

MCS 481 Project One : planar convex hulls, sweeps, and overlays
due Monday 9 February 2009 at 10AM

In the lectures covering the first two chapters we sketched three algorithms to compute the following: (1) the convex hull of points in the plane, (2) intersecting line segments, and (3) computing the overlay of two maps. The goal of this project is to use CGAL (an open source software available from <http://www.cgal.org>) to explore properties of these algorithms.

0. Using CGAL

Version 3.3.1 of CGAL was the most current version until very recently. For this project the differences between versions 3.3.1 or the most recent 3.4 most likely does not matter. CGAL is well documented and we recommend to start working from the examples that come with the installation of the library.

1. 2D Convex Hull

The `"/CGAL-3.3.1/examples/Convex_hull_2/"` directory contains examples to call the planar convex hull algorithms. The program `ch_example_from_cin_to_cout` is executed as

```
$ ch_example_from_cin_to_cout < ch_example_from_cin_cout.cin
```

where `$` is the command prompt. Eventually add `>` to it to redirect the output to a file. Also run `ch_graham_anderson` via the redirection of the input.

We are interested in the average running time of the algorithm and we will investigate if the algorithm is output sensitive, via the following questions:

1. Generate random points on a circle and run the computation of the convex hull. As the number of points n on input increases, report the user cpu time. Do you observe the $O(n \log(n))$ complexity?
2. Compare the running time for two different inputs:
 - (a) All input points lie on a circle, so every point is a vertex.
 - (b) All points lie in a square, so there are only four corners in the convex hull.

Do the running times differ (try large enough sets of input points) for these two input configurations?

2. Line Segment Intersection

The `"/CGAL-3.3.1/examples/Arrangement_2/sweep_line.cpp"` program demonstrates how to use CGAL to compute all intersections of a set of line segments. We will investigate the behavior of the running time of the sweep line algorithms, via the following questions:

1. How does the time increase as the number of segments increases?

Generate segments with both their end points on a circle chosen uniformly at random. Run the program and report the user cpu time and the number of intersection points. Depending on the speed of your computer, you may have to generate fairly large number of points. Interpret if the timings confirm the $O((n + I) \log(n))$ complexity.
2. Is the algorithm robust? Generate the following configurations of line segments:
 - (a) The endpoints of all segments lie on a circle and every segment passes through the center of the circle, so the center of the circle is an intersection point of high multiplicity, but it is the only intersection point.

- (b) Take the highly degenerate configuration of line segments from above, and slightly shift one endpoint so each segment gets very close to the center of the circle, but misses it.

Verify the correctness of the output for both outputs.

For the second shifted configuration, experiment with various small values of the shifting the end points. As with the first assignment, report the number of intersection points as well as the user cpu time for each run.

3. Computing the Overlay of two Maps

The creation of a planar subdivision stored as a doubly-connected edge list is illustrated by the program `"/CGAL-3.3.1/examples/Arrangement.2/overlay.cpp"`.

While the space complexity to intersect n line segments is $O(n)$, the space complexity for the map overlay problem must be higher than this. Use a CGAL program to generate an example representing the worst case space complexity.

As the size of the input dimension n grows, what is more likely to be the bottleneck for the map overlay problem: time or space? Use your experiments to illustrate your answer.

4. the Deadline is Monday 9 February 2009 at 10AM

The report you bring to class at 10AM on Monday 9 February 2009 consists of

1. Source code for the `c++` programs you used in the experiments. Also email the source code of these programs to `jan@math.uic.edu` so I can verify your results.
2. Tables with properly formatting of the essential characteristics for each experiment: the dimension of input and output, and the running time. I do not need the raw data.
3. Your interpretation of the results and conclusions.

This project must be an individual effort. Collaborations are not allowed.

If you have questions or difficulties with the project, feel free to come to my office for help.