CAD practical – programming part 6: Convex hulls

The zip file "CADCG_W6_Code.zip" available on the web page contains the code needed for generating two executables. The two programs "ch_slow" and "ch_graham" which are the topic of the present session. To compile each of these new executables, change the Build Target to "`ch_slow-static`" or "`ch_graham-static`".

Graham's algorithm

Graham's algorithm aims to efficiently find the convex hull of a set of 2D points. Compared to the "naïve" implementation of a convex-hull search, Graham's algorithm is (almost) two orders of magnitude faster. In this session, we will compare the computational times spent by a naïve convex hull algorithm (ch_slow) and Graham's algorithm (ch_graham).

1. In the file "cadcgTP6/test/convex_hull_graham/main.cc", implement the four following functions:

    - `void convex_hull_graham(std::vector<line> &lines, std::vector<npoint2> points);`
      Here implement Graham's algorithm. It receives on entry a set of (unordered) points and returns by reference a set of lines figuring the convex hull (Theory: ppt 6 pp. 33-39).

      Note 1: for ordering the points, you can take advantage of the standard algorithm `std::sort` and the functor `PointsCompare2D`.

    - `bool not_a_right_turn(const npoint2 &a, const npoint2 &b, const npoint2 &c)`
      It tests for you if a candidate point in Graham's algorithm should be kept or rejected. For this, it looks if the two lines `a-b` and `b-c` form a right turn or not (the function `crossprod` in nutil/npoint.h may be useful).

    - `void show_points(data_container &data, const std::vector<npoint2> &points);`
      `void show_lines(data_container &data, const std::vector<line> &lines);`
      They respectively add points and lines to a `data_container` object for visualization purposes. Implement these functions to be able to visualize your result (see the `data_container` class in file nutil/ndata.h).

    - You can compare your result to the one given by the naïve algorithm "ch_slow". Convex hulls should be identical (except for one, "degenerated", point).

2. Compare graphically the performances of the naïve algorithm “ch_slow” and your implementation of Graham’s algorithm “ch_graham”. To this aim, run for each algorithm the two convex hull tests which are already implemented.

   To set the test parameter with Code::Blocks :

   1. go to “project” and click on “Set program’s argument...”
   2. Select the executable “ch_slow” or “ch_graham”
   3. Complete the argument than validate
   4. Build and run with the build target correctly selected
   5. After the tests, the text file with the data is in you build directory

   The arguments for the executable are:

   ```
   test* 10 1000 5
   ```

   Where * is 1 or 2.

   This will test the called algorithm over (pseudo) random sets of 10, 15, 20,…, 1000 points and generate a text file containing some statistics about its running time. For “test1” points are randomly distributed in a square and “for test2” on a circle.

   Write a Matlab script (or other as you like) that loads the four generated text files (use the `load` function) and draw two log-log graphics (one for each test) comparing the running performances of each algorithm (see slides).

   Send the modified file (`main.cc`) and the results of your matlab computations before **November 24, 2023** at **23:59** at a.bolyn@uliege.be in the same form than the other weeks.