUT you must reference where it came from, and clearly indicate WHAT functionality you have added to the original code.

An example applet, and a collection of more examples. Some possible topics:

1. An applet showing the running of the plane-sweep algorithm for decomposing a polygon into X-monotone polygons. The applet should visualize the various vertex types, the sweep status, helper vertices of edges, and the added diagonals.
2. An applet showing the randomized incremental construction of the trapezoidal map, and the search structure that is built up along with it. Could include statistics of "if you inserted the segments in lots of random orders, what is the distribution of the longest search path in the search structure?".

Category Two: Work on Open Problems

This can take the form of a paper (a) describing the current best algorithms for these problems, the best bounds on the running time of these algorithms, and (b) a description of your approaches to these problems. YOU DO NOT have to improve on the best known bounds for these algorithms, but you do have to carefully document what approaches you tried, why they worked/failed. The expected result of this would be a roughly ten page paper (with diagrams and references), or an equivalent length webpage/blog talking about things you tried and how they came out. Check out the list of unsolved problems from the Open Problems Project.

Category Three: Applications

One of the claims I've made is that Computational Geometry is useful for many different fields. Here is your chance to help me prove that statement. For this category, pick a problem relevant to other interests of yours, or other problems of your choosing (e.g. routing of large airplane parts through not so much larger warehouses, or match 3-d shapes from data sets captured by the Kinect). You will need to design an algorithm for your problem, analyze its complexity, provide an implementation, and demonstrate the results on practical inputs.

What to turn in:

Your final document or webpage, should include (in addition to any program/applet you created):

1. Names of people in your group
2. A specific description of the problem that you are trying to solve, **in two sentences or less**. This may be an answer to a question like: What does your applet do? What is your visualization trying to show? What open problem were you trying to address?
3. If you are doing a pedagogical applet
    1. Background information about the problem and approach. Does not need to be long (one or two paragraphs is ok), but state any assumptions that are necessary for your applet/approach to work (e.g. general position, no more than 10 points, etc.)
    2. Describe the pseudo-code of the algorithm that you are creating,
    3. Highlight what parts of that pseudocode you are animating.
    4. Discuss any interesting choices that you made in the implementation of the algorithm.
4. If you are working on an open research problem.
    1. Background information about the problem. What is the current best known algorithm? What are the proven theoretical bounds on the problem?
    2. What approaches have you tried? Can you visualize the results? Show sketches of where proof attempts have failed, or example point sets where your (mostly good) algorithm has a bad case? [I understand that what is appropriate depends heavily on what you tried and what your problem is].
5. If you have an application domain problem:
    1. Background information, what are your specific inputs and outputs (show a visualization, if possible). What are the needs for the problem domain -- is it important that you find the optimal solution? why or why not?
    2. What are the most related projects that others have built?
    3. Describe your algorithm and show example results. Give a complexity analysis of your algorithm, and also show measured running time for different sizes of inputs

If you are submitting an application or demo, I must be able to run your program on a Windows machine with minimal installations (e.g., web browser, bare-bone Java and Python compilers) without needing to install additional packages or libraries. If I am not able to run your program after spending a reasonable effort to make it work, you will not receive any points.