LECTURE 28: DIVIDE-AND-CONQUER ALGORITHM FOR VORONOI DIAGRAMS

PRESENTED BY

SANDEEP CHATTERJEE

17th April 2025, Kolkata

LECTURE 28: DIVIDE-AND-CONQUER ALGORITHM FOR VORONOI DIAGRAMS

- Planar Voronoi Diagrams
- Delaunay triangulation
- Algorithms to compute them
- Divide-and-conquer algorithm
- Analysis
- Computing the contour
- Monotonous single polygonal curve
- Topmost edge of the contour as the perpendicular bisector for the upper tangent.

17th April 2025, Kolkata

SANDEEP CHATTERJEE

Lecture 28: Divide-and-Conquer Algorithm for Voronoi Diagrams

Planar Voronoi Diagrams: Recall that, given n points $P = \{p_1, p_2, \ldots, p_n\}$ in the plane, the Voronoi polygon of a point p_i, $V(p_i)$, is defined to be the set of all points q in the plane for which p_i is among the closest points to q in P. That is,

$$V(p_i) = \{q \ : \ |p_i - q| \leq |p_j - q|, \forall j \neq i\}.$$

The union of the boundaries of the Voronoi polygons is called the *Voronoi diagram* of P, denoted $VD(P)$. The dual of the Voronoi diagram is a triangulation of the point set, called the *Delaunay triangulation*. Recall from our discussion of quad-edge data structure, that given a good representation of any planar graph, the dual is easy to construct. Hence, it suffices to show how to compute either one of these structures, from which the other can be derived easily in $O(n)$ time.

There are a number of algorithms for computing Voronoi diagrams and Delaunay triangulations in the plane. These include:

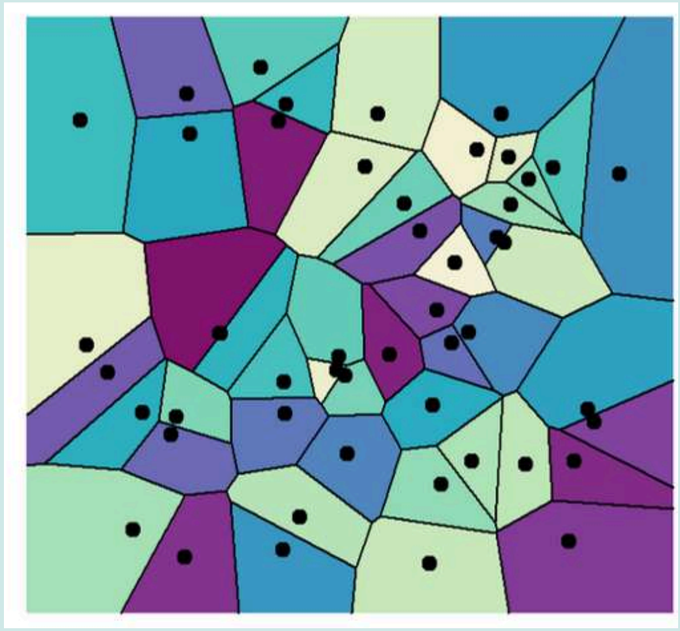
Divide-and-Conquer: (For both VD and DT.) The first $O(n \log n)$ algorithm for this problem. Not widely used because it is somewhat hard to implement. Can be generalized to higher dimensions with some difficulty. Can be generalized to computing Voronoi diagrams of line segments with some difficulty.

Randomized Incremental: (For DT.) The simplest. $O(n \log n)$ time with high probability. Can be generalized to higher dimensions as with the randomized algorithm for convex hulls. Can be generalized to computing Voronoi diagrams of line segments fairly easily.

Fortune's Plane Sweep: (For VD.) A very clever and fairly simple algorithm. It computes a "deformed" Voronoi diagram by plane sweep in $O(n \log n)$ time, from which the true diagram can be extracted easily. Can be generalized to computing Voronoi diagrams of line segments fairly easily.

Reduction to convex hulls: (For DT.) Computing a Delaunay triangulation of n points in dimension d can be reduced to computing a convex hull of n points in dimension $d + 1$. Use your favorite convex hull algorithm. Unclear how to generalize to compute Voronoi diagrams of line segments.

Voronoi Diagrams



Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in $\mathbb{R}^d$, which we call sites.
We want to subdivide $\mathbb{R}^d$ into regions according to the which site is most “influential”.

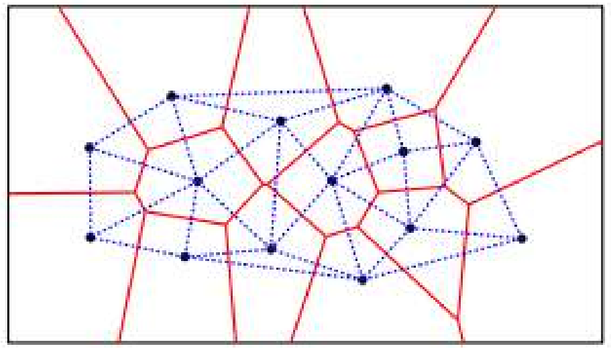
$$\mathcal{V}_P(p_i) = \{q \in \mathbb{R}^d : \|p_i - q\| < \|p_j - q\|,\ \forall j \neq i\}$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial\mathcal{V}_P(p_i)$$

Planar Vononoi Diagrams

Delaunay triangulation as dual of VRD

The dual of the Voronoi diagram is a triangulation of the point set



Voronoi diagram and Delaunay triangulation

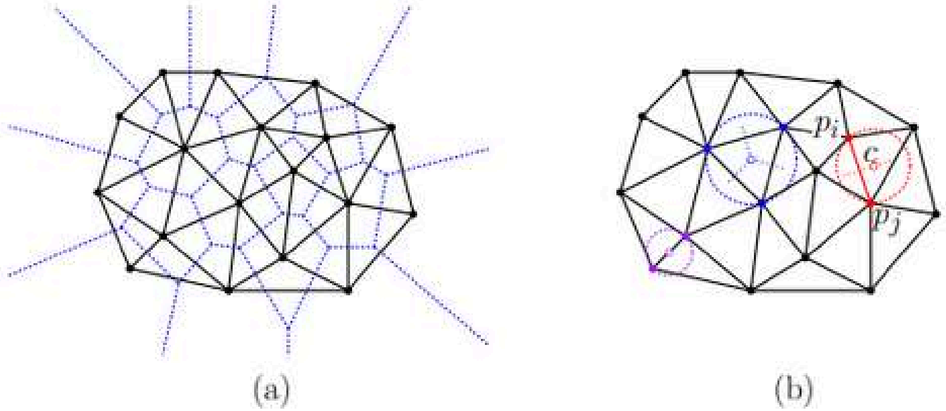
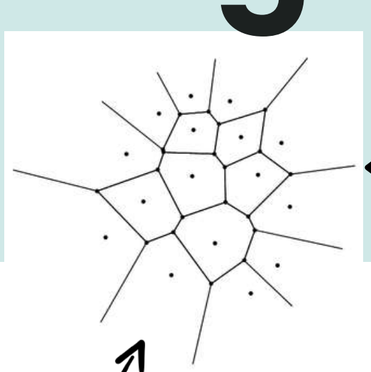
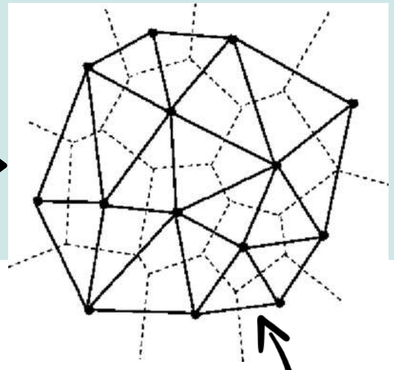


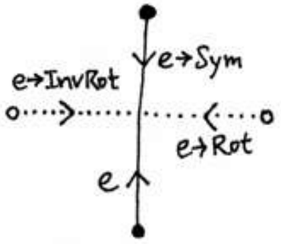
Fig. 62: (a) The Voronoi diagram of a set of sits (broken lines) and the corresponding Delaunay triangulation (solid lines) and (b) circle-related properties.

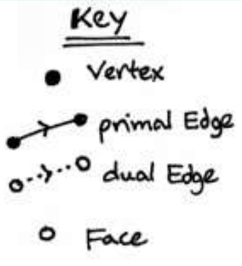
Conversion through Quad Edge operations

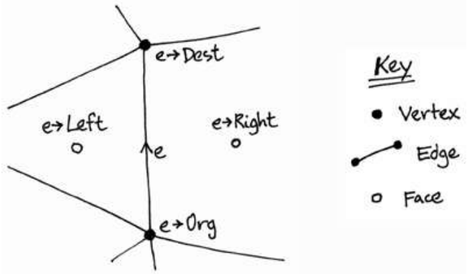
$O(n)$

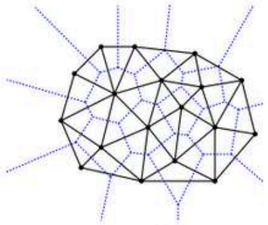




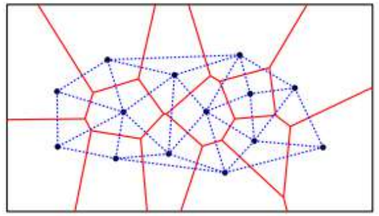




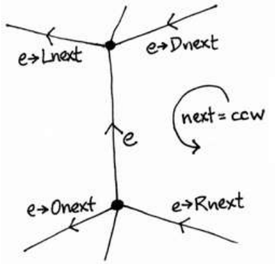




(a)



Voronoi diagram and Delaunay triangulation



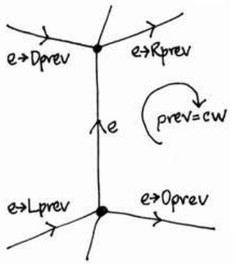


Fig. 62: (a) The Voronoi diagram of a set of sits (broken lines) and the corresponding Delaunay triangulation (solid lines) and (b) circle-related properties.

Quad Edge Data Structure

How to Compute PVD ?

There are a number of algorithms for computing Voronoi diagrams and Delaunay triangulations in the plane. These include:

Naive Algorithm

Divide-and-Conquer

Randomized Incremental

Fortune’s Plane Sweep

Reduction to convex hulls:

How to Compute PVD ?

There are a number of algorithms for computing the Voronoi diagram of a set of n sites in the plane. Of course, there is a naive O(n 2 log n) time algorithm, which operates by computing V(pi) by intersecting the n - 1 bisector halfplanes h(pi , pj), for j != i

$$H(p_i, p_j) = \{q \in \mathbb{R}^d : 2q \cdot (p_j - p_i) = \|p_j\|^2 - \|p_i\|^2\}$$

$$h(p_i, p_j) = \{q \in \mathbb{R}^d : 2q \cdot (p_j - p_i) \leq \|p_j\|^2 - \|p_i\|^2\}$$

$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

$O(1)$ $O(n \cdot logn)$ $O(n^2 \cdot logn)$

Naive Algorithm

How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm

$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm

$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm

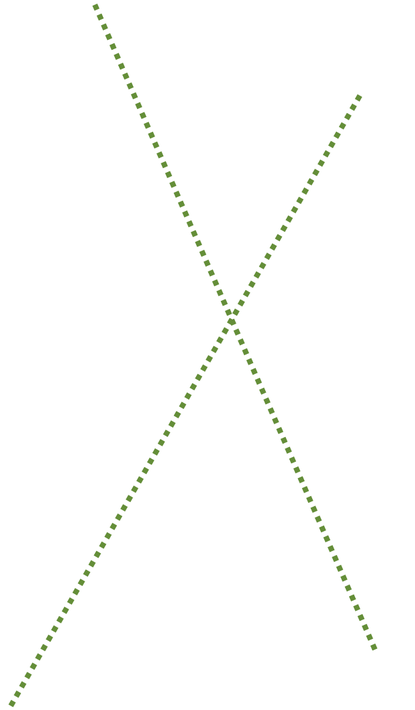
$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm



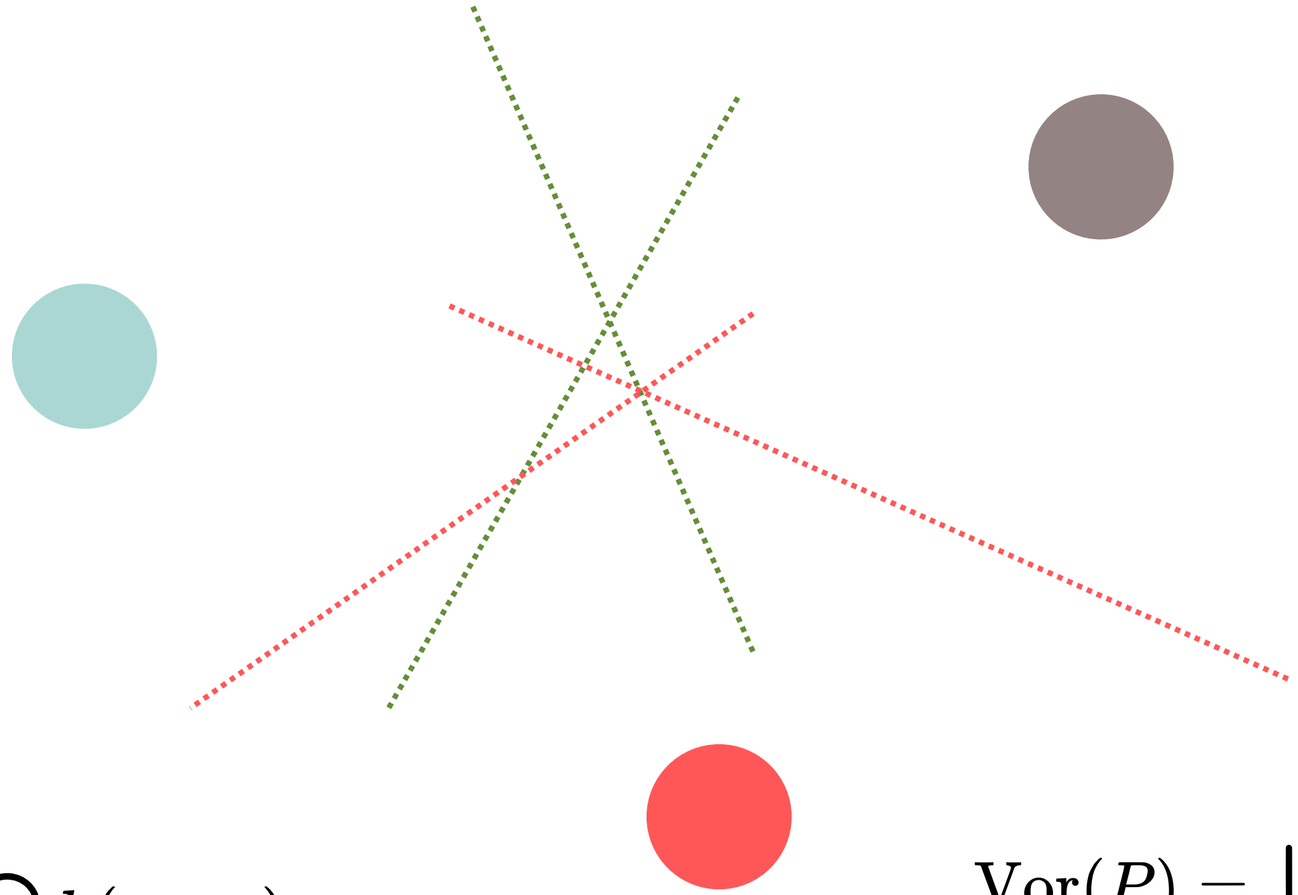
$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm



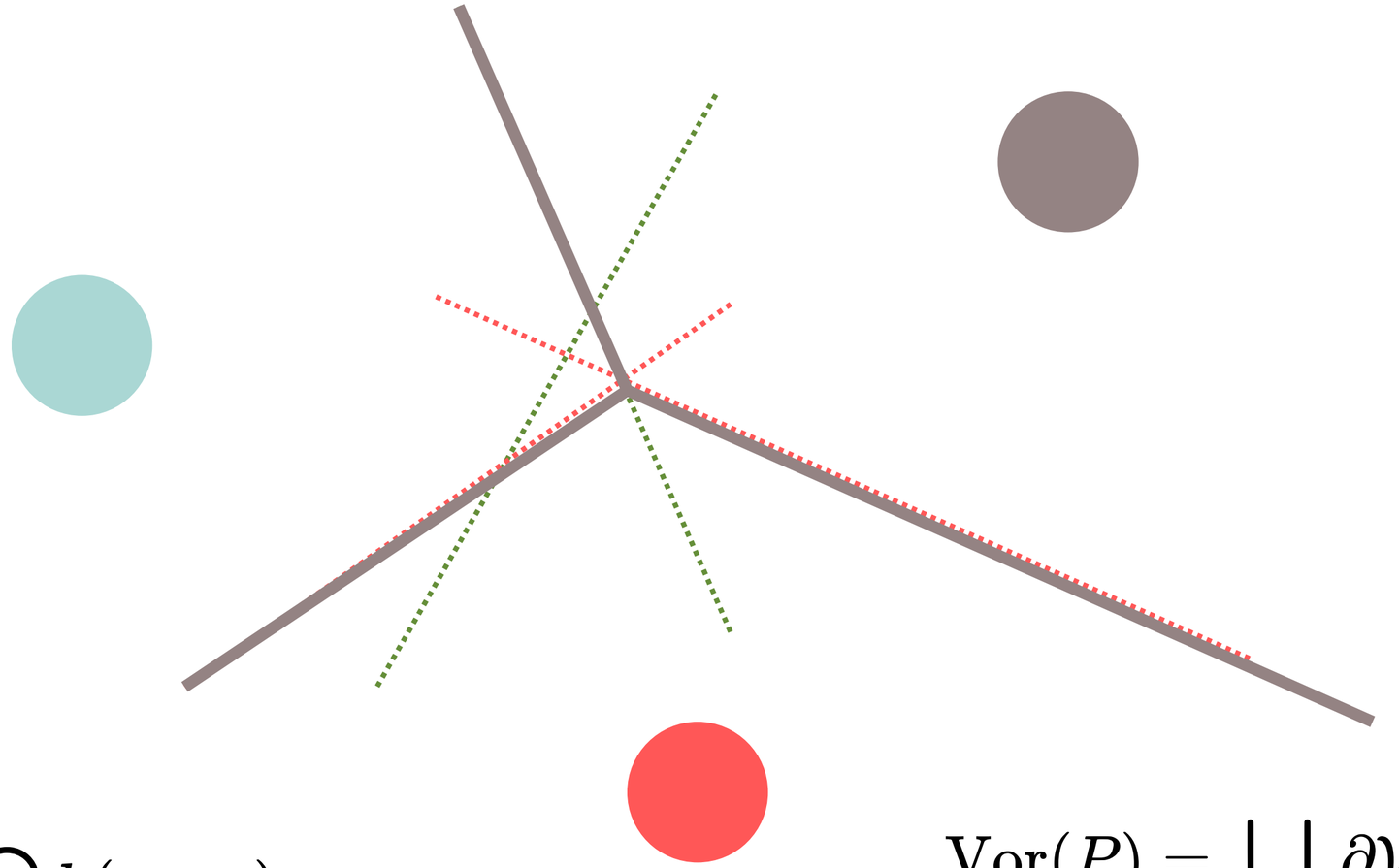
$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm



$$\mathcal{V}_P(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$$

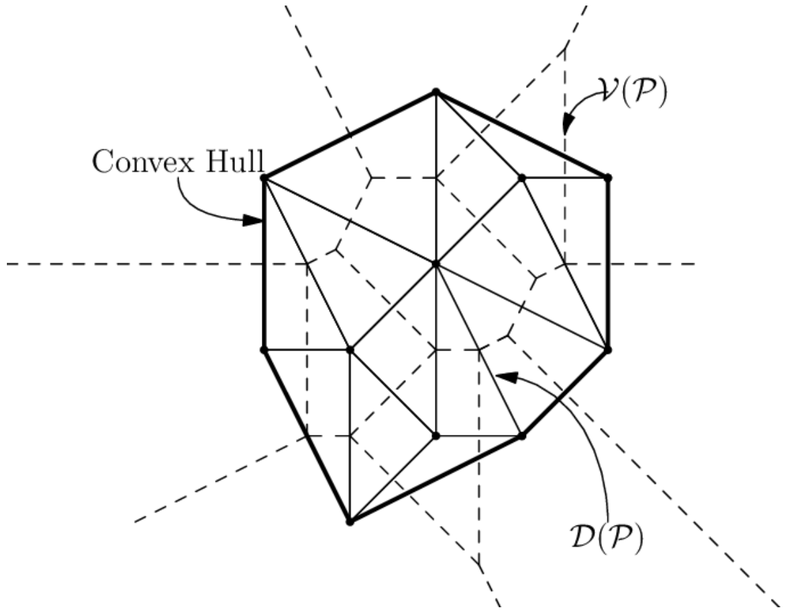
$$\mathrm{Vor}(P) = \bigcup_{p_i \in P} \partial \mathcal{V}_P(p_i)$$

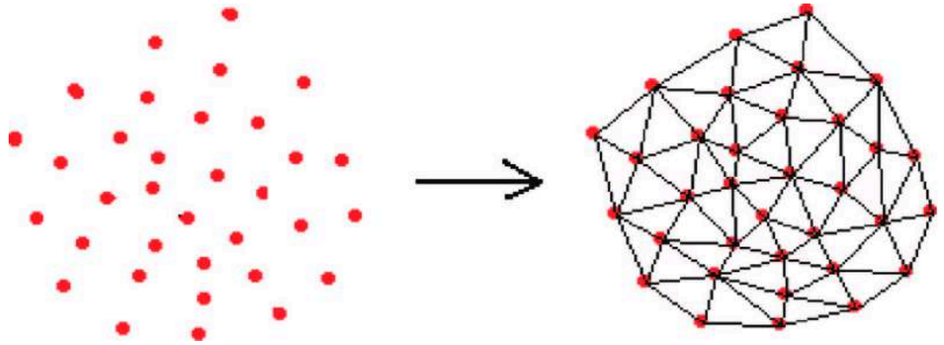
How to Compute PVD ?

$$O(n^2 \cdot logn)$$

Naive Algorithm

What should be a lower bound ?

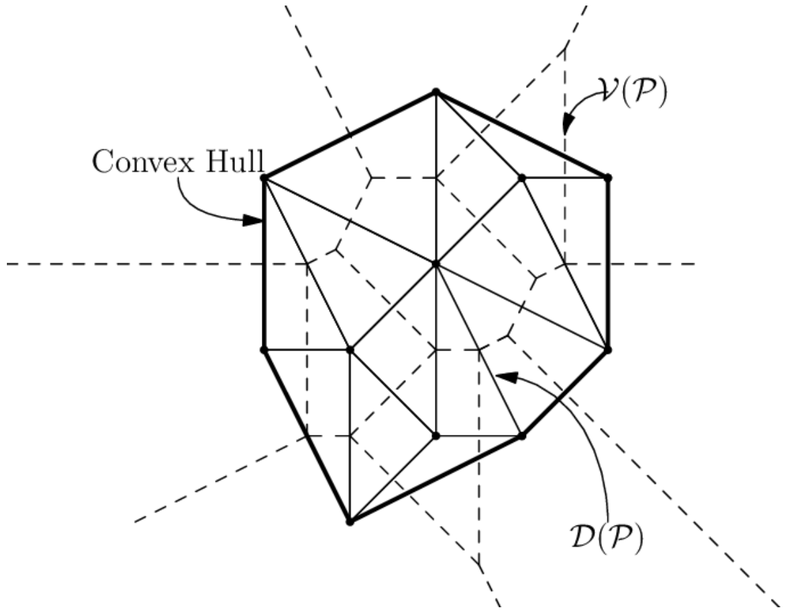




How to Compute PVD ?

$O(n^2 \cdot logn)$ Naive Algorithm

What should be a lower bound ?



P → Vor(P)

$f(n) \quad \in \Omega(n \cdot logn)$

PVD(P) → DelTr(P) →CHull(P)->Sort(P)

$f(n) \quad O(n) \quad O(n) \quad O(n)$

Vor(P) → **Sort(P)** $\in \Omega(n \cdot logn)$

$\leq_P$

Randomized Incremental

🪄 High-level idea

1. **Randomize the input points: Randomize the order of the points p1,p2,...,pn**
2. **Start with a large triangle that contains all points (called the "super triangle").**
3. **Incrementally insert points one by one and update the Delaunay triangulation accordingly.**

Randomized Incremental

- Initialization:
- Start with a triangle that encloses all the input points (this will eventually be discarded).
- This ensures a valid triangulation exists from the beginning.
- Insert points one by one (in random order):
- For each point ppp:
  - Locate the triangle that contains ppp.
  - This can be done using a history DAG (search structure maintained from previous insertions).
  - Split the triangle into three by connecting ppp to the triangle’s vertices.
  - This may violate the Delaunay condition (no point inside the circumcircle of any triangle).
  - Fix violations by "edge flipping": recursively check edges opposite to ppp and flip if needed.
- After all insertions:
- Remove any triangle that includes a vertex from the initial enclosing triangle.
- The remaining triangles form the Delaunay triangulation.
- To get Voronoi diagram:
- Use the dual graph: each triangle's circumcenter becomes a Voronoi vertex.
- Edges connect circumcenters of adjacent triangles.

Fortune’s Plane Sweep:

Later, Steven Fortune discovered a plane sweep algorithm for the problem, which provided a simpler O(n log n) solution to the problem. It is his algorithm that we will discuss. Somewhat later still, it was discovered that the incremental algorithm is actually quite efficient, if it is run as a randomized incremental algorithm. We will discuss a variant of this algorithm later when we talk about the dual structure, called the Delaunay triangulation

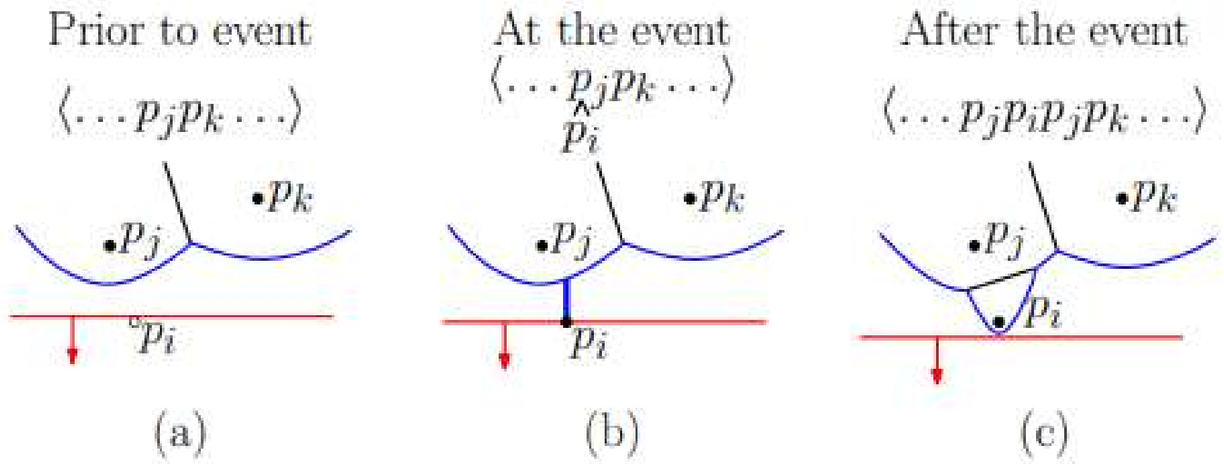


Fig. 60: Site event.

Fortune’s Plane S

Later, Steven Fortune discovered a plane sweep algo
provided a simpler O(n log n) solution to the proble
will discuss. Somewhat later still, it was discov
algorithm is actually quite efficient, if it is run as
algorithm. We will discuss a variant of this algorithm
dual structure, called the Delaunay triangulation

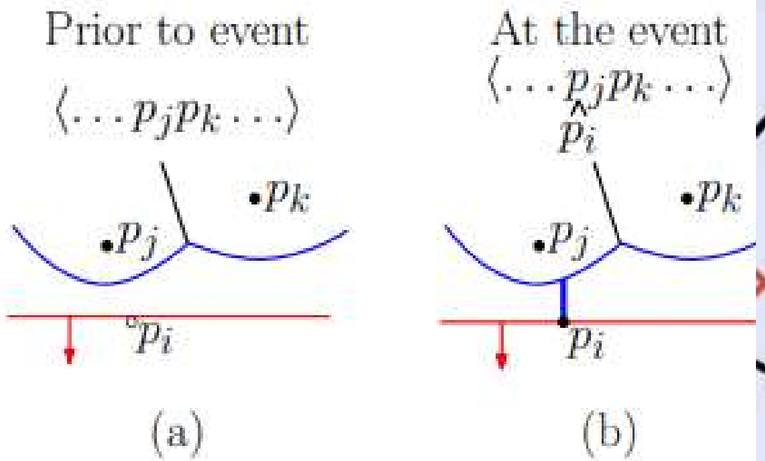
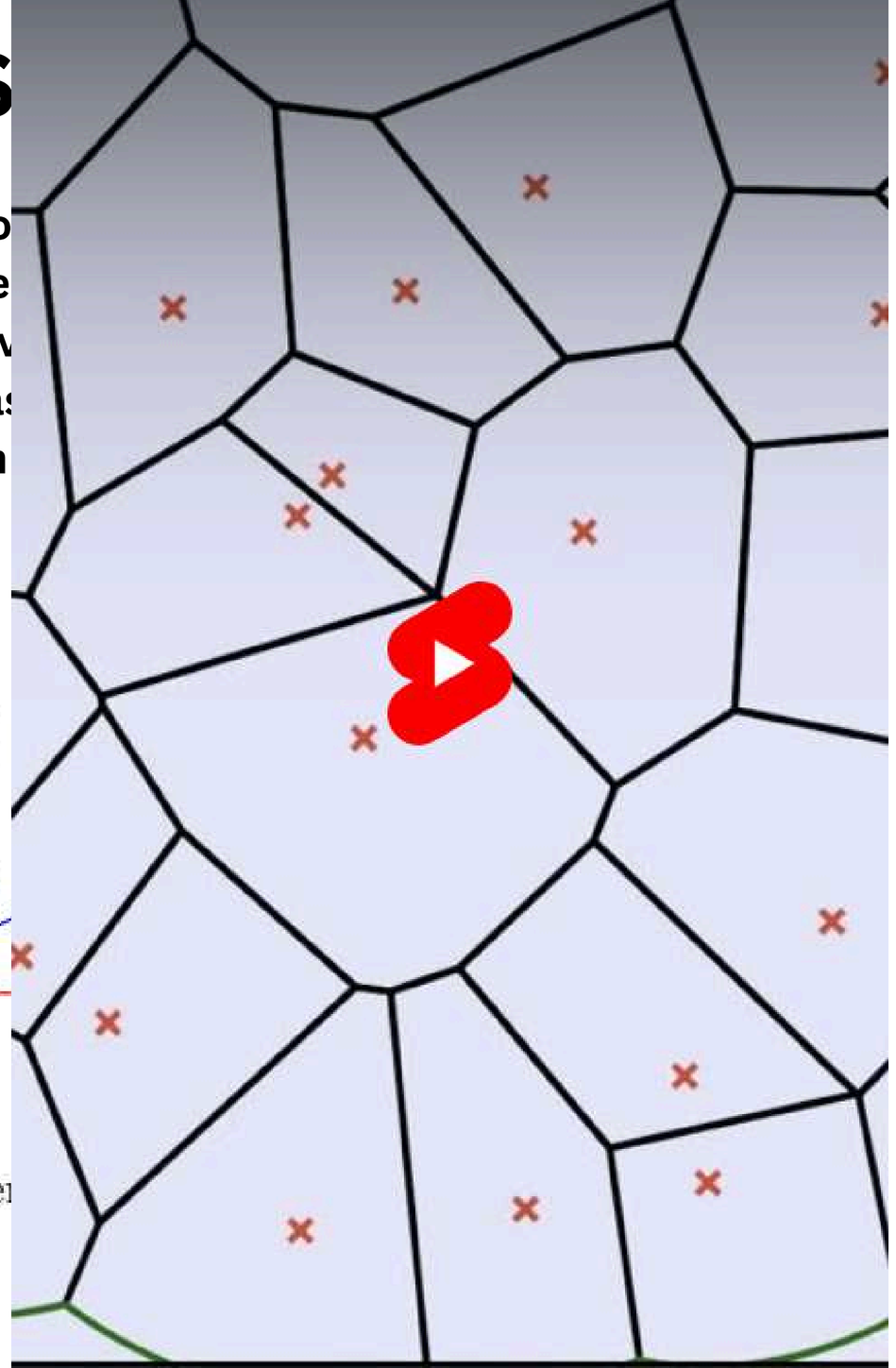


Fig. 60: Site even



Reduction to convex hulls:

🧩 What is the Reduction ?

The key idea is:

> **You can compute the Delaunay triangulation (DT) of a set of points in $\mathbb{R}^d$ by computing the convex hull of a transformed set of points in $\mathbb{R}^{d+1}$.**

In 2D, this means:

DT in 2D ⟶ Compute a convex hull in 3D

🎯 What's the trick (geometrically)?

The transformation maps each point (x, y) in 2D to a point $(x, y, x^2 + y^2)$ in 3D.
This lifts the point onto a **paraboloid** in 3D.

So, if your original point set is $P \subset \mathbb{R}^2$, define:

$$P^{\uparrow} = \{(x, y, x^2 + y^2) \mid (x, y) \in P\}$$

Then:

- Compute the **lower convex hull** of $P^{\uparrow}$ in $\mathbb{R}^3$
- Project the faces of the lower hull **back down** to 2D

The result? ✨ You get the **Delaunay triangulation** of the original set.

Reduction to convex hulls:

🧩 What is the Reduction ?

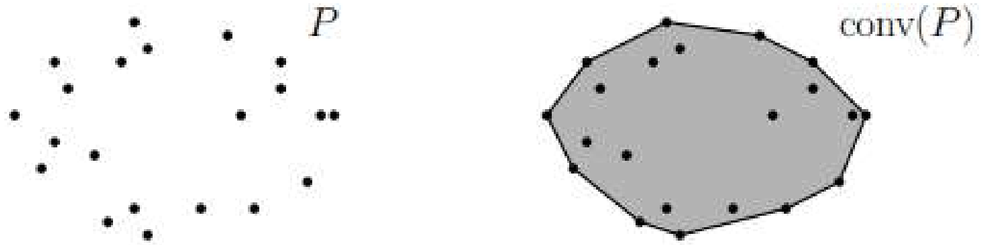
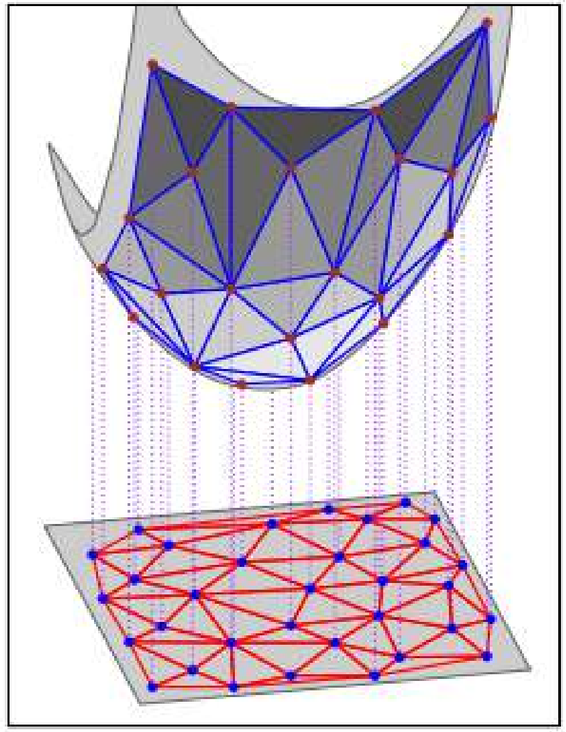


Fig. 7: A point set and its convex hull.



🦉 Why does this work?

The geometry of the paraboloid ensures that:

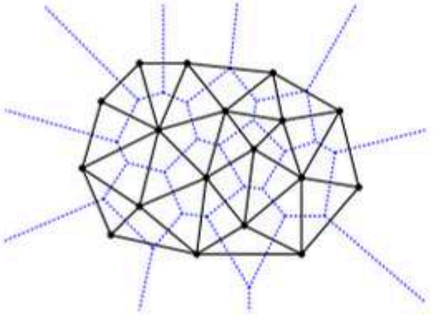
- A face on the lower convex hull in 3D corresponds to a triangle whose **circumcircle contains no other points** in 2D.
- This is exactly the **Delaunay condition**!

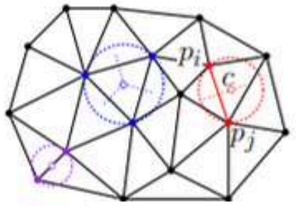
So the geometry of convexity in 3D naturally encodes the Delaunay triangulation in 2D.

Lecture 11: Delaunay Triangulations: General Properties

Delaunay Triangulations: We have discussed the topic of Voronoi diagrams. In this lecture, we consider a related structure, called the *Delaunay triangulation* (DT). The Voronoi diagram of a set of sites in the plane is a planar subdivision, in fact, a cell complex. The *dual* of such subdivision is a cell complex that is defined as follows. For each face of the Voronoi diagram, we create a vertex (corresponding to the site). For each edge of the Voronoi diagram lying between two sites p_i and p_j, we create an edge in the dual connecting these two vertices. Each vertex of the Voronoi diagram corresponds to a face of the dual complex.

Recall that, under the assumption of general position (no four sites are collinear), the vertices of the Voronoi diagram all have degree three. It follows that the faces of the resulting dual complex (excluding the exterior face) are triangles. Thus, the resulting dual graph is a *triangulation* of the sites. This is called the *Delaunay triangulation* (see Fig. 62(a)).

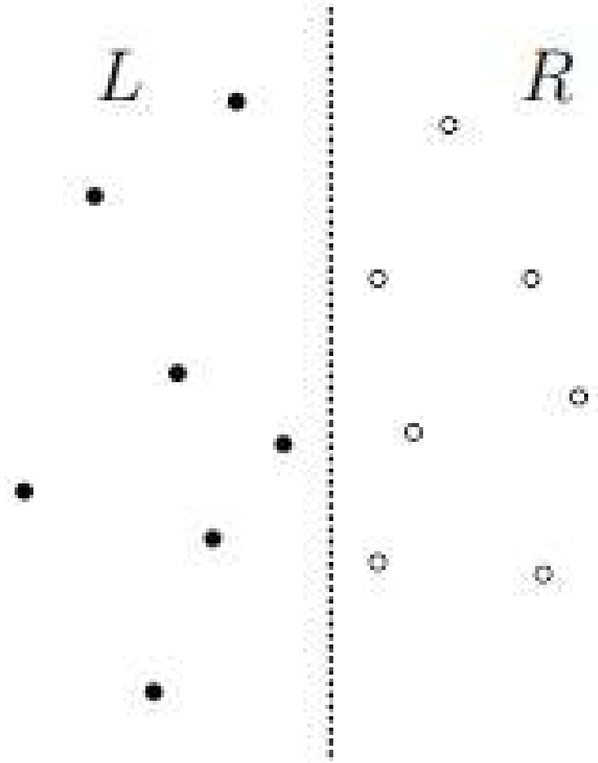




Delaunay Triangulations and Convex Hulls: Let us begin by considering the *paraboloid* Ψ defined by the equation $z = x^2 + y^2$. Observe that the vertical cross sections (constant x or constant y) are parabolas, and whose horizontal cross sections (constant z) are circles. For each point in $\mathbb{R}^2$, $p = (p_x, p_y)$, the *vertical projection* (also called the *lifted image*) of this point onto this Ψ is $p^\uparrow = (p_x, p_y, p_x^2 + p_y^2)$ in $\mathbb{R}^3$.

Given a set of points P in the plane, let $P^\uparrow$ denote the projection of every point in P onto Ψ. Consider the *lower convex hull* of $P^\uparrow$. This is the portion of the convex hull of $P^\uparrow$ which is visible to a viewer standing at $z = -\infty$. We claim that if we take the lower convex hull of $P^\uparrow$, and project it back onto the plane, then we get the Delaunay triangulation of P (see Fig. 67). In particular, let $p, q, r \in P$, and let $p^\uparrow$, $q^\uparrow$, $r^\uparrow$ denote the projections of these points onto Ψ. Then $\triangle p^\uparrow q^\uparrow r^\uparrow$ defines a *face* of the lower convex hull of $P^\uparrow$ if and only if $\triangle pqr$ is a triangle of the Delaunay triangulation of P.

Divide and Conquer

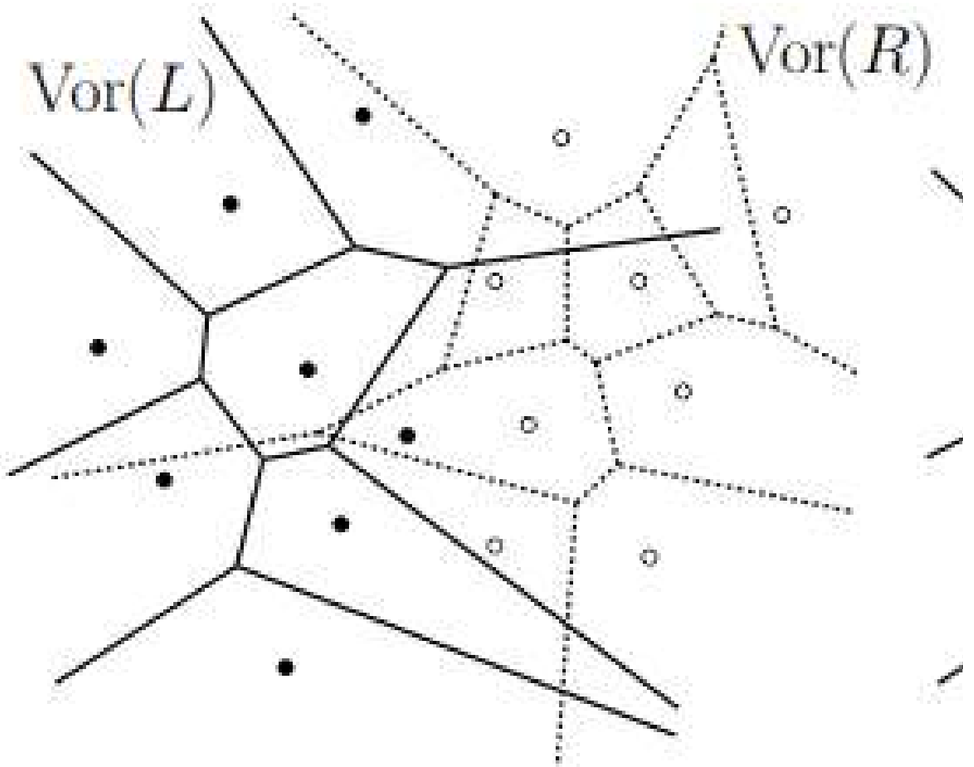


Divide

We split the points according to x-coordinates into two roughly equal sized groups, e.g., by presorting the points by x-coordinate and selecting medians

Conquer

Note that since each diagram alone covers the entire plane, these two diagrams overlap (see Fig. 161(c)). We then merge the resulting diagrams into a single diagram.



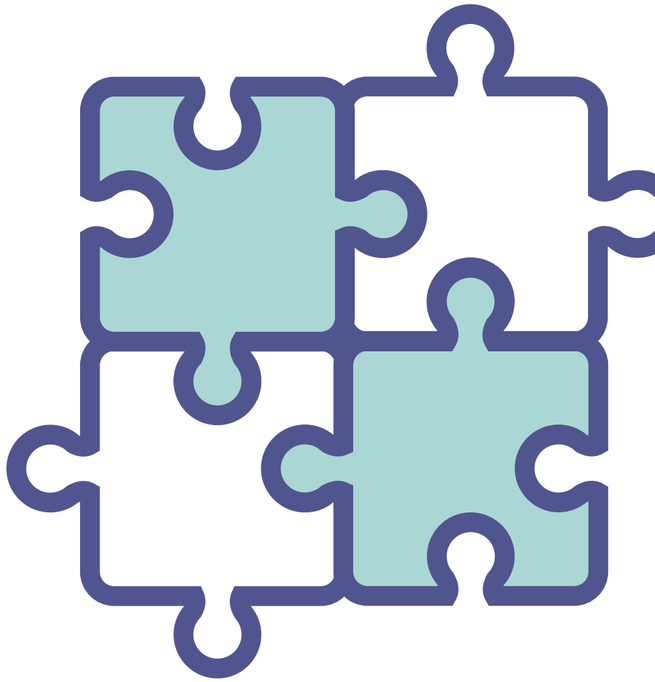
ALGORITHM

and analysis

(0) Presort the points by x-coordinate (this is done once).

(1) Split the point set S by a vertical line into two subsets L and R of roughly equal size.

(2) Compute Vor(L) and Vor(R) recursively. (These diagrams overlap one another.)



(3) Merge the two diagrams into a single diagram, by computing the contour and discarding the portion of the Vor(L) that is to the right of the contour, and the portion of Vor(R) that is to the left of the contour.

Can we can implement step
(3) in O(n) time (where n is the size of the remaining point set)

Mind (the) map

Merge does the work

Divide-and-conquer algorithm: The divide-and-conquer approach works like most standard geometric divide-and-conquer algorithms. We split the points according to x-coordinates into two roughly equal sized groups, e.g., by presorting the points by x-coordinate and selecting medians. We compute the Voronoi diagram of the left side, and the Voronoi diagram of the right side. Note that since each diagram alone covers the entire plane, these two diagrams overlap. We then merge the resulting diagrams into a single diagram.

Divide

Conquer

Merge

Merging

what's the way?

$$T(n) = 2T(n/2) + 1$$

$$\theta(n), \theta(n^2)$$

The merging step is where all the work is done. Observe that every point in the the plane lies within two Voronoi polygons, one in Vor(L) and one in Vor(R). We need to resolve this

$$T(n) = 2 \cdot T(\frac{n}{2}) + n$$

$$\theta(nlogn)$$

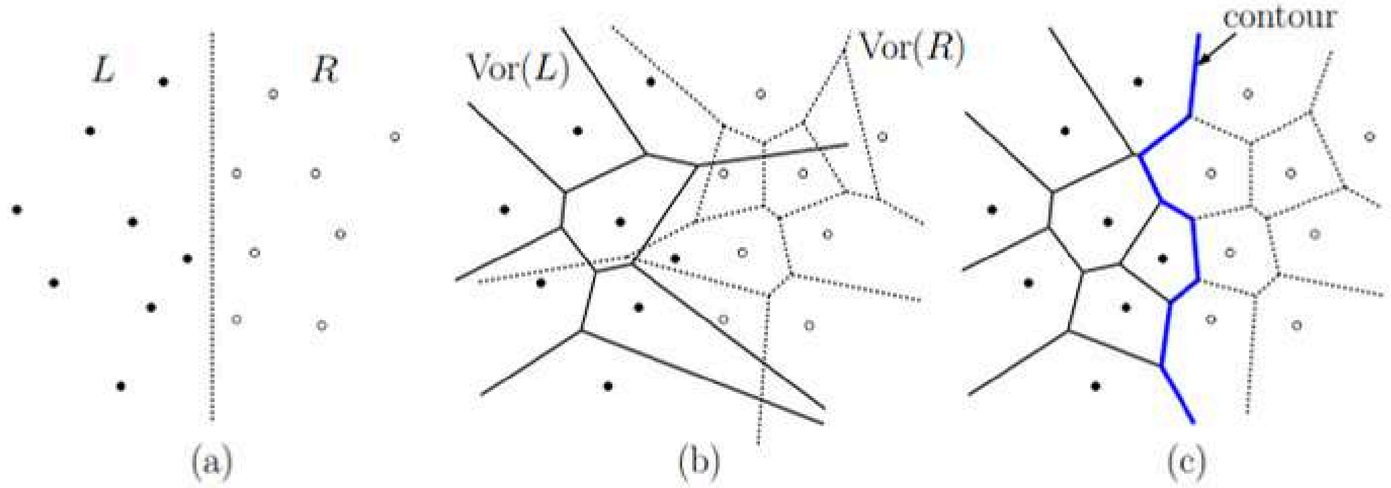


Fig. 161: Voronoi diagrams by divide-and-conquer.

Merging

$$T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$$

$$\rightarrow O(nlogn)$$

Computing the contour: What makes the divide-and-conquer algorithm somewhat tricky is the task of computing the contour. Before giving an algorithm to compute the contour, let us make some observations about its geomtetric structure. Let us make the usual simplifying assumptions that no four points are cocircular.

Lemma: The contour consists of a single polygonal curve (whose first and last edges are semi-infinite) which is monotone with respect to the y-axis.

Lemma: The topmost (bottommost) edge of the contour is the perpendicular bisector for the two points forming the upper (lower) tangent of the left hull and the right hull.

Proof: This follows from the fact that the vertices of the hull correspond to unbounded Voronoi polygons, and hence upper and lower tangents correspond to unbounded edges of the contour.

Merging

These last two theorem suggest the general approach.

We start by computing the upper tangent, which we know can be done in linear time (once we know the left and right hulls, or by prune and search).

Then, we start tracing the contour from top to bottom.
When we are in Voronoi polygons V (l0) and V (r0) we trace the bisector between l0 and r0 in the negative y-direction until its first contact with the boundaries of one of these polygons.

Suppose that we hit the boundary of V (l0) first.

Merging

We start by computing the upper tangent, which we know can be done in linear time (once we know the left and right hulls, or by prune and search).

Then, we start tracing the contour from top to bottom.
When we are in Voronoi polygons V (l0) and V (r0) we trace the bisector between l0 and r0 in the negative y-direction until its first contact with the boundaries of one of these polygons.

Suppose that we hit the boundary of V (l0) first.

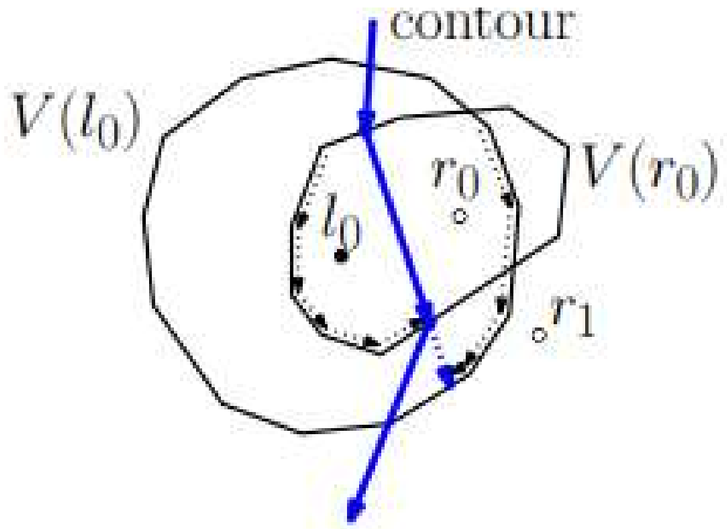


Fig. 162: Tracing the contour.

Merging

Assuming that we use a good data structure for the Voronoi diagram (e.g. quad-edge data structure) we can determine the point l1 lying on the other side of this edge in the left Voronoi diagram. We continue following the contour by tracing the bisector of l1 and r0.

However, in order to insure efficiency, we must be careful in how we determine where the bisector hits the edge of the polygon. We start tracing the contour between l0 and r0 (see Fig. 162). By walking along the boundary of V (l0) we can determine the edge that the contour hits first. This can be done in time proportional to the number of edges in V (l0) (which can be as large as O(n))

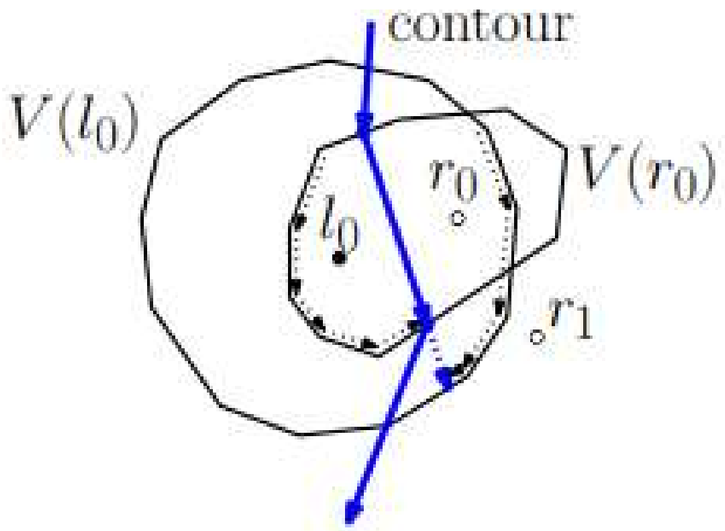


Fig. 162: Tracing the contour.

Merging

However, we discover that before the contour hits the boundary of V (l0) it hits the boundary of V (r0). We find the new point r1 and now trace the bisector between l0 and r1. Again we can compute the intersection with the boundary of V (l0) in time proportional to its size. However the contour hits the boundary of V (r1) first, and so we go on to r2. As can be seen, if we are not smart, we can rescan the boundary of V (l0) over and over again, until the contour finally hits the boundary. If we do this O(n) times, and the boundary of V (l0) is O(n), then we are stuck with O(n^2) time to trace the contou

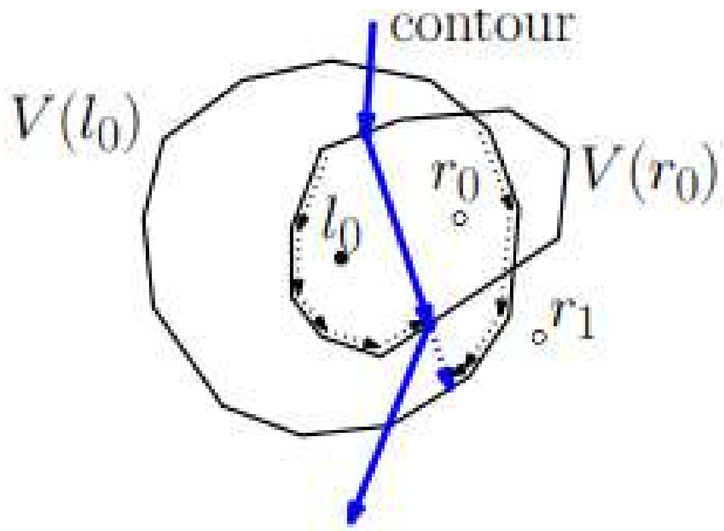


Fig. 162: Tracing the contour.

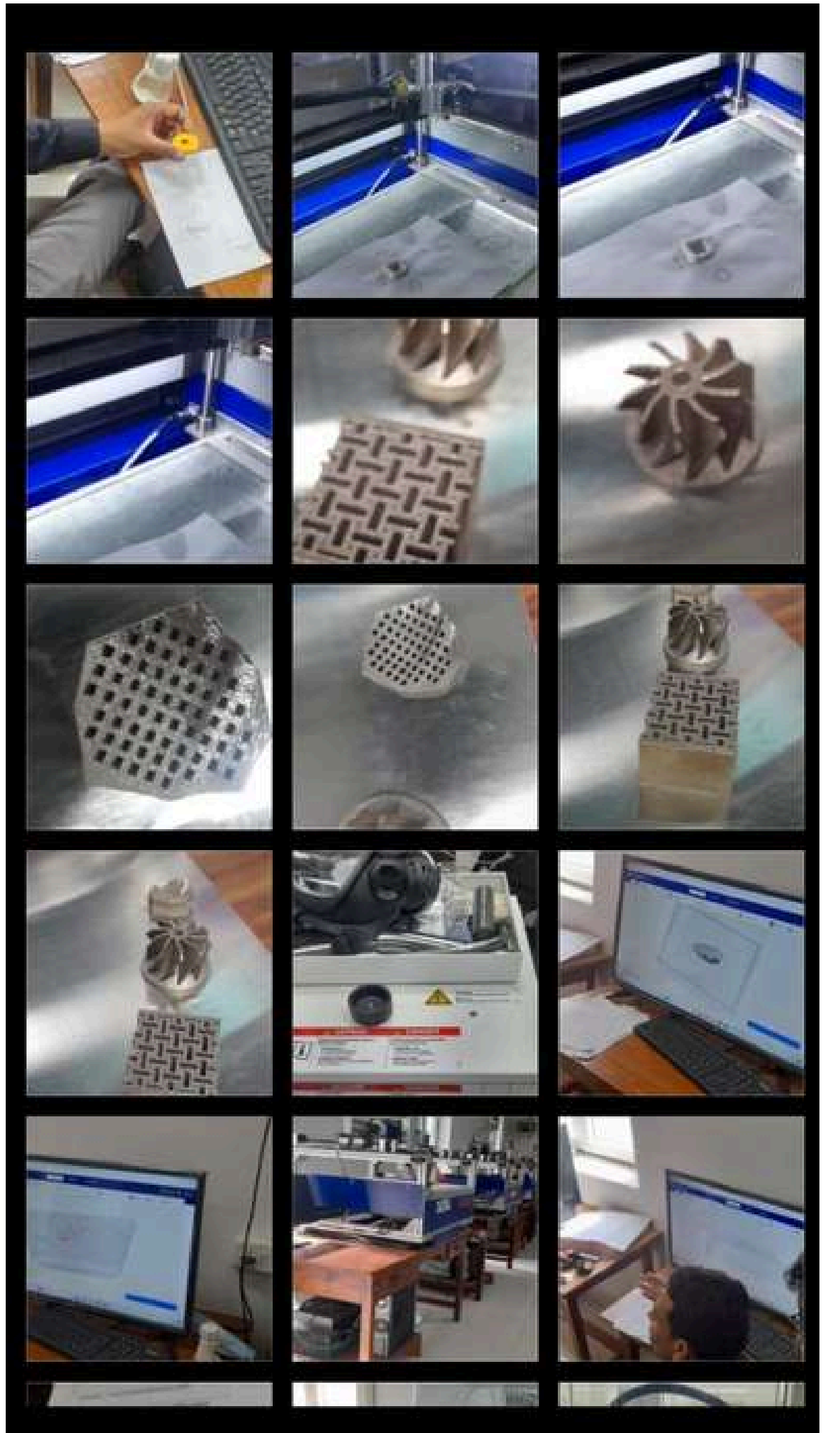
Merging

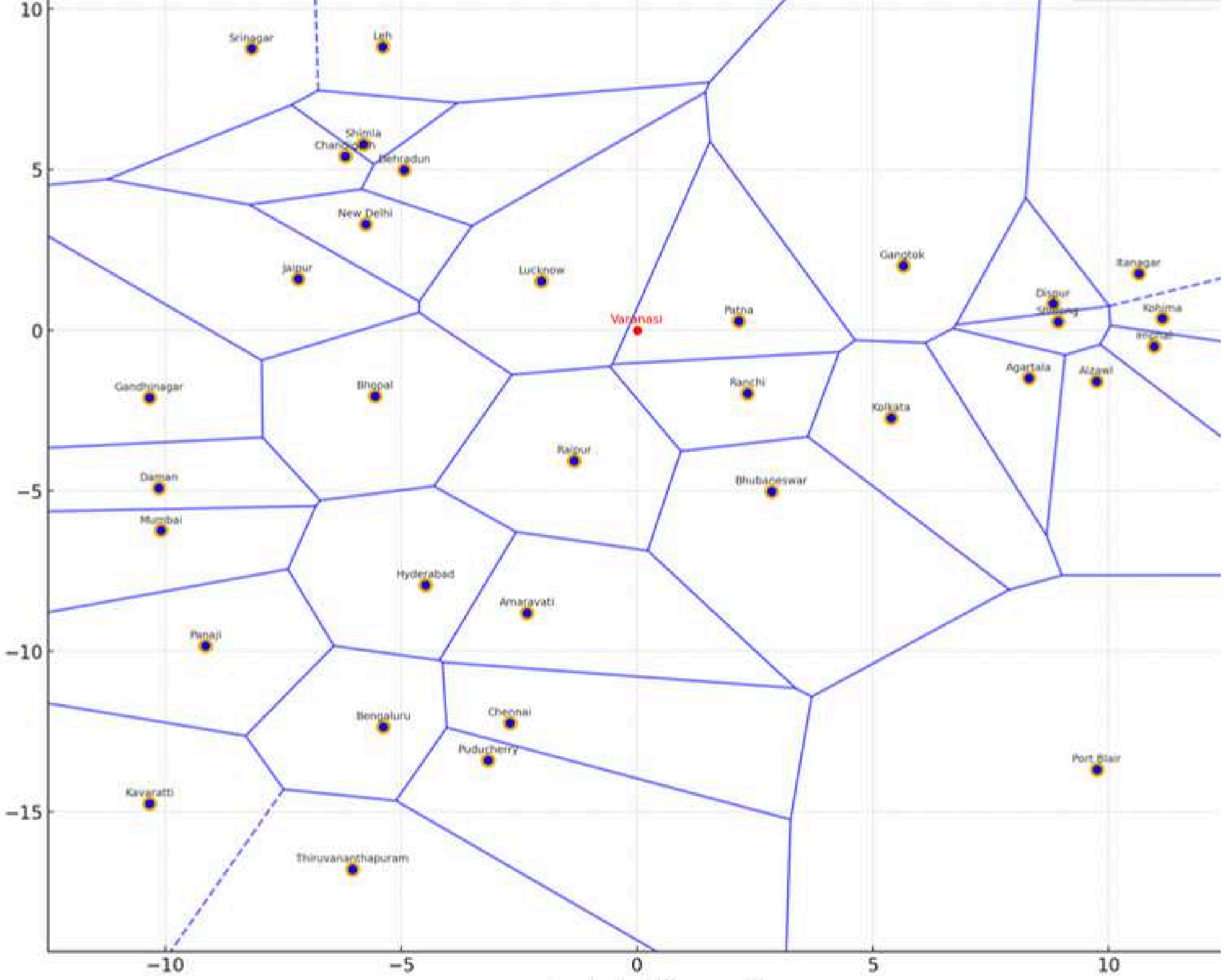
We have to avoid this repeated rescanning. However, there is a way to scan the boundary of each Voronoi polygon at most once.

Observe that as we walk along the contour, each time we stay in the same polygon V (lo), we are adding another edge onto its Voronoi polygon. Because the Voronoi polygon is convex, we know that the edges we are creating turn consistently in the same direction (clockwise for points on the left, and counterclockwise for points on the right). To test for intersections between the contour and the current Voronoi polygon, we trace the boundary of the polygon clockwise for polygons on the left side, and counterclockwise for polygons on the right side. Whenever the contour changes direction, we continue the scan from the point that we left off. In this way, we know that we will never need to rescan the same edge of any Voronoi polygon more than once.

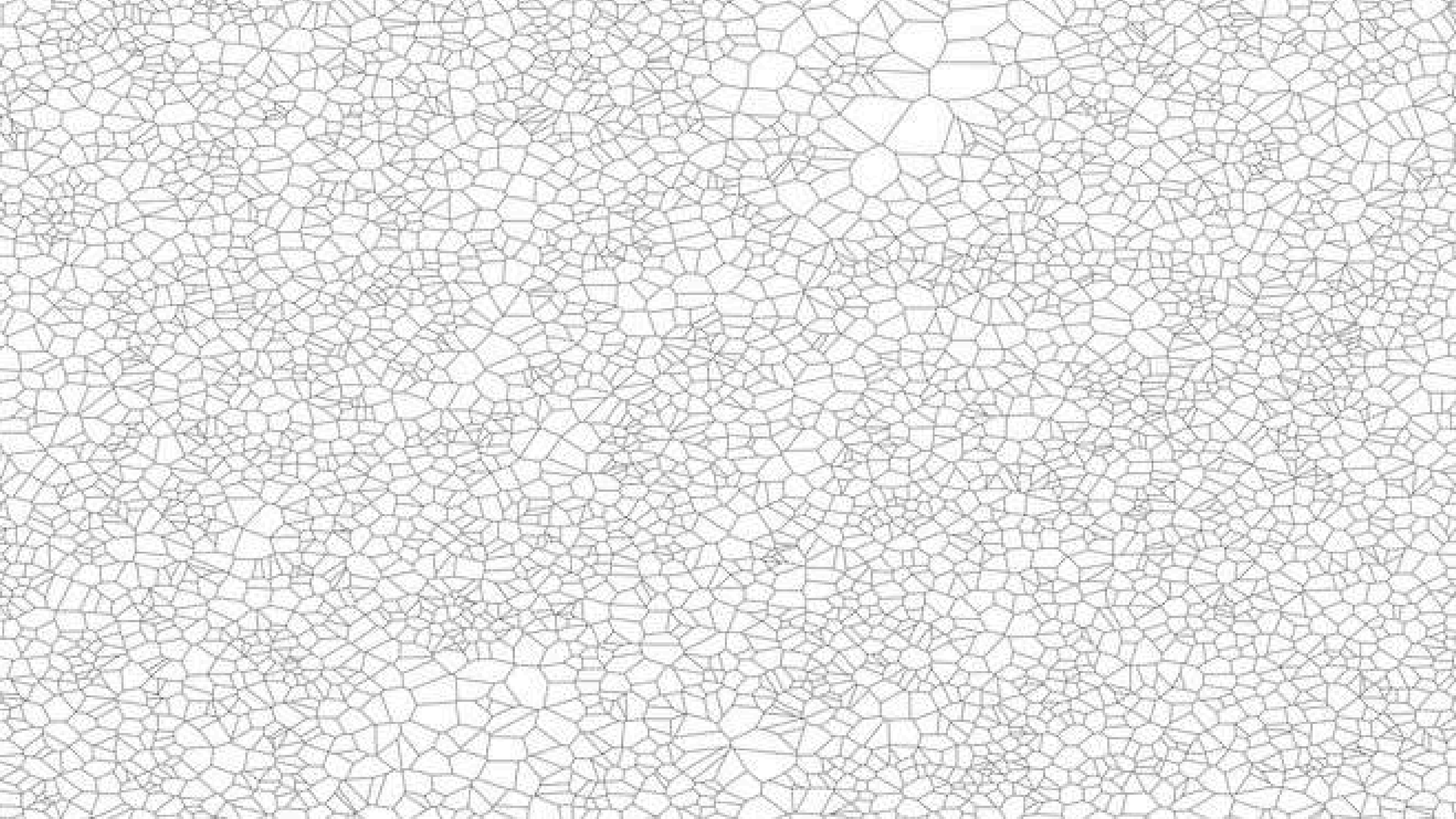
May 31, 2023

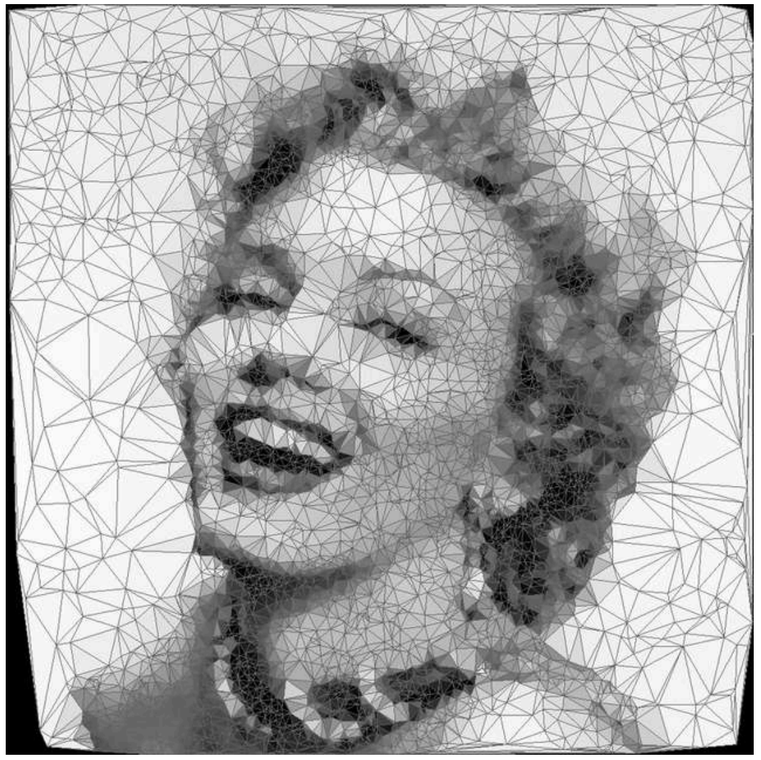
Varanasi

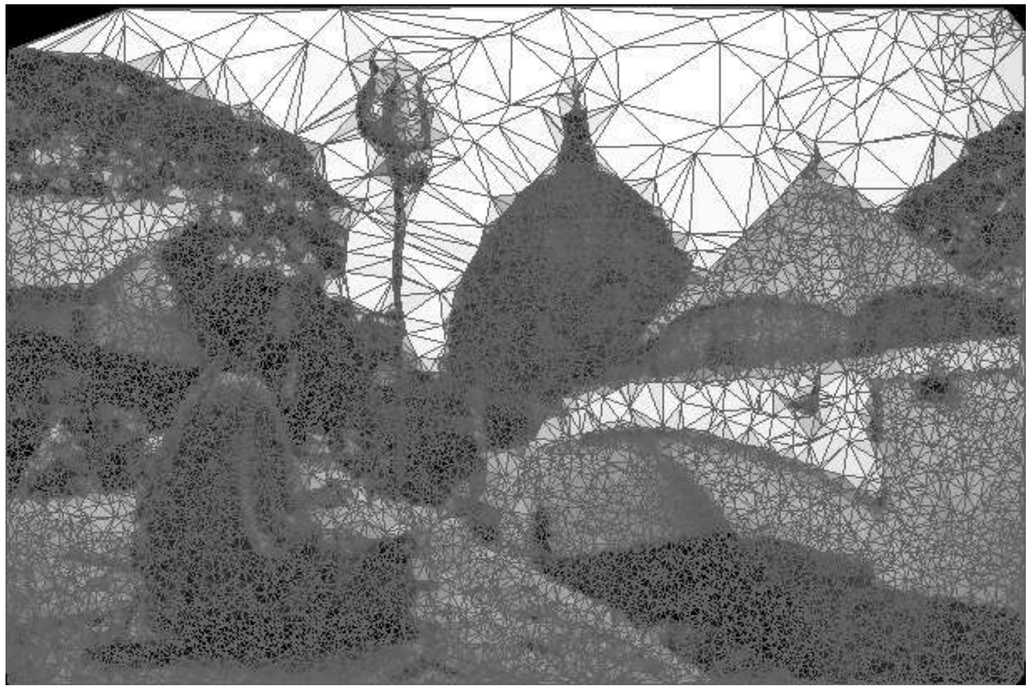


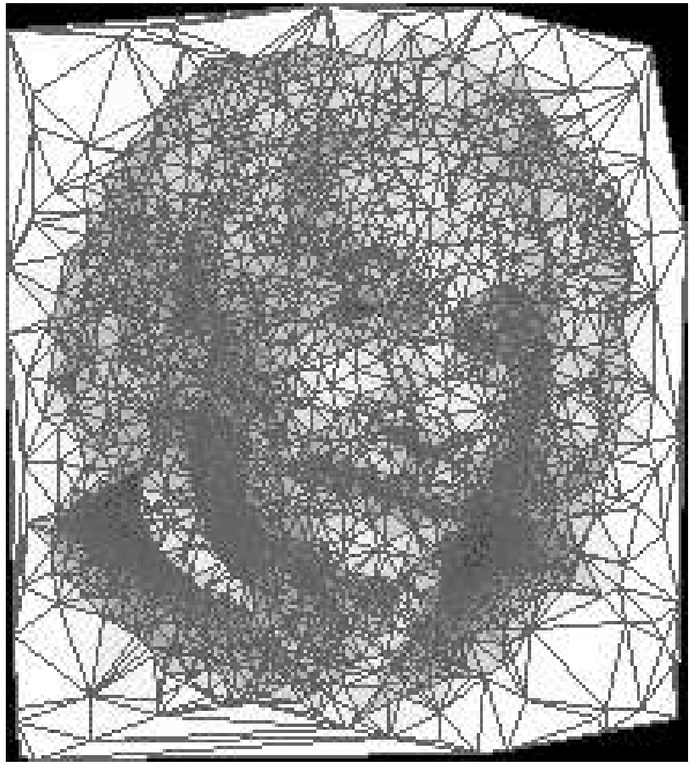

Srinagar
Leh
Shimla
Chandigarh
Dehradun
New Delhi
Jaipur
Lucknow
Varanasi
Patna
Gangtok
Itanagar
Dispur
Kohima
Imphal
Agartala
Aizawl
Gandhinagar
Bhopal
Ranchi
Kolkata
Raipur
Bhubaneswar
Daman
Mumbai
Hyderabad
Amaravati
Panaji
Bengaluru
Chennai
Puducherry
Port Blair
Kavaratti
Thiruvananthapuram
10
5
0
−5
−10
−15
−10
−5
0
5
10

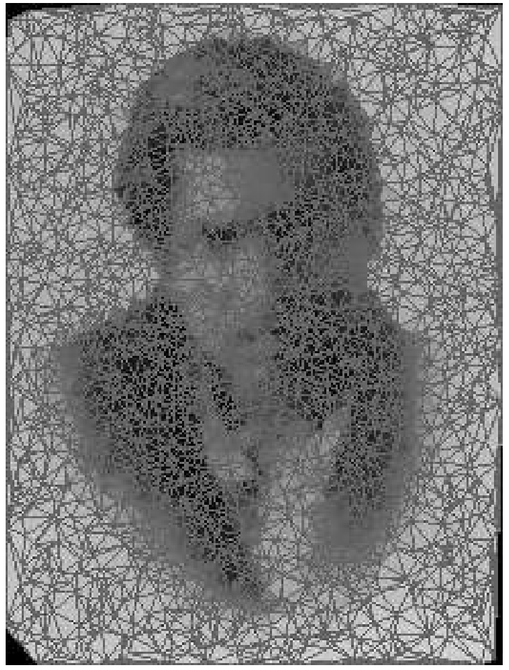

INDIA
ROAD GUIDE & POLITICA
with Road Distances in kms
REFERENCE
INDIA - FACTS & FIGURES
CHINA
NEPAL
BHUTAN
BANGLADESH
MYANMAR
PAKISTAN
SRI LANKA
INDIAN OCEAN
BAY OF BENGAL
ARABIAN SEA
TIBET
MAHARASHTRA
TELANGANA
ANDHRA PRADESH
KARNATAKA
TAMIL NADU
KERALA
ODISHA
WEST BENGAL
JHARKHAND
CHHATTISGARH
MADHYA PRADESH
UTTAR PRADESH
UTTARAKHAND
HIMACHAL PRADESH
PUNJAB
GUJARAT
Latitude Difference (°)
5
0
−5
−10
−15

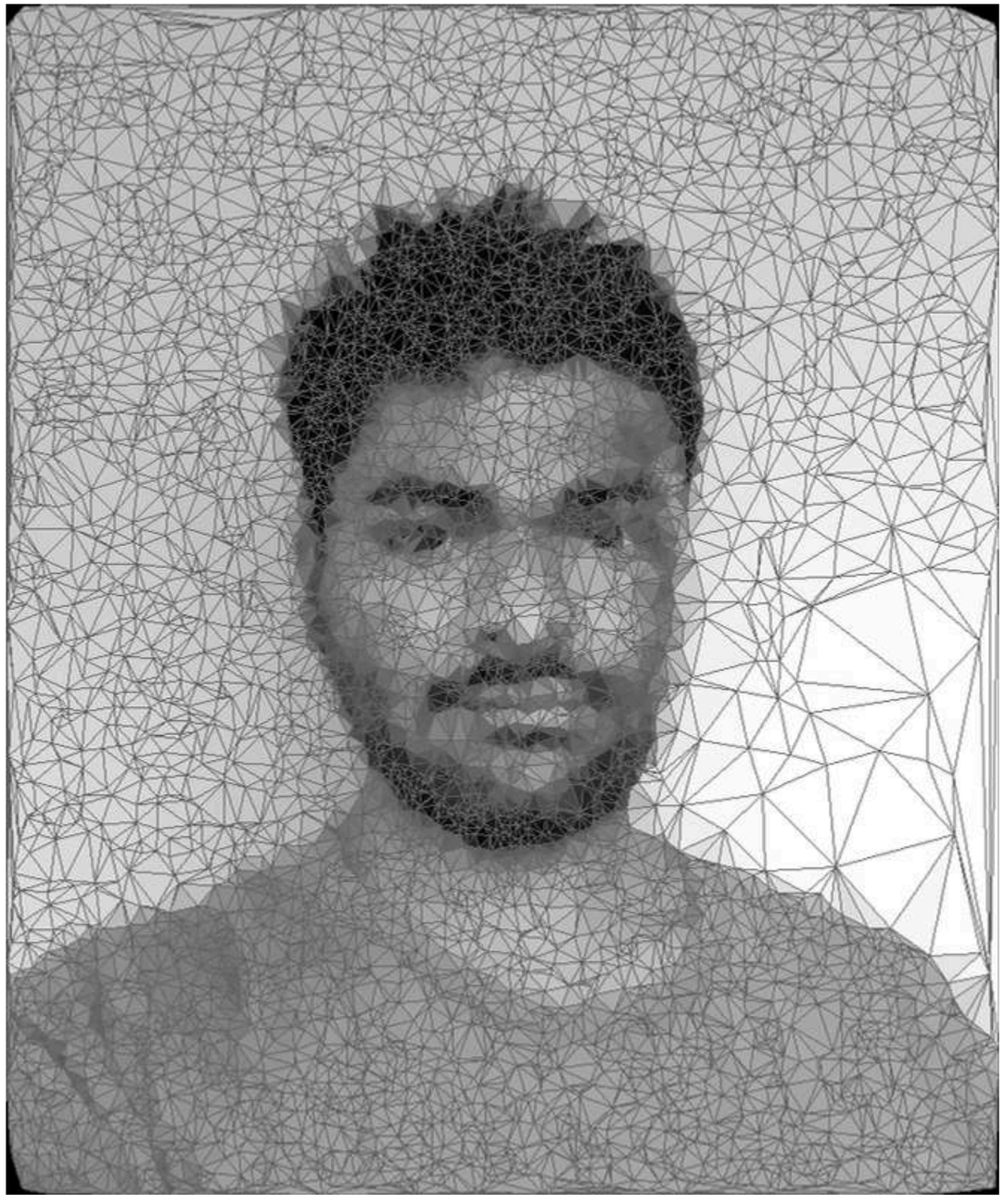


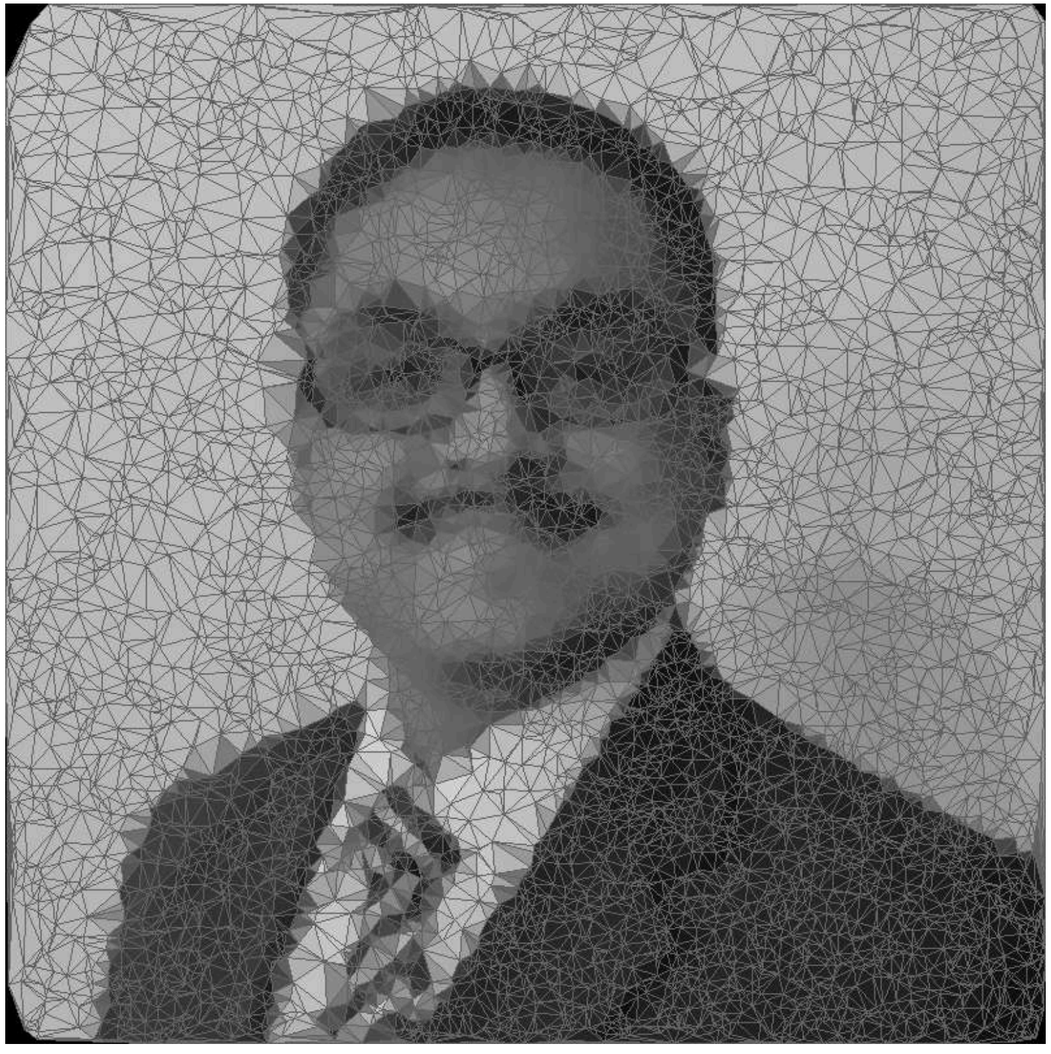


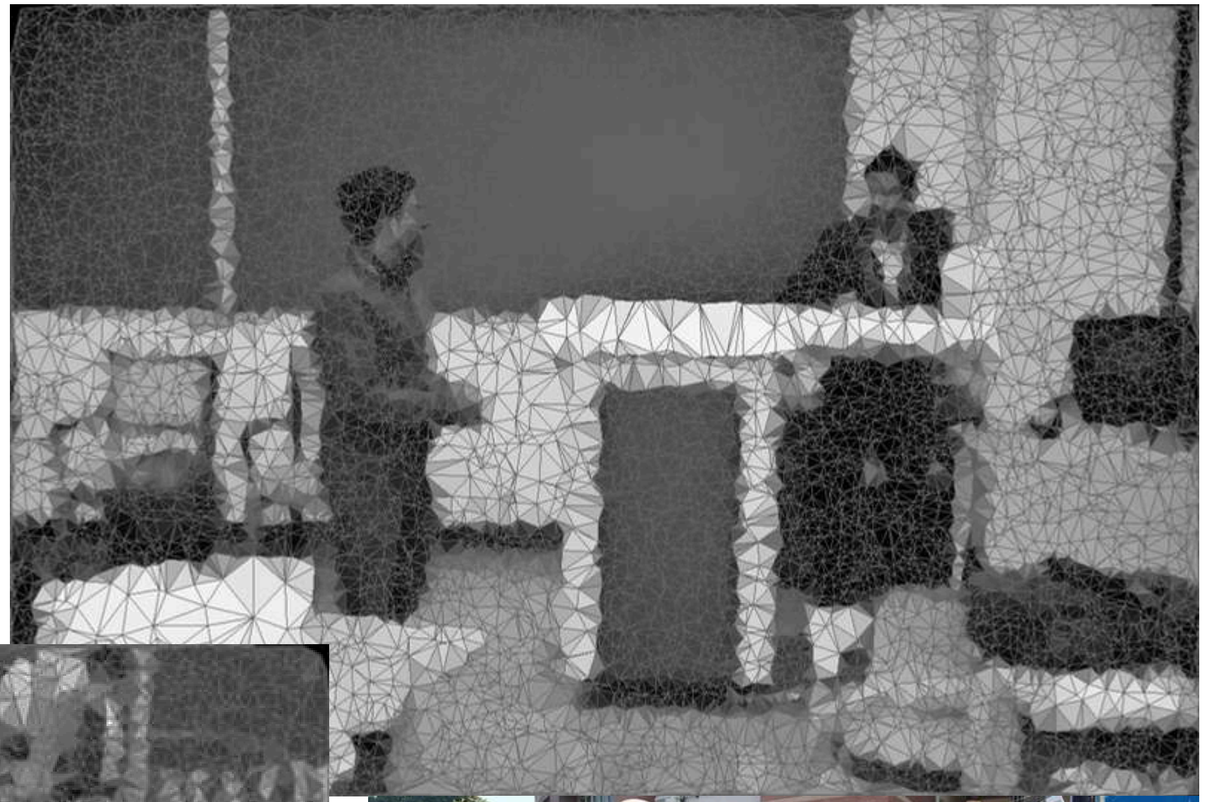


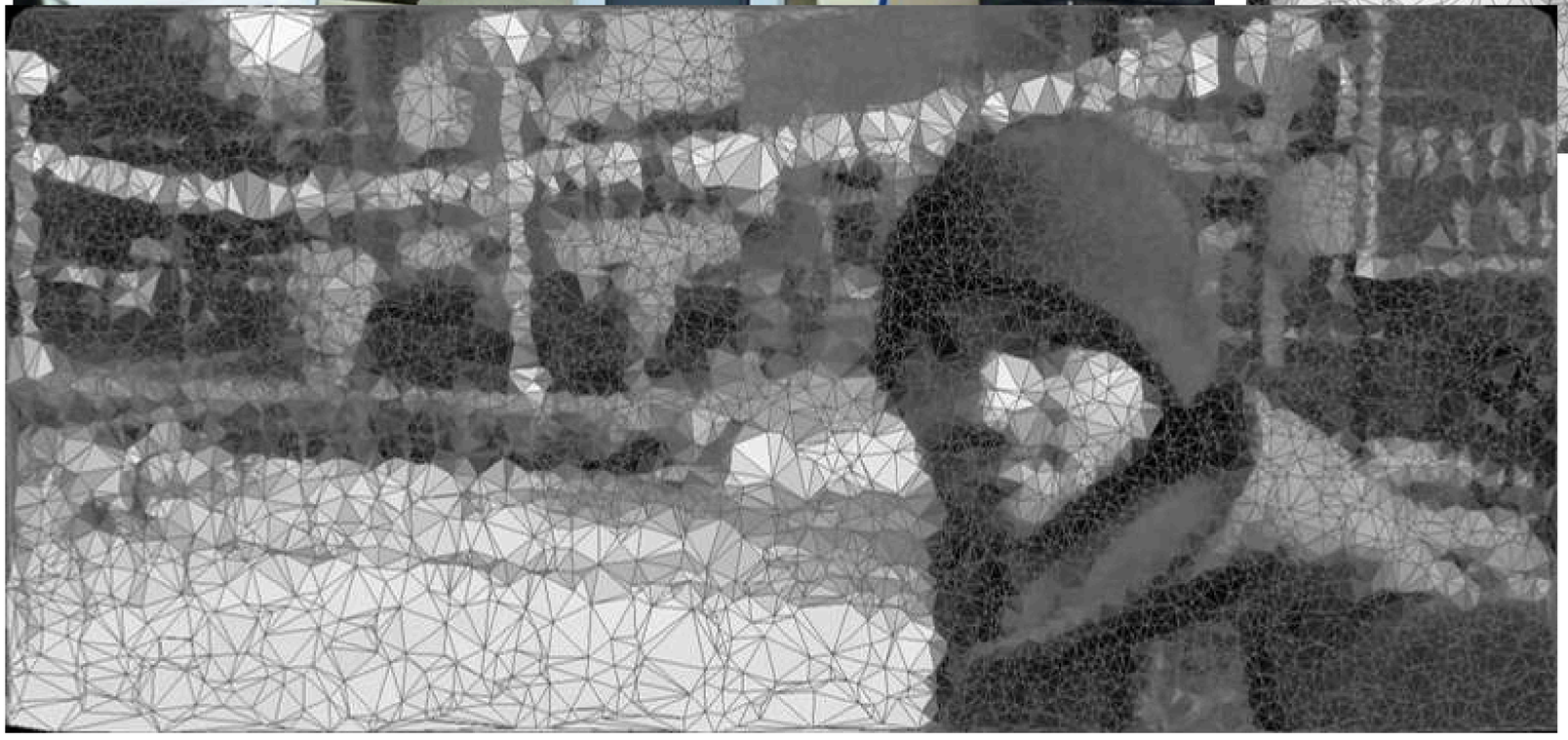




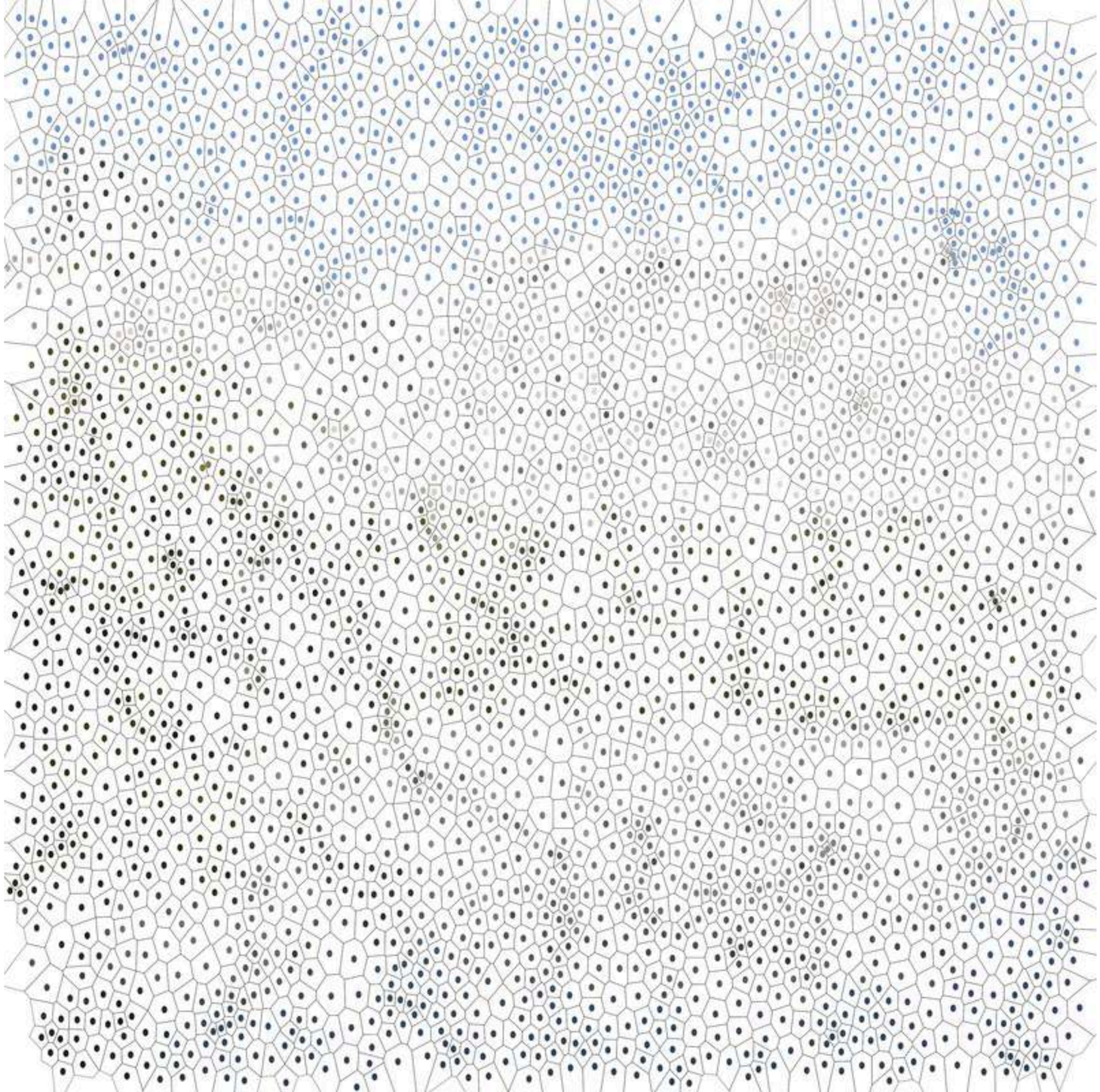


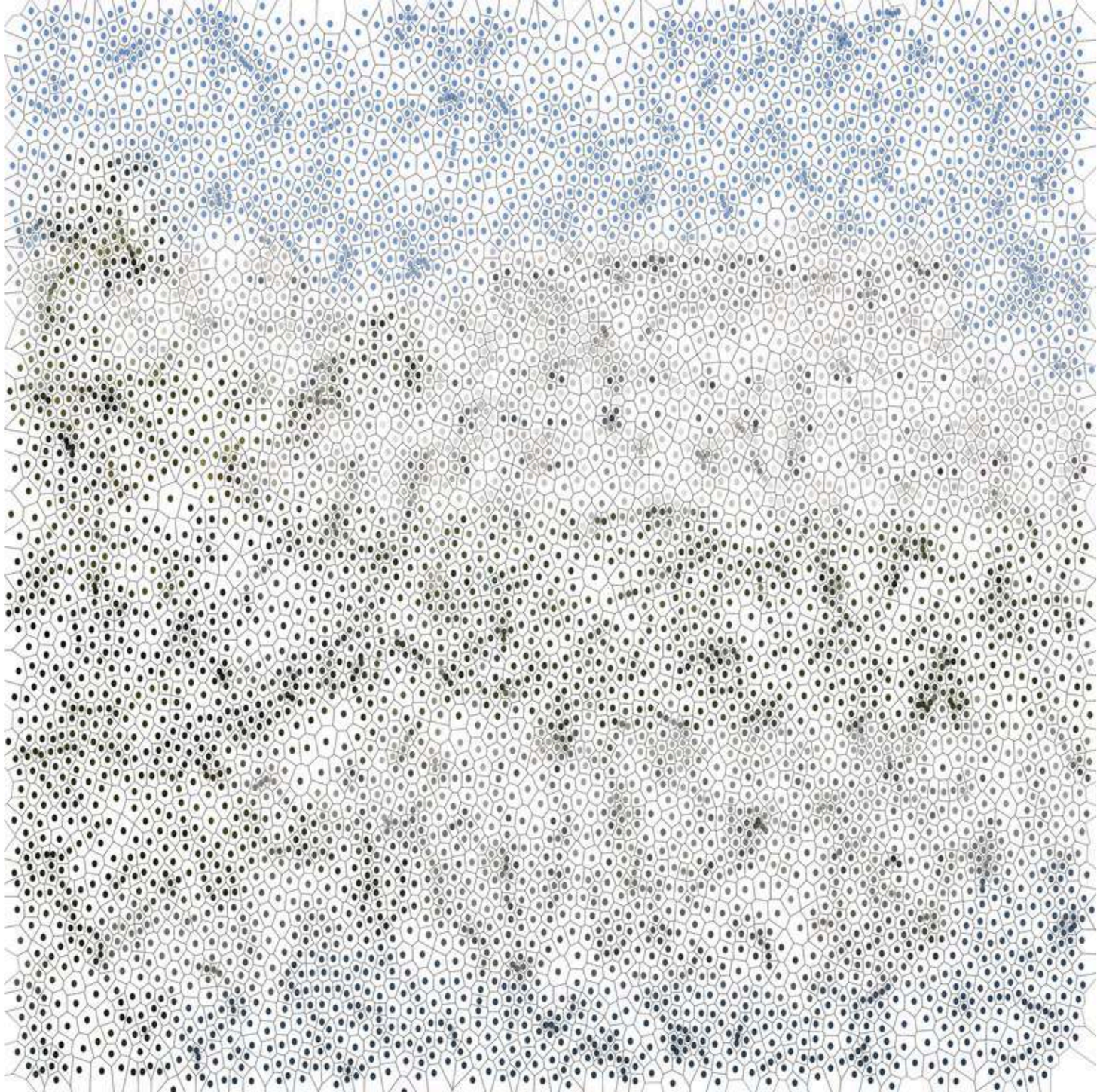


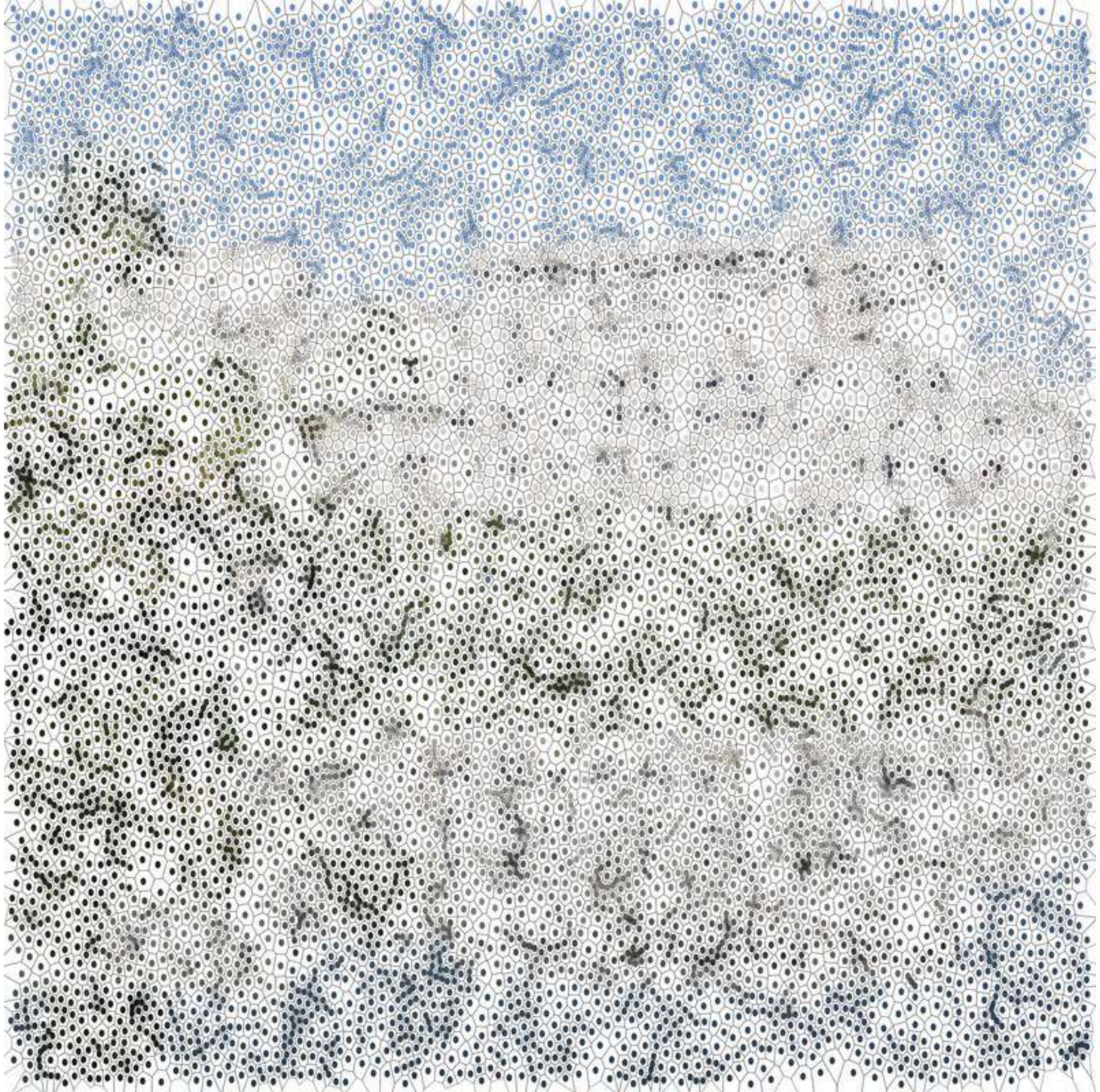


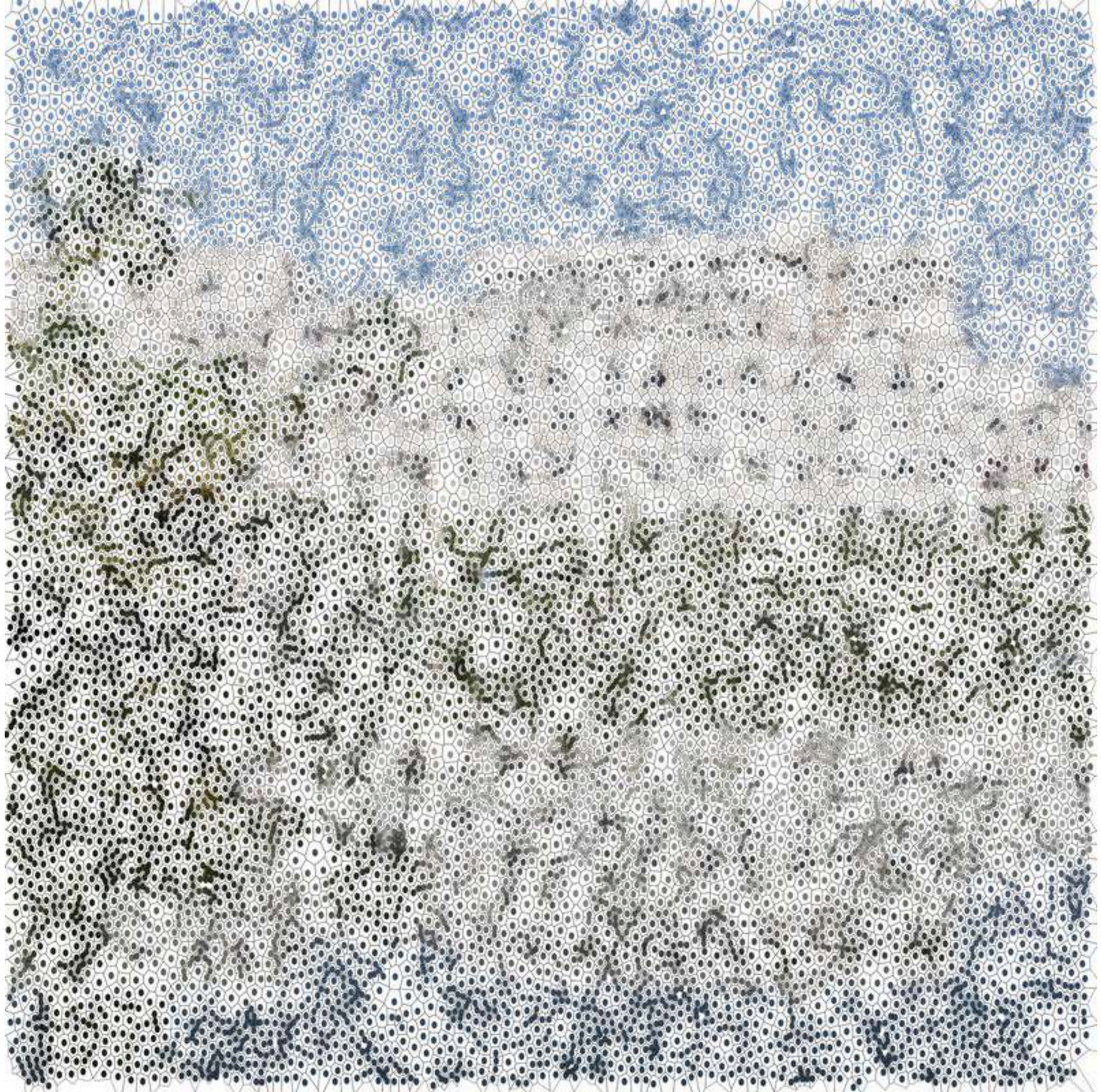


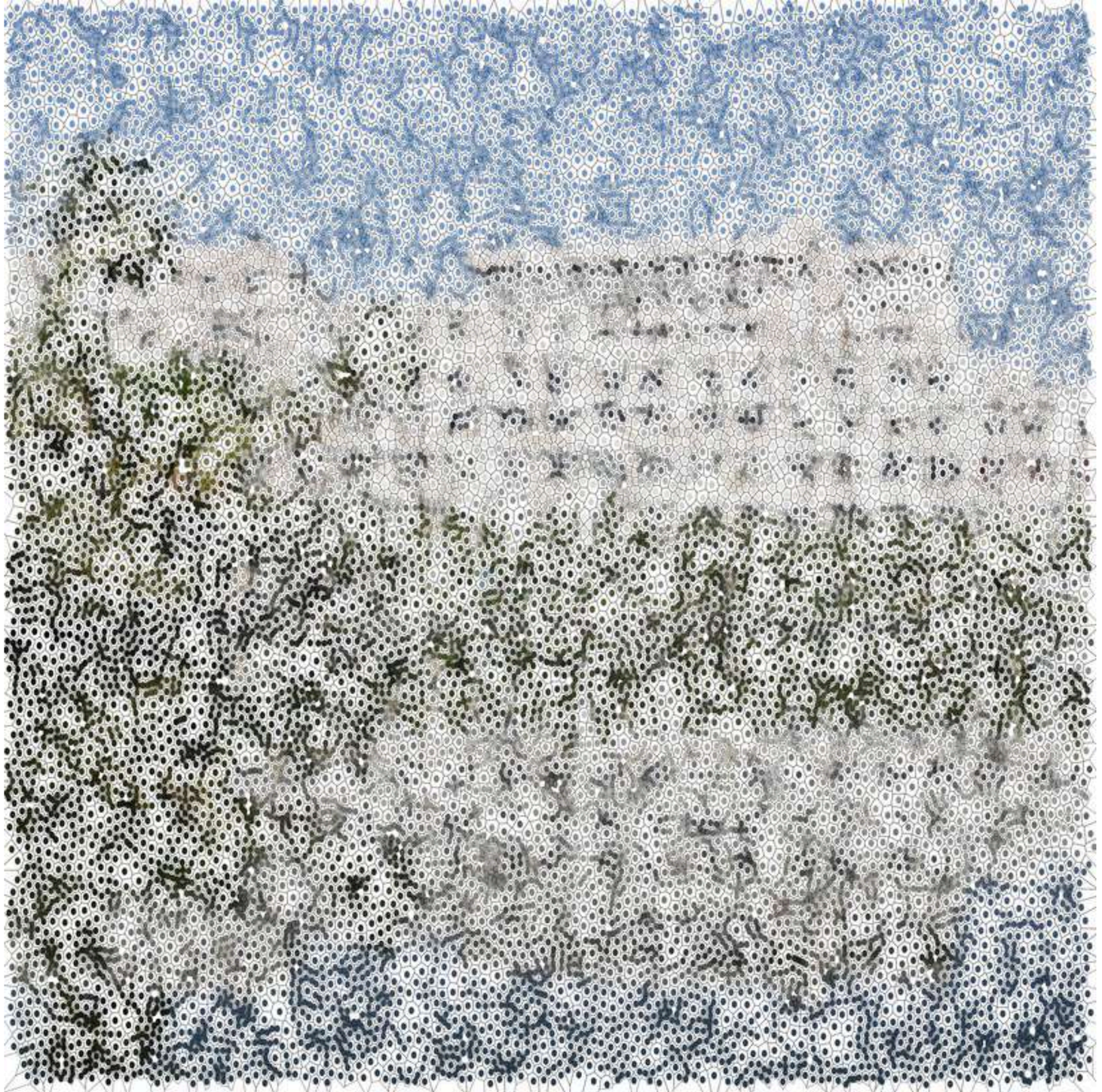

MATH
UNI

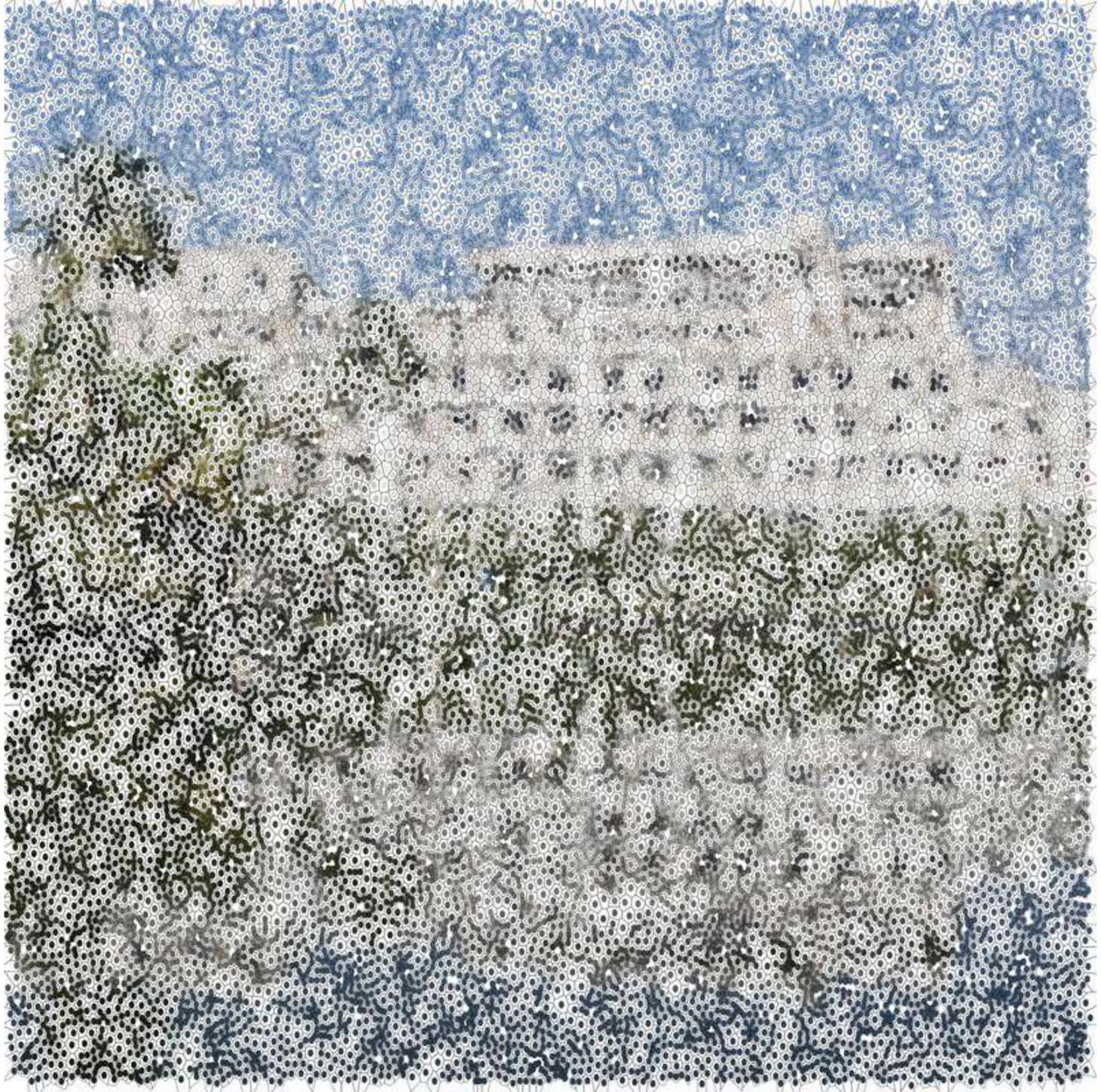


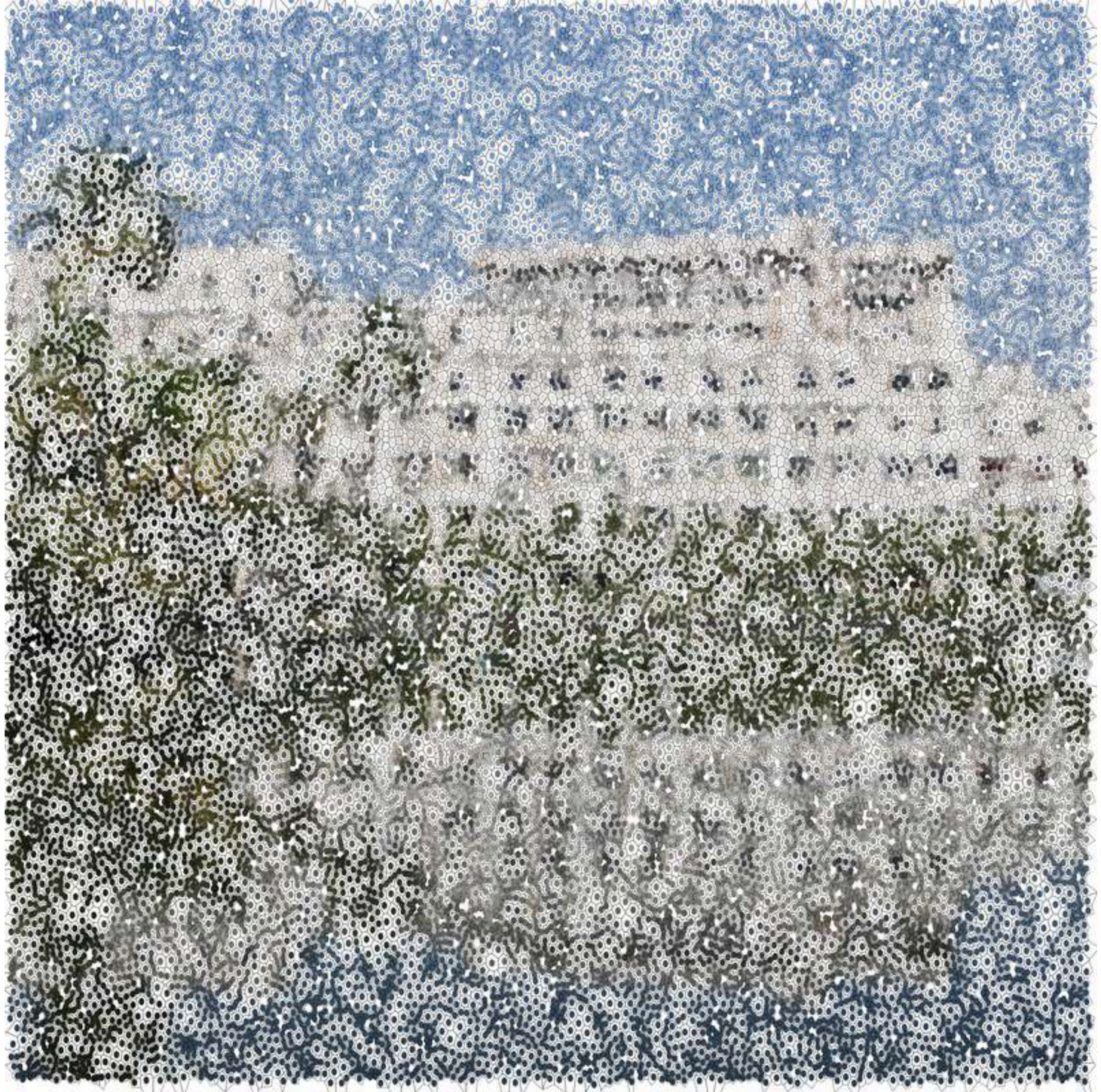


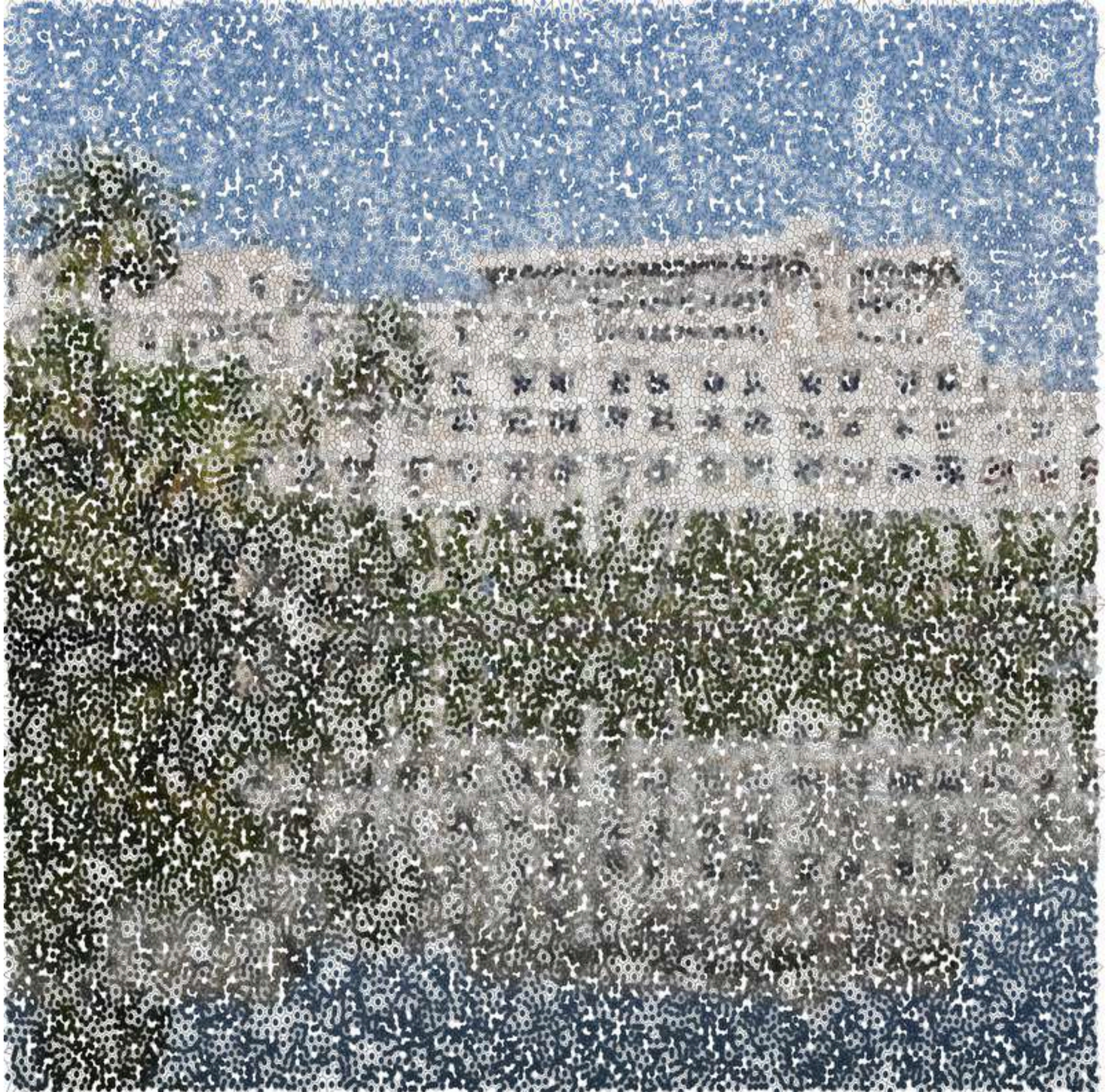












🧠 1. Computational Geometry and Computer Graphics

- Mesh Generation: Used for generating triangular or polygonal meshes in 2D and 3D modeling.
- Pathfinding and Navigation: For shortest path computation in robotics and gaming.
- Geometric Optimization: Helps in facility location problems and optimal point placement.

🌍 2. Geography and Urban Planning

- Service Area Analysis: Dividing cities into regions closest to hospitals, schools, fire stations, etc.
- Territorial Partitioning: Mapping political or administrative boundaries based on proximity.
- Environmental Modeling: Modeling influence zones around features like water bodies or pollutant sources.

🧬 3. Biology and Medicine
Cell Structure Modeling: Used to simulate and analyze cellular structures, especially in tissues and plant leaves.
Epidemiology: Modeling the spread of diseases by spatial proximity to sources.
Medical Imaging: Segmenting anatomical regions based on seed points in images.
.Swarm Optimization: Distributing tasks or resources based on Voronoi partitions

4. ML

- Encoders
- Clustering
- VC dimension, separability

🌐 5. Networking and Wireless Communication

- Cell Tower Coverage: Modeling areas of influence for each tower in mobile networks.
- Sensor Networks: Optimizing the placement of sensors and computing coverage or communication zones.
- Encoding, Hash

🤖 6. Robotics and AI

- Motion Planning: Navigation in obstacle-rich environments using generalized Voronoi diagrams.
- Multi-agent Systems: Dividing responsibility zones among autonomous agents (e.g., drones or robots)

Thank you very much!

THERE IS GEOMETRY IN THE HUMMING OF THE STRINGS, THERE IS MUSIC IN THE SPACING OF THE SPHERES.