Computational topology: Lecture 14

Alberto Paoluzzi

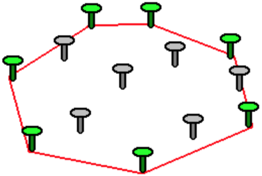
May 3, 2019

Convex hull

Definitions

Given a discrete set S of points:

1. intersection of all convex sets containing S;
2. minimum convex set containing S
3. set spanned by all convex combinations of S points



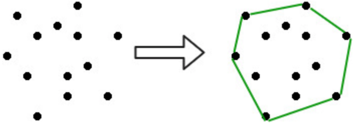
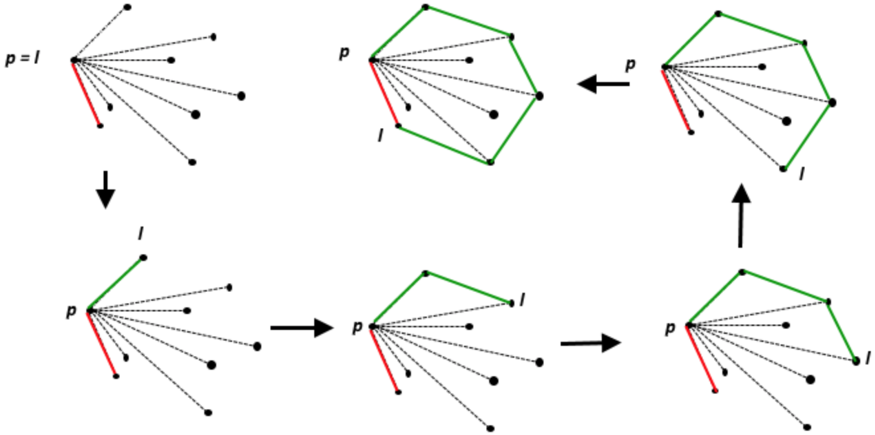


Figure 1: Some examples

Jarvis algorithm (1973)

Gift Wrap Algorithm (Jarvis March Algorithm) to find Convex Hull

$O(nh)$ complexity, where $n = \#S$, and h is the number of points on the convex hull.



Chan's algorithm (1996)

Timothy M. Chan. "Optimal output-sensitive convex hull algorithms in two and three dimensions". *Discrete and Computational Geometry*, Vol. 16, pp.361–368. 1996.

Chan's algorithm in the planar case: the algorithm combines an $O(n \log n)$ algorithm (Graham scan-line, for example) with Jarvis march $O(nh)$, in order to obtain an optimal $O(n \log h)$ time.

Execution demo on Wikipedia

Alpha shapes

Goal: study the shape of a set of points

H. Edelsbrunner, *A short course in computational geometry and topology*, Springer, 2014

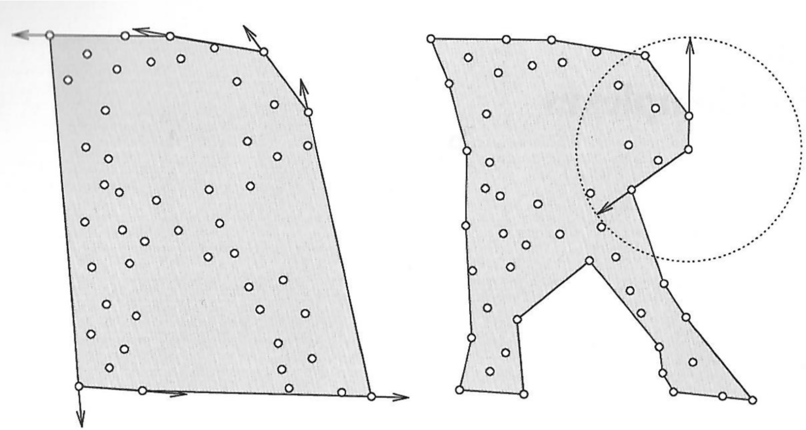


Figure 3: Jarvis construction

α-Hull and α-Shape

H. Edelsbrunner, D. Kirkpatrick and R. Seidel, "On the shape of a set of points in the plane," in *IEEE Transactions on Information Theory*, vol.29, no.4, 1983

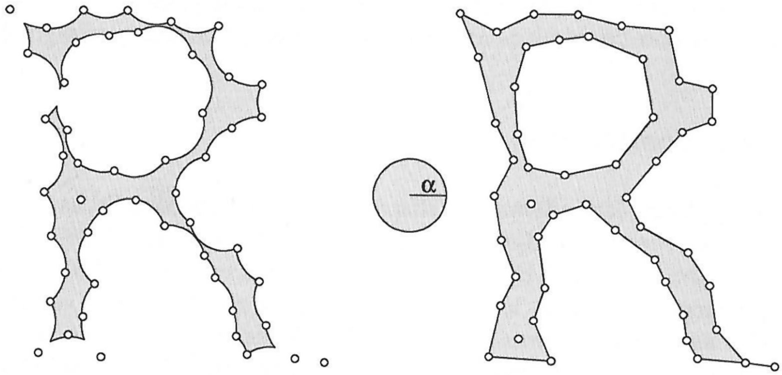


Figure 4: A set of points sampling the letter ‘R’

α-Hull and α-Shape

Let $\alpha \geq 0$ be a fixed radius:

- $D_x(\alpha)$ for a closed disk with center x;
- if $D_x(\alpha) \cap S = \emptyset$: then the disk is empty;
- α-hull of S is the complement of the union of empty disks of radius α;
- $\alpha = 0 \to S$;
- $\alpha = \infty \to \text{conv}(S)$

Union of disks and Voronoi decomposition

Union of disks of same radius α centered on S points:

$$\mathbb{U}_S(\alpha) = \bigcup_{s \in S} D_s(\alpha) = \bigcup_{s \in S} R_s(\alpha), \quad \text{where} \quad R_s(\alpha) = V_S \cap D_s(\alpha)$$

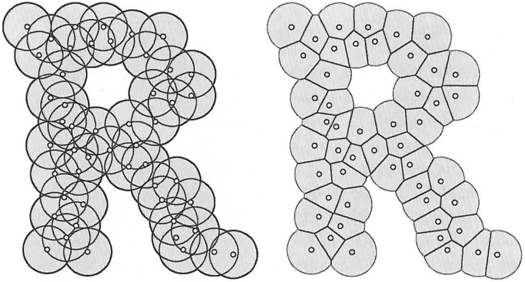


Figure 5: The Voronoi decomposition of the the union

α-Complex (in 2D)

According to Delaunay triangulation, there is an edge between two points if their regions intersect in a common edge, and a triangle between three points if their regions intersect in a common point

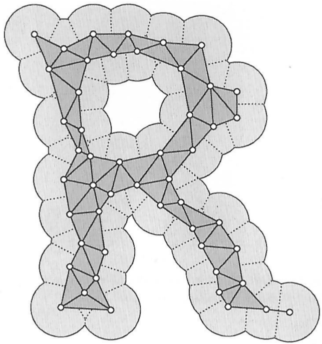


Figure 6: Union of disks decomposed by Voronoi and Delaunay complexes

α-Complex

$$A(\alpha) = \{\sigma \in K | \alpha_\sigma \leq \alpha\},$$

where K is the Delaunay triangulation of S

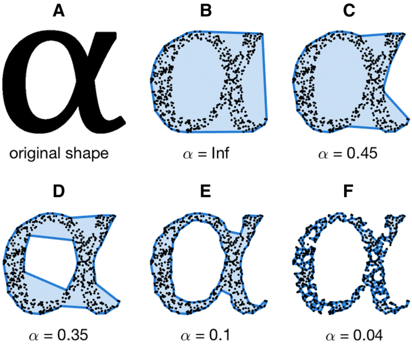


Figure 7: α-Complexes, for varying α

Filtration (to be continued)

aaaa

α-Shapes are closely related to α-complexes, subcomplexes of the Delaunay triangulation of the point set.

Each edge or triangle of the Delaunay triangulation may be associated with a characteristic radius, the radius of the smallest empty circle containing the edge or triangle.

For each real number α, the α-complex of the given set of points is the simplicial complex formed by the set of edges and triangles whose radii are at most $1/\alpha$.

Some implementation

Convex hull

Introduction

Qhull computes the convex hull, Delaunay triangulation, Voronoi diagram, halfspace intersection about a point, furthest-site Delaunay triangulation, and furthest-site Voronoi diagram

- The source code runs in 2-d, 3-d, 4-d, and higher dimensions
- Qhull implements the Quickhull algorithm for computing the convex hull
- It handles roundoff errors from floating point arithmetic
- It computes volumes, surface areas, and approximations to the convex hull.

http://www.qhull.org

Convex hull

2D example

```
julia> using QHull

julia> p2 = rand(10,2)
10×2 Array{Float64,2}:
 0.59823   0.964113
 0.500987  0.656277
 0.664168  0.566141
 0.151405  0.639172
 0.735322  0.198219
 0.103684  0.661002
 0.152175  0.405152
 0.380007  0.180538
 0.515186  0.288242
 0.757414  0.20092

julia> ch2 = chull(p2)
Convex Hull of 10 points in 2 dimensions
Hull segment vertex indices:
[1, 6, 7, 8, 5, 10]
Points on convex hull in original order:
[0.59823 0.964113; 0.735322 0.198219; ... ; ... ]

julia> ch2.points         # original points
10×2 Array{Float64,2}:
 0.59823   0.964113
 0.500987  0.656277
 0.664168  0.566141
 0.151405  0.639172
 0.735322  0.198219
 ....
 0.103684  0.661002
 0.152175  0.405152
 0.380007  0.180538
 0.515186  0.288242
 0.757414  0.20092

julia> ch2.vertices
# indices to line segments forming the convex hull
6-element Array{Int64,1}:
  1
  6
  7
  8
  5
 10

julia> ch2.simplices
# the simplexes forming the convex hull
6-element Array{Array{Int64,1},1}:
 [1, 6]
 [1, 10]
 [7, 6]
 [7, 8]
 [5, 10]
 [5, 8]
```

Convex hull

3D example

```
julia> using QHull

julia> p3 = rand(20,3)

julia> ch3 = chull(p3)
Convex Hull of 20 points in 3 dimensions
Hull segment vertex indices:
[1, 2, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 19]
Points on convex hull in original order:
[0.604637 0.0159875 0.0282537; 0.203942 0.883561 0.665614; ... ;
0.733554 0.946853 0.736748; 0.689781 0.969748 0.688725]

julia> ch3.vertices'
1×13 LinearAlgebra.Adjoint{Int64,Array{Int64,1}}:
 1  2  5  6  7  9  10  11  12  13  14  15  19

julia> @show p3
p3 = [0.604637 0.0159875 0.0282537; 0.203942 0.883561 0.665614;
0.0998071 0.583693 0.387648; 0.322411 0.271389 0.385281; 0.0854707
0.867559 0.407252; 0.544968 0.0609553 0.812328; 0.464687 0.905105
0.74164; 0.588225 0.549453 0.371593; 0.0359141 0.300891 0.357064;
0.138973 0.28496 0.127025; 0.732241 0.90114 0.0442693; 0.684981
0.0743128 0.427316; 0.598535 0.635914 0.881108; 0.623852 0.944785
0.413215; 0.733554 0.946853 0.736748; 0.598131 0.633203 0.293046;
0.37739 0.237786 0.397549; 0.474956 0.483842 0.66596; 0.689781
0.969748 0.688725; 0.487894 0.385103 0.604854]
```

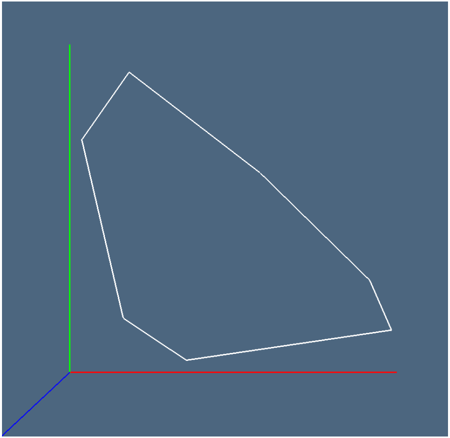
```
julia> @show ch3.simplices;
ch3.simplices = Array{Int64,1}[[3, 8, 7], [3,
[18, 8, 7], [16, 8, 14], [5, 18, 7], [4, 3, 14
18, 11], [9, 18, 8], [9, 16, 11], [9, 16, 8],
 13], [20, 13, 11], [20, 16, 11], [20, 12, 13]
 12, 14], [1, 13, 7], [1, 5, 7], [1, 5, 13],
 13], [19, 18, 11], [19, 5, 18]]
```

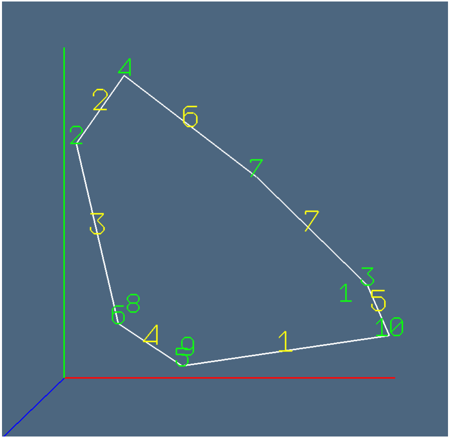
Convex hull (2D)

```
using LinearAlgebraicRepresentation, Plasm
Lar = LinearAlgebraicRepresentation

V2 = convert(Lar.Points, p2')
FV2 = ch2.simplices
Plasm.view(V2, FV2)

Plasm.view( Plasm.numbering(0.5)((V2, [[[k]
    for k=1:size(V2,2)], FV2])) )
```

Convex hull (2D)

Visualization





Convex hull (3D)

```
V3 = convert(Lar.Points, p3')
FV3 = ch3.simplices
Plasm.view(V3, FV3)

EV3 = Lar.simplexFacets(FV3)

Plasm.view( Plasm.numbering(0.25)((V3, [[[k]
    for k=1:size(V3,2)], EV3, FV3])) )
```

Convex hull (3D)

Visualization