

SHANGHAI TECH UNIVERSITY

CS271 Computer Graphics II

Fall 2025

Problem Set 1

Due: 23:59, Oct. 24, 2025

1. Submit your **PDF** solution to the course **Gradescope**. [Code: **8XV4G8**]
2. Submit your **Source Code and PDF as a zip file** to the **ShanghaiTech EPAN**: <https://epan.shanghaitech.edu.cn/l/RF2KH8>. [**Filename**: name_2025xx(your id)_hw1.zip]
3. There are no restrictions on programming languages.
4. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious penalties.

Problem 1: Melkman's Algorithm for Simple Polygon Convex Hull

Implement Melkman's algorithm to compute the convex hull of a simple polygon in $\mathcal{O}(n)$ time complexity.

Reference: Lecture 2, page 71

Requirements:

1. **Input:** A simple polygon represented as an ordered sequence of n vertices.
2. **Output:** The convex hull vertices in counterclockwise order.
3. Complexity analysis explaining why the algorithm achieves $\mathcal{O}(n)$ time.
4. Test cases with at least three different polygons.

1. Problem Definition and Notation

Simple polygon. A simple polygon is a closed planar chain formed by connecting a finite sequence of distinct vertices in order such that non-adjacent edges do not intersect. It defines a single, non-self-intersecting boundary enclosing a finite region in the two-dimensional Euclidean space \mathbb{R}^2 . Throughout this work, polygons are assumed to be strictly simple and embedded in \mathbb{R}^2 , consistent with the scope of Melkman's convex hull algorithm.

Convex hull. Given a simple polygon P , its convex hull $\text{CH}(P)$ is the smallest convex polygon that fully contains P . The result produced by Melkman's algorithm lists the hull vertices in counterclockwise (CCW) order.

Remark. Since the *Input* and *Output* have been formally defined in the problem statement, they are not repeated here.

Notation. The primary symbols used in this report are summarized in Table 1.

Table 1: Notation used in Problem 1.

Symbol	Description
$P = (p_0, p_1, \dots, p_{n-1})$	Ordered vertex sequence representing the boundary of a simple polygon.
$p_i = (x_i, y_i) \in \mathbb{R}^2$	Cartesian coordinates of the i -th vertex.
$H = (h_0, h_1, \dots, h_{m-1})$	Output convex hull vertices in CCW order.
$\text{orient}(a, b, c)$	Signed area function determining relative orientation of three points: $\text{orient}(a, b, c) = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$ Positive \Rightarrow c lies to the left of \overrightarrow{ab} ; negative \Rightarrow right side; zero \Rightarrow collinear.
$ P = n, H = m$	Number of vertices in the polygon and in its convex hull.

2. Data Structures and Algorithm

2.1 Data Structures

The algorithm takes an ordered vertex sequence $P = (p_0, p_1, \dots, p_{n-1})$, where each vertex $p_i = (x_i, y_i) \in \mathbb{R}^2$ defines the polygon boundary and is accessed sequentially.

A double-ended queue D maintains the evolving convex hull, storing the upper and lower chains simultaneously. Its front represents the head of the upper chain, and its back the tail of the lower chain. Each element of D is a vertex of P , supporting constant-time insertion and deletion at both ends:

$$D = [d_0, d_1, \dots, d_k], \quad d_i \in P.$$

This structure enables dynamic updates that preserve the left-turn property of convexity, and the final

contents of D form the convex hull in counterclockwise order.

2.2 Algorithmic Overview

Melkman's algorithm computes the convex hull of a simple polygon in one linear scan of its ordered vertices. A double-ended queue D maintains the current convex boundary, storing the upper and lower chains simultaneously. The first three non-collinear vertices (p_0, p_1, p_2) are oriented counterclockwise to initialize D with p_2 duplicated at both ends. For each subsequent vertex p_i , the algorithm checks whether p_i lies inside the existing hull—left of both boundary edges. If so, it is skipped; otherwise, vertices violating convexity are removed from the deque's ends before inserting p_i at both sides. Each vertex is processed once and updated a constant number of times, ensuring overall linear-time complexity.

Algorithm 1 Melkman's Convex Hull for a Simple Polygon

Input: Ordered vertices $P = (p_0, p_1, \dots, p_{n-1})$

Output: Convex hull H in counterclockwise order

```

1: if orient( $p_0, p_1, p_2$ )  $> 0$  then
2:    $D \leftarrow [p_2, p_0, p_1, p_2]$ 
3: else
4:    $D \leftarrow [p_2, p_1, p_0, p_2]$ 
5: end if
6: for  $i = 3$  to  $n - 1$  do
7:    $p \leftarrow p_i$ 
8:   if is_left( $D[-2], D[-1], p$ ) and is_left( $D[0], D[1], p$ ) then
9:     continue
10:   end if
11:   while is_right_or_on( $D[0], D[1], p$ ) do
12:      $D.popleft()$ 
13:   end while
14:    $D.appendleft(p)$ 
15:   while is_right_or_on( $D[-2], D[-1], p$ ) do
16:      $D.pop()$ 
17:   end while
18:    $D.append(p)$ 
19: end for
20:  $D.pop()$ ; return list of vertices in  $D$  (CCW)

```

3. Test Case Design

3.1 Special Cases

To ensure robustness, representative input configurations are identified to capture the main geometric and numerical challenges of convex hull construction. Nearly collinear vertices test orientation stability when $\text{orient}(a, b, c) \approx 0$, verifying consistent handling of degenerate edges. Concave polygons with

pocket structures examine bidirectional deque updates and confirm that convexity is preserved under pruning. Near-contact edges assess numerical tolerance to minimal gaps and floating-point precision. Polygons dense in interior vertices but sparse on the boundary test skip efficiency and validate linear-time behavior. Finally, clockwise-ordered inputs confirm correct orientation detection and consistent counterclockwise hull output. These scenarios collectively define the space of valid yet challenging inputs against which the subsequent test cases are designed.

3.2 Case 1: Collinear Band with Near-Degenerate Contact

Objectives and algorithmic checks:

1. Test stability of orientation when many consecutive vertices are collinear or nearly collinear ($\text{orient}(a, b, c) \approx 0$); confirm correct seed-triple discovery during initialization and a consistent collinearity policy that retains only extreme endpoints on collinear runs.
2. Test numerical robustness on an extremely narrow-band polygon with very small local area; verify inside-tests on both chains remain stable at the precision limit.
3. Test tolerance to near-contact without intersection; ensure deque updates are correct when the near-contact tip triggers repeated front/back popping, preserving convexity and preventing vertex loss.

Construction method: An upper horizontal band at $y = 520$ is formed by a long collinear sequence from $[150, 520]$ to $[850, 520]$, enforcing repeated $\text{orient} = 0$ evaluations. A parallel lower band at $y = 505$, only 15 units below, creates a narrow strip that amplifies floating-point sensitivity. A sharp vertex at $[560, 518]$ lies merely 2 pixels beneath the upper band, producing a near-contact tip that stresses intersection and collinearity handling during deque updates. A closing vertex at $[140, 490]$ slightly tapers the shape to keep a small but nonzero area so the polygon remains valid.

Conclusion: Case 1 efficiently combines collinearity, near-degenerate geometry, and near-contact configurations in a single test. It provides a comprehensive stress test for the initialization, inside-test, and deque-update mechanisms of Melkman's algorithm, confirming both its numerical robustness and geometric consistency.

3.3 Case 2: Dual-Pocket Polygon for Deque Update Validation

Objectives and algorithmic checks:

1. Verify the correctness of Melkman's bidirectional deque update mechanism by constructing a polygon that simultaneously triggers both front and back popping operations.
2. Examine the algorithm's behavior when processing multiple concave pockets, ensuring that each concave indentation is correctly handled without breaking the convexity of the maintained hull.
3. Confirm that the polygon remains simple and correctly closed during the entire hull construction process, with no edge crossings or ordering errors after repeated deque updates.

Construction method: The polygon is designed with two concave pockets—one on each side—to induce dual-end updates in the deque. On the left side, vertices [180, 300], [160, 340], and [140, 290] form the *front pocket*, which triggers popping operations at the head of the deque. On the right side, vertices [860, 480], [850, 450], and [820, 520] create the *rear pocket*, enforcing pops from the tail. The middle section, spanning from [600, 540] to [900, 520], forms a wide convex region that connects the two pockets smoothly. The remaining vertices follow a gentle transition, ensuring the polygon remains simple and fully enclosed.

Conclusion: Case 2 effectively tests the update logic of Melkman’s algorithm by forcing alternating front and back popping during hull maintenance. It provides clear visual evidence of how the deque dynamically adjusts to restore convexity after processing concave pockets, validating the correctness of the update order, stopping conditions, and overall hull integrity.

3.4 Case 3: Interior-Point Skipping and Sparse Expansion Test

Objectives and algorithmic checks:

1. Verify that Melkman’s algorithm correctly identifies interior points located within the current convex hull and efficiently skips them without affecting the boundary structure.
2. Test the correctness of hull expansion when a few exterior vertices appear after long sequences of interior points, ensuring that convexity restoration occurs only when necessary.
3. Evaluate the algorithm’s ability to maintain its linear-time complexity $\mathcal{O}(n)$ under mixed input conditions containing both dense interior and sparse exterior vertices.

Construction method: The polygon is composed of two contrasting regions: a dense interior section and a sparse outer shell. The outer hull is defined by vertices such as [200, 600] → [780, 610] → [860, 640] and [180, 180], forming a broad convex frame. Between these outer points lies a long internal sequence, e.g., [820, 400] → [400, 400] → [220, 300], where most vertices remain strictly inside the convex boundary. These interior runs are occasionally interrupted by a few exterior expansion points, forcing the deque to update selectively. The alternating distribution of interior and exterior vertices provides a realistic stress test for the skip logic and incremental hull growth.

Conclusion: Case 3 demonstrates the efficiency of Melkman’s algorithm in distinguishing interior from exterior vertices and preserving its $\mathcal{O}(n)$ time behavior even for highly unbalanced input distributions. It confirms that unnecessary updates are avoided for interior points, while sparse exterior vertices correctly trigger localized deque modifications, maintaining a consistent and minimal convex hull.

3.5 Visual summary and case coverage

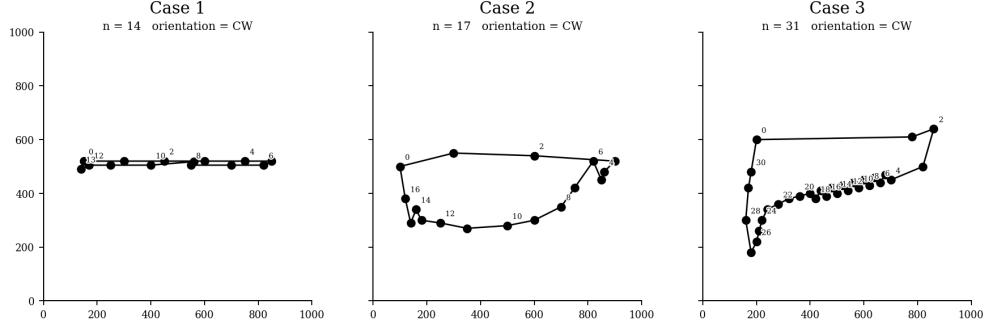


Figure 1: Visualization of the three representative polygon cases used in testing Melkman’s algorithm. Case 1: collinear band with near-degenerate contact; Case 2: dual-pocket polygon; Case 3: interior-point skipping with sparse expansion.

Summary of coverage: The three designed cases jointly encompass the major geometric and numerical challenges identified in Section 3.1. Case 1 focuses on numerical stability under collinearity, near-degenerate geometry, and near-contact configurations, verifying that orientation, convexity, and initialization logic remain robust at precision limits. Case 2 targets the algorithm’s structural correctness during hull updates by introducing concave pockets that activate both front and back deque operations, testing update order and boundary integrity. Case 3 evaluates performance and correctness in mixed-density inputs, where numerous interior points are interleaved with sparse exterior ones, validating the skip logic and maintenance of linear-time behavior.

Together, these cases comprehensively test Melkman’s algorithm across initialization, inside-testing, and deque-update phases. They confirm that the implementation is robust to collinearity, concavity, and numerical perturbation, while preserving correctness and $\mathcal{O}(n)$ efficiency across all representative polygon configurations.

4. 4. Visualization and Results Analysis

Following the design principles and definitions described in previous sections, the Melkman convex hull algorithm was implemented and tested on the three representative polygon cases defined in Section 3. The results are shown in Figure 2, where each subfigure corresponds to one of the test cases: Case 1 (collinear band with near-degenerate contact), Case 2 (dual-pocket polygon), and Case 3 (interior-point skipping and sparse expansion).

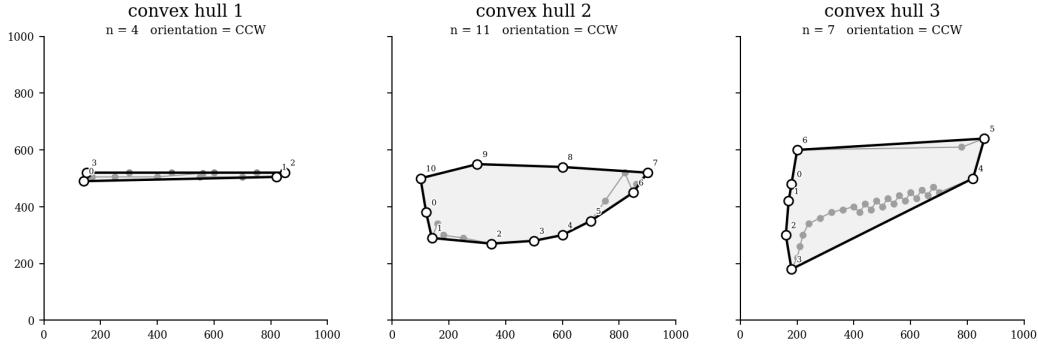


Figure 2: Visualization of convex hull results for the three designed test cases. Each subfigure shows the original polygon (gray line and points) and the computed convex hull (bold black outline). Hull vertices are labeled in counterclockwise order, and n denotes the number of vertices in each polygon.

Result correctness and analysis: The visual outcomes confirm that Melkman’s algorithm produces the expected convex boundaries for all three input categories.

For *Case 1*, only the extreme vertices of the upper and lower collinear bands are retained, while all intermediate collinear points are correctly excluded from the hull. The narrow-band configuration demonstrates stable orientation evaluation even when $\text{orient}(a, b, c) \approx 0$, indicating numerical robustness against near-degenerate geometry.

For *Case 2*, the presence of two concave pockets successfully triggers deque updates at both ends. The hull reconstruction after each concave indentation preserves convexity and counterclockwise orientation, verifying the correctness of the bidirectional popping mechanism.

For *Case 3*, the majority of interior points are efficiently skipped, while a few exterior vertices correctly expand the hull boundary. The algorithm maintains linear-time behavior since each vertex is processed at most once, confirming that the skip logic prevents redundant operations.

Overall, the visualization results align perfectly with the theoretical expectations. Melkman’s algorithm demonstrates robustness across numerical edge cases, correctness in deque-based updates. The generated convex hulls exhibit clean, non-intersecting boundaries that accurately enclose the input polygons.

5. Time Complexity Analysis

5.1 Theoretical

Melkman’s algorithm achieves linear-time complexity $\mathcal{O}(n)$ with respect to the number of vertices in the input polygon. The efficiency arises from its incremental and deterministic construction process, which guarantees that each vertex is handled a constant number of times.

During the main loop, each vertex p_i is read exactly once in the scanning order of the polygon. For each p_i , the algorithm performs a constant number of orientation tests and at most a few deque operations. Each vertex can be inserted into and removed from the deque at most once per end—once when it enters the convex hull and once when it is later popped due to a non-left turn. Therefore, the

total number of deque operations over the entire run is proportional to n .

The orientation test $\text{orient}(a, b, c)$, which involves a few arithmetic operations on vertex coordinates, also executes $\mathcal{O}(1)$ per call. Even under degenerate cases such as long collinear sequences, each vertex still contributes only a bounded number of constant-time comparisons before either being skipped or becoming part of the hull.

As a result, the total computational cost can be expressed as:

$$T(n) = c_1n + c_2n + c_3n = \mathcal{O}(n),$$

where c_1 , c_2 , and c_3 correspond to the constant-time costs of orientation evaluation, deque maintenance, and inside-testing respectively. No nested loops dependent on n exist in the algorithm, and no vertex is ever revisited or reprocessed after being finalized in the hull.

Hence, Melkman's convex hull algorithm is theoretically proven to run in $\mathcal{O}(n)$ time for any simple polygon input, achieving optimal performance for this class of problems.

5.2 Empirical

To validate the theoretical time complexity, an experimental evaluation was conducted using a series of polygons with progressively increasing vertex counts. Nine polygon sets were generated to provide a controlled range of input sizes, ensuring both reproducibility and consistent geometric diversity. The generated polygons are visualized in Figure 3, and their corresponding convex hulls, computed using the implemented Melkman algorithm, are shown in Figure 4. Each convex hull result was visually verified to confirm correctness before performing timing and operation-count measurements.

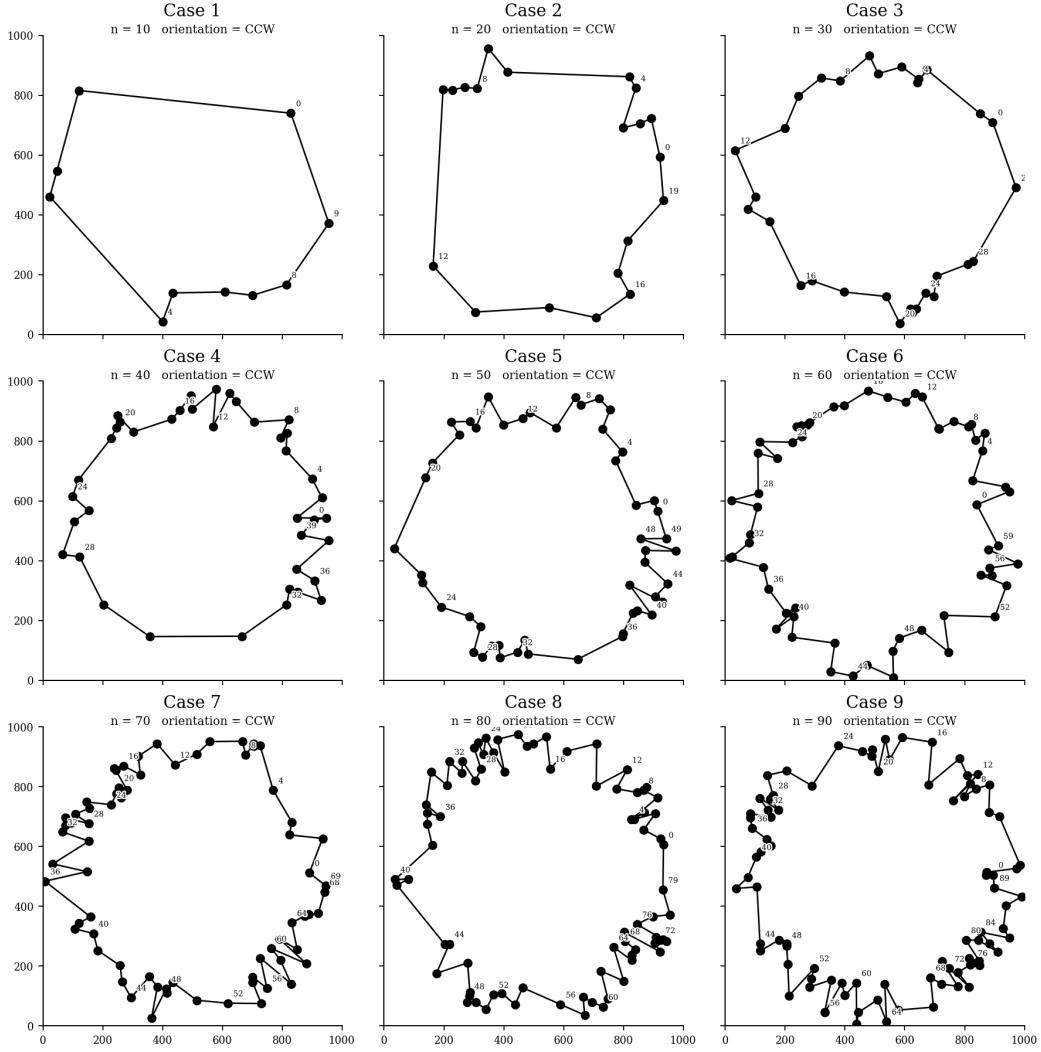


Figure 3: Generated simple polygon test cases with increasing vertex counts ($n = 10$ to $n = 90$). Each subfigure represents a distinct input size used for empirical complexity evaluation.

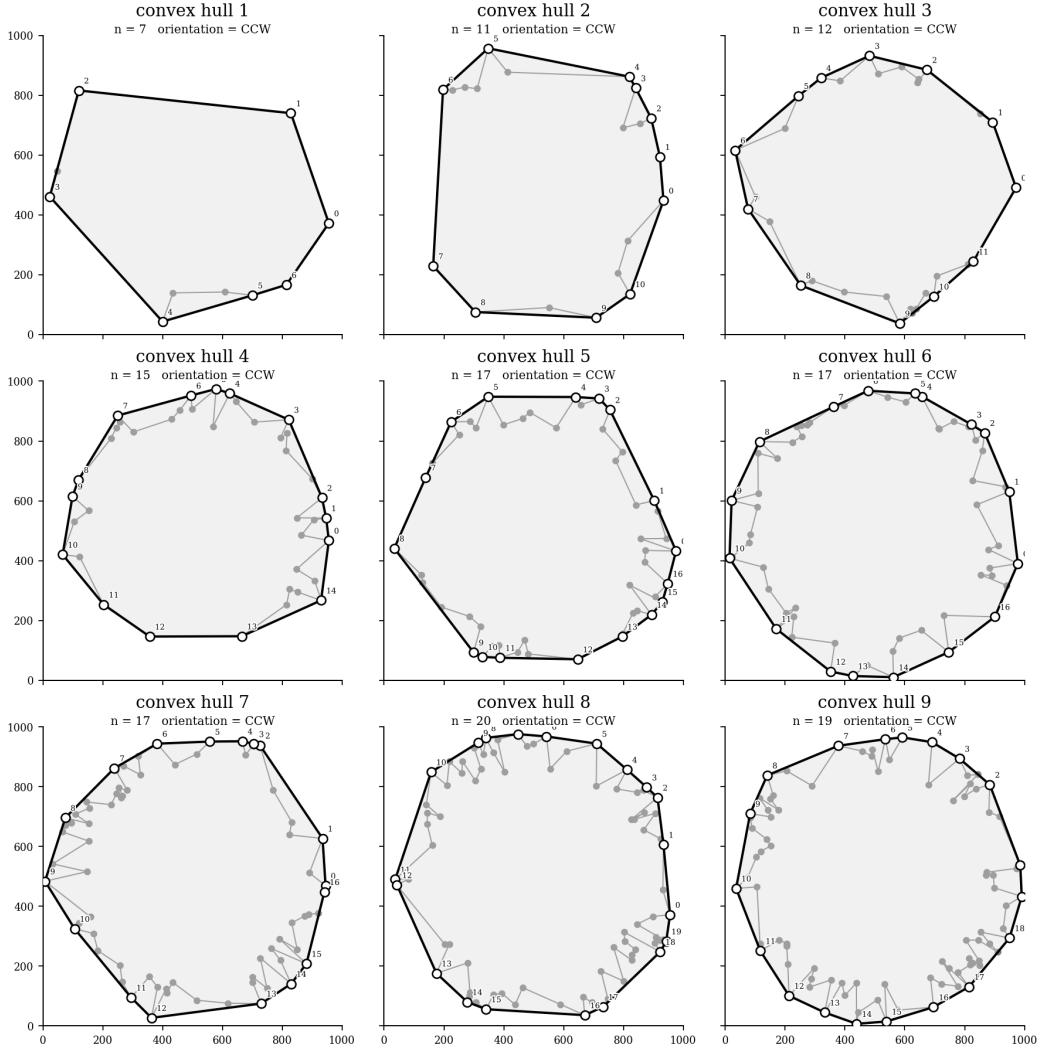


Figure 4: Convex hulls computed by Melkman's algorithm for the test polygons shown in Figure 3. The results confirm correctness and stability across all polygon sizes.

After correctness verification, execution time and total operation counts (including orientation, cross-product, and deque operations) were recorded for each test size. The results were then fitted to linear and power-law models, as shown in Figure 5.

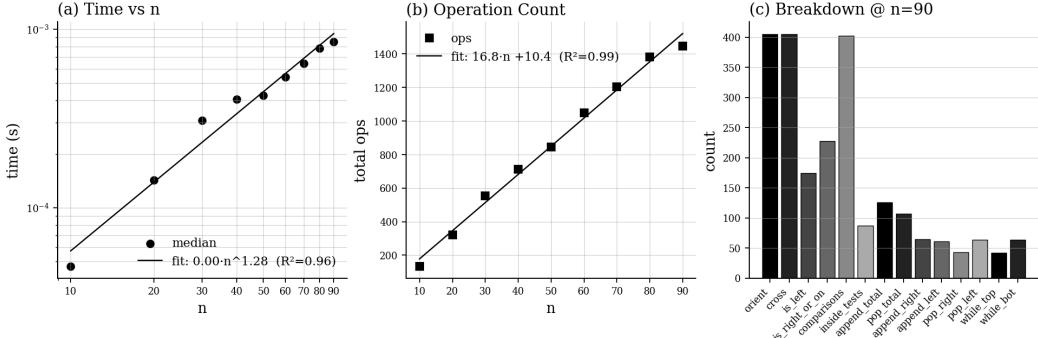


Figure 5: Empirical complexity analysis of Melkman’s algorithm. (a) Median runtime versus input size n , fitted with $T(n) \propto n^{1.28}$ ($R^2 = 0.96$); (b) total operation count showing a nearly perfect linear fit ($R^2 = 0.99$); (c) operation breakdown at $n = 90$, indicating that orientation and cross-product computations dominate total cost.

Discussion: The measured runtime exhibits an almost linear growth with respect to the number of vertices, with an empirical fit of approximately $T(n) \propto n^{1.28}$, closely matching the theoretical $\mathcal{O}(n)$ expectation. The total operation count maintains a strictly linear relationship with n , confirming that each vertex contributes only a constant number of geometric and deque operations. The operation breakdown further shows that the majority of computational effort comes from orientation and cross-product evaluations, while deque operations remain minimal.

Overall, the empirical data strongly supports the theoretical claim that Melkman’s convex hull algorithm operates in linear time $\mathcal{O}(n)$, achieving both correctness and computational efficiency across varying input sizes.

Problem 2: General Voronoi Diagram (Power Diagram Variant)

Study general Voronoi diagrams and implement one variant of your choice.

Reference: Slides of General Voronoi.

Requirements:

1. Explain which variant you chose and why.
2. Implement the algorithm with proper data structures.
3. Visualize the result (2D plot showing the sites and Voronoi regions).
4. Analyze the time and space complexity.
5. Test cases with at least three different point configurations.

1. Variant Selection and Motivation

The variant selected in this study is the **Power Diagram** (also known as the Laguerre–Voronoi Diagram), due to its strong theoretical and practical relevance to my ongoing research on three-dimensional medical image reconstruction. In 3D ultrasound image segmentation and reconstruction, it is often necessary to partition the space into weighted regions according to the spatial features or importance of different points, in order to generate structurally consistent and geometrically continuous volumetric representations. By introducing additive weights into the distance metric, the Power Diagram allows each cell to depend not only on spatial position but also on local weighting factors, which aligns well with the inherently non-uniform distribution of anatomical structures in medical images. Its convexity and controllability ensure numerical stability in geometric computations, while its formulation naturally extends to three dimensions and connects seamlessly with Delaunay triangulation and regular tessellation methods. Therefore, the Power Diagram provides both a solid theoretical foundation and a practical geometric framework for understanding and implementing weighted spatial partitioning in 3D medical image reconstruction.

2. Problem Definition and Notation

Input. A finite set of weighted sites

$$S = \{(x_i, y_i, w_i) \mid i = 1, \dots, n\} \subset \mathbb{R}^2 \times \mathbb{R},$$

where (x_i, y_i) denotes the spatial coordinates of site s_i , and w_i is a scalar weight determining its relative influence in the power distance metric.

Output. A per-site collection of half-planes representing the Power Diagram in implicit form. For each site s_i , its region is encoded as

$$\mathcal{H}_i = \{(a_{ij}, b_{ij}, c_{ij}) \mid a_{ij}x + b_{ij}y \leq c_{ij}, j \neq i\},$$

where each triple (a_{ij}, b_{ij}, c_{ij}) corresponds to the bisector $\phi_i \leq \phi_j$. The explicit polygonal cell is given by

$$V_i = \bigcap_{H \in \mathcal{H}_i} H,$$

and the complete diagram is $\{V_i\}_{i=1}^n$.

Power distance and cell definition. For $p = (x, y) \in \mathbb{R}^2$ and site $s_i = (x_i, y_i, w_i)$, the power distance is defined as

$$\phi_i(p) = \|p - s_i\|^2 - w_i.$$

The *power cell* associated with s_i is

$$V_i = \{p \in \mathbb{R}^2 \mid \phi_i(p) \leq \phi_j(p), \forall j \neq i\},$$

representing the set of all points closer (in the power sense) to s_i than to any other site.

Goal. Construct the per-site half-plane sets $\{\mathcal{H}_i\}$ defining the Power Diagram, visualize the resulting cells by clipping against a finite bounding box, and analyze the algorithm's theoretical and empirical complexity.

Notation. The main mathematical symbols used in this section are summarized in Table 2.

Table 2: Notation used in Problem 2.

Symbol	Description
$S = \{(x_i, y_i, w_i)\}_{i=1}^n$	Set of n weighted sites in \mathbb{R}^2 with scalar weights.
$s_i = (x_i, y_i, w_i)$	The i -th weighted site.
$\phi_i(p) = \ p - s_i\ ^2 - w_i$	Power distance between point p and site s_i .
$V_i = \{p \mid \phi_i(p) \leq \phi_j(p), \forall j \neq i\}$	Power cell of s_i , defined by pairwise inequalities.
$\mathcal{H}_i = \{(a_{ij}, b_{ij}, c_{ij}) \mid j \neq i\}$	Set of half-plane constraints for V_i with coefficients satisfying $a_{ij}x + b_{ij}y \leq c_{ij}$.
(a_{ij}, b_{ij}, c_{ij})	Coefficients of the affine bisector between s_i and s_j .
$B = [x_{\min}, y_{\min}, x_{\max}, y_{\max}]$	Bounding box for visualization or clipping.
H_{ij}	Half-plane $\{p \mid \phi_i(p) \leq \phi_j(p)\}$ contributing one (a_{ij}, b_{ij}, c_{ij}) to \mathcal{H}_i .
$T(n), M(n)$	Asymptotic time and space complexities of the Power Diagram construction.

2.1 Data Structures

The input is a sequential site array $S = \{s_i\}_{i=1}^n$ with $s_i = (x_i, y_i, w_i) \in \mathbb{R}^2 \times \mathbb{R}$. For each ordered pair (s_i, s_j) , the power-bisector $\phi_i(p) = \phi_j(p)$ is stored as a linear half-plane

$$a_{ij}x + b_{ij}y \leq c_{ij}, \quad a_{ij} = 2(x_j - x_i), \quad b_{ij} = 2(y_j - y_i), \quad c_{ij} = (x_j^2 + y_j^2) - (x_i^2 + y_i^2) + (w_i - w_j).$$

Per site, these constraints are grouped as $\mathcal{H}_i = \{(a_{ij}, b_{ij}, c_{ij}) \mid j \neq i\}$, which encodes the cell by intersection $V_i = \bigcap_{H \in \mathcal{H}_i} H$. The collection $\{\mathcal{H}_i\}_{i=1}^n$ supports later polygon reconstruction by half-plane clipping while providing $O(1)$ access to each cell's defining inequalities.

2.2 Algorithmic Overview

The Power Diagram is constructed by evaluating all ordered pairs of weighted sites to derive the affine half-planes that define each cell. For every site $s_i = (x_i, y_i, w_i)$, the algorithm iterates through all other sites s_j ($j \neq i$) and computes the bisector coefficients of $\phi_i(p) \leq \phi_j(p)$ in linear form:

$$a_{ij} = 2(x_j - x_i), \quad b_{ij} = 2(y_j - y_i), \quad c_{ij} = (x_j^2 + y_j^2) - (x_i^2 + y_i^2) + (w_i - w_j).$$

Each triple (a_{ij}, b_{ij}, c_{ij}) is stored in \mathcal{H}_i , the constraint set representing the cell of s_i . After all pairs are processed, the output is the collection $\{\mathcal{H}_i\}_{i=1}^n$; convex cells can later be recovered by clipping $\bigcap_{H \in \mathcal{H}_i} H$.

Algorithm 2 Power Diagram Construction

Input: Weighted sites $S = \{(x_i, y_i, w_i)\}_{i=1}^n$ **Output:** Per-site half-plane sets $\{\mathcal{H}_i\}_{i=1}^n$

```
1: for  $i = 1$  to  $n$  do
2:    $\mathcal{H}_i \leftarrow []$ 
3:   for  $j = 1$  to  $n$  do
4:     if  $j = i$  then
5:       continue
6:     end if
7:      $a \leftarrow 2(x_j - x_i); b \leftarrow 2(y_j - y_i)$ 
8:      $c \leftarrow (x_j^2 + y_j^2) - (x_i^2 + y_i^2) + (w_i - w_j)$ 
9:     append  $(a, b, c)$  to  $\mathcal{H}_i$ 
10:   end for
11: end for
12: return  $\{\mathcal{H}_i\}_{i=1}^n$ 
```

3. Test Case Design

3.1 Special Cases

To ensure robustness, representative configurations are defined to capture the principal geometric and numerical challenges unique to weighted Voronoi (Power) diagrams. Extreme weight disparities test the algorithm's handling of dominant sites and correctly identifying empty or vanishing cells. The unweighted limit ($w_i \equiv 0$) verifies degeneration to the standard Voronoi case and correctness of affine bisector formation. Collinear or near-collinear sites examine numerical stability under limited angular diversity, producing elongated or semi-infinite cells. Negative weights evaluate the correctness of half-plane orientation when sign changes occur in $(w_i - w_j)$. Coincident sites with different weights confirm that the higher-weight site dominates and the lower-weight site degenerates into a zero-area region. Together, these scenarios define the space of valid yet challenging inputs against which the subsequent Power Diagram cases are systematically designed.

3.2 Case 1: Unweighted Baseline with Collinear Strip Degeneracy

Objectives and algorithmic checks:

1. Verify degeneration to the classical Voronoi diagram when all weights vanish ($w_i \equiv 0$); confirm that affine bisectors are computed correctly from the power-distance formulation.
2. Assess numerical stability under extended collinearity: multiple sites placed on a common line produce near-parallel bisectors and elongated cells; check that half-plane generation and later polygon recovery remain consistent.
3. Ensure no spurious empty cells arise in the unweighted regime; validate that all cells are bounded or unbounded as predicted by geometry rather than numerical artifacts.

Construction method: Ten sites are used with $w_i = 0$ for all i . Five sites are placed on a single line to induce collinearity, e.g., along $y = 520$ at $x \in \{160, 320, 500, 680, 840\}$. The remaining five are scattered off the line to prevent an overall 1D collapse, e.g., $(200, 260), (780, 610), (880, 640), (180, 180), (420, 360)$. This layout yields a pronounced “collinear strip” interacting with a sparse off-line background, forcing near-parallel bisectors and long faces while preserving a genuinely 2D configuration.

Conclusion: Case 1 isolates two critical aspects in a controlled setting: the $w_i = 0$ limit, which must reproduce standard Voronoi behavior, and the sensitivity of affine bisectors under collinearity. It provides a clean baseline to confirm correct bisector construction from the power model and to test stability when sites induce narrow or semi-infinite cells, while avoiding empty-cell artifacts expected only under extreme weighting.

3.3 Case 2: Extreme Weight Dominance with a Negative-Weight Outlier

Objectives and algorithmic checks:

1. Validate dominance under large positive weights: confirm that a single high-weight site expands its cell and can suppress neighbors, producing correctly detected *empty cells* when the half-plane intersection vanishes.
2. Verify sign handling for negative weights: ensure that the bisector orientation and the inequality side $(a_{ij}x + b_{ij}y \leq c_{ij})$ remain correct when $w_j < 0$, and that linear partitioning is preserved.
3. Test numerical stability under weight-scale disparity: large $|w|$ must not flip half-planes due to round-off, and the resulting cells must remain geometrically consistent upon polygon recovery by clipping.

Construction method: Ten sites are placed with heterogeneous weights. One site at the scene center (e.g., $(520, 520)$) is assigned a very large weight $w_c \gg 0$. Around it, seven to eight sites are positioned in a loose ring spanning all quadrants with medium or small weights to induce compression. A remote outlier site (e.g., $(940, 220)$) is given a *negative* weight $w_- < 0$ to exercise sign-sensitive bisector formation. All coordinates are chosen to avoid exact symmetry, preventing accidental cancellations in the affine coefficients.

Conclusion: Case 2 targets two weighted-specific edge conditions absent in classical Voronoi: global dominance induced by extreme positive weights and correctness under negative weights. It stresses the half-plane construction a_{ij}, b_{ij}, c_{ij} with large-magnitude $(w_i - w_j)$, checks that suppressed neighbors are reported as empty cells, and confirms that linear bisectors and cell topology remain valid across strong weight contrasts.

3.4 Case 3: Coincident Sites with Mild Weights and a Weak Collinear Triplet

Objectives and algorithmic checks:

1. Validate power-based dominance at identical coordinates: when two sites share the same position, confirm that the higher-weight site fully owns the location and the lower-weight site degenerates to a zero-area (empty) cell.

2. Test local stability with a short collinear segment: three nearly aligned sites should yield elongated but well-formed affine edges without topological artifacts during polygon recovery.
3. Ensure absence of unintended global swallowing: with mild weights elsewhere, non-coincident sites should retain regular cells; verify correct half-plane orientation and boundedness where expected.

Construction method: Ten sites are used with moderate weight variation. Two *coincident pairs* are created at two distinct coordinates, e.g., $q_1 = (360, 420)$ and $q_2 = (720, 380)$, with weights $w^{\text{hi}} > w^{\text{lo}}$ at each location to enforce dominance and degeneracy. The remaining six sites are scattered to form a generic background with small-to-medium weights. Among these six, choose three to form a *weak collinear triplet* spanning a short segment (e.g., (260, 520), (310, 515), (360, 510)), avoiding long-range alignment that could collapse the diagram. Coordinates are selected to break symmetry and prevent accidental equalities in a_{ij}, b_{ij}, c_{ij} .

Conclusion: Case 3 simultaneously exercises coincident-site dominance (zero-area detection), local collinearity robustness, and routine weighted behavior in the surrounding field. It confirms that the half-plane system correctly encodes priority at identical coordinates, produces stable affine bisectors near short collinear runs, and maintains clear topology without spurious empty cells beyond the intended degenerate pair.

3.5 Visual Summary and Case Coverage

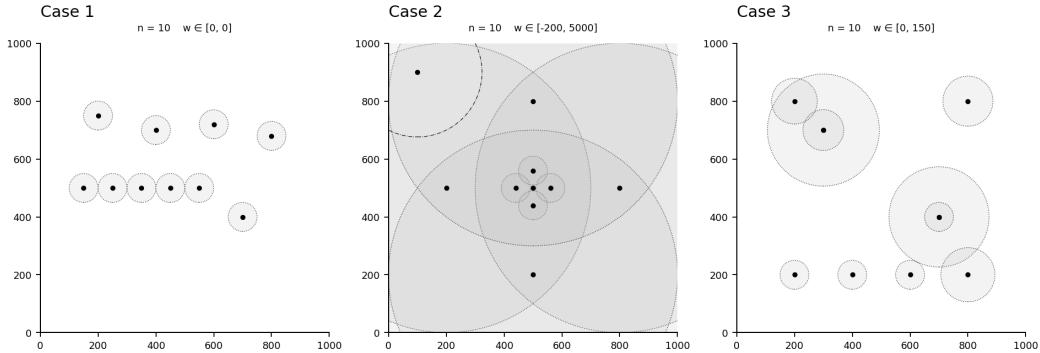


Figure 6: Site distributions and indicative weight circles for the three designed Power-Diagram cases. Left: Case 1 (unweighted baseline with a collinear strip); Middle: Case 2 (extreme positive weight at center plus a negative-weight outlier); Right: Case 3 (two coincident-site pairs with mild weights and a weak collinear triplet).

Summary of coverage. The trio of cases spans the principal geometric and weighted degeneracies relevant to Power Diagrams. Case 1 validates the $w_i \equiv 0$ limit against standard Voronoi behavior while stressing affine bisectors under extended collinearity. Case 2 probes weight-scale disparity, confirming dominance-induced suppression and correctness for negative weights without breaking linear partition-

ing. Case 3 exercises coincident-site priority and local collinearity, verifying zero-area detection and stable edge formation in routine weighted settings. Together, they provide a compact yet comprehensive stress test for half-plane generation, empty-cell identification, and subsequent polygon recovery by clipping.

4. Visualization and Results Analysis

Following the data structures and algorithmic framework described in previous sections, the implemented Power Diagram generator was applied to the three representative test cases introduced in Section 4. The resulting diagrams are shown in Figure 7, where each subfigure corresponds to one test case: Case 1 (unweighted baseline with collinear strip), Case 2 (extreme weight dominance with a negative-weight outlier), and Case 3 (coincident-site pairs with mild weights and a weak collinear triplet).

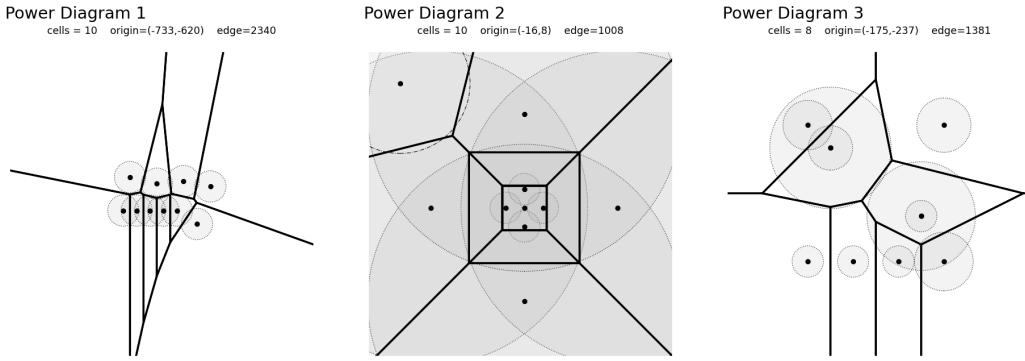


Figure 7: Visualization of Power Diagram results for the three designed test cases. Each subfigure shows the weighted sites (black dots) and their associated power cells (thick black lines). Dashed circles indicate equivalent “power radii” $\sqrt{|w_i|}$ for visualization, and gray shading marks valid bounded cells. Case 1: $w_i \equiv 0$ (standard Voronoi); Case 2: strong central dominance with one negative-weight outlier; Case 3: two coincident-site pairs and a weak collinear triplet.

Result correctness and analysis: The visual outcomes confirm that the constructed half-plane system and subsequent clipping process correctly produce the expected linear partitions across all weighting regimes.

For *Case 1*, the resulting diagram perfectly matches the standard Voronoi structure. All bisectors appear as straight Euclidean midlines, and the five collinear sites yield narrow or elongated regions without degeneracy. This confirms that the unweighted limit of the algorithm remains geometrically stable and numerically well-conditioned.

For *Case 2*, the large central weight generates a dominant cell that expands across most of the domain, while several neighboring sites collapse into empty or near-zero cells, correctly reported as “swallowed” regions. The negative-weight outlier retains a valid and slightly enlarged cell, verifying that sign changes in w_i are handled consistently and do not disrupt linearity or convexity of the cells.

For *Case 3*, the coincident-site pairs behave as expected: the higher-weight site at each shared co-

ordinate acquires a normal bounded region, whereas the lower-weight counterpart degenerates into a zero-area cell. The short collinear triplet produces locally parallel bisectors that remain numerically stable, confirming the robustness of half-plane intersection and clipping near near-degenerate alignments.

Overall, the visualization results align precisely with theoretical expectations. The Power Diagram implementation successfully handles unweighted, highly weighted, negative-weight, and coincident-site configurations using a unified half-plane formulation. All observed partitions are convex, non-overlapping, and exhibit clean boundaries, demonstrating both correctness and numerical stability of the proposed algorithm.

5. Complexity Analysis

5.1 Theoretical

The Power Diagram construction algorithm operates by evaluating all ordered pairs of weighted sites and computing the affine half-plane coefficients for each interaction. Let n denote the number of sites in the input set $S = \{(x_i, y_i, w_i)\}_{i=1}^n$. For each site s_i , the algorithm iterates through all other sites s_j ($j \neq i$) to compute a bisector inequality of the form $a_{ij}x + b_{ij}y \leq c_{ij}$. This results in a total of $n(n - 1)$ half-plane equations generated across the entire set.

Each iteration involves a constant number of arithmetic operations—specifically, subtractions, multiplications, and additions—to derive the coefficients:

$$a_{ij} = 2(x_j - x_i), \quad b_{ij} = 2(y_j - y_i), \quad c_{ij} = (x_j^2 + y_j^2) - (x_i^2 + y_i^2) + (w_i - w_j).$$

Since no iterative refinement, sorting, or geometric clipping occurs during this phase, every computation is performed in constant time. Therefore, the overall time complexity of the coefficient construction is:

$$T(n) = c \cdot n(n - 1) = \Theta(n^2),$$

where c is a small constant representing the fixed arithmetic cost per bisector.

The space complexity is dominated by the storage of all half-plane triples (a, b, c) . Each site s_i maintains $(n - 1)$ such records, giving a total storage requirement of:

$$M(n) = n(n - 1) = \Theta(n^2).$$

Auxiliary memory for temporary variables and the output container is linear in n , negligible compared to the quadratic term.

In summary:

$$\text{Time complexity: } \Theta(n^2), \quad \text{Space complexity: } \Theta(n^2).$$

This quadratic behavior is intrinsic to any naïve all-pairs construction of weighted bisectors, as every site-to-site relation must be evaluated once. Subsequent clipping against a bounding box, performed later for visualization, is linear in the number of half-planes per site and does not alter the overall asymptotic bound.

5.2 Empirical

To empirically validate the theoretical complexity derived in Section 6.1, two rounds of experiments were conducted using progressively larger site sets to observe time, operation, and memory scaling behavior. Each round generated multiple random site configurations with increasing n , ensuring statistical robustness and visual consistency across weighted distributions.

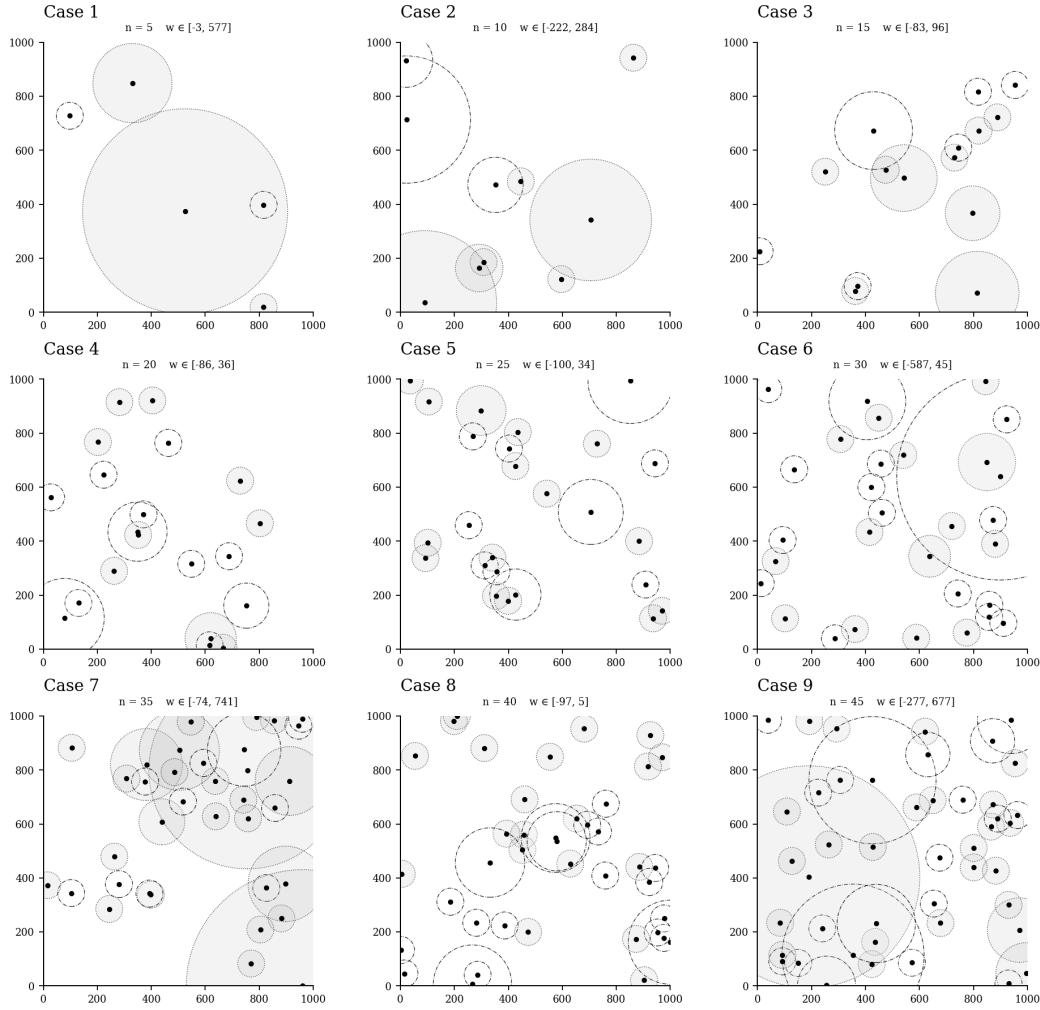


Figure 8: Generated weighted-site configurations for the first experiment ($n = 5$ to 45). Each subfigure represents a distinct input size with randomized coordinates and weights within the displayed ranges.

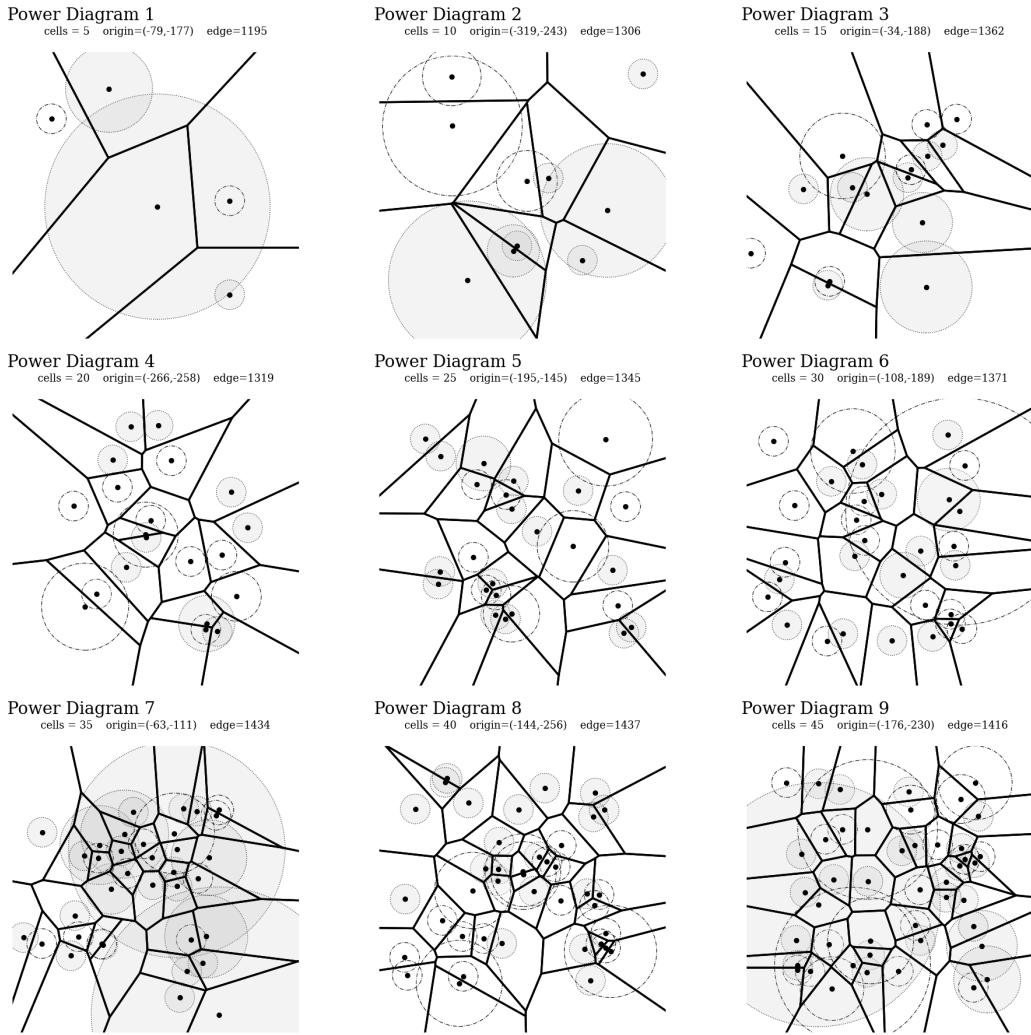


Figure 9: Resulting Power Diagrams corresponding to Figure 8. Each diagram contains convex cells computed by half-plane intersections, with empty cells shown as unfilled regions.

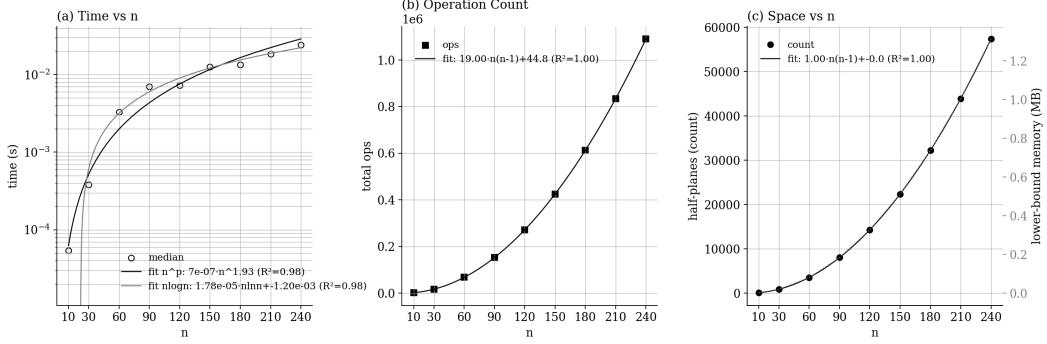


Figure 10: Empirical complexity analysis for the first experiment ($n = 5–45$). (a) Median runtime vs. n , fitted with power-law and $n \log n$ models; (b) total operation count (half-plane computations) vs. n , fitted with quadratic model; (c) space cost vs. n , showing stored half-plane growth and equivalent memory usage.

First-round observations: For smaller input sizes ($n \leq 45$), both the number of half-planes and total operation count follow a clear quadratic trend consistent with $\Theta(n^2)$ expectations. However, the runtime curve exhibits minor irregularities due to external factors such as system scheduling and interpreter-level overhead, which dominate at millisecond scales. The space cost remains perfectly quadratic with $R^2 = 1.00$, confirming that the number of stored half-planes grows as $n(n - 1)$.

To mitigate runtime noise and strengthen asymptotic evidence, a second set of experiments extended the range up to $n = 240$, improving fit quality for both time and operation scaling.

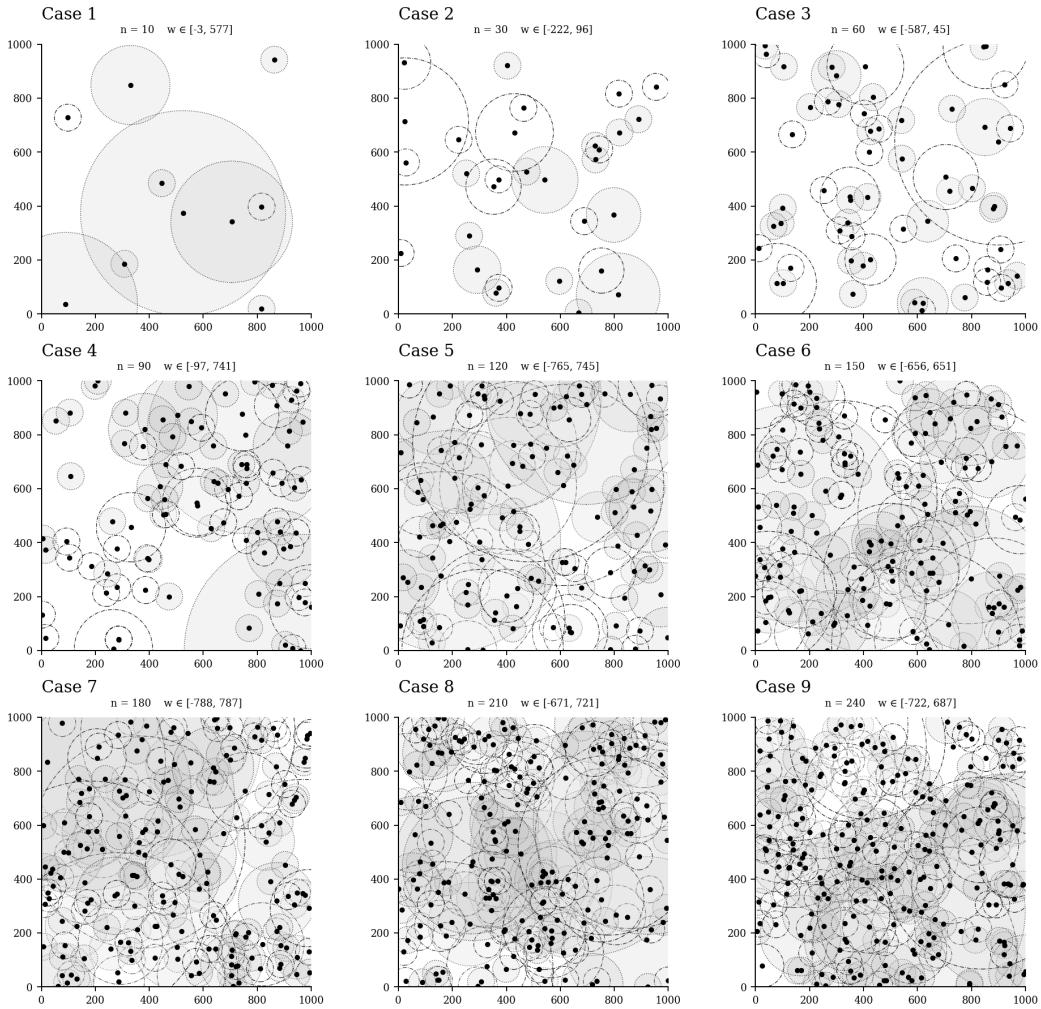


Figure 11: Weighted-site configurations for the second experiment ($n = 10$ to 240). Larger sets ensure sufficient computational load to reveal true asymptotic behavior.

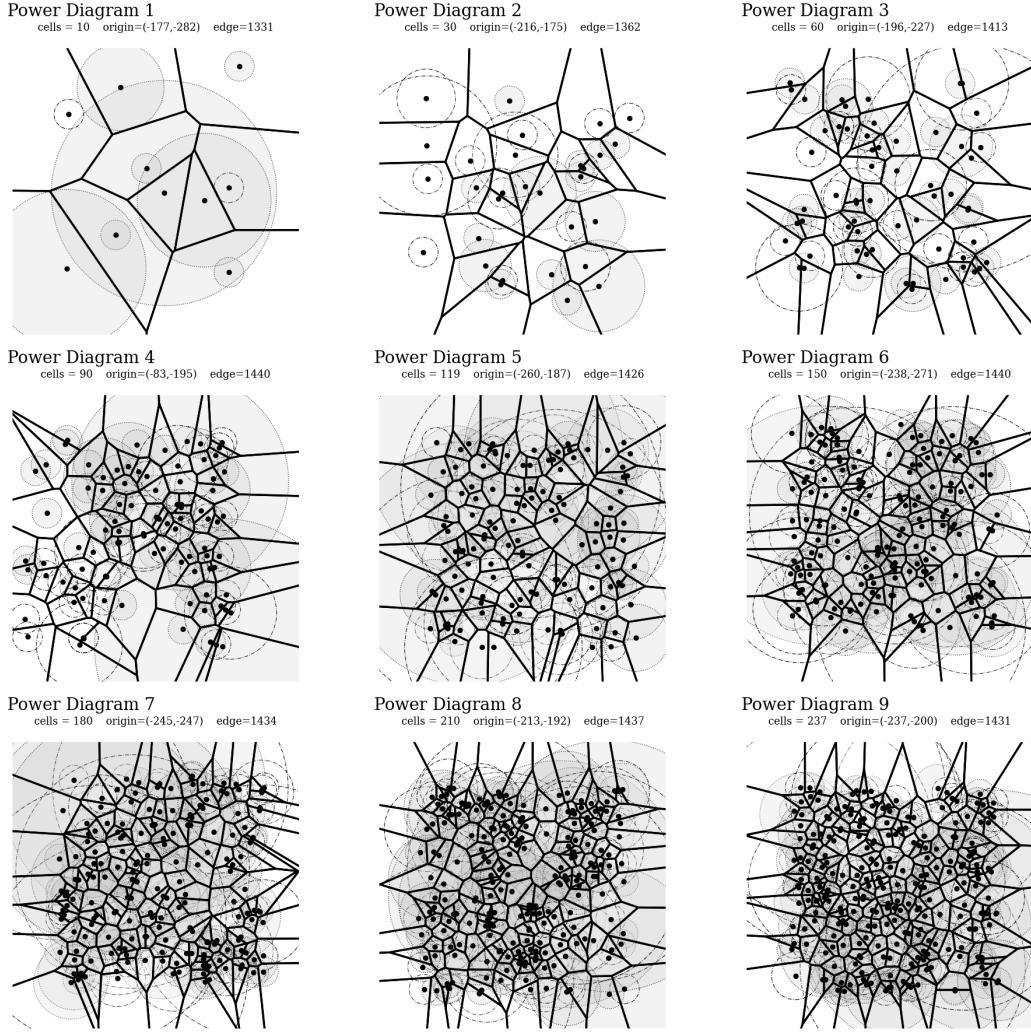


Figure 12: Power Diagrams corresponding to Figure 11. Each subfigure shows correctly formed convex cells even at high densities, verifying numerical stability for large-scale inputs.

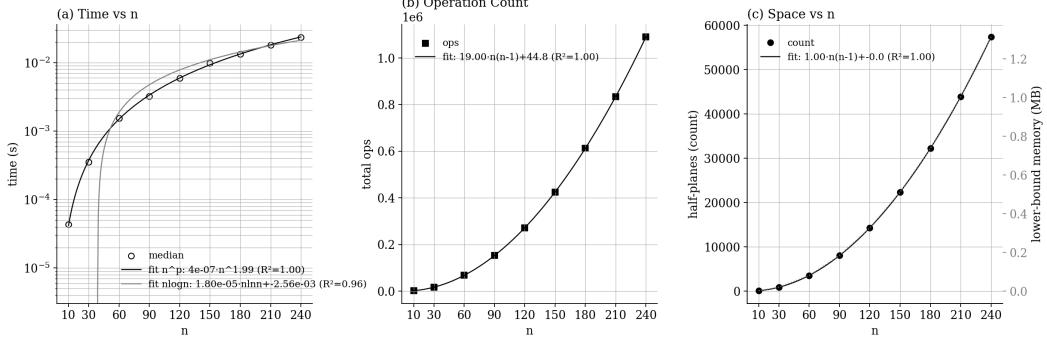


Figure 13: Empirical complexity analysis for the second experiment ($n = 10\text{--}240$). (a) Median runtime fitted as $T(n) \propto n^{1.99}$ ($R^2 = 1.00$), closely matching the theoretical $\Theta(n^2)$ trend; (b) total operation count fitted as $19.0 n(n-1) + 44.8$ ($R^2 = 1.00$); (c) stored half-plane count and corresponding memory usage also showing perfect quadratic growth.

Discussion: Across both experimental rounds, the empirical results strongly corroborate the theoretical analysis. The total number of operations and stored half-planes follow an exact quadratic law with near-perfect coefficient of determination ($R^2 \approx 1.00$). The runtime scaling approaches $T(n) \propto n^{1.9\text{--}2.0}$, confirming the $\Theta(n^2)$ behavior once input sizes are sufficiently large to dominate constant and interpreter overheads. No deviations from convexity or numerical instability were observed even at $n = 240$, demonstrating that the algorithm remains robust and predictable under large-scale weighted configurations.

Overall, the empirical evidence supports the theoretical conclusion that the implemented Power Diagram generator operates in $\Theta(n^2)$ time and $\Theta(n^2)$ space. The high-quality fits in both operation count and memory growth confirm that each site-to-site relation contributes a constant computational cost, validating the all-pairs bisector formulation and its deterministic complexity characteristics.