Computational Geometry Lectures

Olivier Devillers & Monique Teillaud

Homework and Exam

`https://members.loria.fr/Monique.Teillaud/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 and Software projects.**
  A list of research papers and software projects will be available on the web site November 10th. Each student must prepare
  — A presentation (12 mn+questions) of a research paper
  OR
  — A software project, + a talk to present your code with a demo (12mn+questions).

  You have to choose your article/project before November 22nd (by mail to Olivier.Devillers@inria.fr and Monique.Teillaud@inria.fr).

  Please synchronize, two students are not allowed to choose the same project/article.

Software projects

Note regarding the graphic interface:

- *For projects in 2D, start from the CGAL demo of the* `Triangulation_2` *package (the demo is accessible in GraphicsView), 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.*
- *For projects in 3D, we advise to start from the* `Triangulation_3_Geomview_demos` *of the* `Triangulation_3` *package, whose code is easier to read and modify than the code of* `T3_demo`.

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: Benchmarking walking in a triangulation

There are several strategies to walk between the vertices of a triangulation (see lecture on probability).

- Generate a big random set of points inside $[-0.1, 1.1] \times [-0.3, 0.3]$ and add points $(0, 0)$ and $(1, 0)$.
- Compute the Delaunay triangulation
- Compute several paths between $(0, 0)$ and $(1, 0)$: the shortest path, the upper path, the Voronoi path, the greedy path.
- Compute the Euclidean length of these paths

Repeating the experiment, give approximations of the expected values of these lengths. Your code must be parameterized by the number of points and provide a graphical view of the different paths.

Shortest path: the shortest amongst all possible paths.

Upper path: the upper boundary of the union of all triangles intersected by the line segment from $(0,0)$ to $(1,0)$.

Voronoi path: the sequence of nearest neighbors of a point moving linearly from $(0,0)$ to $(1,0)$.

Greedy path: the incremental path starting at $(0,0)$ where the successor of p is the Delaunay neighbor that minimize the angle with the line segment from p to $(1,0)$.

3 Software project: Enumerating neighbors

Given a set of points S, code an iterator that enumerates the points by increasing distance from a query point q. Notice that the k^{th} neighbor is a Delaunay neighbor of the j^{th} neighbor for some $j < k$. Thus implement a priority queue of candidates for the next neighbor that contains all the Delaunay neighbors of the already seen points.

Your code has to allow to enter the point set by clicking or by choosing an option “random point set”. The query should be entered by clicking.

You have to encapsulate this in an iterator, not just to enumerate the neighbors.

4 Software project: Minimum spanning tree

Given a set of points S, compute the Euclidean minimum spanning tree using the algorithm explained during lecture on Delaunay triangulation.

Your code has to allow to enter the point set by clicking or by choosing an option “random point set”.

5 Software project: Lloyd’s algorithm

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. The user should be able to enter input sites eiter by clicking or by choosing an option “random point set”. The user should be able to choose whether (s)he wants to proceed to the next step of the iterative algorithm either by clicking with another mouse button or if the program should do it automatically (in that case, adding a pause between two consecutive steps will help the user better visualize the progress made).

Optional: consider the same algorithm for a non-uniform mass distribution.

6 Software project: A disk moving among obstacles

Let us consider a simple polygonal domain P of the plane. A set of obstacles in P is represented by a set S of points on their boundaries. Let us consider an open disk D of constant radius r moving inside the polygonal domain P: its center c follows edges of the Voronoi diagram of S. An edge of the Voronoi diagram is said to be free if the open disk D does not contain any point of S while c traverses the whole edge (its boundary is allowed to contain points of S). The set of free edges form a (non necessarily connected) graph in the plane. A free path for D is defined as a path in this graph.

Using the 2D Delaunay triangulation package of CGAL, implement a program that, for two points p, q on the graph of free edges, computes and displays a free path from p to q for D, or answers that there is no free path.

The radius r of D can be left as a constant, say 1 (then the disk should be displayed to the user before (s)he starts entering the domain P and the obstacles). The user should be allowed to enter

- the polygonal domain P
- and the set S

by clicking points or line segments, which the program will automaticaly discretize. (S)he will also click to enter the two positions p, q.

Optional:

1- Note that some edges may be only partially free. Consider such partially free edges.

2- Consider the case when p, q are not on the graph of free edges.

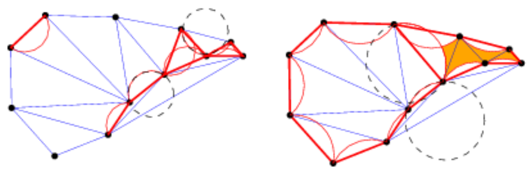
7 Software project: 3D Alpha Shapes

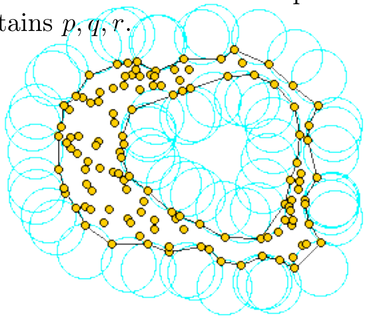
Alpha shapes can be seen a generalization of the convex hull of a finite point set.

Let S be a set of points in $\mathbb{R}^3$ and $\alpha > 0$ a real number.

Three points $p, q, r \in S$ form a facet of the alpha shape of S iff there exists an open ball of radius $\sqrt{\alpha}$ empty of any point of S and whose boundary contains p, q, r.

(illustration in 2D)





Propose a simple method to compute the alpha shape of S from the Delaunay triangulation of S computed with CGAL, *without using the alpha shapes package of CGAL.*

Implement your method. Your implementation should allow a user to choose the value of α. You can test your software on the input files contained in the zip file `data.zip`.

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] 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`.

[4] 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`.

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

[6] 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`.

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

[8] 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`.

[9] 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`.

[10] 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`.

[11] 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`.