Computational topology: Lecture 7

Alberto Paoluzzi

March 22, 2019

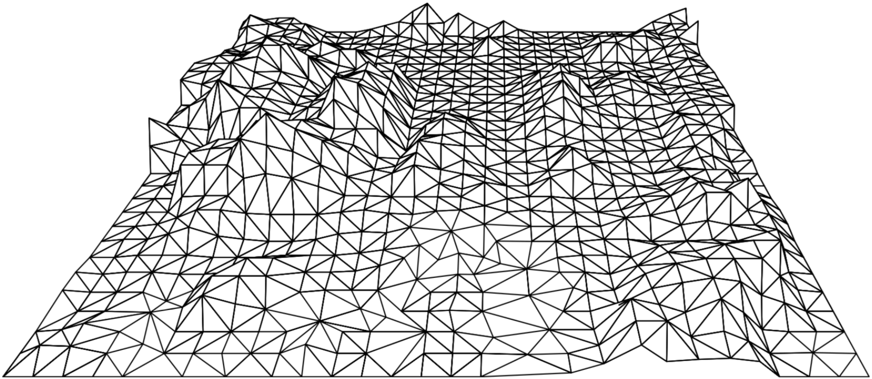
1. Delaunay triangulations[1]
2. Voronoi complexes[2]
3. Julia Packages
4. Examples and implementation

[1] from: de Berg, Otfried Cheong, van Kreveld, Overmars: Computational Geometry, Algorithms and Applications, Third Edition, Springer.
[2] idem.

Delaunay triangulations[3]

[3]from: de Berg, Otfried Cheong, van Kreveld, Overmars: Computational Geometry, Algorithms and Applications, Third Edition, Springer.

Triangulation example (terrain map)



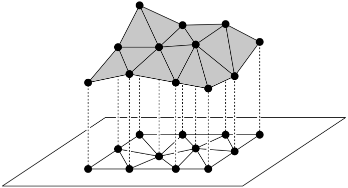
Triangulation example (terrain map)

We first determine a triangulation of P: a planar subdivision whose bounded faces are triangles and whose vertices are the points of P.

We then lift each sample point to its height, mapping every triangle in the triangulation to a triangle in 3-space

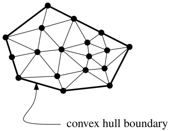
We get is a polyhedral terrain, the graph of a piecewise linear continuous function

The polyhedral terrain as an approximation of the original terrain.



Triangulations of Planar Point Sets

Let $P := \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane. To be able to formally define a triangulation of P, we first define a *maximal planar subdivision* as a subdivision $\mathcal{S}$ such that no edge connecting two vertices can be added to $\mathcal{S}$ without destroying its planarity. In other words, any edge that is not in $\mathcal{S}$ intersects one of the existing edges. A *triangulation* of P is now defined as a maximal planar subdivision whose vertex set is P.



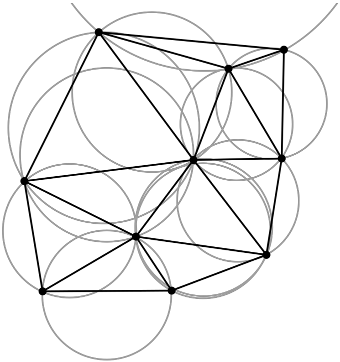
Theorem 9.1 *Let P be a set of n points in the plane, not all collinear, and let k denote the number of points in P that lie on the boundary of the convex hull of P. Then any triangulation of P has $2n-2-k$ triangles and $3n-3-k$ edges.*

The Delaunay Triangulation

Definition

Delaunay triangulation for a set P of discrete points in a plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$

Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid sliver triangles.



Delaunay Triangulation properties

The Delaunay triangulation is a triangulation which is equivalent to the nerve of the cells in a Voronoi diagram,

it is the triangulation of the convex hull of the points in the diagram in which every circumcircle of a triangle is an empty circle

an edge is illegal if we can locally increase the smallest angle by flipping that edge.

A Delaunay triangulation is unique iff the circumcircle of every triangle contains exactly three points on its circumference: the vertices of the triangle.

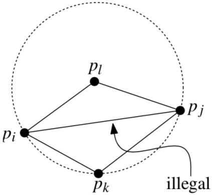
For instance, the Delaunay diagram of the four vertices of a square is a square, and can be converted into a triangulation in two different ways

Computing the Delaunay Triangulation

Observation 9.3 *Let $\mathcal{T}$ be a triangulation with an illegal edge e. Let $\mathcal{T}'$ be the triangulation obtained from $\mathcal{T}$ by flipping e. Then $A(\mathcal{T}') > A(\mathcal{T})$.*

It turns out that it is not necessary to compute the angles $\alpha_1, \ldots, \alpha_6, \alpha'_1, \ldots, \alpha'_6$ to check whether a given edge is legal. Instead, we can use the simple criterion stated in the next lemma. The correctness of this criterion follows from Thales's Theorem.

Lemma 9.4 *Let edge $\overline{p_i p_j}$ be incident to triangles $p_i p_j p_k$ and $p_i p_j p_l$, and let C be the circle through p_i, p_j, and p_k. The edge $\overline{p_i p_j}$ is illegal if and only if the point p_l lies in the interior of C. Furthermore, if the points p_i, p_j, p_k, p_l form a convex quadrilateral and do not lie on a common circle, then exactly one of $\overline{p_i p_j}$ and $\overline{p_k p_l}$ is an illegal edge.*



Computing the Delaunay Triangulation

We define a *legal triangulation* to be a triangulation that does not contain any illegal edge. From the observation above it follows that any angle-optimal triangulation is legal. Computing a legal triangulation is quite simple, once we are given an initial triangulation. We simply flip illegal edges until all edges are legal.

Algorithm LEGALTRIANGULATION($\mathcal{T}$)
Input. Some triangulation $\mathcal{T}$ of a point set P.
Output. A legal triangulation of P.

```
while 𝒯 contains an illegal edge p_i p_j
    do (* Flip p_i p_j *)
        Let p_i p_j p_k and p_i p_j p_l be the two triangles adjacent to p_i p_j.
        Remove p_i p_j from 𝒯, and add p_k p_l instead.
return 𝒯
```

Computing the Delaunay Triangulation: Divide and conquer

Recursively draws a line to split the vertices into two sets

The Delaunay triangulation is computed for each set, and then the two sets are merged along the splitting line.

A divide and conquer paradigm to performing a triangulation in d dimensions is presented in "DeWall: A fast divide and conquer Delaunay triangulation algorithm in E^d" by P. Cignoni, C. Montani, R. Scopigno.

Delaunay triangulations rely on fast operations for detecting if a point is within a triangle's circumcircle and an efficient data structure for storing triangles and edges (LAR is OK?)

In 2D, wheater point D lies in the circumcircle of A, B, C is tested by evaluating a 3×3 determinant

Voronoi complexes[13]

[13] idem.

Convex polygons

To test if a polygon is convex, every point of the polygon should be level with or behind each line segment

Consider each set of three points along the polygon. If every angle is $\leq \pi$ you have a convex polygon

Definition and Basic Properties

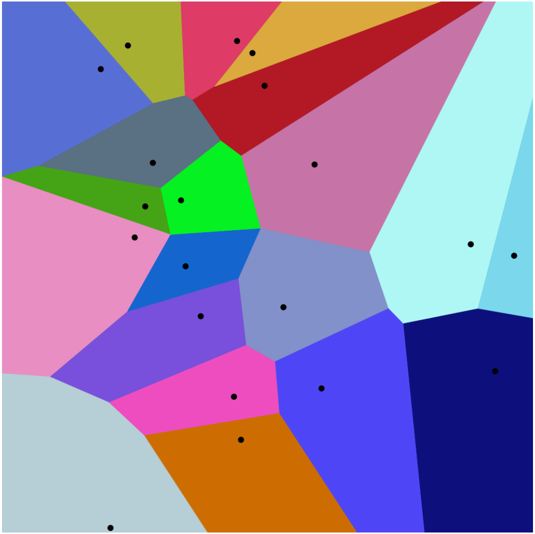
Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane.

That set of points (called seeds, sites, or generators) is specified beforehand, and for each seed there is a corresponding region consisting of all points closer to that seed than to any other.

These regions are called Voronoi cells.

The Voronoi diagram of a set of points is dual to its Delaunay triangulation.

Voronoi diagrams



Voronoi properties

The dual graph for a Voronoi diagram (in the case of a Euclidean space with point sites) corresponds to the Delaunay triangulation for the same set of points

The closest pair of points corresponds to two adjacent cells in the Voronoi diagram

Then two points are adjacent on the convex hull if and only if their Voronoi cells share an infinitely long side

Computing the Voronoi Diagram

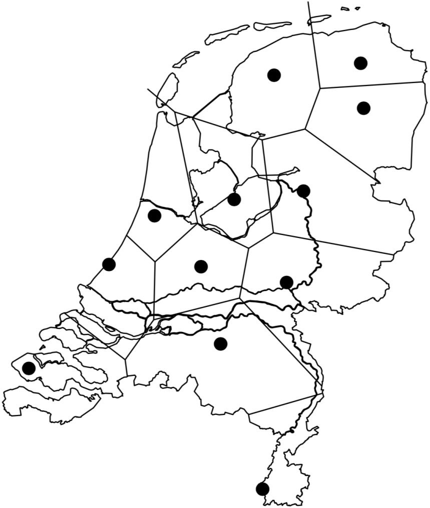
Fortune's algorithm, an $O(n \log(n))$ algorithm for generating a Voronoi diagram from a set of points in a plane

Lloyd's algorithm and its generalization via the Linde–Buzo–Gray algorithm (aka k-means clustering), utilize Voronoi tessellations in spaces of arbitrary dimension to iteratively converge towards a specialized form of the Voronoi diagram, called Centroidal Voronoi tessellation, where each site is also the geometric center (barycenter) of its cell

Starting with a Delaunay triangulation (obtain the dual):

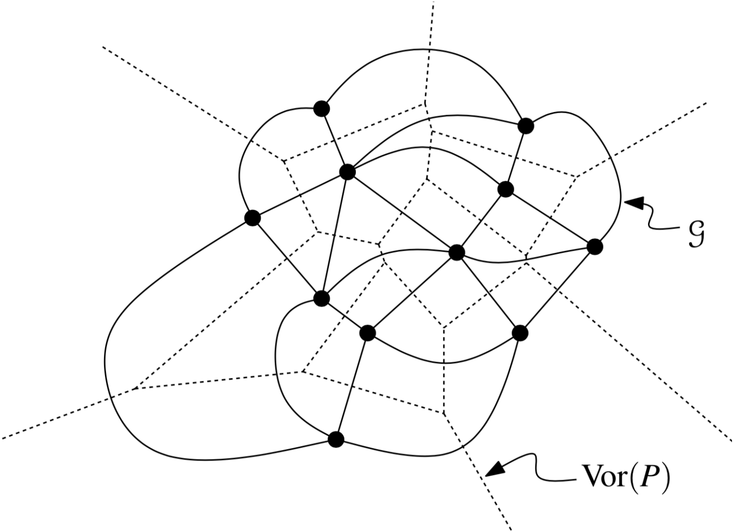
Bowyer–Watson algorithm, an $O(n \log(n))$ to $O(n^2)$ algorithm for generating a Delaunay triangulation in any number of dimensions, from which the Voronoi diagram can be obtained

Post office problem



Planar graphs

V, E, F one-to-one to F, E, V

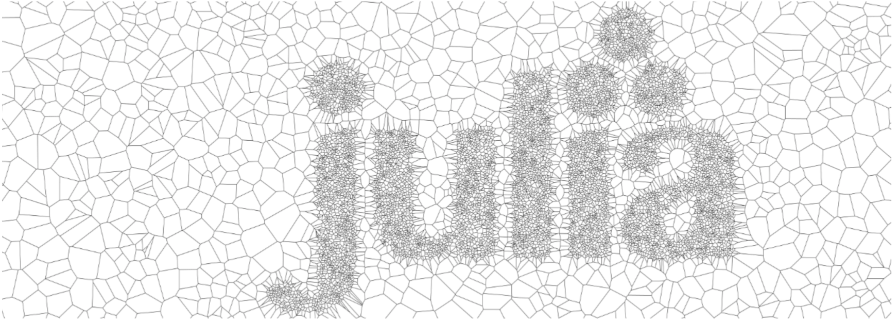


Julia Packages

https://github.com/JuliaGeometry/VoronoiDelaunay.jl

VoronoiDelaunay.jl

build passing | Julia v0.6 v0.2.0 Tests Pass | coverage 73%



Fast, robust construction of 2D Delaunay and Voronoi tessellations on generic point types. Implementation follows algorithms described in the Arepo paper and used (for e.g.) in the Illustris Simulation. License: MIT. Bug reports welcome!

Read the source !!

Look and enjoy the coding style ...

728 lines (641 sloc) | 22.8 KB

Raw Blame History

```julia
module VoronoiDelaunay

# Fast, robust 2D Voronoi/Delaunay tesselation
# Implementation follows algorithms described in http://arxiv.org/abs/0901.4107
# and used (for e.g.) in the Illustris Simulation
# http://www.illustris-project.org/
#
# Author: Ariel Keselman (skariel@gmail.com)
# License: MIT
# Bug reports welcome!

export
DelaunayTessellation, DelaunayTessellation2D, sizehint!, isexternal,
min_coord, max_coord, locate, movea, moveb, movec,
delaunayedges, voronoiedges, voronoiedgeswithoutgenerators,
iterate, findindex, push!,
Point, Point2D, AbstractPoint2D, getx, gety, geta, getb, getc,
getgena, getgenb, getplotxy

using GeometricalPredicates
import GeometricalPredicates: geta, getb, getc
```

Examples and implementation

Exercise

1. generate a set of random points within $[0, 1]^2$

Exercise

1. generate a set of random points within $[0,1]^2$
2. generate a Delaunay tessellation with `JuliaGeometry/VoronoiDelaunay.jl`

Exercise

1. generate a set of random points within $[0, 1]^2$
2. generate a Delaunay tessellation with `JuliaGeometry/VoronoiDelaunay.jl`
3. import the result in `cvdlab/LinearAlgebraicRepresentation.jl` aka `Lar`

Exercise

1. generate a set of random points within $[0, 1]^2$
2. generate a Delaunay tessellation with `JuliaGeometry/VoronoiDelaunay.jl`
3. import the result in `cvdlab/LinearAlgebraicRepresentation.jl` aka `Lar`
4. visualize with `cvdlab/Plasm.jl`

Exercise

1. generate a set of random points within $[0, 1]^2$
2. generate a Delaunay tessellation with `JuliaGeometry/VoronoiDelaunay.jl`
3. import the result in `cvdlab/LinearAlgebraicRepresentation.jl` aka `Lar`
4. visualize with `cvdlab/Plasm.jl`
5. as above: generate, export and visualize the corresponding Voronoi complex