Computational Geometry Lectures 2018-19

Olivier Devillers & Marc Pouget

Homework and Exam

`https://members.loria.fr/Olivier.Devillers/Master2-ENS-Lyon/`

The validation of the course will rely on the homework and a personal work presented in a written report and an oral presentation. This personal work can be a the presentation of a research article or a software project using CGAL.

- **Homework.**
  A homework sheet will be given at the end of the four first lectures, and must be completed for the next friday.
- **Research papers**
  A list of research papers will be available on the web site November 1st. Each student must prepare
  — A presentation (12 mn+questions) of a research paper

  You have to choose your article/project before November 15th (by mail to Olivier.Devillers@inria.fr).

  Please synchronize, two students are not allowed to choose the same paper.

Research papers for presentations

[1] Siu-Wing Cheng, Tamal K Dey, Herbert Edelsbrunner, Michael A Facello, and Shang-Hua Teng. Silver exudation. *Journal of the ACM (JACM)*, 47(5):883–904, 2000. `doi:10.1145/355483.355487`.

[2] L. P. Chew and S. Fortune. Sorting helps for Voronoi diagrams. *Algorithmica*, 18:217–228, 1997. `doi:10.1007/BF02526034`.

[3] Jeff Erickson. Dense point sets have sparse Delaunay triangulations or "...but not too nasty". *Discrete & Computational Geometry*, 33:83–115, 2005. `doi:10.1007/s00454-004-1089-3`.

[4] Leonidas Guibas and David Marimont. Rounding arrangements dynamically. *Internat. J. Comput. Geom. Appl.*, 8:157–176, 1998. `doi:10.1142/S0218195998000096`.

[5] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989. `doi:10.1016/0020-0190(89)90136-1`.

[6] John Hershberger. Stable snap rounding. *Computational Geometry*, 46(4):403–416, 2013. `doi:10.1016/j.comgeo.2012.02.011`.

[7] David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986. `doi:10.1137/0215021`.

[8] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991. `doi:10.1016/0925-7721(91)90012-4`.

[9] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997. URL: `https://www.cs.cmu.edu/~quake/robust.html`.

[10] Richard Shewchuk. Star splaying: an algorithm for repairing Delaunay triangulations and convex hulls. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 237–246. ACM, 2005. URL: `http://www.cs.berkeley.edu/~jrs/papers/star.pdf`.

Software projects

Note regarding the graphic interface:

- *Start from the CGAL demo of the* `Triangulation_2`, and replace the action done by one of its buttons by the action requested for the project. This is easier than trying to add a new button.

1 Software project: Crust

The algorithm (see lecture 'Reconstruction') is the following:
Input: S a set of n points in the plane.
Output: A set of line segments between points of S.

- Let V be the set of vertices of the Voronoi diagram of S
- Compute $Del(V \cup S)$
- Output the edges of $Del(V \cup S)$ between two points of S

1.1 Static version

Using CGAL, code the above algorithm. Be careful that you need to have two types of points in the $Del(V \cup S)$. The user of your program should have to click the input points and ask for the computation.

1.2 Incremental version

Propose an incremental version. When a new point is added the crust must be updated (without recomputing everything from scratch).

1.3 Dynamic version

Propose a dynamic version. The user should be able to add a new point or to remove an existing point (without recomputing everything from scratch).

2 Software project: experiments on Poisson Delaunay triangulation

Generate a random point set according to Poisson point distribution of density 1 and extract several parameters from its Delaunay triangulation.

Make several trial (compute average value and standard deviation). Look at how the results depend from the density.

2.1 Extremal values

In a window $[0, \sqrt{n}]^2$ compute the highest degree of a vertex, the smallest distance between two points, the longest Delaunay edge.

2.2 Origin neighborhood

Compute the distance between the origin and its closest neighbor, its second closest neighbor,$\ldots$, 20th closest neighbor. Compute the sum of the lengthes of the edges of all triangles in conflict with the origin and the perimeter of the union of these triangles. Propose a way to generate points in a lazy manner to improve running times.

3 Software project: Moving points

Let S be a set of points in the plane. The usual way to compute the Delaunay triangulation is to "spatial sort" the points (i.e. put them in an order the preserve locality, still being random enough for randomized complexity) and to incrementally insert the points, starting the point location using the previously inserted point as hint.

The nearest neighbor of the point would be a better hint but we usually don't know this point. If the point set moves (slowly) we may compute the new triangulation inserting the points in the same order using as hint the new position of the nearest neighbor (at insertion time) of the previous position of the currently inserted point. This idea is presented in [1].

3.1

Using CGAL, code the above algorithm. The user should be able to enter input sites eiter by clicking or by choosing an option "random point set". Then the motion should start. Running times must be printed (adding a pause between two consecutive steps will help the user better visualize the progress made).

It should be necessary to restart from scratch from time to time when points have moved too much.

Several possibility can be used for the motion:
— Brownian motion (random at each time step)
— Jumping balls (random direction and reflect on the boundary)
— Lloyd (see below)

3.2 Lloyd motion

Let S be a given set of sites in a 2D square. Lloyd's algorithm works iteratively as follows: at each step,

1. Compute the Voronoi diagram of S.
2. Move each site $p \in S$ to the center of mass of its Voronoi cell. (When the cell of p is not contained in the square, move p to the center of mass of its Voronoi cell clipped by the square.)
3. Update S to the set of moved sites and restart at 1.

Implement this algorithm using the 2D Delaunay triangulation package of CGAL.

Reference

[1] Olivier Devillers. Delaunay Triangulation of Imprecise Points, Preprocess and Actually Get a Fast Query Time. *Journal of Computational Geometry*, 2(1):30–45, 2011. URL: `https://hal.inria.fr/inria-00595823`, `doi:10.20382/jocg.v2i1a3`.