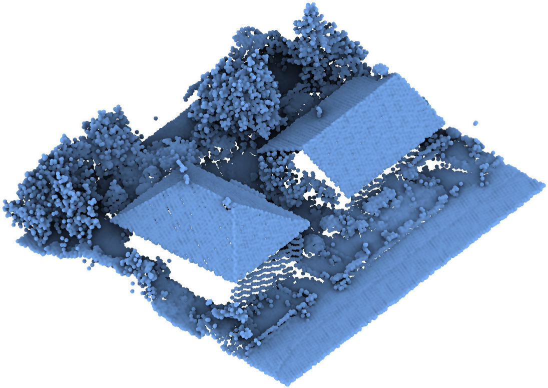
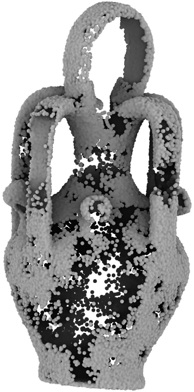
An Introduction to Point Cloud Processing

Computational Geometry (LMECA2170)

Point Clouds are the simplest kind of geometric data

$$P = \{p_1, ..., p_N\} \subset \mathbb{R}^d \qquad d = 2 \text{ or } 3$$





https://github.com/potree/potree

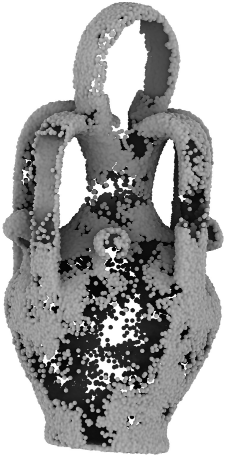
Point Cloud Acquisition

Retz (Austria)

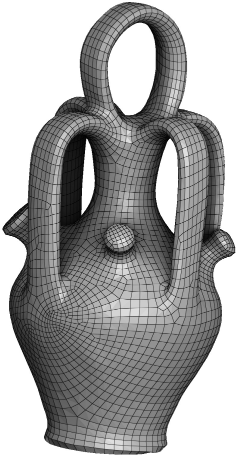
Lidar scanners

Our final goal: Surface reconstruction

From point clouds...

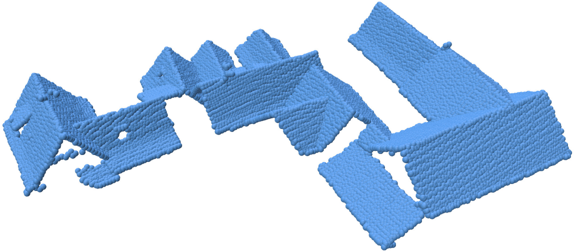


... to surface meshes

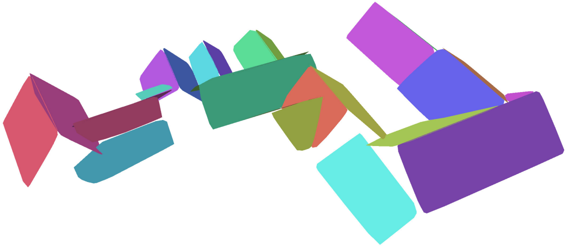


Our final goal: Surface reconstruction

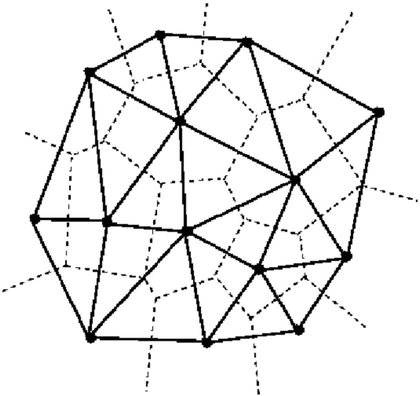
From point clouds...



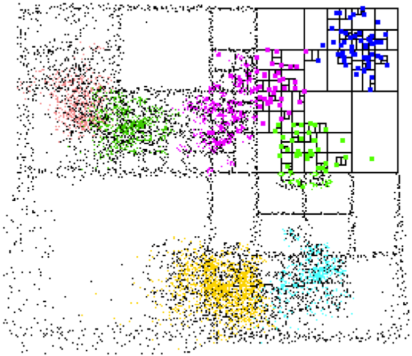
... to surface meshes



Requirements



Delaunay triangulation



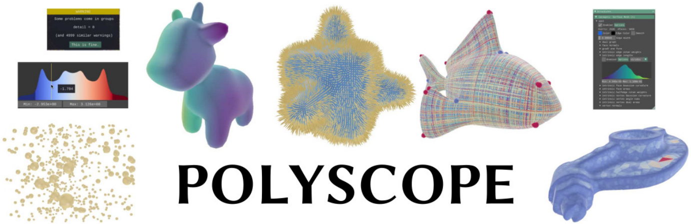
Efficient k-nearest neighbors search

... and basic linear algebra (matrices and eigenvalues)

Software Environment

Python using:

- *numpy* and *scipy*
- *mouette*[1] for data loading and geometry utilities
- *polyscope*[2] for visualization

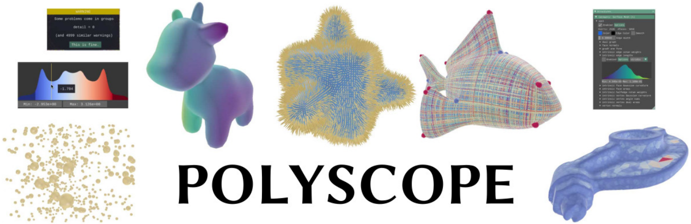


[1] https://gcoiffier.github.io/mouette/
[2] https://polyscope.run/py/

Software Environment

Python using:

- *numpy* and *scipy*
- *mouette*[1] for data loading and geometry utilities
- *polyscope*[2] for visualization



DEMO

[1] https://gcoiffier.github.io/mouette/
[2] https://polyscope.run/py/

Part 1: The Point Cloud Processing Toolbox

1. Nearest Neighbors and the k-NN graph
2. Estimating Normals
   - Best fitting hyperplane
   - Consistent Orientation
3. Estimating Density
4. Clustering and Primitive Fitting
   - The RANSAC Algorithm

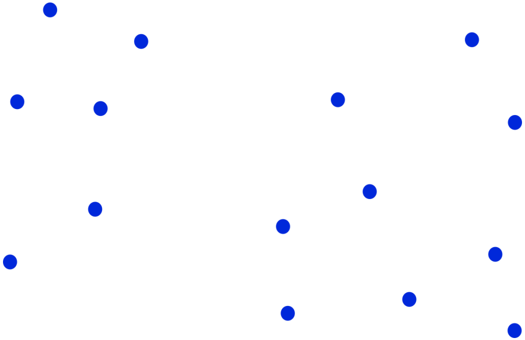
Nearest Neighbors and the k-NN graph

The k-nn graph: Finding structure through locality

Definition

A graph $G = (P, E)$ where:

- $P = \{p_1, ..., p_N\}$ are the N points of the cloud
- Oriented edge $p_i \rightarrow p_j \in E \Leftrightarrow p_j$ is among the k nearest points of p_i

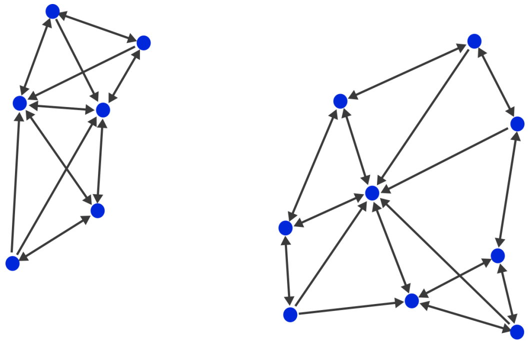


The k-nn graph: Finding structure through locality

Definition

A graph $G = (P, E)$ where:

- $P = \{p_1, ..., p_N\}$ are the N points of the cloud
- Oriented edge $p_i \rightarrow p_j \in E \Leftrightarrow p_j$ is among the k nearest points of p_i



The k-nn graph: Finding structure through locality

Definition

A graph $G = (P, E)$ where:

- $P = \{p_1, ..., p_N\}$ are the N points of the cloud
- Oriented edge $p_i \to p_j \in E \Leftrightarrow p_j$ is among the k nearest points of p_i

Properties of the k-NN graph

- The graph is k-regular
- The graph is **sparse**: $|E| = kN = \Theta(N)$

The k-nn graph: Finding structure through locality

Definition

A graph $G = (P, E)$ where:

- $P = \{p_1, ..., p_N\}$ are the N points of the cloud
- Oriented edge $p_i \to p_j \in E \Leftrightarrow p_j$ is among the k nearest points of p_i

Properties of the k-NN graph

- The graph is k-regular
- The graph is **sparse**: $|E| = kN = \Theta(N)$

DEMO

The k-nn graph: Finding structure through locality

Representation

Adjacency list $L \in \mathbb{N}^{N \times 3}$

$L_{i,j}$ = index of the j-th nearest neighbor of point p_i.

Adjacency matrix $A \in \mathbb{R}^{N \times N}$

$$A_{i,j} = \begin{cases} 1 & \text{if } j \text{ is among the } k \\ & \text{nearest neighbors of } p_i \\ 0 & \text{otherwise} \end{cases}$$

Can be weighted (replace 1 by $d(p_i, p_j)$)

The k-nn graph: Finding structure through locality

Representation

Adjacency list $L \in \mathbb{N}^{N\times 3}$

$L_{i,j}$ = index of the j-th nearest neighbor of point p_i.

Adjacency matrix $A \in \mathbb{R}^{N\times N}$

$$A_{i,j} = \begin{cases} 1 & \text{if } j \text{ is among the } k \text{ nearest neighbors of } p_i \\ 0 & \text{otherwise} \end{cases}$$

Can be weighted (replace 1 by $d(p_i, p_j)$)

Variants

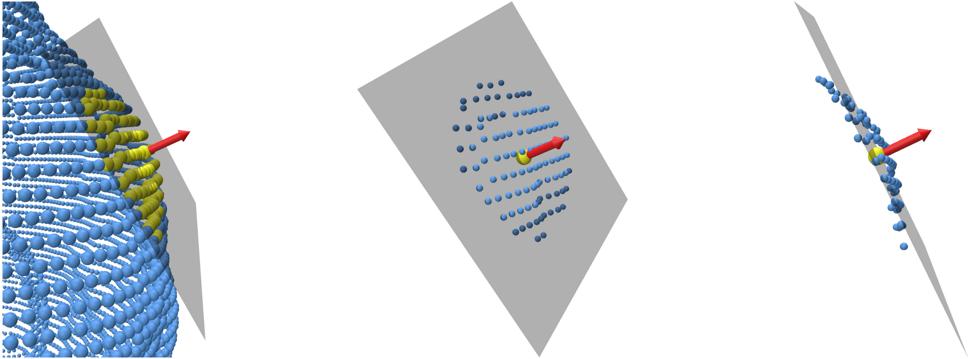
Unoriented: Consider edge (i, j) if p_j is among the k nearest points of p_i or vice-versa (Not k-regular anymore!)

Weighted: Add to edge (i, j) a weight $w_{i,j} \in \mathbb{R}$. Typically, $w_{i,j} = d(p_i, p_j)$

Estimating Normals

Estimating normals

The normal vector is the orthogonal vector of the tangent plane at point p_i
$\hookrightarrow$ Find the plane that best interpolates the points locally.



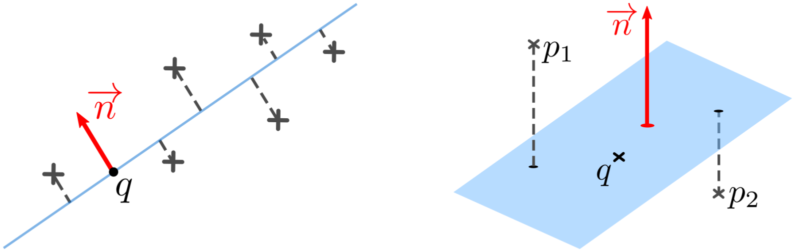
DEMO

Best fitting hyperplane

A hyperplane (line in 2D, plane in 3D) Π is completely defined by a normal **unitary** vector $\overrightarrow{n} \in \mathbb{R}^d$ and any point $q \in \mathbb{R}^d$ by the following implicit equation:

$$\Pi = \left\{ p \in \mathbb{R}^d \;\middle|\; (p - q).\overrightarrow{n} = 0 \right\}$$

The distance to plane Π is given by $d(p, \Pi) = |(p - q).\overrightarrow{n}|$



Best fitting hyperplane

A hyperplane (line in 2D, plane in 3D) Π is completely defined by a normal **unitary** vector $\overrightarrow{n} \in \mathbb{R}^d$ and any point $q \in \mathbb{R}^d$ by the following implicit equation:

$$\Pi = \left\{ p \in \mathbb{R}^d \;|\; (p-q).\overrightarrow{n} = 0 \right\}$$

The distance to plane Π is given by $d(p,\Pi) = |(p-q).\overrightarrow{n}|$

Problem formulation

Given points $p_1, ..., p_N \in \mathbb{R}^d$, find the plane $\Pi(\overrightarrow{n}, q)$ that minimizes the sum of square distances to each point:

$$\min_{\overrightarrow{n},q} E(p_i, \overrightarrow{n}, q) \quad \text{s.t.} \quad ||\overrightarrow{n}|| = 1 \quad \text{where} \quad E = \sum_{i=1}^{N} ((p_i - q).\overrightarrow{n})^2$$

Best fitting plane resolution 1: Finding the origin point

Let $c = \frac{1}{N}\sum_{i=1}^{N} p_i$ be the *centroid* of the point cloud.

> The best fitting plane goes through c. As a consequence, we can reduce the search to planes of the form $\Pi(\overrightarrow{n}, c)$.

Proof. Compute:

$$\frac{\partial E}{\partial q} = -2\sum_i ((p_i - q).n)\, n = -2\left((\sum_i p_i - q).n\right) n = -2N\,[(c-q).n]\, n$$

and since $||n|| = 1$, solving for $\frac{\partial E}{\partial q} = 0$ implies that $(c-q).n = 0$ which means that c belongs to the plane. □

Principal Axes and Best-Fit Planes, with Applications, Brown, 1976

Best fitting plane resolution 2: Finding the best normal vector

Matrix Notation

Define the **scatter matrix** M:

$$M = \begin{pmatrix} p_1 - c \\ p_2 - c \\ \vdots \\ p_N - c \end{pmatrix} \in \mathbb{R}^{N \times d}$$

and the **covariance matrix** K:

$$K = M^T M \in \mathbb{R}^{d \times d}$$

Reformulation

The problem of finding the best fitting plane can be written as:

$$\min_{n \in \mathbb{R}^d} ||Mn||^2 \quad \text{s.t.} \quad ||n|| = 1$$

or

$$\min_{n \in \mathbb{R}^d} n^T K n \quad \text{s.t.} \quad n^T n = 1$$

The function E is minimized when n is a (unitary) eigenvector associated with the smallest eigenvalue of K.

The function E is minimized when n is a (unitary) eigenvector associated with the smallest eigenvalue of K.

Proof. K is symetric positive definite. Let $0 < \lambda_1 < \lambda_2 < ... < \lambda_d$ be its eigenvalues.
We can write: $K = U^T \Lambda U$ with $U^T U = I$ and $\Lambda = \text{diag}(\lambda_1, ..., \lambda_d)$.
Since U is invertible, if we write $y = Un$, we have:

$$\min_n n^T K n = \min_y y^T \Lambda y = \min_y \sum_i \lambda_i y_i^2 \geqslant \sum_i \lambda_1 y_i^2 = \lambda_1 ||y||^2.$$

Since U is orthogonal, $||y|| = ||n|| = 1$ so the minimum of $n^T K n$ is $\geqslant \lambda_1$. But this value is achieved for $y = (1, 0, ..., 0)$. In that case, $n = U^T y$ is the first column of U, i.e. an eigenvector associated to λ_1. □

Algorithm for finding the best fitting plane

Best-fitting plane

input: points $\{p_1, ..., p_N\} \subset \mathbb{R}^d$

1. Compute the scatter matrix $M = (p_i - c) \in \mathbb{R}^{N \times d}$
2. Compute the correlation matrix $K = M^T M \in \mathbb{R}^{d \times d}$
3. Compute the eigen decomposition $(\lambda_1, e_1), (\lambda_2, e_2), (\lambda_3, e_3)$ of K $(\lambda_1 \geqslant \lambda_2 \geqslant \lambda_3)$

return e_1

Algorithm for finding the best fitting plane

Best-fitting plane

input: points $\{p_1, ..., p_N\} \subset \mathbb{R}^d$

1. Compute the scatter matrix $M = (p_i - c) \in \mathbb{R}^{N \times d}$
2. Compute the correlation matrix $K = M^T M \in \mathbb{R}^{d \times d}$
3. Compute the eigen decomposition $(\lambda_1, e_1), (\lambda_2, e_2), (\lambda_3, e_3)$ of K $(\lambda_1 \geqslant \lambda_2 \geqslant \lambda_3)$

return e_1

Best-fitting plane (variant)

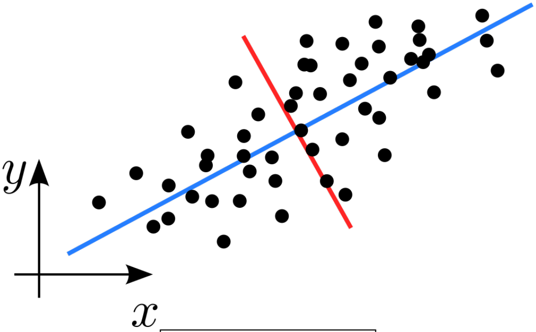
A mathematically equivalent formulation (saves the multiplication $M^T M$):

1. Compute the scatter matrix $M = (p_i - c) \in \mathbb{R}^{N \times d}$
2. Compute the SVD of M^T: $M^T = U \Sigma V^T$

return the first line of U

Digression: The Principal Component Analysis

Singular values of $M = U\Sigma V^T$ and lines of U give us the directions where the dataset varies the most/the less.



DEMO

Back to normal estimation

Normal estimation algorithm

For each point p_i in the point cloud:

- Compute $q_1, ...q_k$ the k nearest neighbors of p_i in P.
- Find the best fitting plane Π through $\{p_i, q_1, ..., q_k\}$
- The normal at p_i is the normal of the best fitting plane.

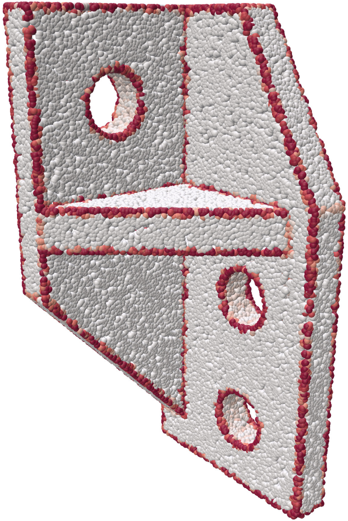
DEMO

Digression: Curvature Estimation and Edge Detection

Let $\lambda_1 \leqslant \lambda_2 \leqslant \lambda_3$ be the eigenvalues of the correlation matrix K.

$$\eta = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}$$

is a good indicator of how much the considered points deviate from a perfect plane.



Efficient Simplification of Point-Sampled Surfaces, Pauly et al., 2002

Digression: Curvature Estimation and Edge Detection

Crease Detection

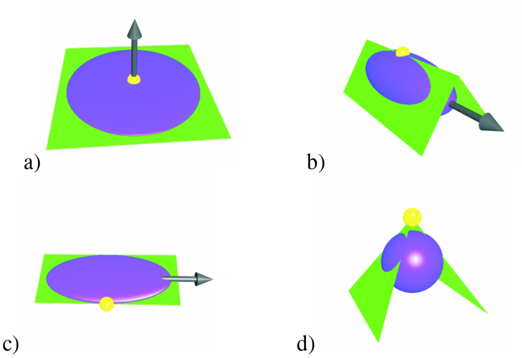
$$\eta_{\text{crease}} = \frac{\max\{\lambda_2 - \lambda_1, |\lambda_3 - \lambda_2 - \lambda_1|\}}{\lambda_3}$$

Edge Detection

$$\eta_{\text{edge}} = \frac{|\lambda_3 - 2\lambda_2|}{\lambda_3}$$

Corner detection

$$\eta_{\text{edge}} = \frac{\lambda_3 - \lambda_1}{\lambda_3}$$



Feature Extraction from Point Clouds, Gumhold et al., n.d.

Consistent Orientation

The best fitting plane only provides the normal's *direction* but not orientation.
Finding a globally consistent orientation is NP-complete.
$\hookrightarrow$ Approximation via propagation to neighbors

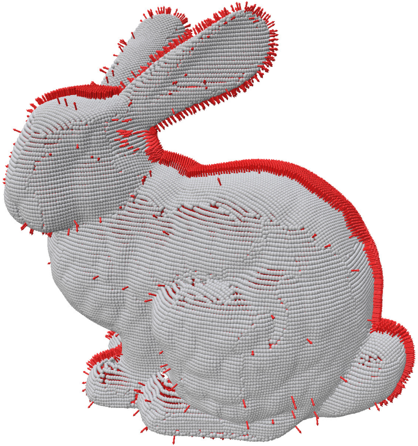
Consistent Normal Orientation

1. Compute the *unoriented* k-NN graph $G = (P, E)$.
2. Compute a minimal spanning tree of G with weights $w_{ij} = 1 - |n_i \cdot n_j|$
3. Set an arbitrary root r and traverse the spanning tree.
   For each edge (i, j) where i is the parent of j:
   if $-n_i \cdot n_j > n_i \cdot n_j$ then set $n_i = -n_i$

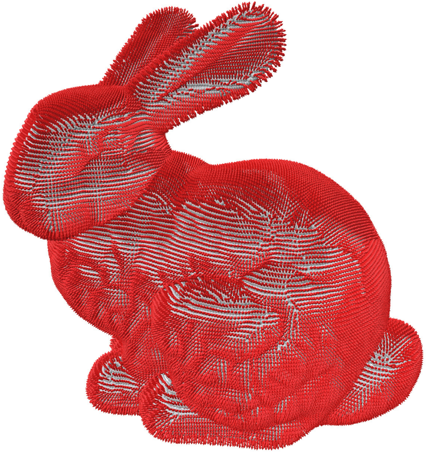
DEMO

Surface Reconstruction from Unorganized Points, Hoppe et al., 1992

Consistent Orientation



Faulty normal orientation
(raw hyperplane fitting)

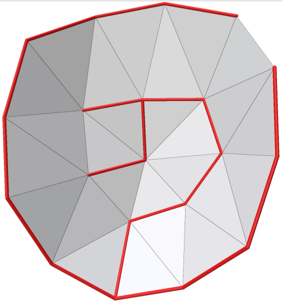


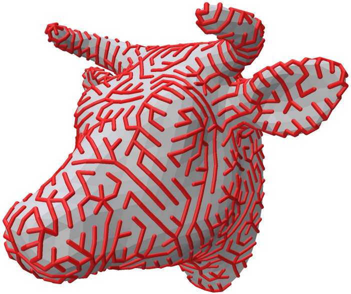
Consistent (outward) normal orientation

Minimal Spanning Tree

Let $G = (V, E)$ be a connected graph with weights $w_{ij} \in \mathbb{R}$ on edge (i, j).
A **Minimal Spanning Tree** $T = (V, E')$ is a subgraph of G such that:

- T is connected (there exists a path of edges between every vertex)
- $E' \subset E$ such that $\sum_{(i,j)\in E'} w_{ij}$ is minimal





Minimal Spanning Tree on surface meshes with Euclidean distance as edge weights

Minimal Spanning Tree

Let $G = (V, E)$ be a connected graph with weights $w_{ij} \in \mathbb{R}$ on edge (i, j).
A **Minimal Spanning Tree** $T = (V, E')$ is a subgraph of G such that:

- T is connected (there exists a path of edges between every vertex)
- $E' \subset E$ such that $\sum_{(i,j)\in E'} w_{ij}$ is minimal

Kruskal's algorithm

Input: A graph $G = (V, E)$
Output: A minimal spanning tree T of G

- $T = \varnothing$ set of sedges
- For each edge $(a, b) \in E$ in increasing weight order:
- If a and b were not already connecting by the tree T[a]:
- Add edge (a, b) to T

[a]Fast with a Union-Find data structure (https://fr.wikipedia.org/wiki/Union-find)

Estimating Density

Local lengths

- Distance to the closest neighbor
- Distance to the n-th neighbor
- Mean of distances to the k nearest neighbors
- etc.

Local areas

1. Consider the k nearest neighbors of a point p (or points at a distance $< r$ from p)
2. Project all points onto the tangent plane of p ($\perp$ to $\overrightarrow{n}_p$)
3. Compute the 2d Delaunay triangulation
4. Return the sum of areas of the Delaunay triangles

Fast Winding Numbers for Soups and Clouds, Barill et al., 2018

Local lengths

- Distance to the closest neighbor
- Distance to the n-th neighbor
- Mean of distances to the k nearest neighbors
- etc.

Local areas

1. Consider the k nearest neighbors of a point p (or points at a distance $< r$ from p)
2. Project all points onto the tangent plane of p ($\perp$ to $\overrightarrow{n}_p$)
3. Compute the 2d Delaunay triangulation
4. Return the sum of areas of the Delaunay triangles

DEMO

Fast Winding Numbers for Soups and Clouds, Barill et al., 2018

Outlier removal

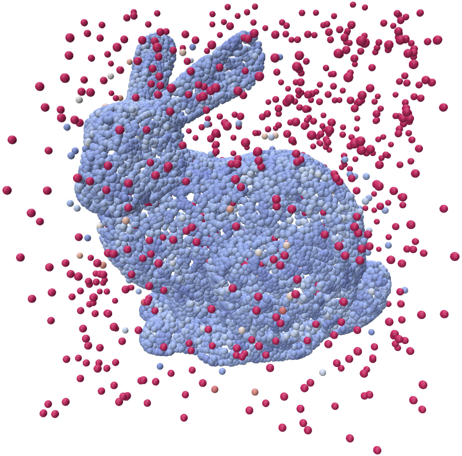
Observation

Density of outlier points is usually lower than density of inliers.

Simple outlier detection

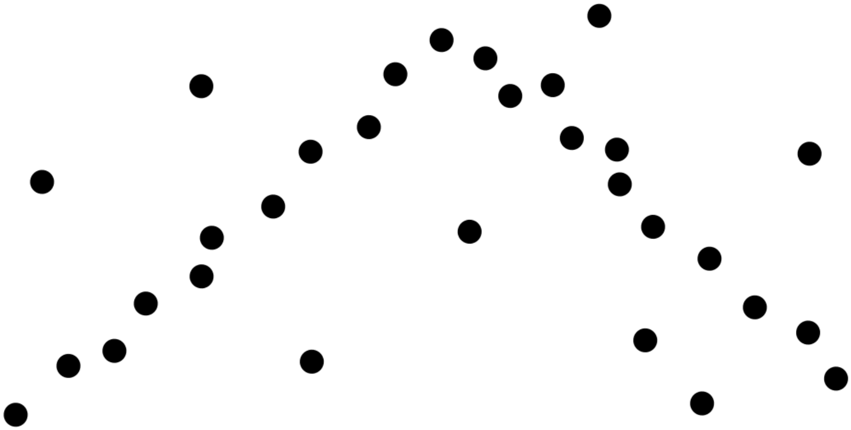
Input: points $P = \{p_1, ..., p_N\}$
Remove points whose local area is $> \eta$

DEMO

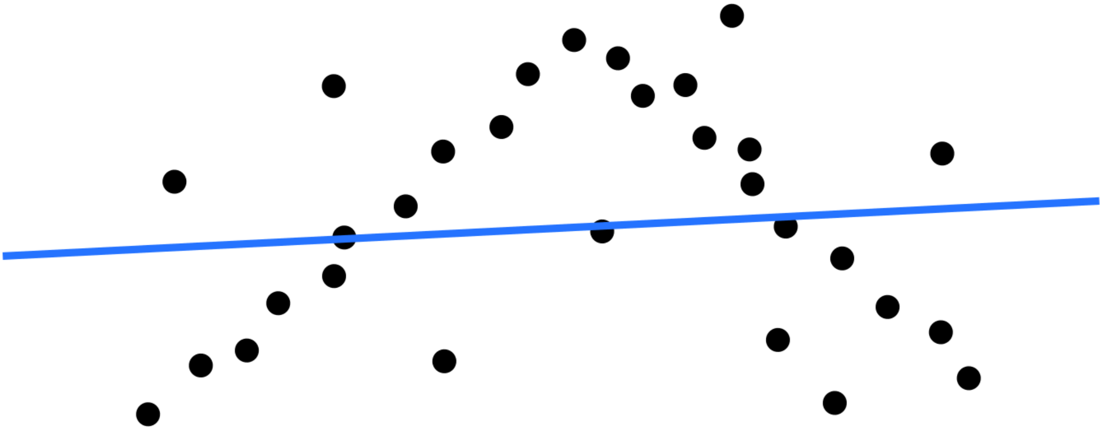


Clustering and Primitive Fitting

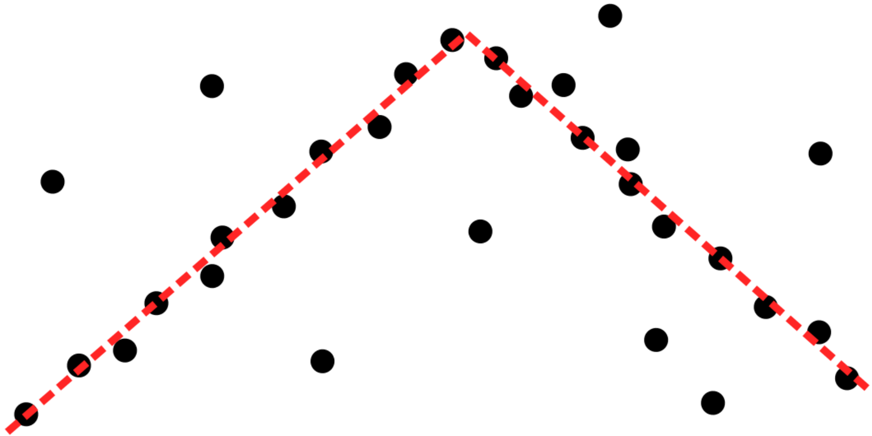
How to fit a plane in the presence of many outliers?



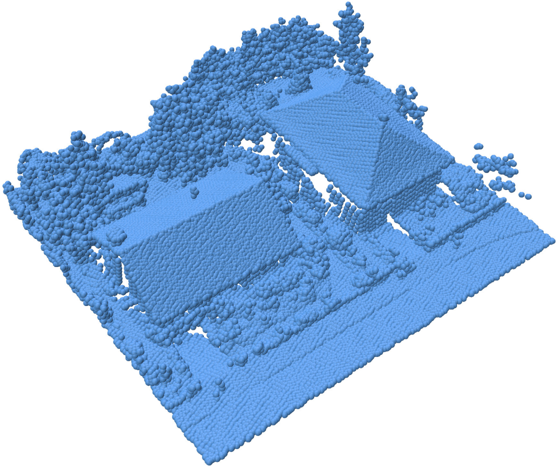
How to fit a plane in the presence of many outliers?

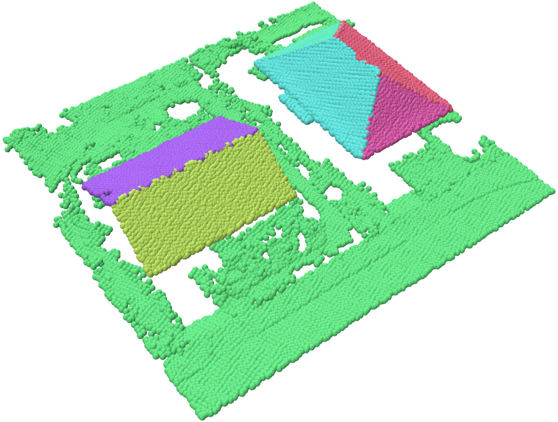


How to fit a plane in the presence of many outliers?



How to fit a plane in the presence of many outliers?





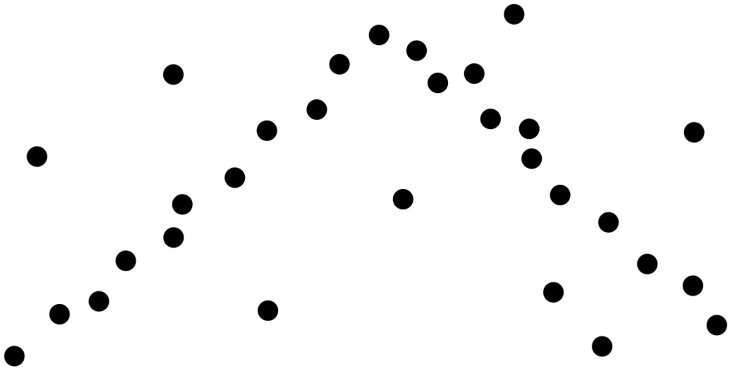
General Idea: Sample and reject random candidates

RANSAC Algorithm (RANdom SAmple Consensus)

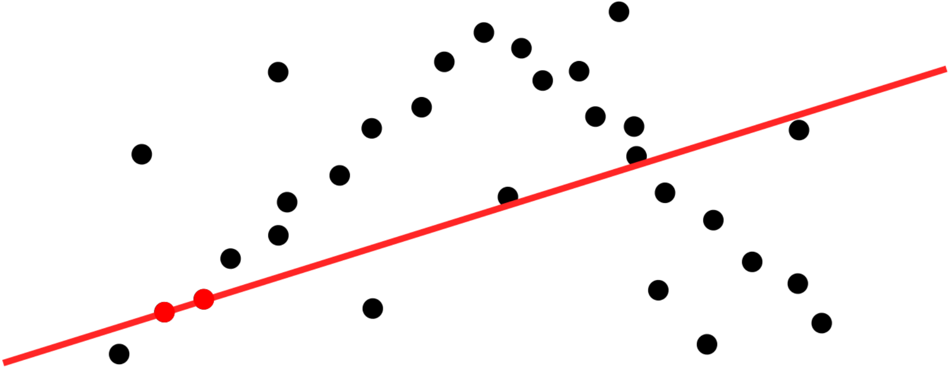
Input: points $P = \{p_1, ..., p_N\}$
Parameters: sample size K, minimal cluster size S, inlier threshold τ
Iterate:

- Sample K planes (triplets of points in P)
- For each plane i, compute its set P_i of inliers (points in P at distance $< \tau$)
- Find the plane i_0 with the most inliers.
- If $|P_{i_0}| \geqslant S$:
  Add the plane to the found primitives
  Remove the points P_{i_0} from the samplable points ($P \leftarrow P \backslash P_i$)

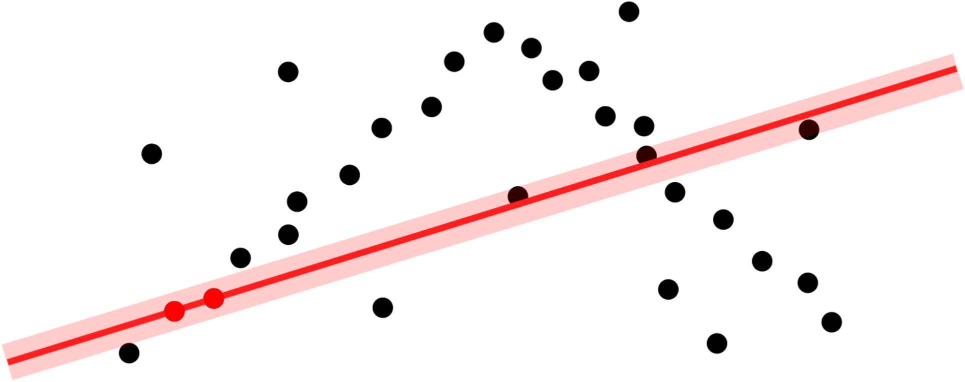
Executing the RANSAC algorithm on a 2D example



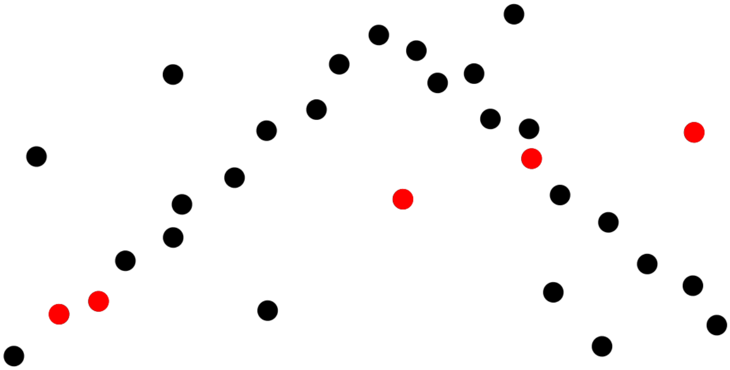
Executing the RANSAC algorithm on a 2D example



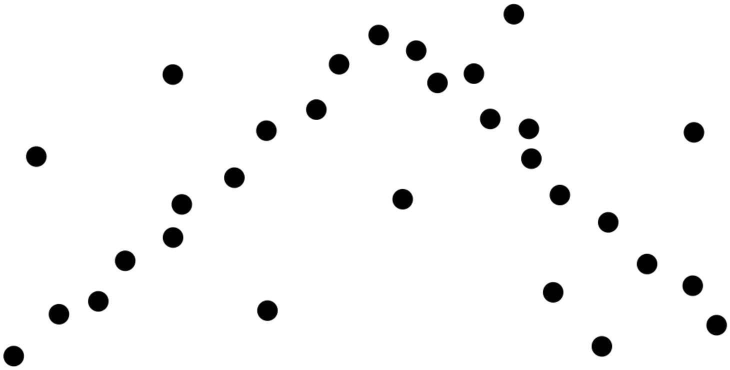
Executing the RANSAC algorithm on a 2D example



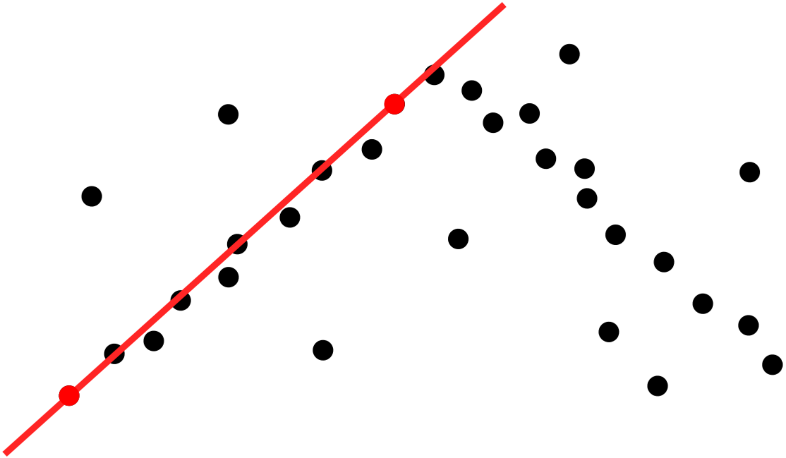
Executing the RANSAC algorithm on a 2D example



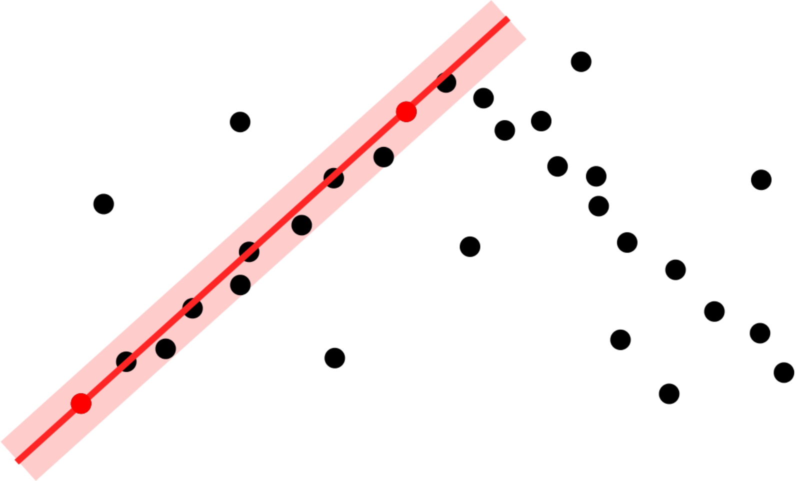
Executing the RANSAC algorithm on a 2D example



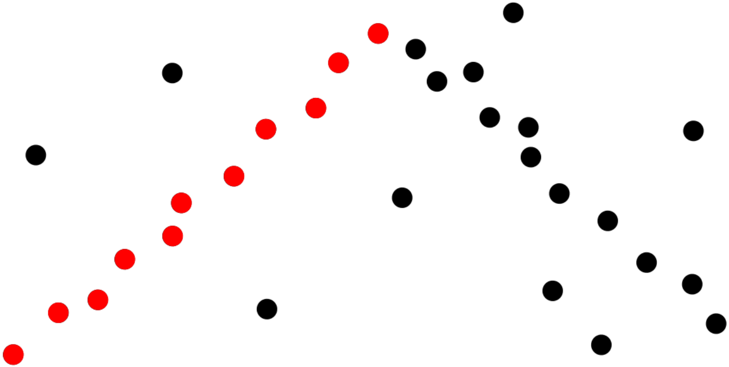
Executing the RANSAC algorithm on a 2D example



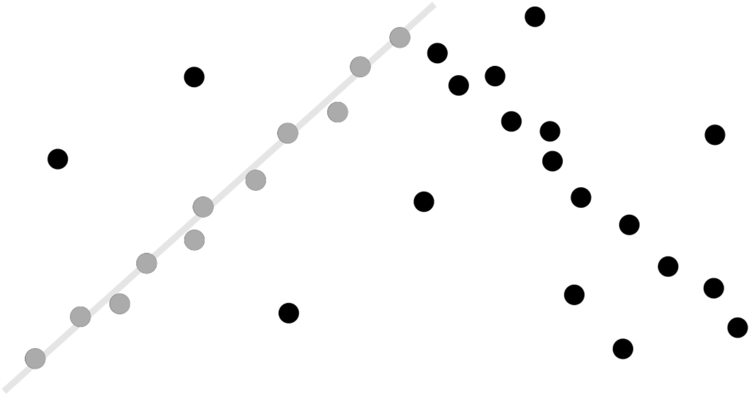
Executing the RANSAC algorithm on a 2D example



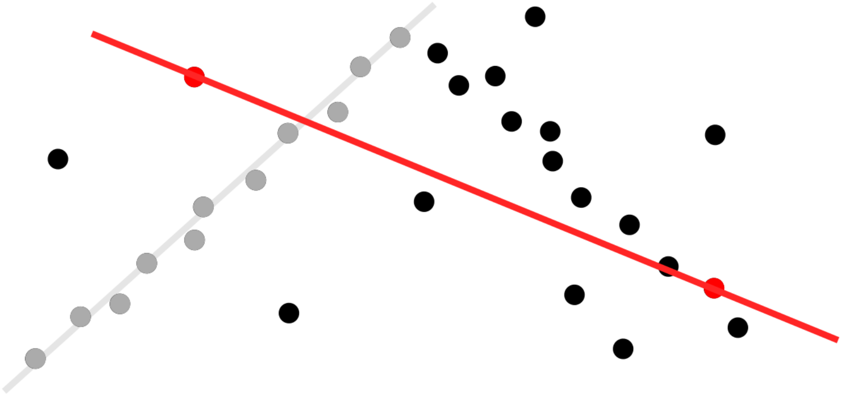
Executing the RANSAC algorithm on a 2D example



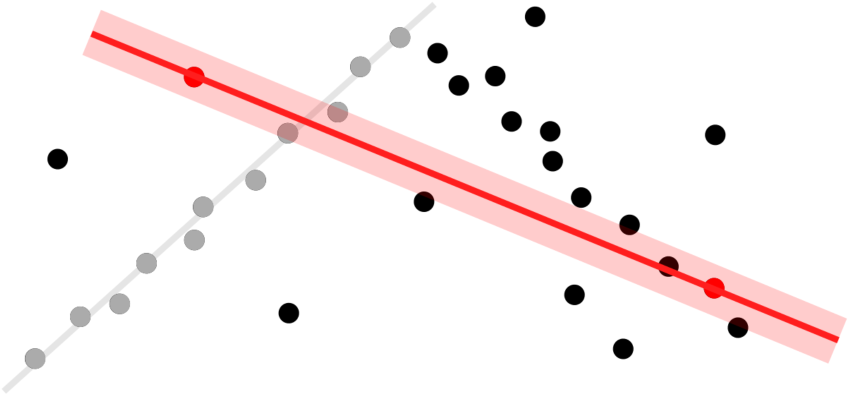
Executing the RANSAC algorithm on a 2D example



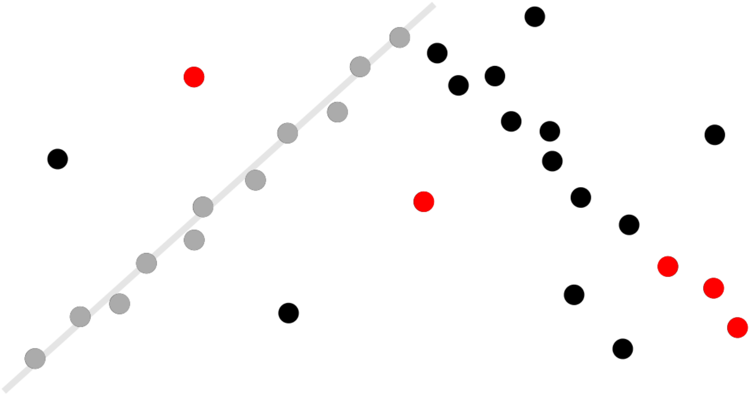
Executing the RANSAC algorithm on a 2D example



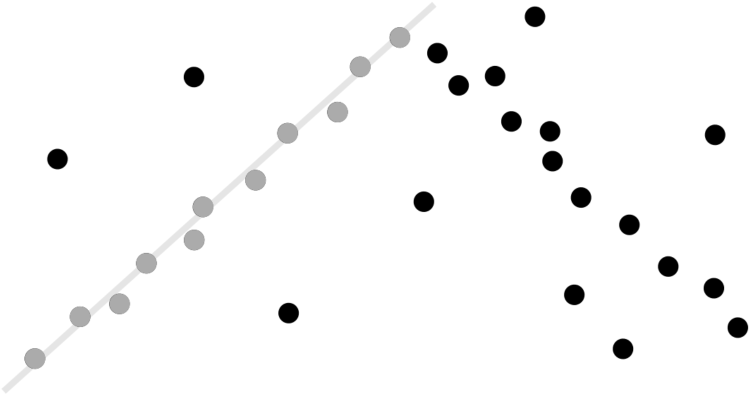
Executing the RANSAC algorithm on a 2D example



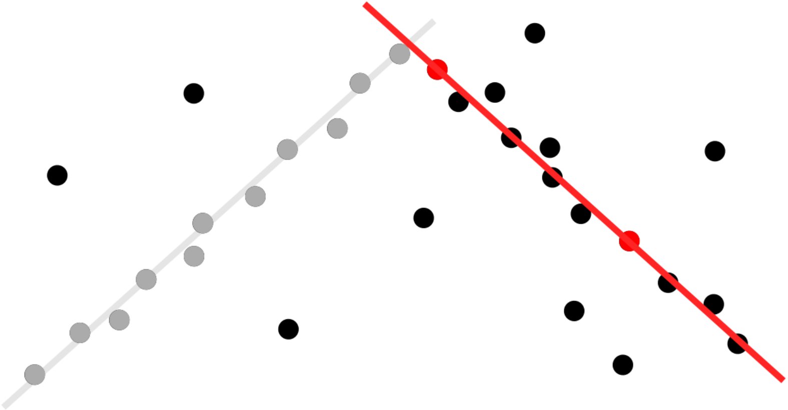
Executing the RANSAC algorithm on a 2D example



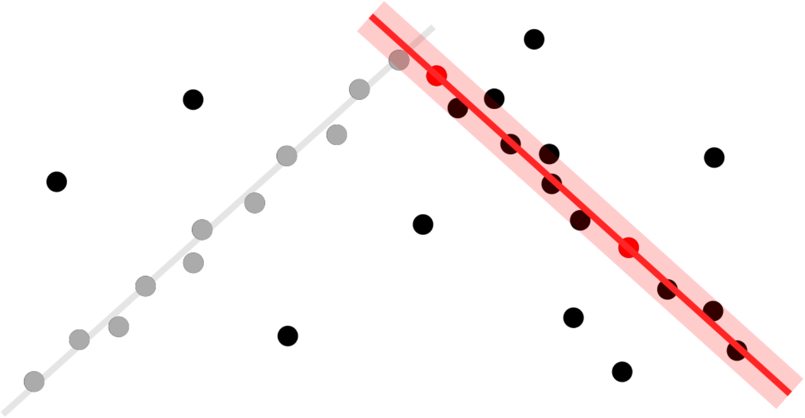
Executing the RANSAC algorithm on a 2D example



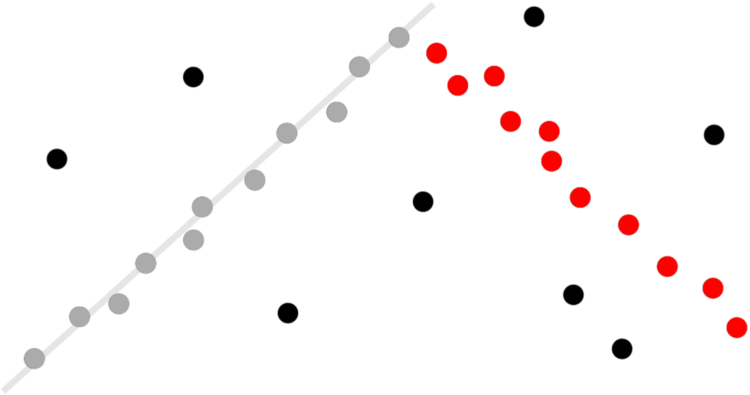
Executing the RANSAC algorithm on a 2D example



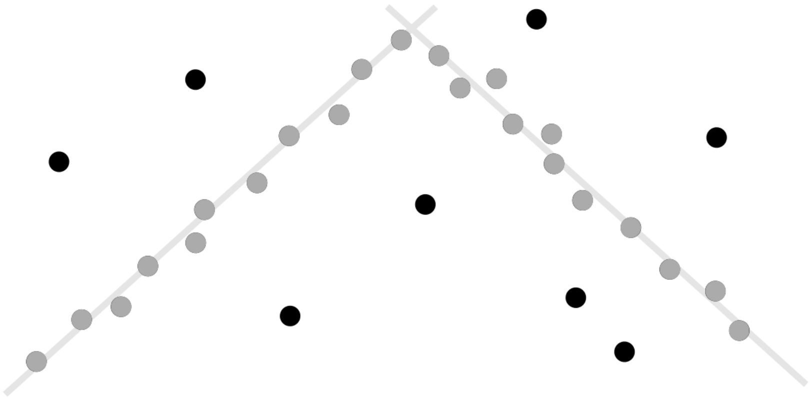
Executing the RANSAC algorithm on a 2D example



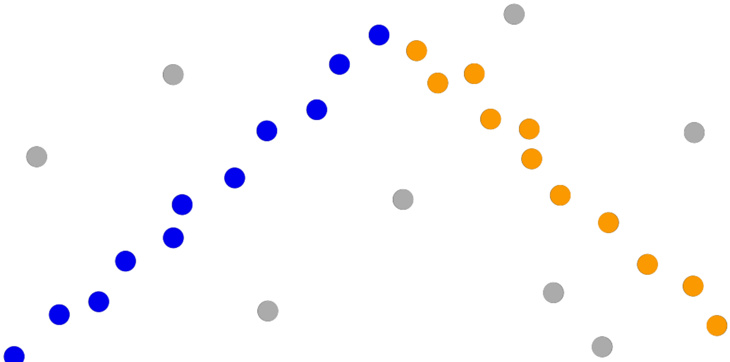
Executing the RANSAC algorithm on a 2D example



Executing the RANSAC algorithm on a 2D example



Executing the RANSAC algorithm on a 2D example



Improvements

Locality bias

Planes have more chance to be a valid primitive if all three points are "close" together. $\hookrightarrow$ Sample first point and two others among its nearest neighbors

Consider connected components

When computing inliers, only consider points in the largest connected component of the k-NN graph

Consider normals

A point is an inlier if small distance and its normal aligns (dot product with plane's normal $> \delta$)

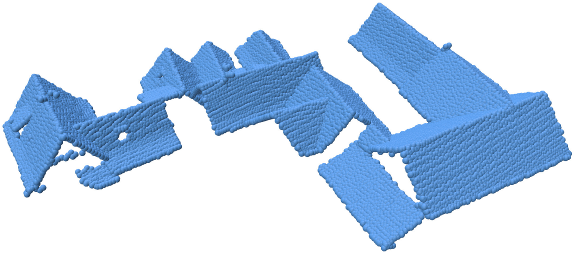
Other primitives

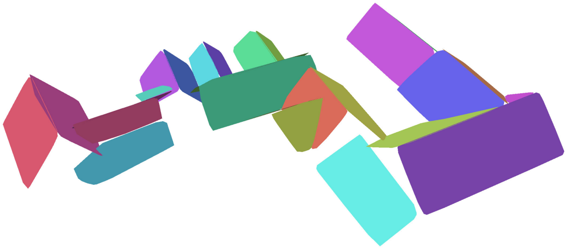
Also works when sampling spheres (4 points), cylinders, cones, etc...

Efficient RANSAC for Point-Cloud Shape Detection, Schnabel et al., 2007

Presentation of the Project

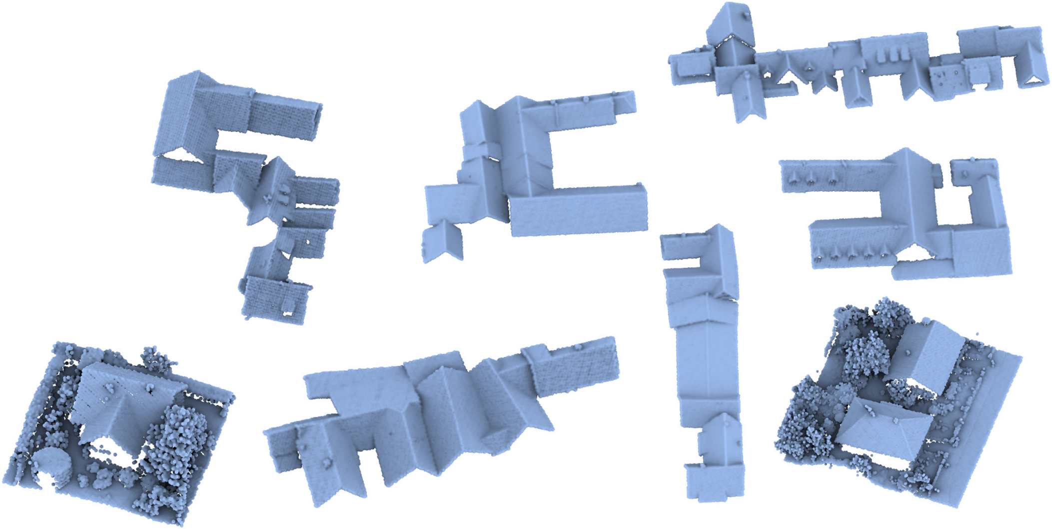
Finding plane primitives in aerial LIDAR point clouds





DEMO

Input Data: List of (x, y, z) coordinates in a text file



Features to implement in python (baseline)

- K-nearest neighbors
- Normal estimation
- RANSAC with some heuristics
- Delaunay triangulation of the primitives

... and lots of possible improvements

Features to implement in python (baseline)

- K-nearest neighbors
- Normal estimation
- RANSAC with some heuristics
- Delaunay triangulation of the primitives

... and lots of possible improvements

Evaluation

An oral evaluation with us where you show us a demo (and we ask you questions!)

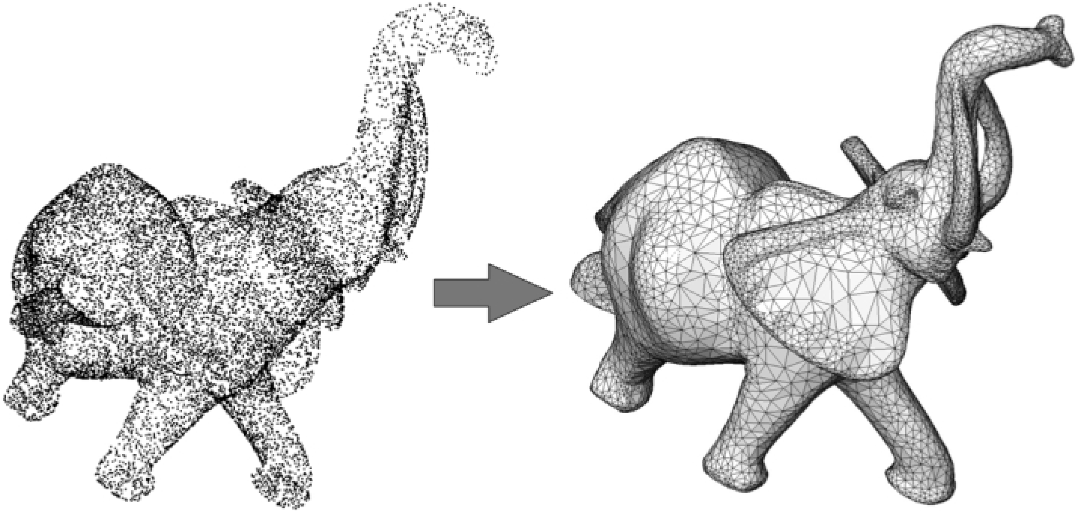
https://www.youtube.com/watch?v=X0vC3slzDmc

Part 2: Surface Reconstruction

5. Reconstruction algorithms based on Delaunay triangulations
    - α-shapes
    - Ball Pivoting
    - CRUST
    - Implicit Representations of Geometry

6. Reconstruction algorithm based on implicit representations
    - Poisson Surface Reconstruction
    - Generalized Winding Number

From point cloud to surface meshes



https://doc.cgal.org/Manual/3.5/doc_html/cgal_manual/Surface_reconstruction_points_3/Chapter_main.html

Reconstruction algorithms based on Delaunay triangulations

How to choose k in the k-NN graph?

Observation

The edges/faces that we want in the reconstruction are in the k-NN graph for some k large enough.

DEMO

How to choose k in the k-NN graph?

Observation

The edges/faces that we want in the reconstruction are in the k-NN graph for some k large enough.

DEMO

A naive reconstruction algorithm (2D)

Keep edges of the 2-NN graph of $P = \{p_1, ..., p_N\}$

What about Delaunay?

Observation

The edges/faces that we want in the reconstruction are edges/faces of the Delaunay triangulation/tetrahedrization.

DEMO

What about Delaunay?

Observation

The edges/faces that we want in the reconstruction are edges/faces of the Delaunay triangulation/tetrahedrization.

DEMO

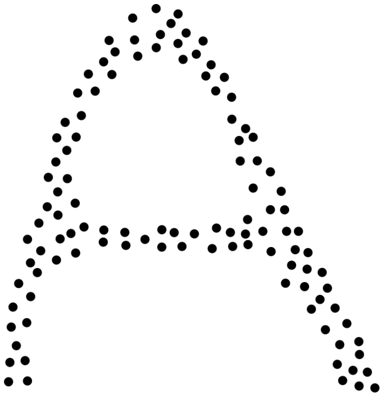
A (slightly less) naive reconstruction algorithm

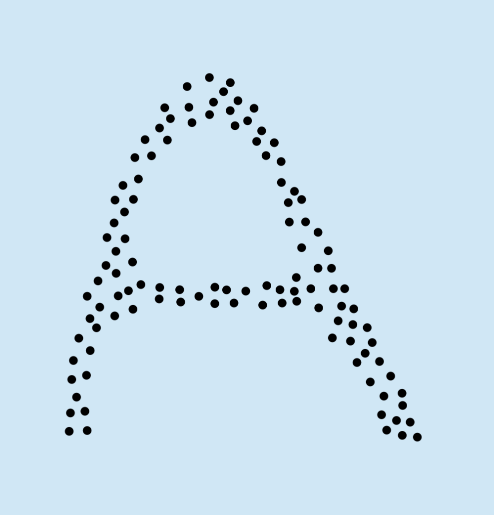
1. Compute the Delaunay Triangulation of $P = \{p_1, ..., p_N\}$
2. Keep edges with length $< L$

α-shapes

The intuition

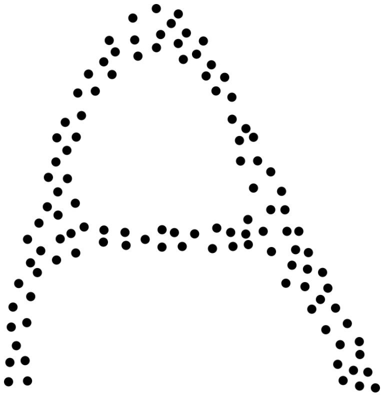
Given points $P = \{p_1, ..., p_N\} \in \mathbb{R}^d$, remove all possible spheres of radius α that does not contain any point.

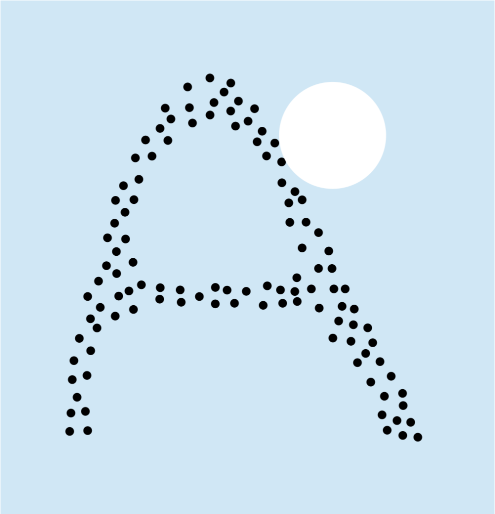




The intuition

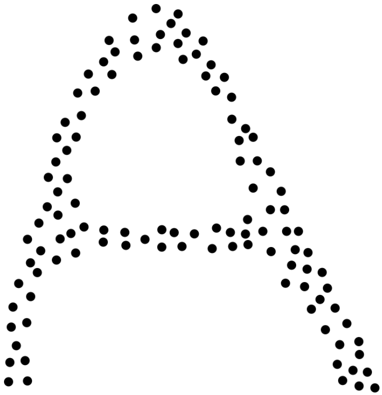
Given points $P = \{p_1, ..., p_N\} \in \mathbb{R}^d$, remove all possible spheres of radius α that does not contain any point.

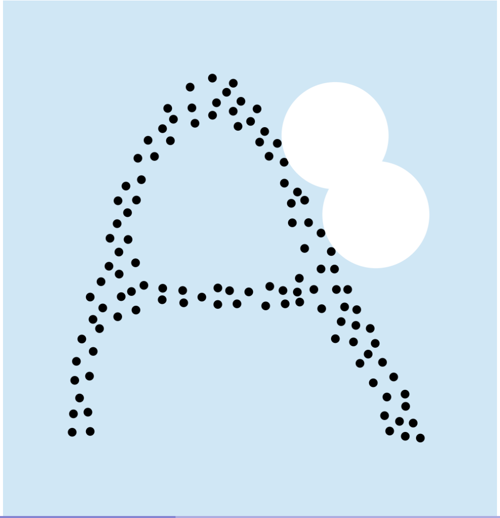




The intuition

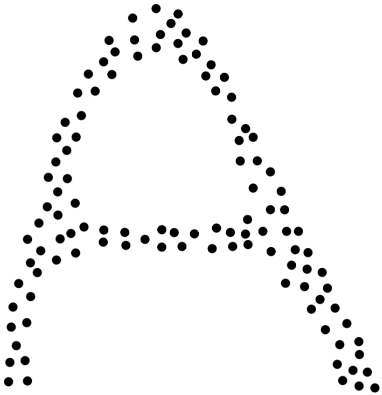
Given points $P = \{p_1, ..., p_N\} \in \mathbb{R}^d$, remove all possible spheres of radius α that does not contain any point.

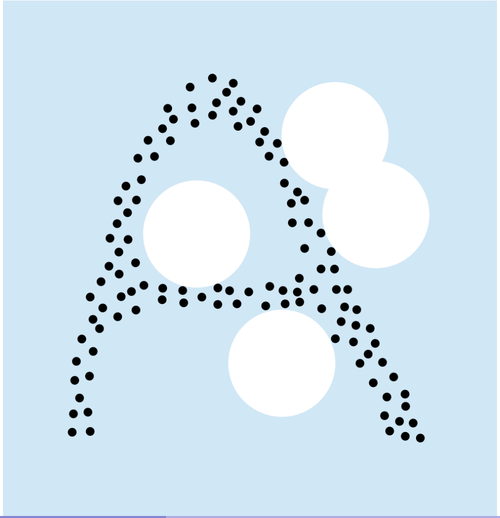




The intuition

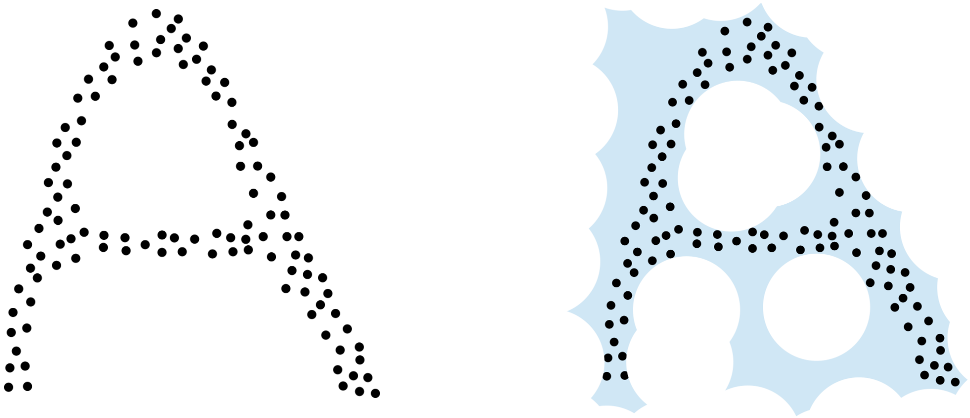
Given points $P = \{p_1, ..., p_N\} \in \mathbb{R}^d$, remove all possible spheres of radius α that does not contain any point.





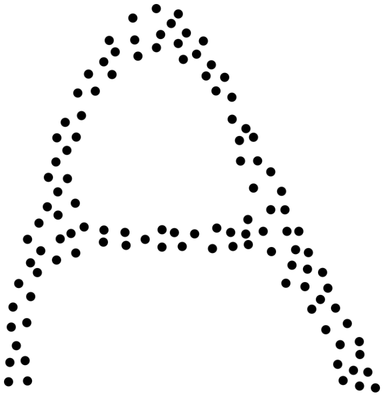
The intuition

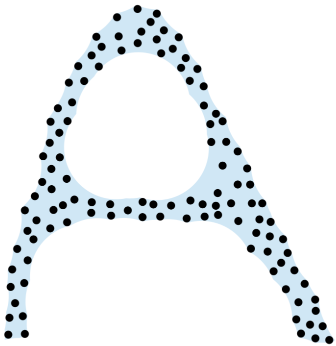
Given points $P = \{p_1, ..., p_N\} \in \mathbb{R}^d$, remove all possible spheres of radius α that does not contain any point.



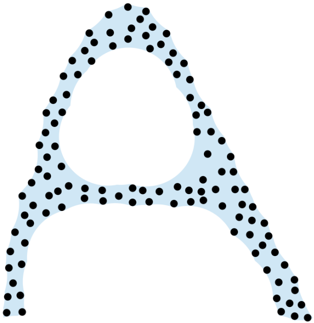
The intuition

Given points $P = \{p_1, ..., p_N\} \in \mathbb{R}^d$, remove all possible spheres of radius α that does not contain any point.





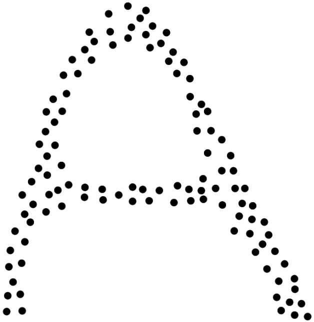
α-hull



The α-**hull** of P is what's left when we have removed all possible empty circles of radius α

On the Shape of a Set of Points in the Plane, Edelsbrunner et al., 1983

α-hull

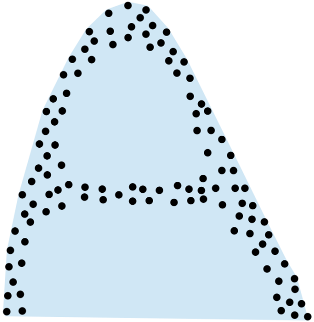


The α-**hull** of P is what's left when we have removed all possible empty circles of radius α

When $\alpha \to 0$, we get only P

On the Shape of a Set of Points in the Plane, Edelsbrunner et al., 1983

α-hull



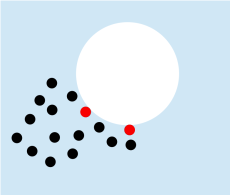
The α-**hull** of P is what's left when we have removed all possible empty circles of radius α

When $\alpha \to 0$, we get only P

When $\alpha \to \infty$, we get the convex-hull of P

On the Shape of a Set of Points in the Plane, Edelsbrunner et al., 1983

α-shape



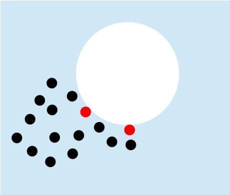
A point $p \in P$ is an α-**exposed** if there exists an empty circle of radius α such that p is on its boundary.

An edge (p_1, p_2) is α-**exposed** if there exists an empty circle of radius α such that *both* points are on its boundary.

On the Shape of a Set of Points in the Plane, Edelsbrunner et al., 1983

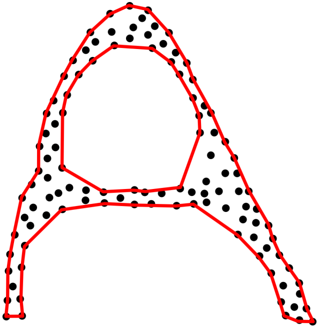
Three-dimensional alpha shapes, Edelsbrunner and Mücke, 1994

α-shape



A point $p \in P$ is an α-**exposed** if there exists an empty circle of radius α such that p is on its boundary.

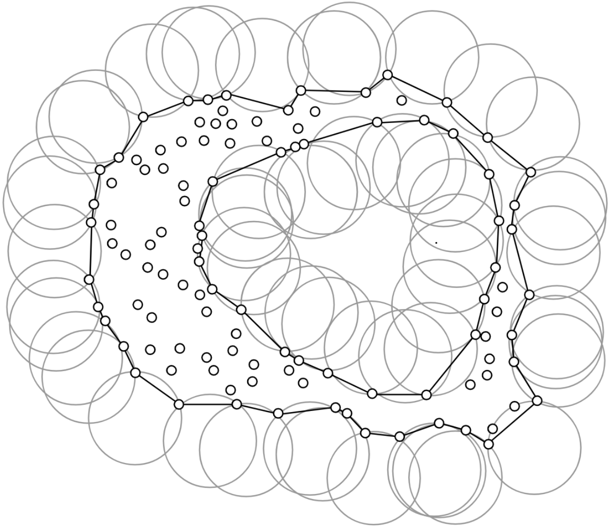
An edge (p_1, p_2) is α-**exposed** if there exists an empty circle of radius α such that *both* points are on its boundary.



The α-**shape** is the straight line graph made of all α-exposed edges between two points of P.

On the Shape of a Set of Points in the Plane, Edelsbrunner et al., 1983
Three-dimensional alpha shapes, Edelsbrunner and Mücke, 1994

α-shape



https://graphics.stanford.edu/courses/cs268-11-spring/handouts/AlphaShapes/as_fisher.pdf

Properties of α-shapes

Theorem

For any α, the α-shape of P is a subgraph of the Delaunay triangulation of P.

Properties of α-shapes

Theorem

For any α, the α-shape of P is a subgraph of the Delaunay triangulation of P.

Let p and q be two α-exposed points. There exists a circle of radius α not containing any point $r \in P$ such that p and q are on its boundary. Let c be the center of this circle. Clearly, $d(p,c) = d(q,c) \leqslant d(r,c)$ for any $r \in P \backslash \{p,q\}$. This means that c is in both the Voronoi cells $Vor(p)$ and $Vor(q)$, which means that those cells are touching. In other words, p and q are neighbors in the (dual) Delaunay triangulation.

Properties of α-shapes

Theorem

For any α, the α-shape of P is a subgraph of the Delaunay triangulation of P.

Let p and q be two α-exposed points. There exists a circle of radius α not containing any point $r \in P$ such that p and q are on its boundary. Let c be the center of this circle. Clearly, $d(p,c) = d(q,c) \leqslant d(r,c)$ for any $r \in P \backslash \{p,q\}$. This means that c is in both the Voronoi cells $Vor(p)$ and $Vor(q)$, which means that those cells are touching. In other words, p and q are neighbors in the (dual) Delaunay triangulation.

Consequence

There exists only a finite number of different α-shapes of P when α goes from 0 to ∞.

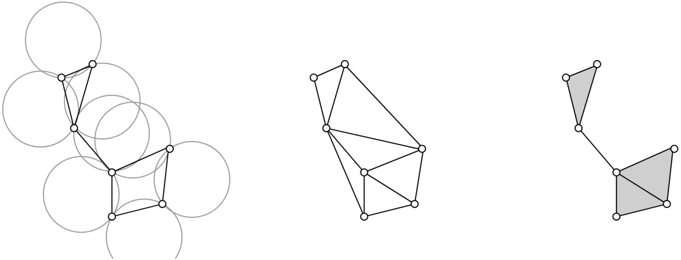
Computing α-shapes: the α-complex

https://demonstrations.wolfram.com/AlphaComplexAndUnionOfGrowingDisks/

Which Delaunay triangle belong inside the α-shape?

Let S be a simplex (segment, triangle, tetrahedra,...). Let its circumsphere be centered at μ_S with radius σ_S.
S is in the α-**complex** if:

- S is on the boundary of a simplex S' of the α-complex, or
- $\sigma_S < \alpha$ and the sphere centered in μ_S of radius σ_S is empty



Computing α-shapes: the α-complex

The α-shape is the boundary of the α-complex

Three-dimensional alpha shapes, Edelsbrunner and Mücke, 1994

Computing α-shapes: the α-complex

The α-shape is the boundary of the α-complex

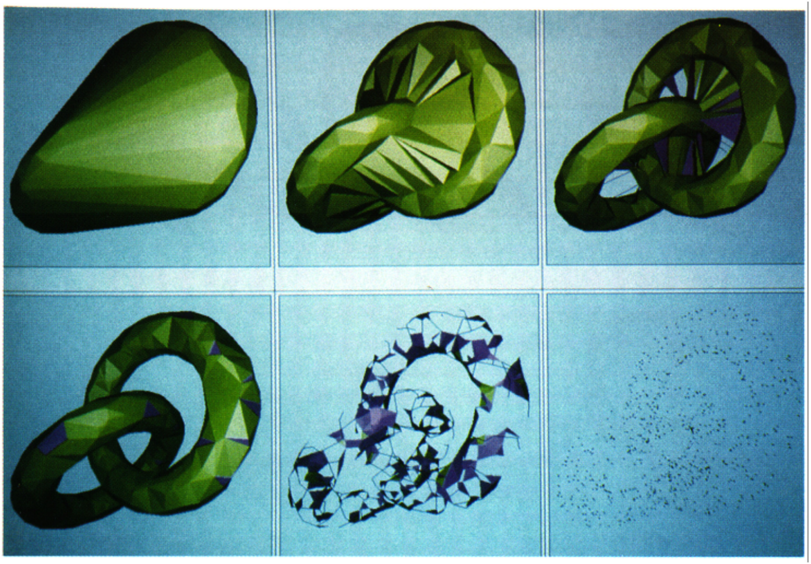
Edelsbrunner's algorithm

Input: points $P = \{p_1, ..., p_N\}$

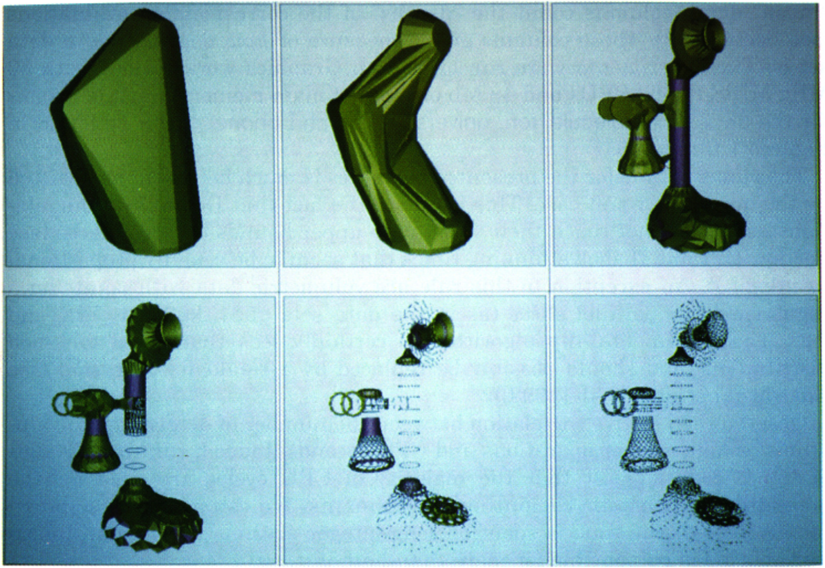
1. Compute the Delaunay triangulation $D = (V, E, T)$ of P
2. Determine the set $\mathcal{C}_\alpha$ of triangles T inside the α-shape
3. Return the outside boundary of $\mathcal{C}_\alpha$

DEMO

Three-dimensional alpha shapes, Edelsbrunner and Mücke, 1994



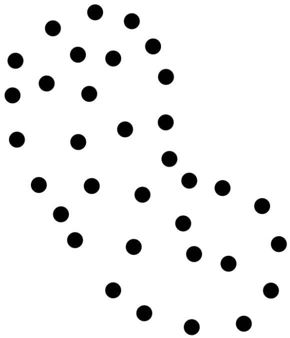
Three-dimensional alpha shapes, Edelsbrunner and Mücke, 1994



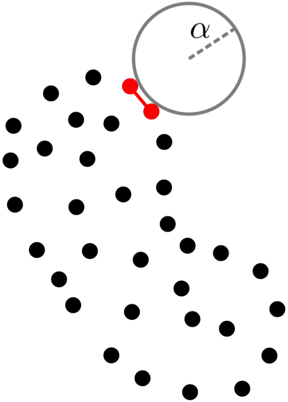
Three-dimensional alpha shapes, Edelsbrunner and Mücke, 1994

Ball Pivoting

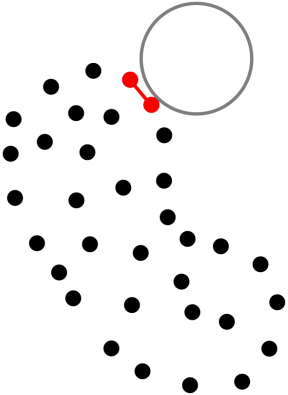
2D example



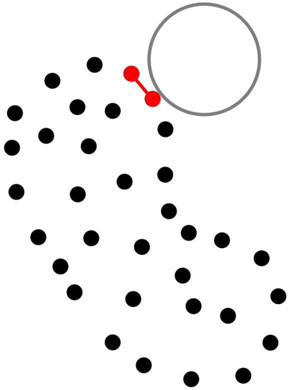
2D example



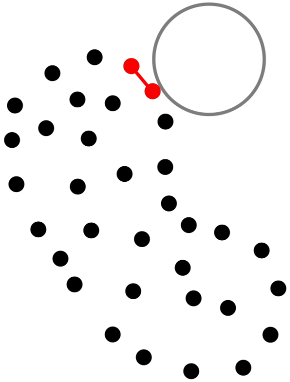
2D example



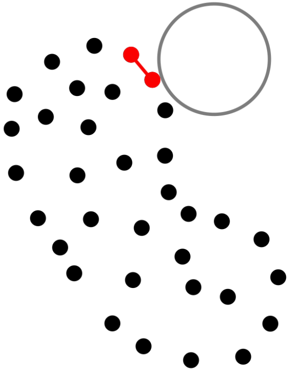
2D example



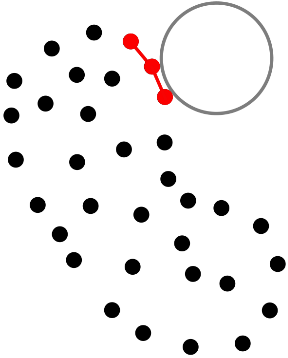
2D example



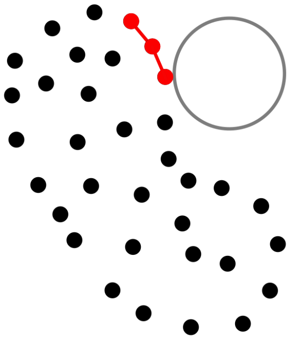
2D example



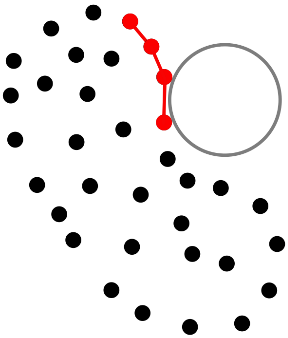
2D example



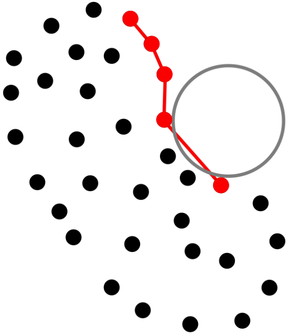
2D example



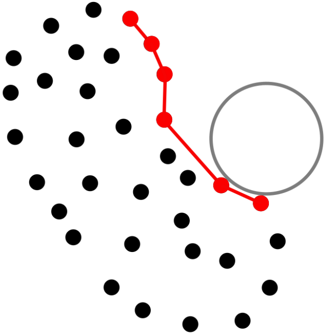
2D example



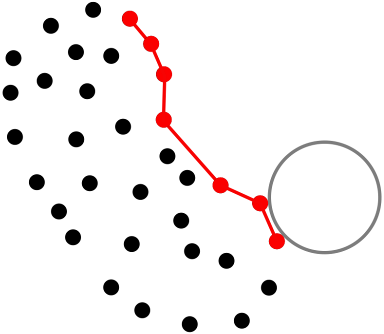
2D example



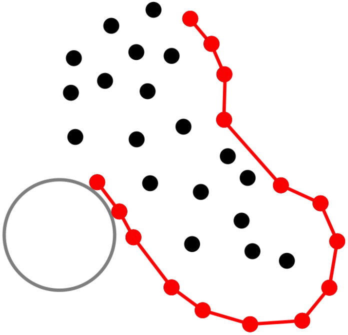
2D example



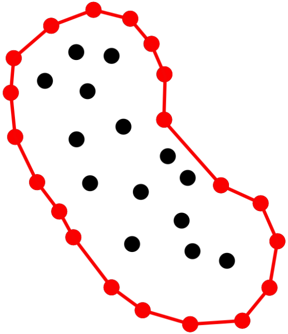
2D example



2D example



2D example



Ball Pivoting

Algorithm in 2D

Consider a ball of radius α.
Find an α-exposed edge $e = (p_0, p_1)$. Add edge e to reconstructed edges.
Set $p = p_1$ as the pivot.
Iterate:

- Pivot the ball around point p until another unvisited point q is touched
- If no point q can be reached, stop.
- Otherwise, add (p, q) as a reconstructed edge. Set p as the new pivot.

The Ball-Pivoting Algorithm for Surface Reconstruction, Bernardini et al., 1999

Ball Pivoting

Algorithm in 2D

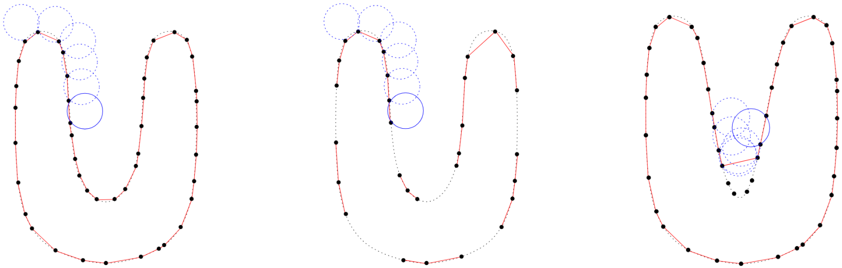
Consider a ball of radius α.
Find an α-exposed edge $e = (p_0, p_1)$. Add edge e to reconstructed edges.
Set $p = p_1$ as the pivot.
Iterate:

- Pivot the ball around point p until another unvisited point q is touched
- If no point q can be reached, stop.
- Otherwise, add (p, q) as a reconstructed edge. Set p as the new pivot.

It generalizes naturally to surfaces in 3D (pivot around an edge), though normals are needed to repair some bad cases

The Ball-Pivoting Algorithm for Surface Reconstruction, Bernardini et al., 1999

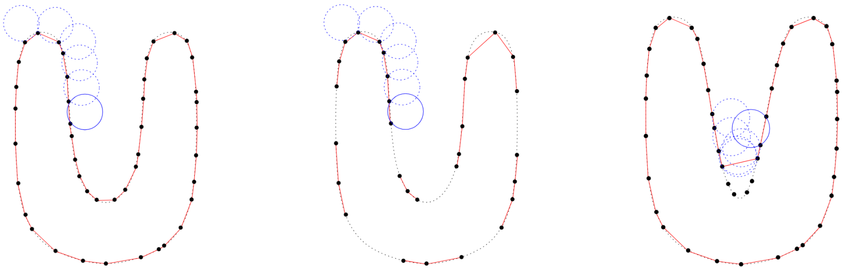
Properties



(a) (b) (c)

Every edge found by the BPA with radius α is α-exposed

Properties



(a) (b) (c)

Suppose the existence of an underlying manifold M from which the points are sampled. If:

- The intersection of any ball of radius α with M is a topological disk and
- Any ball of radius α centered on M contains at least one point

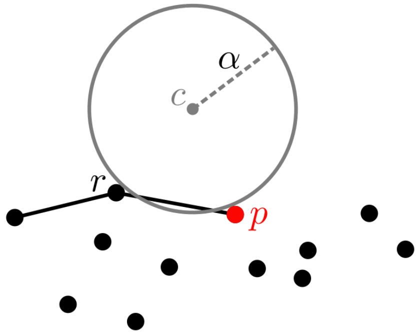
then the reconstructed surface is manifold with the correct topology.

Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

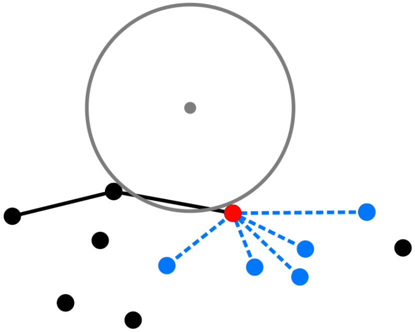


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, \dots, q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, \dots, c_k$ of the touching spheres for each of them
- Sort $c_1, \dots c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

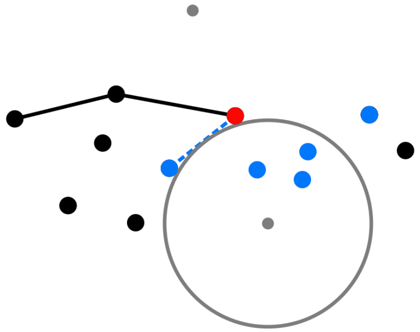


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

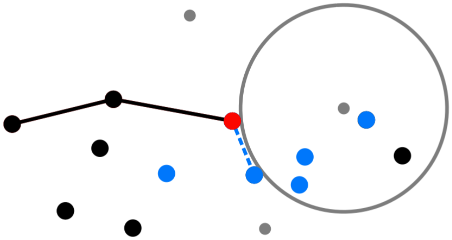


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

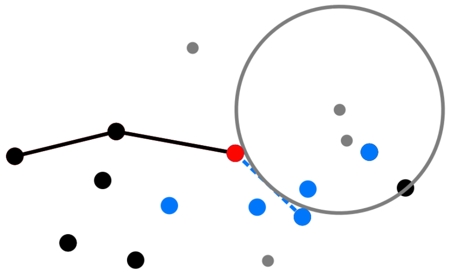


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

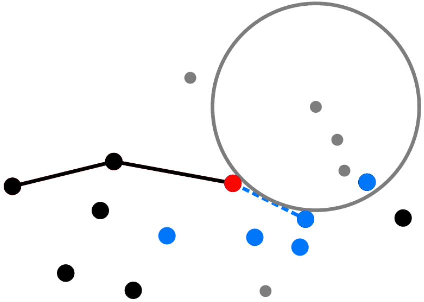


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, \dots, q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, \dots, c_k$ of the touching spheres for each of them
- Sort $c_1, \dots c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

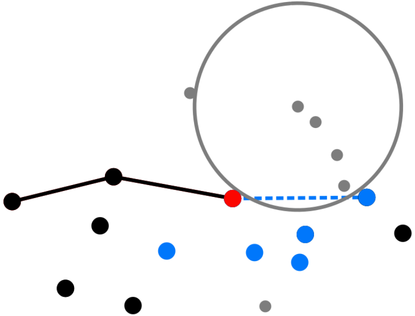


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, \ldots, q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, \ldots, c_k$ of the touching spheres for each of them
- Sort $c_1, \ldots c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

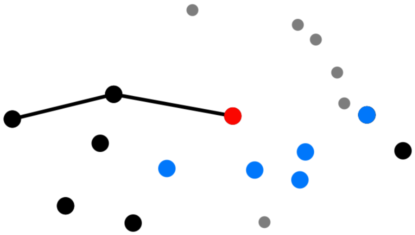


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

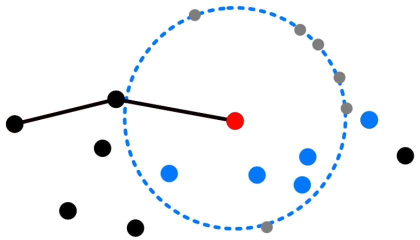


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, \ldots, q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, \ldots, c_k$ of the touching spheres for each of them
- Sort $c_1, \ldots c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

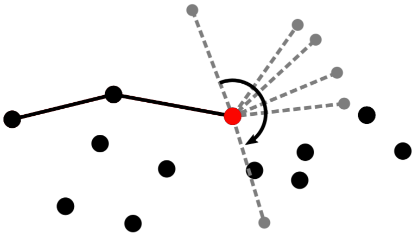


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, \ldots, q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, \ldots, c_k$ of the touching spheres for each of them
- Sort $c_1, \ldots c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

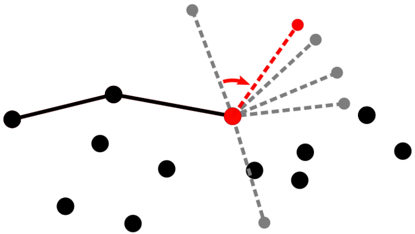


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

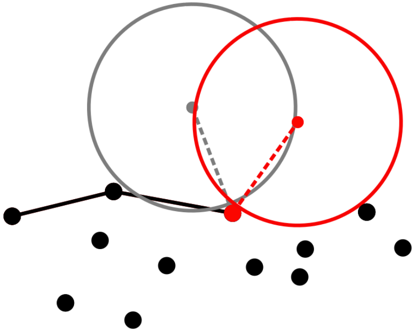


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

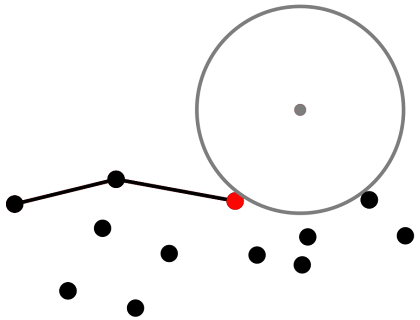


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, \ldots, q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, \ldots, c_k$ of the touching spheres for each of them
- Sort $c_1, \ldots c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot

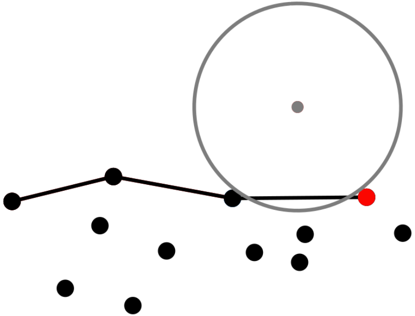


Algorithmic perspective

Computing the next intersected point

Let p be the current pivot, r be the previous pivot and c being the center of the α-ball.

- Query the unvisited neighbors $q_1, ..., q_k$ of p at distance $< 2\alpha$
- Compute the centers $c_1, ..., c_k$ of the touching spheres for each of them
- Sort $c_1, ...c_k$ by increasing order of oriented angle w.r.t. c around p.
- Select the minimum c_{i_0} as the new ball position and q_{i_0} as the new pivot



The Ball-Pivoting Algorithm for Surface Reconstruction, Bernardini et al., 1999

CRUST

CRUST

Idea

Delaunay reconstruction but prevent edges from "crossing" inside the domain

CRUST algorithm

Input: points $P = \{p_1, ..., p_N\}$

1. Compute the Voronoi diagram V of P.
2. Compute the Delaunay triangulation D of points $P \cup S$ where S are the vertices of the Voronoi diagram V
3. Return edges of D that link two points of P

A New Voronoi-based Surface Reconstruction Algorithm, Amenta et al., 1998

CRUST

Idea

Delaunay reconstruction but prevent edges from "crossing" inside the domain

CRUST algorithm

Input: points $P = \{p_1, ..., p_N\}$

1. Compute the Voronoi diagram V of P.
2. Compute the Delaunay triangulation D of points $P \cup S$ where S are the vertices of the Voronoi diagram V
3. Return edges of D that link two points of P

DEMO

A New Voronoi-based Surface Reconstruction Algorithm, Amenta et al., 1998

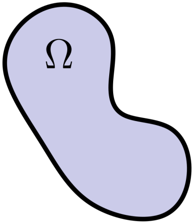
Reconstruction algorithm based on implicit representations

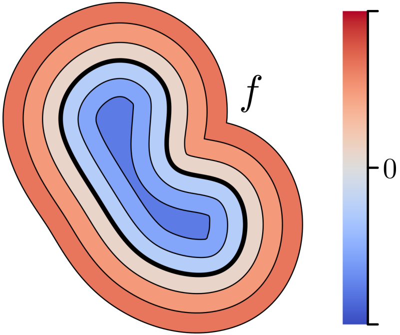
Implicit Representations of Geometry

Implicit Representation of Geometry

Represent a compact object $\Omega \subset \mathbb{R}^d$ as a level set of a continuous function:

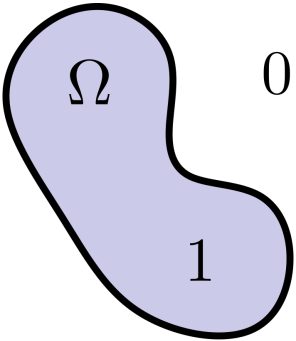
$$\Omega = \{x \in \mathbb{R}^d \mid f(x) \leqslant 0\}$$





Indicator Function

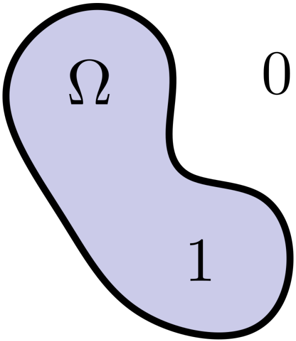
$$f(x) = \begin{cases} 1 \text{ if } x \in \Omega \\ 0 \text{ otherwise} \end{cases}$$



Indicator Function

$$f(x) = \begin{cases} 1 \text{ if } x \in \Omega \\ 0 \text{ otherwise} \end{cases}$$

- Simplest possible function
- Not differentiable...
- Not always easy to compute



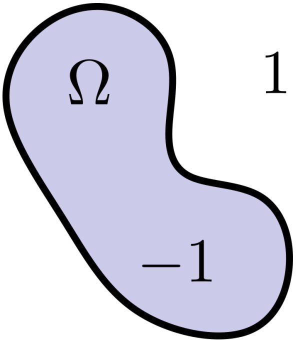
Indicator Function

$$f(x) = \begin{cases} 1 \text{ if } x \in \Omega \\ 0 \text{ otherwise} \end{cases}$$

- Simplest possible function
- Not differentiable...
- Not always easy to compute

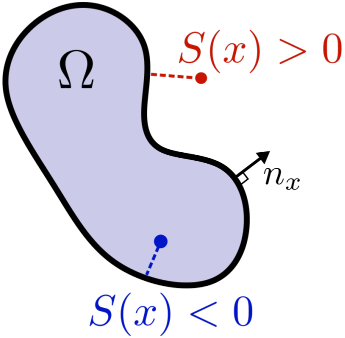
A variant: the sign function

$$y(x) = 1 - 2f(x) = \begin{cases} -1 \text{ if } x \in \Omega \\ 1 \text{ otherwise} \end{cases}$$



Signed Distance Function

$$S(x) = \begin{cases} -d(x, \partial\Omega) & \text{if } x \in \Omega \\ d(x, \partial\Omega) & \text{otherwise} \end{cases}$$

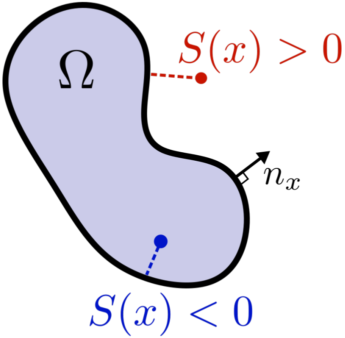


Signed Distance Function

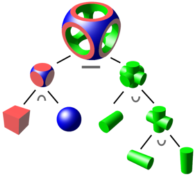
$$S(x) = \begin{cases} -d(x, \partial\Omega) & \text{if } x \in \Omega \\ d(x, \partial\Omega) & \text{otherwise} \end{cases}$$

Eikonal equation

$$\begin{cases} ||\nabla S(x)|| & = 1, \quad \forall x \in \mathbb{R}^d \\ S(x) & = 0, \quad \forall x \in \partial\Omega \\ \nabla S(x) & = n_x, \quad \forall x \in \partial\Omega \end{cases}$$

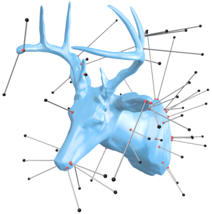


Applications



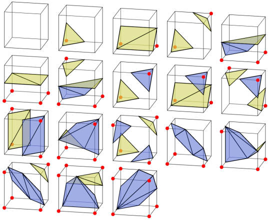
Constructive Solid Geometry

[Ricci (1973)]



Closest Point Query

[Sharp and Jacobson (2022)]

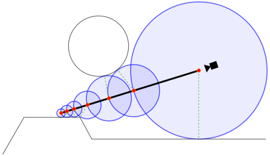


Marching Cubes

[Lorensen and Cline (1987)]

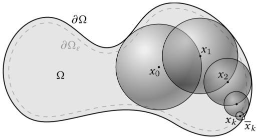
Rendering

Snail shader by Inigo Quilez



Empty Sphere Query

[Hart (1995)]



Monte-Carlo Simulation

[Sawhney and Crane (2020)]

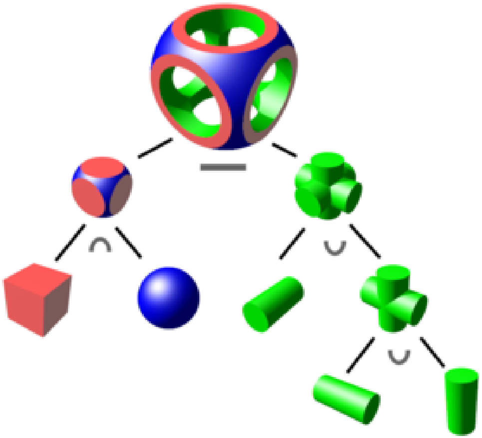
Constructive Solid Geometry

If $\Omega_a \leftrightarrow S_a$ and $\Omega_b \leftrightarrow S_b$:

- $-S_a$ represents $\overline{\Omega_a}$
- $\min(S_a, S_b)$ represents $\Omega_a \cap \Omega_b$
- $\max(S_a, S_b)$ represents $\Omega_a \cup \Omega_b$

$\min(S_a, S_b)$ is **not** a distance field[a]

[a] https://iquilezles.org/articles/interiordistance/

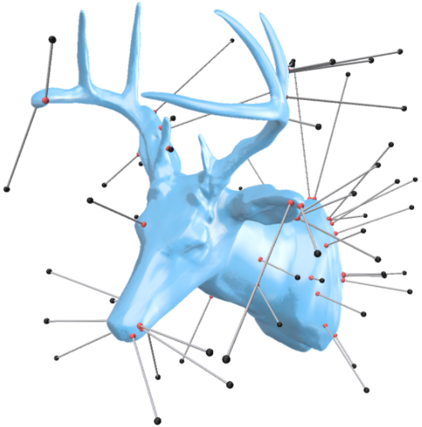


Geometrical Queries

For $x \in \mathbb{R}^d$ and $\Omega \leftrightarrow S$:

$$p = x - S(x)\nabla S(x)$$

is the closest point from x on Ω



[Sharp and Jacobson (2022)]

Geometrical Queries

For $x \in \mathbb{R}^d$ and $\Omega \leftrightarrow S$:

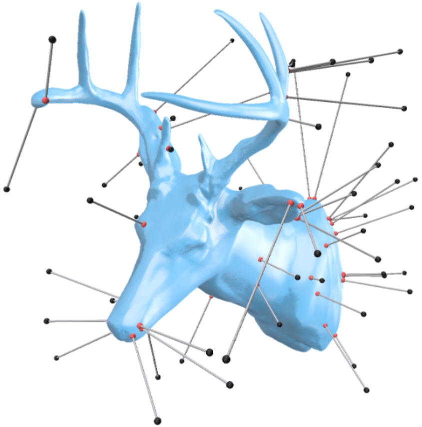
$$p = x - S(x)\nabla S(x)$$

is the closest point from x on Ω

Approximated signed distance fields

This also works if $|f(x)| < S(x)$ and

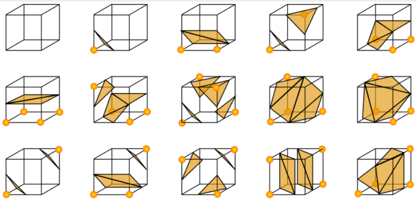
$$x_{n+1} = x_n - f(x_n)\nabla f(x_n)$$

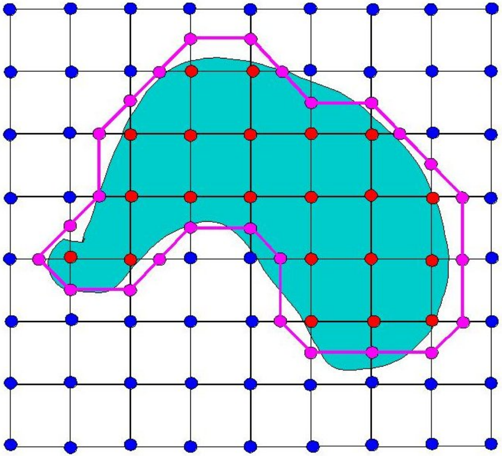


[Sharp and Jacobson (2022)]

From implicit functions to surface meshes: the Marching Cubes Algorithm

- Sample your implicit function over a grid
- For each cell, determine which points are inside/outside
- Mesh the cell according to a finite set of templates
- Possible improvements depending on the value of the function at grid vertices





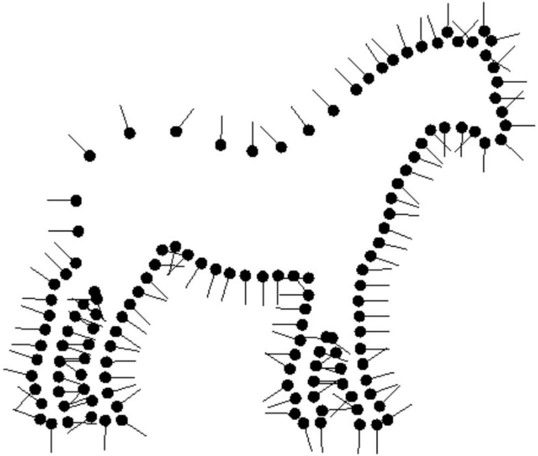
Marching Cubes: A High Resolution 3D Surface Construction Algorithm, Lorensen and Cline, 1987

Reconstruction algorithm based on implicit representations

Poisson Surface Reconstruction

Poisson Surface Reconstruction: the setting

Input: points with normals

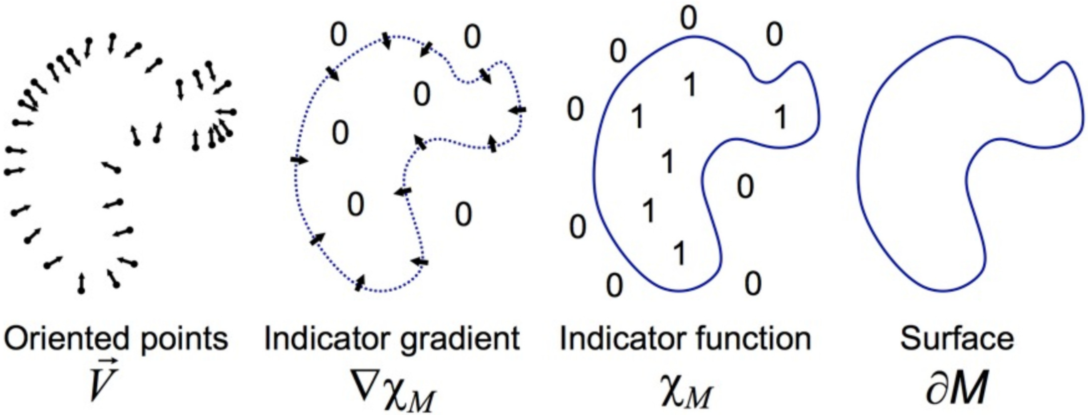


The idea

- Consider that each normal $\overrightarrow{n_p}$ at point p is the gradient ∇f of some implicit function f
- Integrate this gradient into the function f
- Recover the surface via marching cubes

Poisson surface reconstruction, Kazhdan et al., 2006

Poisson Surface Reconstruction: Principle



https://slides.cgg.unibe.ch/GP20/06_Surface_Reconstruction.html

Points $p_1, ...p_N$ with normals $\overrightarrow{n_1}, ..., \overrightarrow{n_N}$
Approximate a vector field v by $v(p_i) = \overrightarrow{n_i}$ and $v(x) = 0$ otherwise.
Solve:

$$\min_f \int_\Omega ||\nabla f(x) - \overrightarrow{n}(x)||^2 dx$$

i.e.:

$$\min_f \sum_{i=1}^{N} ||\nabla f(p_i) - \overrightarrow{n_i}||^2$$

Points $p_1, ...p_N$ with normals $\overrightarrow{n_1}, ..., \overrightarrow{n_N}$
Approximate a vector field v by $v(p_i) = \overrightarrow{n_i}$ and $v(x) = 0$ otherwise.
Solve:

$$\min_f \int_\Omega ||\nabla f(x) - \overrightarrow{n}(x)||^2 dx$$

i.e.:

$$\min_f \sum_{i=1}^{N} ||\nabla f(p_i) - \overrightarrow{n_i}||^2$$

Poisson Problem

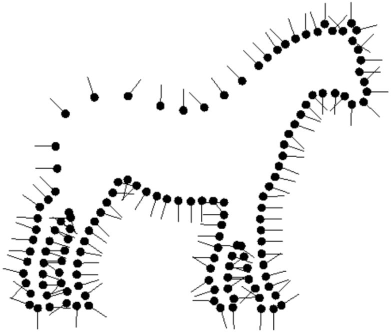
Vector field v is not integrable in general.
We apply the **divergence** operator (Euler-Lagrange equation):

$$\nabla.\nabla f = \Delta f = \nabla.\overrightarrow{n}$$

We recover a Poisson problem (of form $\Delta f = a$)

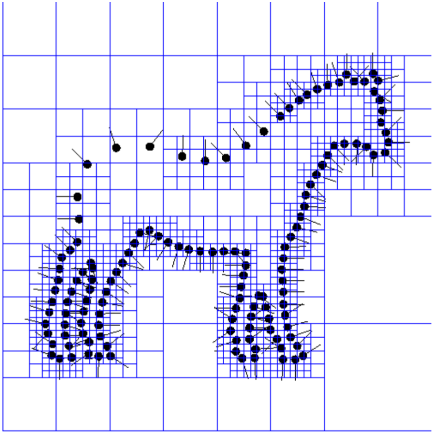
Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i



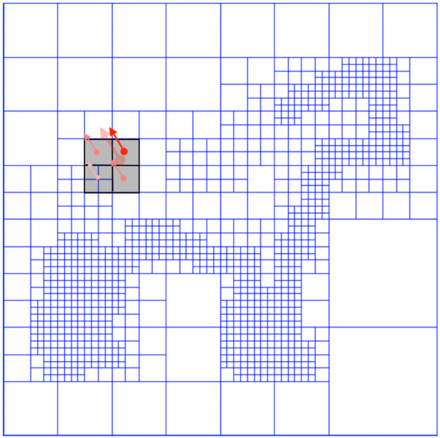
Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i
2. Setup an octree containing a single point per cell



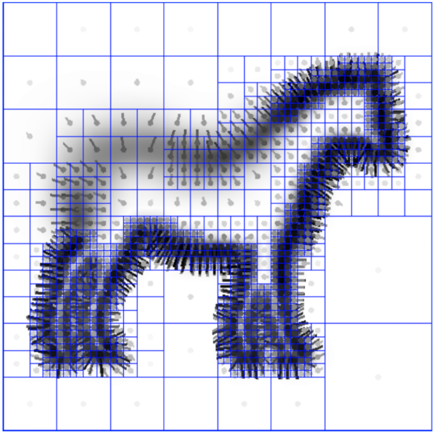
Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i
2. Setup an octree containing a single point per cell
3. Splat the samples (define a FEM basis)



Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i
2. Setup an octree containing a single point per cell
3. Splat the samples (define a FEM basis)



Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i
2. Setup an octree containing a single point per cell
3. Splat the samples (define a FEM basis)
4. Solve the Poisson problem and recover indicator function

Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i
2. Setup an octree containing a single point per cell
3. Splat the samples (define a FEM basis)
4. Solve the Poisson problem and recover indicator function
5. Recover the interface via marching cubes

Implementation: Solving the Poisson problem on an octree via FEM

1. Consider a dataset of points p_i with normals n_i
2. Setup an octree containing a single point per cell
3. Splat the samples (define a FEM basis)
4. Solve the Poisson problem and recover indicator function
5. Recover the interface via marching cubes

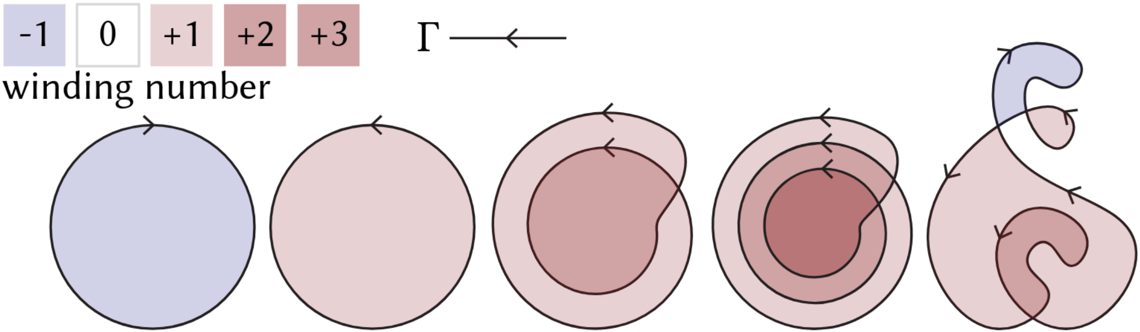
DEMO

- Result is a watertight manifold surface
- Can also be implemented on a mesh instead of an octree

- Needs consistently oriented normals
- Implementation is not trivial on an octree

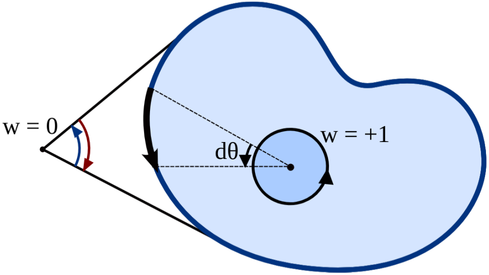
Generalized Winding Number

Winding Number: another possible implicit representation

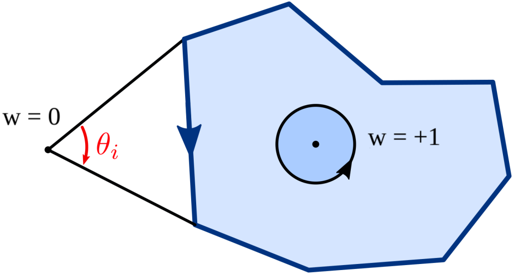


https://nzfeng.github.io/research/WNoDS/PerspectivesOnWindingNumbers.pdf

Winding Number: another possible implicit representation



$$w(p) = \frac{1}{2\pi} \oint_{\mathcal{C}} d\theta$$

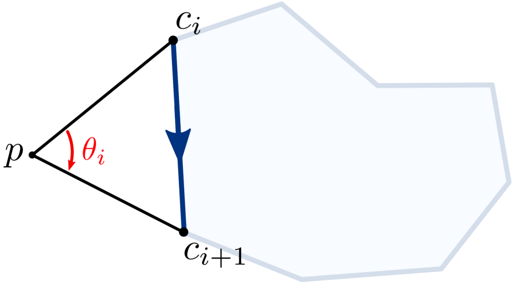


$$w(p) = \frac{1}{2\pi} \sum_i \theta_i$$

Computing Winding Number of Polylines

Let $a = c_i - p$ and $b = c_{i+1} - p$.

$$\tan(\theta_i) = \frac{\det(a,b)}{a.b} = \frac{a_x b_y - a_y b_x}{a_x b_x + a_y b_y}$$



$$w(p) = \frac{1}{2\pi} \sum_i \theta_i$$

Generalizing for imperfect geometries

Idea

Winding number is a sum of "solid angle" weighted by "area":

Robust Inside-Outside Segmentation Using Generalized Winding Numbers, Jacobson et al., 2013

Generalizing for imperfect geometries

Idea

Winding number is a sum of "solid angle" weighted by "area":

Generalized Winding Number

For points $p_1, ..., p_N$ with normals n_i and local areas a_i, the generalized winding number at point q is:

$$w(q) = \frac{1}{4\pi} \sum_i a_i \frac{(q - p_i).n_i}{||q - p_i||^3}$$

Robust Inside-Outside Segmentation Using Generalized Winding Numbers, Jacobson et al., 2013

Generalizing for imperfect geometries

Idea

Winding number is a sum of "solid angle" weighted by "area":

Generalized Winding Number

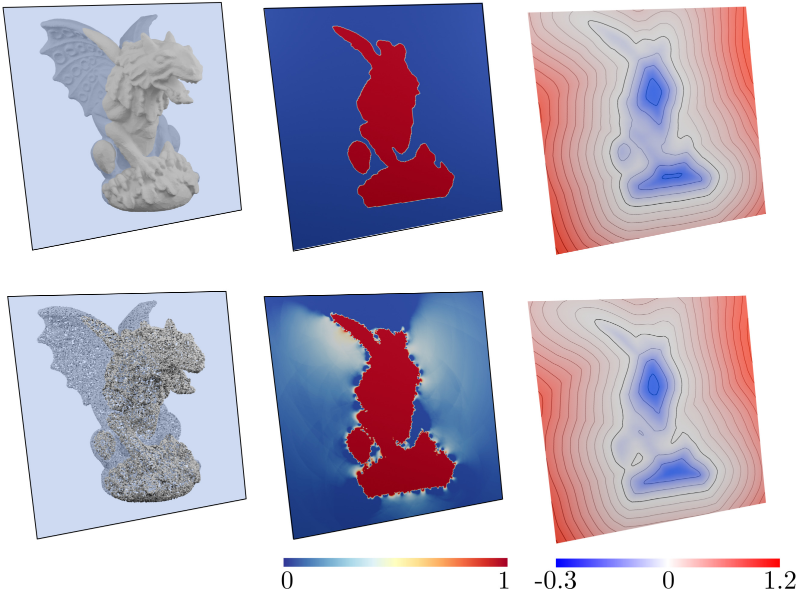
For points $p_1, ..., p_N$ with normals n_i and local areas a_i, the generalized winding number at point q is:

$$w(q) = \frac{1}{4\pi} \sum_i a_i \frac{(q - p_i).n_i}{||q - p_i||^3}$$

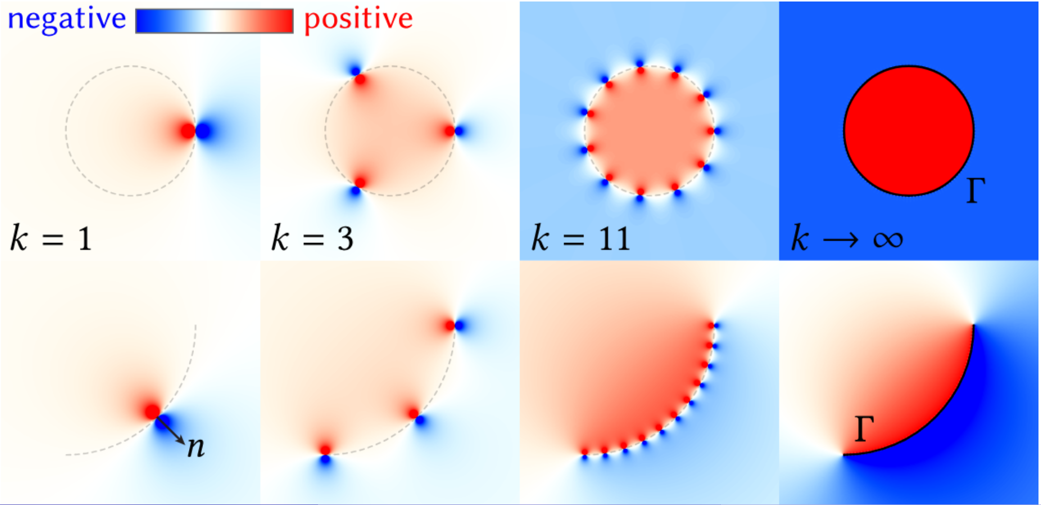
A good representation of the underlying surface of $p_1, ..., p_N$ is the isovalue $\frac{1}{2}$ of w.

https://observablehq.com/@rreusser/fast-generalized-winding-numbers-in-2d

Robust Inside-Outside Segmentation Using Generalized Winding Numbers, Jacobson et al., 2013


0
1
-0.3
0
1.2

The generalized winding number corresponds to the electric potential of infinitely many dipoles scattered on the surface ⇒ It's a harmonic function ($\Delta w = \rho$)



Relation with Poisson Surface Reconstruction

The two methods are actually equivalent in theory!

- Both solve a Laplace equation $\Delta f = a$ under constraint that f "jumps" from 0 to 1 at the interface
- Surface in Poisson is the discontinuity of the function. Surface of GWN is isovalue $\frac{1}{2}$
- Only the implementations differ. GWN does not need to splat the points $\Rightarrow$ more precise
- Both need normals, but GWN also needs an approximation of the density

Fast Winding Numbers for Soups and Clouds, Barill et al., 2018

References I

Brown, C. (1976). Principal Axes and Best-Fit Planes, with Applications.

Pauly, M., Gross, M., & Kobbelt, L. (2002). Efficient simplification of point-sampled surfaces. *IEEE Visualization, 2002. VIS 2002.*, 163–170. https://doi.org/10.1109/VISUAL.2002.1183771

Gumhold, S., Wang, X., & MacLeod, R. (n.d.). Feature Extraction from Point Clouds.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., & Stuetzle, W. (1992). Surface reconstruction from unorganized points. *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, 71–78. https://doi.org/10.1145/133994.134011

Barill, G., Dickson, N. G., Schmidt, R., Levin, D. I. W., & Jacobson, A. (2018). Fast winding numbers for soups and clouds. *ACM Transactions on Graphics*, *37*(4), 1–12. https://doi.org/10.1145/3197517.3201337

References II

Schnabel, R., Wahl, R., & Klein, R. (2007). Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, *26*(2), 214–226. https://doi.org/10.1111/j.1467-8659.2007.01016.x

Edelsbrunner, H., Kirkpatrick, D., & Seidel, R. (1983). On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, *29*(4), 551–559. https://doi.org/10.1109/TIT.1983.1056714

Edelsbrunner, H., & Mücke, E. P. (1994). Three-dimensional alpha shapes. *ACM Transactions On Graphics (TOG)*, *13*(1), 43–72.

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., & Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, *5*(4), 349–359. https://doi.org/10.1109/2945.817351

Amenta, N., Bern, M., & Kamvysselis, M. (1998). A new Voronoi-based surface reconstruction algorithm. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '98*, 415–421. https://doi.org/10.1145/280814.280947

References III

Ricci, A. (1973). A constructive geometry for computer graphics. *The Computer Journal, 16*(2), 157–160. https://doi.org/10.1093/comjnl/16.2.157

Sharp, N., & Jacobson, A. (2022). Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics, 41*(4), 1–16. https://doi.org/10.1145/3528223.3530155

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics, 21*(4), 163–169. https://doi.org/10.1145/37402.37422

Hart, J. (1995). Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer, 12*. https://doi.org/10.1007/s003710050084

Sawhney, R., & Crane, K. (2020). Monte Carlo geometry processing: A grid-free approach to PDE-based methods on volumetric domains. *ACM Transactions on Graphics, 39*(4), 123:123:1–123:123:18. https://doi.org/10.1145/3386569.3392374

Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson surface reconstruction. *Proceedings of the fourth Eurographics symposium on Geometry processing, 7*(4).

References IV

Jacobson, A., Kavan, L., & Sorkine-Hornung, O. (2013). Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics*, *32*(4), 33:1–33:12. https://doi.org/10.1145/2461912.2461916