TD10: Convex Hulls and Digital Convex Hulls

In this TP, the idea is to implement a convex hull algorithm and to experiment its complexity (number of edges) on digital objects.

Exercise 1 Orientation predicate

Question: Implement the *Orientation*(p, q, r) predicate as discussed in the lecture.

- To avoid numerical issues, we encourage you to consider points in $\mathbb{Z}^2$ on a digital domain (instead of $\mathbb{R}^2$). Furthermore, it would help outputs since you will be able to reuse DGtal board mechanism.
- To display a line in `DGtal` between `Points` p and q, you can use the following method of the `Board` class: `board.drawLine( p[0], p[1], q[0], q[1]);`

Question: Test the *Orientation* predicate with an implementation of the segment-segment intersection detection (cf lecture). Experiment your intersection test on all cases (regular intersection, alignement, no intersection, intersection point is a vertex, ...).

The rest of the TP focuses on the implementation and the experimentation of a *convex hull algorithm*. You can choose any variant of the algorithm you want to implement. At the end we would like you:

- To test the convex hull construction on point sets defined by the digitization (at a given resolution h) of a disc (cf TP5) or an ellipse (cf DM) defined as digital sets.
- To plot (using `gnuplot`) the number of edges when $h \to 0$ in log-scale. The aim is to observe the $N^{2/3}$ behavior we discussed in the lecture for convex hull in $N \times N$ domains (also, express the relationships between N, h and the slope of the best fitting straight line in log-scale) ?

The first variant is a Graham's scan implementation on the point set ($O(n.logn)$), the second variant is based on the Melkman's algorithm on the contour points (extracted using a contour tracker) in $O(n)$. Both should only use the *Orientation*(p, q, r) predicate and point coordinate comparisons.

Exercise 2 (Rookie Mode) - Convex hull

Questions:

- Implement the Graham's scan algorithm as described in the lecture:
  - First, sort the points by polar angle (cf the `qsort` C function man page or the C++ `std::sort` to do the sort). As discussed in the lecture, you would just have to replace the comparison function/functor by the Orientation predicate with fixed p_0.
  - The second step consists in a simple stack based removal (using for example `std::queue`, cf previous TP).

Note that Graham's scan can be speed up just considering border point and not interior points (for disc/ellipse experiment).

Exercise 2 (Advanced Mode) - Convex hull

Questions:

- To implement the Melkman's algorithm, you will first have to construct a sequence of points but you should have such function from previous TP. Implement Melkman's technique (cf lecture) using the `std::deque` as data structure to store the convex hull vertices.

The overall code is not more complex than Graham's one but you'll have to dig previous TP for finding the appropriate border extraction code from a digital set.