Scalable GPU Construction of 3D Voronoi and Power Diagrams

Scalable GPU Construction of 3D Voronoi and Power Diagrams

BERNARDO TAVEIRA*, Chalmers University of Technology, Sweden and Zenseact, Sweden
CARL LINDSTRÖM*, Chalmers University of Technology, Sweden and Zenseact, Sweden
MARYAM FATEMI, Zenseact, Sweden
LARS HAMMARSTRAND, Chalmers University of Technology, Sweden
FREDRIK KAHL, Chalmers University of Technology, Sweden

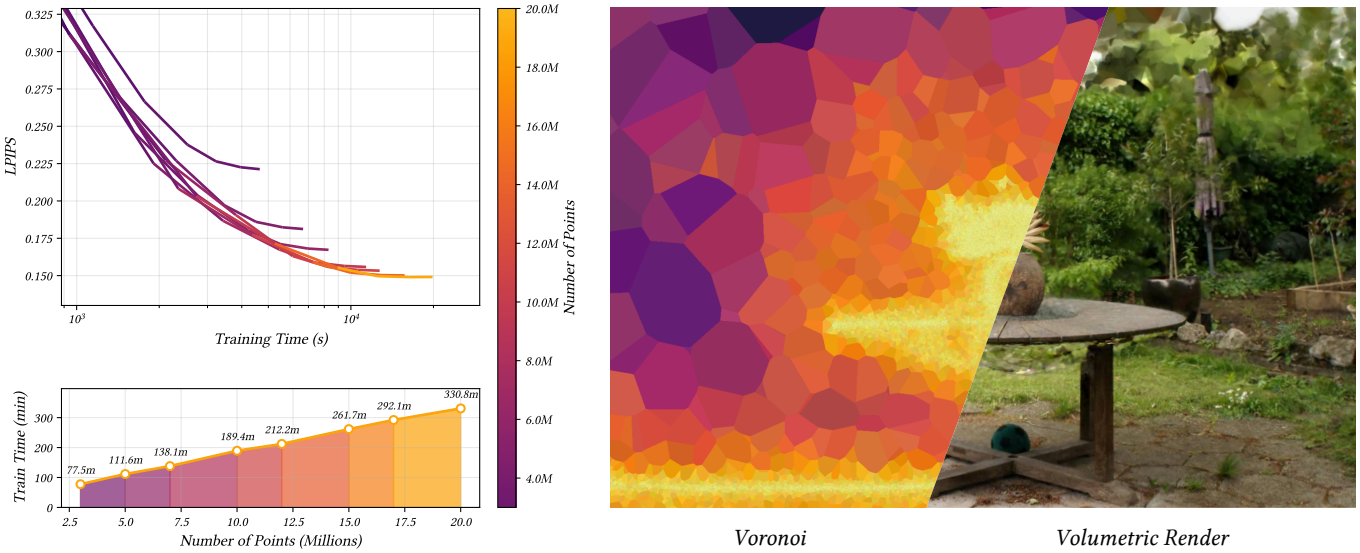


Fig. 1. **Scalability and perceptual quality.** Our efficient construction of 3D Voronoi diagrams enables mesh-based neural rendering to scale to 20 million cells, and reveals a strong correlation between model capacity and perceptual quality. We apply our method to the Radiant Foam [Govindarajan et al. 2025] on the challenging "Garden" scene from the mip-NeRF 360 dataset [Barron et al. 2022]. **(Left)** The bottom plot shows that total training time (wall-clock, including all optimization steps) increases approximately linearly with the point-count limit. The top plot tracks the evolution of the LPIPS perceptual error across multiple training runs with different point budgets, demonstrating how higher point capacities allow for better visual convergence during training. **(Right)** A split view of the rendered image and a 2D slice of the Voronoi mesh, color-coded to reflect cell size, highlights the massive and highly varying cell density required to model this scene.

Voronoi diagrams, and their more general weighted counterpart, power diagrams, are fundamental geometric constructs with wide-ranging applications in biology, physics simulation, and computer graphics. Recently, they have gained renewed attention in mesh-based neural rendering. Despite being extensively studied, the construction of 3D Voronoi diagrams for large-scale point sets remains computationally expensive, limiting their adoption in large-scale applications. Existing CPU-based approaches typically rely on computing its dual, the Delaunay tetrahedralization, but are prohibitively slow for large diagrams, while GPU-based methods either struggle to scale efficiently to large point sets or assume homogeneous point distributions. The weighted case, power diagrams, is even less explored in this context. Existing approaches are typically tailored to the application at hand, assuming homogeneous point distributions and small weight variations, making them unsuitable for general use in more complex heterogeneous data.

In this paper, we present a highly parallelizable GPU algorithm for the fast construction of large-scale 3D Voronoi and power diagrams. Our approach constructs each convex cell from a weighted 3D point by progressively clipping an initial cell volume against bisecting planes induced by candidate neighboring points. To efficiently identify candidate neighbors under arbitrary spatial distributions, we introduce a culling criterion based on directional geometric bounds of the evolving cell, combined with a hierarchical best-first traversal of bounding volumes.

We achieve performance on par with state-of-the-art Delaunay tetrahedralization methods on small and moderate problem sizes, while exhibiting robust scalability to large point sets and diverse spatial distributions. Moreover, our method naturally generalizes to power diagrams without additional

*Both authors contributed equally to the paper.

Authors' Contact Information: Bernardo Taveira, Chalmers University of Technology, Gothenburg, Sweden and Zenseact, Gothenburg, Sweden, bernardo.taveira@chalmers.se; Carl Lindström, Chalmers University of Technology, Gothenburg, Sweden and Zenseact, Gothenburg, Sweden, carl.lindstrom@chalmers.se; Maryam Fatemi, Zenseact, Gothenburg, Sweden, maryam.fatemi@zenseact.com; Lars Hammarstrand, Chalmers University of Technology, Gothenburg, Sweden, lars.hammarstrand@chalmers.se; Fredrik Kahl, Chalmers University of Technology, Gothenburg, Sweden, fredrik.kahl@chalmers.se.

SIGGRAPH Conference Papers '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2554-8/26/07
https://doi.org/10.1145/3799902.3811229

assumptions. To facilitate reproducibility and future research, we release our source code, see https://research.zenseact.com/publications/paragram.

CCS Concepts: • **Computing methodologies → Massively parallel algorithms**.

Additional Key Words and Phrases: Computational Geometry, Power Diagrams, Weighted Delaunay Triangulation, Voronoi, GPU Algorithms, Neural Rendering, Spatial Acceleration Structures

ACM Reference Format:
Bernardo Taveira, Carl Lindström, Maryam Fatemi, Lars Hammarstrand, and Fredrik Kahl. 2026. Scalable GPU Construction of 3D Voronoi and Power Diagrams. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3799902.3811229

1 Introduction

Voronoi diagrams and their dual, Delaunay triangulations, stand as pillars of computational geometry. For decades, these have been a fundamental concept to computer graphics, enabling mesh generation, fluid simulation, collision detection and surface reconstruction [Aurenhammer 1991; Brochu et al. 2010; Shewchuk 2002]. The generalization to weighted points—known as power diagrams and weighted Delaunay triangulation—further extends their utility, enabling the modeling of poly-disperse aggregates and volume preserving partitions. Extensive research, particularly for unweighted cases, has produced reliable and precise methods for computing these geometric structures.

The advent of differentiable rendering has shifted the computational demands from the rendering task into the optimization of the underlying representation. Recent mesh-based neural rendering methods like Radiant Foam [Govindarajan et al. 2025] and Radiance Meshes [Mai et al. 2025] demonstrate high quality results and valuable applications by representing 3D space as a mutable volumetric mesh, derived from Voronoi or Delaunay topology. However, these methods introduce a constraint that geometric solvers were never designed to meet: computing diagrams of massive scale (exceeding 10^6 sites) within iterations of an optimization problem. As we show in our experiments, the mesh generation frameworks currently employed face scalability and generality limitations that constrain their broader adoption.

This computational bottleneck now limits the scaling of volumetric neural rendering methods. While CPU-based solvers are well-studied and precise (e.g., [The CGAL Project 2025]), they lack the efficiency to scale. GPU alternatives remain underexplored, with existing methods exhibiting stability issues, failing to scale beyond a few million points, or relying on restrictive assumptions on the point distribution. As the compute hardware improves and neural rendering pushes towards higher resolutions, finer details and larger scenes, the inability to efficiently generate Delaunay and Voronoi meshes becomes the limiting factor on visual quality.

We address this challenge by introducing a highly parallelizable, unified framework for computing 3D power diagrams, weighted Delaunay, and their special cases, Voronoi and Delaunay. Our method is designed to tackle large-scale problems beyond the degree used by current applications. By decoupling cell computation into independent threads, designing a directional culling criteria and utilizing a bounding volume hierarchy tree (BVH) for rapid geometric search and traversal, our algorithm harnesses the massive parallel capabilities of current and future GPUs.

Despite being a general-purpose algorithm, we demonstrate its potential in the context of neural rendering. By replacing the Radiant Foam Voronoi generation method with our drop-in replacement, we effectively remove the computational bottleneck. This allows us to scale the explicit representation from the previous practical limit by 5x, to over 20 million points. We show that this increased capacity alone directly results in improved reconstruction quality.

2 Related work

We review prior work on the construction of Voronoi and power diagrams, organized by algorithmic strategy, and conclude with a discussion of mesh-based neural rendering, the application driving our scalability requirements.

2.1 Voronoi and power diagram construction

The construction of Voronoi diagrams, power diagrams, and their Delaunay duals is a long-studied problem in computational geometry. Existing solvers fall into three broad algorithmic approaches: incremental insertion, convex-hull lifting, and cell-oriented clipping.

Incremental insertion. Incremental algorithms build the diagram one site at a time, locally retriangulating the cavity of conflicting simplices. The classical Bowyer–Watson method [Bowyer 1981; Watson 1981] underpins mature CPU libraries such as CGAL [The CGAL Project 2025] and Geogram [Lévy 2025], both of which ship multi-core implementations. Parallel formulations of this approach have also been the subject of dedicated research [Chrisochoides and Nave 2003]. A carefully engineered multi-threaded implementation by Marot et al. [2019] tetrahedralizes three billion points on a single workstation in under a minute, and their HXT library [Marot and Remacle 2020] represents the current CPU state of the art. On the GPU, gDel3D [Cao et al. 2014] adapts this strategy to the massive parallelism the hardware permits, but its resource footprint limits scalability to a few million sites on consumer hardware.

Convex-hull lifting. Lifting sites onto a paraboloid in $\mathbb{R}^{d+1}$ reduces Voronoi and power diagram construction to a lower-convex-hull problem [Aurenhammer 1987]. This formulation naturally accommodates the weighted case, and is used by the Quickhull implementation QHull [Barber et al. 1996] and the Delaunay routines of SciPy [Virtanen et al. 2020], which rely on QHull internally. However, the added dimension makes it uncompetitive for the large 3D point sets we target.

Cell-oriented clipping. Cell-oriented methods construct each cell independently by progressively clipping an initial convex volume against bisecting half-spaces induced by neighboring sites. This formulation was popularized by the CPU library Voro++ [Rycroft 2009], later extended to multi-threaded execution [Lu et al. 2023]. A key concept for efficient cell clipping is the radius of security criterion of Lévy and Bonneel [2013], which guarantees that the cell has been clipped by enough neighbors, allowing early termination of per-cell construction. Sainlot et al. [2017] build on this idea, proposing corner validation as an alternative termination criterion.

Ray et al. [2018] and the subsequent work of Basselin et al. [2021] brought the cell-oriented strategy to the GPU, targeting meshless volume integration. Their approach assumes a near-uniform distribution of sites. Neighbor search is driven by a k-nearest-neighbor heuristic and terminates at a fixed multiple of the current cell radius. These assumptions break down in the heterogeneous and weighted settings we target, where local density, and thus the required search radius, varies by orders of magnitude across the domain.

Our method builds on Basselin et al. [2021]. We preserve the per-cell independence that makes this formulation well suited to massive parallelism, and replace the isotropic radius criterion with a *directional* culling criterion derived from the evolving axis-aligned bounds of each cell, combined with a hierarchical best-first traversal over a BVH augmented for power-distance queries. Together, these components enable a GPU-native construction that scales to tens of millions of sites under arbitrary spatial and weight distributions.

2.2 Mesh-based view synthesis

Research on novel view synthesis was significantly accelerated by Neural Radiance Fields (NeRF) [Mildenhall et al. 2021], which represents scenes as continuous volumetric radiance fields optimized via differentiable volume rendering. Subsequent work has focused on improving rendering efficiency by adopting explicit scene representations and point-based rasterization, most notably through 3D Gaussian Splatting (3DGS) [Kerbl et al. 2023].

Given the foundational role of meshes in computer graphics, several approaches have explored the use of meshes for both NeRF-style implicit fields [Wang et al. 2021; Yariv et al. 2021, 2023] and 3DGS-style explicit representations [Choi et al. 2024; Gao et al. 2024; Lin et al. 2024]. More recent work has investigated differentiable, mesh-based scene representations that explicitly discretize space. Radiant Foam [Govindarajan et al. 2025] represents scenes as an optimizable 3D Voronoi diagram. To support accurate ray-tracing, it requires frequent updates of cell adjacencies during training via Delaunay triangulation. Similarly, Radiance Meshes [Mai et al. 2025] partitions space into tetrahedral cells derived from gDel3D's [Cao et al. 2014] Delaunay tetrahedralization.

These methods typically initialize cell sites using structure-from-motion [Schönberger and Frahm 2016], then optimize their positions via gradient descent on photometric reconstruction losses, a process requiring up to 20k optimization steps. Heuristics are also employed to densify and prune the representation throughout training.

While these approaches enable explicit control over scene discretization, they critically depend on efficient triangulation algorithms, as connectivity must be updated frequently during optimization. Moreover, visual fidelity scales with the number of sites, motivating the need for larger geometric meshes and scalable construction methods.

3 Preliminaries

To understand our method, we begin by formally defining the geometric structures that underlie it. Let $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ be a set of n points, each with an associated scalar weight $w_i \in \mathbb{R}$. The *power diagram* of P partitions space into convex cells according to the power distance, where each cell site p_i is

$$C_i = \{x \in \mathbb{R}^d \mid \|x - p_i\|^2 - w_i \leq \|x - p_j\|^2 - w_j,\ \forall j \neq i\}. \quad (1)$$

When all weights are equal, the problem is reduced to the standard Voronoi diagram, which partitions space by nearest neighbors. The sites p_i and p_j are considered adjacent if their cells share a polygonal face. The resulting adjacency graph defines the connectivity of the dual *regular* (weighted) Delaunay triangulation.

Equation 1 reveals a key property for parallel construction. Each power cell C_i is computationally independent, defined solely by the intersection of half-spaces derived from its neighbors. This independence implies that the construction of the power diagram of P can be decomposed into n local clipping tasks. However, computational efficiency depends on identifying relevant neighbors from the large set P that contribute to the cell boundary.

4 Method

Given the set of weighted points P, our goal is to compute the power diagram where each point's cell is given by Eq. 1 and forms a convex polyhedron defined by the intersection of half-spaces induced by bisecting planes with neighboring cells. Note that the special case of equal weights reduces to the Voronoi diagram and its dual, the Delaunay triangulation.

To this end, we present a new GPU-accelerated algorithm for computing 3D power diagrams at scale. Our method combines three key components:

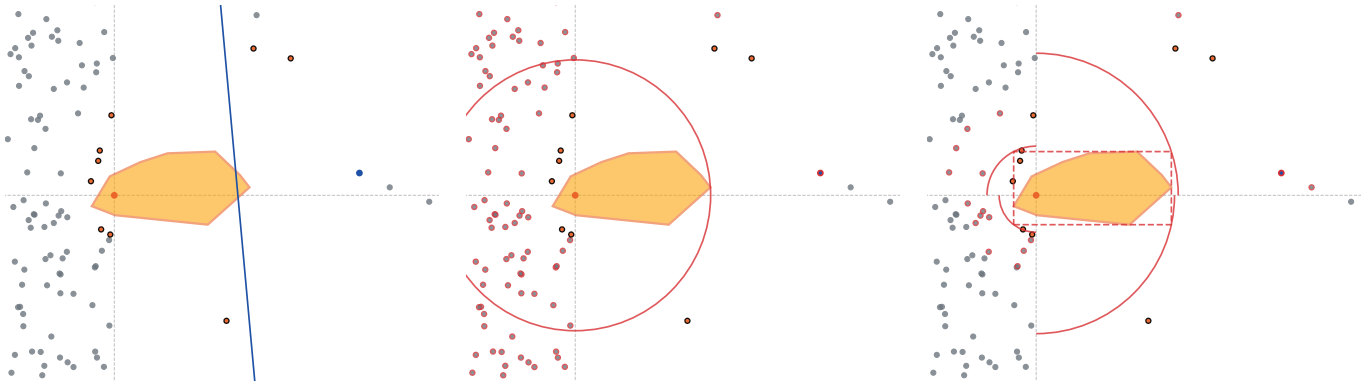
(1) a convex cell clipping algorithm to construct individual diagram cells (Sec. 4.1),
(2) a directional culling criterion that efficiently identifies relevant neighbor sites (Sec. 4.2), and
(3) a hierarchical spatial data structure that enables rapid neighbor queries (Sec. 4.3).

Our key technical innovation efficiently discards large volumes of irrelevant neighbor sites using tight geometric bounds and a hierarchical data structure, while guaranteeing correctness.

4.1 Convex cell clipping

The core of our algorithm lies in computing each cell independently as the intersection of a set of half-spaces, following [Basselin et al. 2021]. In a power diagram, the half-spaces of a cell are defined by the bisection planes between a site and its neighbors. Consequently, given the adjacency of a site, the corresponding convex polyhedron can be constructed directly via half-plane intersection. Naively, if a cell is iteratively clipped by all bisecting planes induced by other sites in the point set, and the sites whose planes contribute to the final boundary are recorded, the resulting convex polyhedron and its adjacency are recovered.

The clipping procedure follows [Basselin et al. 2021]. For each candidate neighbor, the vertices lying outside the corresponding half-space are removed. The intersection between the bisecting plane and the current convex cell is then computed, producing a polygonal boundary whose vertices are added to the cell. If the bisecting plane lies entirely outside the current cell, the intersection is empty and the cell remains unchanged. See the supplementary



(a) Convex cell clipping by a bisecting plane. The blue site induces a plane that intersects the current cell (orange) and therefore must be considered during clipping.

(b) Isotropic culling using the maximum cell radius. All sites within the global radius are retained as candidates (red), including many whose bisecting planes cannot intersect the cell.

(c) Directional culling using AABB-based geometric bounds yields a significantly tighter candidate set (red).

Fig. 2. Effect of geometric bounds on neighbor culling during convex cell construction. The current cell is shown in orange, with previously accepted neighbor sites highlighted. (a) A site whose bisecting plane intersects the cell (blue) must be selected in the candidate set for cell clipping. (b) An isotropic bound based on the maximum cell radius correctly includes the site, but also admits many irrelevant candidates. (c) Directional geometric bounds derived from an AABB of the cell provide a tighter criterion, reducing the number of unnecessary neighbors while preserving correctness.

material for a detailed description. Fig.2(a) demonstrates the complementary case where a non-empty intersection triggers clipping. To achieve optimal performance, clipping should begin with neighbors that are most likely to contribute to the final cell. Early reduction of the cell volume increases the likelihood that subsequent clipping operations result in no modification and can be skipped.

4.2 Geometric bounds and culling criteria

The brute-force cell sculpting procedure described in Sec. 4.1 guarantees the correct diagram construction, but becomes prohibitively expensive for large point sets if all points are treated as potential neighbors. As the cell is progressively clipped, however, many neighbors can be rejected early if their bisecting planes are too distant to intersect the current cell. Since exact intersection tests are costly, we instead rely on conservative geometric bounds.

For a site $p_i \in \mathbb{R}^3$ with weight $w_i \in \mathbb{R}$ and a candidate neighbor $p_j \in \mathbb{R}^3$ with weight $w_j \in \mathbb{R}$, the (power) distance from p_i to their bisecting plane is

$$d_{ij} = \frac{||p_i - p_j||^2 + w_i - w_j}{2||p_i - p_j||}. \tag{2}$$

Given a geometric bound r_i on the extent of the cell C_i, the candidate neighbor C_j can be safely discarded if $d_{ij} > r_i$. Provided that r_i upper-bounds the distance to all vertices relevant to that neighbor, this criterion preserves correctness.

4.2.1 Directional culling. Prior work [Basselin et al. 2021] employs an isotropic bound defined as the maximum distance from the cell site to any current cell vertex, see Fig.2(b). While effective for roughly uniform point distributions, this bound becomes overly conservative for more complex configurations, when the cell's spatial extent varies significantly across directions.

We address this limitation with a *directional geometric bound* that depends on the relative position of the candidate neighbor. The key observation is that a bisecting plane can only intersect a subset of the cell faces depending on the direction vector $p_j - p_i$. By restricting the bound based on those faces, we obtain a tighter and more informative culling criterion.

Specifically, we define a *directional cell radius* using the distances from the cell site to the corners of an axis-aligned bounding box (AABB) enclosing the current cell vertices, Fig.2(c). The appropriate bound for a candidate neighbor is determined by identifying the octant of space in which the neighbor p_j lies relative to the cell site p_i. This octant uniquely determines which subset of AABB corners can possibly intersect with the corresponding bisecting plane.

The directional radius is the maximum distance to these corners, giving a conservative upper bound on the cell's extent toward the candidate neighbor. This significantly reduces unnecessary clipping while retaining the efficiency of a max over a small, fixed set.

4.2.2 Bounding-volume culling. Evaluating the directional culling criterion for every individual site can still be inefficient for large point sets. Prior work [Basselin et al. 2021] mitigates this by iteratively querying the k nearest neighbors in a growing search radius, which works well for homogeneous point distributions and small weight differences between cells. In contrast, we make no assumptions on point distributions, and instead formulate a general strategy for discarding entire volumes of candidates.

We extend directional geometric bounds to spatial groups of candidates by testing entire axis-aligned bounding volumes at once. Specifically, we determine whether any point p_j within a volume $B = [\, b_{min}, b_{max} \,]$, where $b_{min}, b_{max} \in \mathbb{R}^3$, could clip the current cell defined by p_i. Since we assume no symmetry, size, or placement of the volume, the test naturally supports directional discarding and applies to arbitrary spatial partitions.

Each volume B is associated with the maximum weight $w_{max} = \max_j w_j$ of any site it contains. This represents a worst case, as higher weights induce bisecting planes closer to the cell site. We conservatively assume the closest potential candidate lies on the surface of B at the location minimizing its distance to p_i, giving a lower bound on all bisecting-plane distances d_{ij} for points p_j in B.

Because B may span multiple octants relative to p_i, we compute the directional radius for each occupied octant and take their maximum as a conservative bound. This upper-bounds the cell extent in all directions from which a candidate in B could affect it.

Finally, we compare the bisecting-plane distance induced by the closest possible site on the boundary of B, using the maximum weight w_{max}, against the upper-bound r_i. If d is the distance between the closest possible boundary point of B and p_i, then

$$d_{ij} = \frac{||p_i - p_j||}{2} + \frac{w_i - w_j}{2||p_i - p_j||} \geq d/2 + \frac{w_i - w_{max}}{2d}$$

provided $w_i \leq w_{max}$, and otherwise, $d_{ij} \geq d/2$ is a valid lower bound. If this lower bound on d_{ij} exceeds r_i, no candidate in B can clip the current cell, and the entire volume is discarded.

4.3 Hierarchical neighbor search

Efficiently identifying the subset of sites that define the cell boundary requires a spatial acceleration structure that supports rapid culling of distant candidates. Furthermore, the amount of clips required to reach the final cell polyhedra is highly dependent on the order in which planes are processed. Processing the nearest neighbors first can rapidly shrink the cell, tightening the bounding box and allowing us to discard large portions of the search space early.

To partition the sites, we employ a bounding volume hierarchy (BVH). This structure inherently supports hierarchical, multi-scale culling: a single test at a high-level node can discard vast regions containing millions of sites, while deeper traversals allow precise culling of smaller subsets. This spatial adaptivity is crucial for distributions of drastically varying densities, where uniform grids or voxelization approaches [Basselin et al. 2021] struggle to balance skipping large empty volumes with processing high-density clusters. To support power diagrams, we augment each BVH node to not only store its volumetric bounds but also the maximum weight of any site in its subtree, enabling the culling test in Sec. 4.2.2.

Standard BVH traversals usually employ a depth-first search (DFS) using a LIFO (last-in-first-out) stack. However, DFS does not ensure that nodes are visited in order of proximity, leading to wasted cell clipping operations against distant sites that are later occluded and discarded as adjacent. Instead, we employ a best-first search, using a local priority queue ordered by distance. This approach prioritizes proximity, ensuring that we clip the most restrictive planes early in the process, rapidly shrinking the cell.

During the traversal and clipping process we maintain and update an axis-aligned bounding box (AABB) of the current partially clipped cell. As described in Sec. 4.2, this bounding box is used to determine if any point within a volume could potentially clip the cell bounding box. The same signed distance function is employed to pick the closest of the node's children. By using this signed function instead of the euclidean distance to the site seed, we prioritize the neighbors most likely to induce a cut on the current bounding volume. The signed distance metric effectively measures how deep a candidate site can potentially penetrates the current validity region of the cell. By prioritizing based on this penetration depth, we ensure that the algorithm processes the nodes most likely to clip the cell in a significant manner. This strategy maximizes the rate of volume reduction, as the most intrusive planes are clipped immediately. Pseudo-code of this best-first neighbor traversal is provided in Alg. 1.

ALGORITHM 1: Best-First Cell Neighbor Traversal

Input: BVH tree $\mathcal{T}$, Seed point P, Initial Radii R

```
stack ← ∅
node ← T.root
while true do
    // Traverse down until a leaf or dead-end is reached
    while not IsLeaf(node) do
        n0, n1 ← node.children
        r0 ← ComputeDirectionalRadius(R, P, n0.bounds)^2
        r1 ← ComputeDirectionalRadius(R, P, n1.bounds)^2
        d0 ← NodeSqrDist(n0);   d1 ← NodeSqrDist(n1)
        δ0 ← d0 − r0;   δ1 ← d1 − r1
        if min(δ0, δ1) > 0 then
            node ← NULL          // Both children too far
            break
        // Traverse near child, push far child if
           potentially valid
        if δ0 < δ1 then
            node ← n0;   far ← {n1, d1};   δ_far ← δ1
        else
            node ← n1;   far ← {n0, d0};   δ_far ← δ0
        if δ_far ≤ 0 then
            stack.Push(far)
    if node ≠ NULL then
        // We reached a valid leaf
        R ← ProcessLeaf(node.prims, R)
    // Backtrack: Pop next candidate
    while true do
        if stack is empty then
            return
        {idx, dist} ← stack.Pop()
        r_new ←
          ComputeDirectionalRadius(R, P, T[idx].bounds)^2
        // Check if node is still valid with potentially
           shrunk R
        if dist − r_new ≤ 0 then
            node ← T[idx]
            break                    // Resume traversal
```

5 Experiments

We evaluate our method across synthetic datasets and a real-world neural rendering application to demonstrate both computational efficiency and practical applicability. Our experiments are designed to evaluate our method's performance across different spatial distributions, how it scales with an increasing number of points, and

its effectiveness in existing applications. We describe the datasets used for our experiments in Sec. 5.1, and present the corresponding results in Sec. 5.2. Finally, we demonstrate how our method can scale up neural rendering applications in Sec. 5.3.

5.1 Datasets

Synthetic test cases. To evaluate the computational efficiency of our method across different point set distributions, we generate three synthetic test configurations. All configurations sample points within the domain $\Omega = [-10, 10]^3$. The first configuration samples points uniformly:

$$p_i \sim \mathcal{U}(\Omega). \tag{3}$$

This represents generic spatial distributions with no inherent structure but varying density.

The second configuration samples points from multiple Gaussian clusters to simulate scenarios with local density variations and large empty regions. We first sample K cluster centers uniformly in Ω, then distribute points evenly among clusters:

$$c_j \sim \mathcal{U}(\Omega), \quad p_i \sim \mathcal{N}(c_j, \sigma^2 \mathbf{I}), \tag{4}$$

where $\sigma = 0.1$ and points are clamped to Ω. We evaluate with $K \in \{5, 10\}$ clusters.

In the third test configuration, we sample points with a linear density gradient along the x-axis using inverse transform sampling:

$$p(x) \propto (x - a), \quad x \in [a, b] \tag{5}$$

$$\Rightarrow x = a + (b - a)\sqrt{u}, \quad u \sim \mathcal{U}(0, 1) \tag{6}$$

$$y, z \sim \mathcal{U}(a, b) \tag{7}$$

where $[a, b] = [-10, 10]$. This configuration produces higher point density at $x = 10$ and lower density at $x = -10$.

For each synthetic test configuration, we evaluate performance with point counts ranging from 0.1M to 15M, spanning from moderate to large-scale problems. Fig. 3 shows a visualization of the synthetic test cases.

Real-world test cases. To evaluate our method on realistic point distributions that arise in practical applications, we use trained checkpoints from Radiant Foam [Govindarajan et al. 2025], a neural rendering model that uses 3D Voronoi diagrams as its differentiable scene representation. We obtain publicly available checkpoints from the original paper, trained on seven scenes from the Mip-NeRF 360 dataset [Barron et al. 2022] and two scenes from the Deep Blending dataset [Hedman et al. 2018]. Each checkpoint contains the seed point positions of the final Voronoi diagram configuration after optimization convergence. These checkpoints provide representative examples of moderately large point sets (ranging from 2M to 4.2M sites, depending on the scene) with complex spatial distributions that emerge from volumetric rendering optimization. These test cases exhibit realistic patterns of local clustering, empty space, and varying density that reflect the underlying scene geometry. Fig. 7 shows a visualization of the real-world test cases.

Power diagrams. To validate that our method remains efficient for the more general case of weighted Voronoi diagrams (i.e., power diagrams), we evaluate on both the synthetic and real-world test cases with weighted point sets. We sample weights from a Gaussian distribution:

$$w_i \sim \mathcal{N}(0, (\frac{d_{\text{nn}}^2}{3})^2) \tag{8}$$

where d_{nn} is the median nearest-neighbor distance in the point set. We use identical point positions as in the unweighted experiments to isolate the impact of weights on performance.

5.2 Results

We evaluate our algorithm on the datasets presented in Sec. 5.1, measuring computational efficiency on both consumer-grade GPUs (NVIDIA RTX 5090) and enterprise GPUs (NVIDIA H200). We compare against two GPU-based algorithms for 3D Delaunay triangulation, gDel3D [Cao et al. 2014] and the implementation from Radiant Foam [Govindarajan et al. 2025]. Further, we include our own CUDA implementation of [Basselin et al. 2021], adapted to facilitate explicit mesh construction. Additionally, we evaluate CPU-based implementations from SciPy [Virtanen et al. 2020], CGAL [The CGAL Project 2025], Geogram [Lévy 2025], Voro++ [Rycroft 2009] and HXT GmSH [Marot and Remacle 2020]. For CPU methods, we use 16 CPU cores from an AMD Epyc 9534. We run three warm-up iterations followed by ten timed runs from which we compute the average runtime. We impose a 300-second timeout limit. All reported timing results are end-to-end; no preprocessing or preallocation is excluded.

Computational efficiency. For the synthetic test cases, we evaluate point counts of 0.1M, 0.5M, 1M, 2M, 5M, 10M, and 15M, presenting results in Fig. 3. Detailed numerical results for Fig. 3 are provided in the supplementary material. We also measure runtime on the trained checkpoints from Radiant Foam (described in Sec. 5.1) and present results in Tab. 1. Finally, we show aggregated results of the best performing methods in Fig. 5.

As illustrated in Fig. 5, our algorithm is competitive across all point distributions and scales. Notably, it is the fastest across all point sizes for both the Poisson-distributed cases (density gradient and white noise) and the real-world scenes, and the fastest overall for large point sets (≥5M). Among the GPU-baselines, gDel3D achieves the best efficiency on the small clustered point sets, but exhausts memory for larger scenes even on enterprise hardware. On real-world scenes it is on average ∼ 55% slower than our method on the H200 and ∼ 137% slower on the 5090. Our implementation of [Basselin et al. 2021] scales to large point sets on Poisson-distributed data, but fails to complete any other test cases due to the limitations of KNN-based search on irregular distributions. HXT is the best-performing CPU-baseline, closely followed by Geogram. HXT performs very close to our method on the clustered data and scales well to larger point sets, but is on average ∼ 55% slower than our method (H200) on the Poisson-distributed data and ∼ 129% slower on real-world scenes. Overall, our method outperforms both CPU and GPU baselines on both real-world scenes and Poisson-distributed data across all point set sizes, while remaining competitive on the clustered data, particularly for large point sets. Further details on precision and timing breakdown of our method are provided in the supplementary material.

Table 1. **Voronoi diagram creation/Delaunay triangulation**. Runtime results (in seconds) on point sets obtained from Radiant Foam checkpoints trained on the Mip-NeRF 360 dataset. Color coding denotes fastest, second fastest, and third fastest.

		bicycle 4.2M	bonsai 2.0M	counter 2.0M	drjohnson 2.8M	garden 4.1M	kitchen 2.0M	playroom 3.2M	room 1.9M	stump 4.2M
CPU	SciPy	279.710	121.480	127.273	187.908	260.904	133.174	207.022	119.729	268.345
	Voro++	-	-	-	-	-	-	-	-	-
	CGAL	20.445	9.591	9.366	13.522	19.205	9.781	15.016	8.877	19.559
	CGAL Parallel	26.986	18.900	97.873	21.249	20.066	57.639	49.721	40.528	40.466
	Geogram	1.593	0.826	0.859	1.244	1.732	0.865	1.370	0.812	1.560
	HXT GmSH	1.243	0.730	0.720	0.928	1.299	0.821	0.980	0.675	1.375
5090	Basselin*	-	-	-	-	-	-	-	-	-
	RF Del	124.929	25.299	24.980	24.201	40.413	24.762	93.342	16.889	44.777
	gDel3D	1.204	0.570	0.592	0.826	1.168	0.604	0.931	0.542	1.197
	Ours	0.570	0.245	0.248	0.331	0.496	0.235	0.359	0.240	0.491
H200	Basselin*	-	-	-	-	-	-	-	-	-
	RF Del	250.688	22.501	24.800	25.142	39.551	25.851	122.038	18.020	43.680
	gDel3D	0.893	0.428	0.444	0.620	0.862	0.462	0.678	0.424	0.896
	Ours	0.684	0.298	0.256	0.355	0.574	0.297	0.371	0.293	0.564

Generalization to power diagrams. As our method generalizes to power diagrams, we validate performance on the weighted version of our datasets, comparing against CPU-based algorithms for Regular Delaunay triangulation from SciPy, CGAL and Geogram. We also include results from our implementation of [Basselin et al. 2021] on the white noise and density gradient cases; however, their KNN-based search cannot produce valid results for the remaining distributions. Neither HXT, gDel3D nor the Radiant Foam implementation supports the weighted case. We present results from the synthetic data in Fig. 4 and real-world data in Tab. 2. The results demonstrate that our method generalizes well to the weighted case across all datasets, achieving similar runtimes and scaling behavior as for the unweighted case. We provide further results on how our method scales with varying weight distributions in the supplementary material.

5.3 Scaling mesh-based neural rendering

To demonstrate the practical benefits of efficient diagram construction at scale, we study the scaling behavior of a recent mesh-based neural rendering model, Radiant Foam [Govindarajan et al. 2025]. Radiant Foam represents scenes using differentiable 3D Voronoi diagrams and therefore requires frequent updates of cell adjacencies via Delaunay triangulation as the sites move during optimization. While the authors show that reconstruction quality improves consistently with an increasing number of points, indicating that higher geometric resolution leads to better reconstructions, their evaluation is limited to 2M points.

In this context, the Voronoi diagram serves as a scene's spatial discretization. Each cell stores learnt parameters for density and appearance effects. Volumetric rendering is performed by casting rays through the diagram and accumulating the per-cell contributions, producing a final color for the pixel. During training, site positions and each site's attributes are optimized via gradient descent, which demands the mesh to be recomputed exhaustively, to reflect the changing geometry. The repeated reconstruction of the mesh is the computational bottleneck of scaling our method addresses. While Radiant Foam uses incremental updates when possible, we benchmark full reconstruction for a controlled comparison.

Table 2. **Power diagram creation/regular Delaunay triangulation**. Runtime results (in seconds) on point sets obtained from Radiant Foam checkpoints trained on the Mip-NeRF 360 dataset. Color coding denotes fastest, second fastest, and third fastest.

		bicycle 4.2M	bonsai 2.0M	counter 2.0M	drjohnson 2.8M	garden 4.1M	kitchen 2.0M	playroom 3.2M	room 1.9M	stump 4.2M
CPU	SciPy	212.538	94.919	117.140	174.597	230.981	124.567	188.740	110.533	214.884
	CGAL	16.000	7.754	8.606	12.016	17.151	8.931	13.742	8.033	16.669
	CGAL Parallel	-	110.926	141.335	49.209	269.222	118.085	174.701	63.181	154.698
	Geogram	2.939	1.368	1.444	1.776	2.670	1.379	2.666	1.321	3.206
5090	Basselin*	-	-	-	-	-	-	-	-	-
	Ours	1.134	0.319	0.266	0.346	0.613	0.248	0.385	0.264	0.836
H200	Basselin*	-	-	-	-	-	-	-	-	-
	Ours	1.697	0.827	0.414	0.419	1.038	0.465	0.462	0.344	1.280

To enable Radiant Foam to scale beyond this regime, we replace their Delaunay triangulation step with our algorithm and retrain the model on scenes from the Mip-NeRF 360 dataset. We vary the final number of Voronoi sites from 3M to 17M, making no other changes to the method, training procedure, learning rate, or hyperparameters.

Fig. 1 reports the training time of Radiant Foam as a function of the number of points, along with LPIPS [Zhang et al. 2018] scores measured on the validation set as a function of training time for different final resolutions. The results highlight two key observations. First, mesh-based neural rendering models exhibit familiar scaling behavior, with reconstruction quality improving as the number of geometric primitives increases. Second, our approach enables such scaling with near-linear complexity, making large-scale Voronoi-based scene representations practically feasible.

6 Conclusion

We presented a GPU-based algorithm for efficiently and robustly constructing 3D Voronoi and power diagrams that supports large-scale point sets with complex spatial distributions while remaining competitive with state-of-the-art methods on small to moderate problem sizes.

Our approach is based on convex cell clipping, which naturally maps to massively parallel GPU execution. Combined with a culling criterion based on directional geometric bounds and a hierarchical best-first search strategy, this design yields an efficient and general algorithm that scales robustly with the number of points and across diverse spatial distributions.

We evaluated our method on both synthetic and real-world datasets, demonstrating performance comparable to or exceeding prior work on Delaunay triangulation across a wide range of distributions and scales. In several scenarios, our method achieves the best reported performance and naturally extends to the weighted case of power diagram construction.

Finally, we demonstrated the practical impact of our approach in the context of large-scale mesh-based neural rendering, where efficient and scalable construction of 3D Voronoi diagrams directly translates into improved reconstruction quality. These results underscore the importance of scalable Voronoi and power diagram

algorithms as foundational tools for emerging large-scale graphics and vision applications.

In summary, we introduced a scalable and general method for 3D Voronoi and power diagram construction that matches or exceeds prior Delaunay-based approaches and enables efficient processing of substantially larger point sets.

Limitations and Future Work. Our current implementation targets single-GPU execution and static point sets, and its scalability is ultimately bounded by available device memory rather than computation. Supporting incremental updates to diagrams, multi-GPU execution, and distributed construction remains an important direction for future work. In addition, our method relies on floating-point arithmetic and does not target the exact geometric guarantees of CPU-based libraries. Finally, further study of extreme weight distributions could provide additional insight into the limits of our culling strategies.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Computational resources were provided by NAISS at NSC Berzelius, partially funded by the Swedish Research Council, grant agreement no. 2022-06725.

References

Franz Aurenhammer. 1987. Power diagrams: properties, algorithms and applications. *SIAM journal on computing* 16, 1 (1987), 78–96.

Franz Aurenhammer. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM computing surveys (CSUR)* 23, 3 (1991), 345–405.

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. 1996. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* 22, 4 (Dec. 1996), 469–483. doi:10.1145/235815.235821

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. 2022. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5470–5479.

Justine Basselin, Laurent Alonso, Nicolas Ray, Dmitry Sokolov, Sylvain Lefebvre, and Bruno Lévy. 2021. Restricted Power Diagrams on the GPU. *Computer Graphics Forum* (2021). doi:10.1111/cgf.142610

A. Bowyer. 1981. Computing Dirichlet tessellations*. *Comput. J.* 24, 2 (01 1981), 162–166. arXiv:https://academic.oup.com/comjnl/article-pdf/24/2/162/967239/240162.pdf doi:10.1093/comjnl/24.2.162

Tyson Brochu, Christopher Batty, and Robert Bridson. 2010. Matching fluid simulation elements to surface geometry and topology. In *ACM SIGGRAPH 2010 papers*. 1–9.

Thanh-Tung Cao, Ashwin Nanjappa, Mingcen Gao, and Tiow-Seng Tan. 2014. A GPU accelerated algorithm for 3D Delaunay triangulation. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 47–54.

Jaehoon Choi, Yonghan Lee, Hyungtae Lee, Heesung Kwon, and Dinesh Manocha. 2024. Meshgs: Adaptive mesh-aligned gaussian splatting for high-quality rendering. In *Proceedings of the Asian Conference on Computer Vision*. 3310–3326.

Nikos Chrisochoides and Démian Nave. 2003. Parallel Delaunay mesh generation kernel. *Internat. J. Numer. Methods Engrg.* 58, 2 (2003), 161–176. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.765 doi:10.1002/nme.765

Lin Gao, Jie Yang, Bo-tao Zhang, Jia-mu Sun, Yu-jie Yuan, Hongbo Fu, and Yu-kun Lai. 2024. Real-time large-scale deformation of gaussian splatting. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–17.

Shrisudhan Govindarajan, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. 2025. Radiant Foam: Real-Time Differentiable Ray Tracing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 4135–4145.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. 2018. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)* 37, 6 (2018), 1–15.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Bruno Lévy and Nicolas Bonneel. 2013. Variational anisotropic surface meshing with Voronoi parallel linear enumeration. In *Proceedings of the 21st international meshing roundtable*. Springer, 349–366.

Ancheng Lin, Yusheng Xiang, Paul Kennedy, and Jun Li. 2024. Direct learning of mesh and appearance via 3d gaussian splatting. *arXiv preprint arXiv:2405.06945* (2024).

Jiayin Lu, Emanuel A Lazar, and Chris H Rycroftz. 2023. An extension to Voro++ for multithreaded computation of Voronoi cells. *Computer Physics Communications* 291 (2023), 108832.

Bruno Lévy. 2025. Geogram: A Programming Library with Geometric Algorithms. https://github.com/BrunoLevy/geogram. Accessed: 2025.

Alexander Mai, Trevor Hedstrom, George Kopanas, Janne Kontkanen, Falko Kuester, and Jonathan T. Barron. 2025. Radiance Meshes for Volumetric Reconstruction. arXiv:2512.04076 [cs.GR] https://arxiv.org/abs/2512.04076

Célestin Marot, Jeanne Pellerin, and Jean-François Remacle. 2019. One machine, one minute, three billion tetrahedra. *Internat. J. Numer. Methods Engrg.* 117, 9 (2019), 967–990.

Célestin Marot and Jean-François Remacle. 2020. Quality tetrahedral mesh generation with HXT. *arXiv preprint arXiv:2008.08508* (2020).

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

Nicolas Ray, Dmitry Sokolov, Sylvain Lefebvre, and Bruno Lévy. 2018. Meshless voronoi on the GPU. *ACM Trans. Graph.* 37, 6, Article 265 (Dec. 2018), 12 pages. doi:10.1145/3272127.3275092

Chris Rycroft. 2009. VORO++: A three-dimensional Voronoi cell library in C++. (2009).

Maxime Sainlot, Vincent Nivoliers, and Dominique Attali. 2017. Restricting Voronoi diagrams to meshes using corner validation. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 81–91.

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jonathan Richard Shewchuk. 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry* 22, 1-3 (2002), 21–74.

The CGAL Project. 2025. *CGAL User and Reference Manual* (6.1 ed.). CGAL Editorial Board. https://doc.cgal.org/6.1/Manual/packages.html

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. doi:10.1038/s41592-019-0686-2

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689* (2021).

D. F. Watson. 1981. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*. *Comput. J.* 24, 2 (01 1981), 167–172. arXiv:https://academic.oup.com/comjnl/article-pdf/24/2/167/967258/240167.pdf doi:10.1093/comjnl/24.2.167

Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume rendering of neural implicit surfaces. *Advances in neural information processing systems* 34 (2021), 4805–4815.

Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. 2023. Bakedsdf: Meshing neural sdfs for real-time view synthesis. In *ACM SIGGRAPH 2023 conference proceedings*. 1–9.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

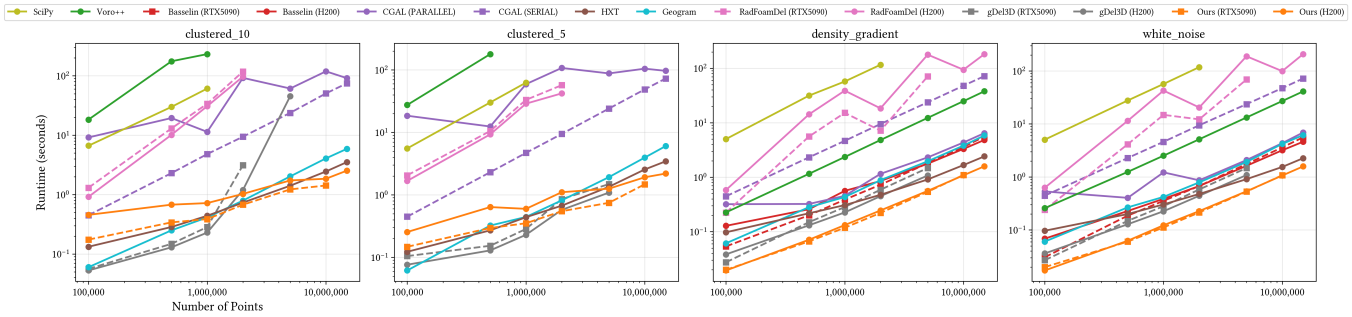


Fig. 3. Runtime results (in seconds) for Voronoi diagram creation/Delaunay triangulation on the synthetic generated point sets described in Sec. 5.1. Missing data points indicate that the method could not finish within the maximum allowed time limit of 300 seconds.

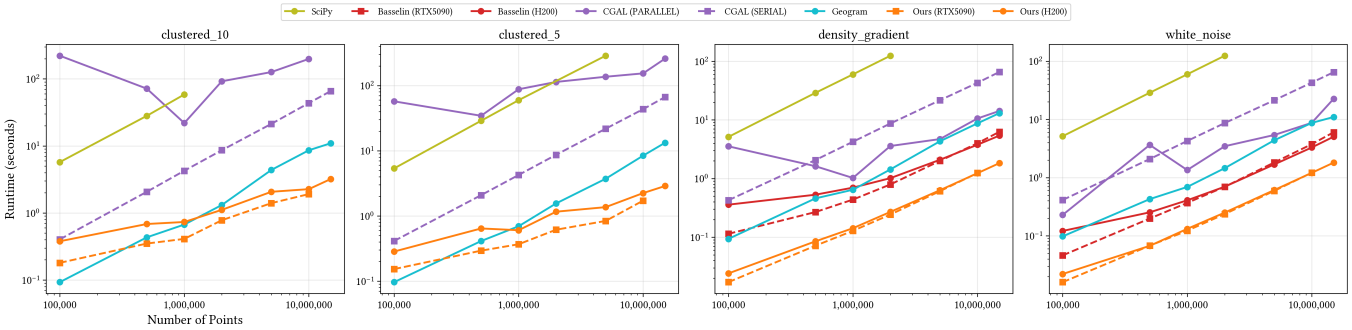


Fig. 4. Runtime results (in seconds) for power diagram creation/regular Delaunay triangulation on the weighted synthetic generated point sets described in Sec. 5.1. Missing data points indicate that the method could not finish within the maximum allowed time limit of 300 seconds.

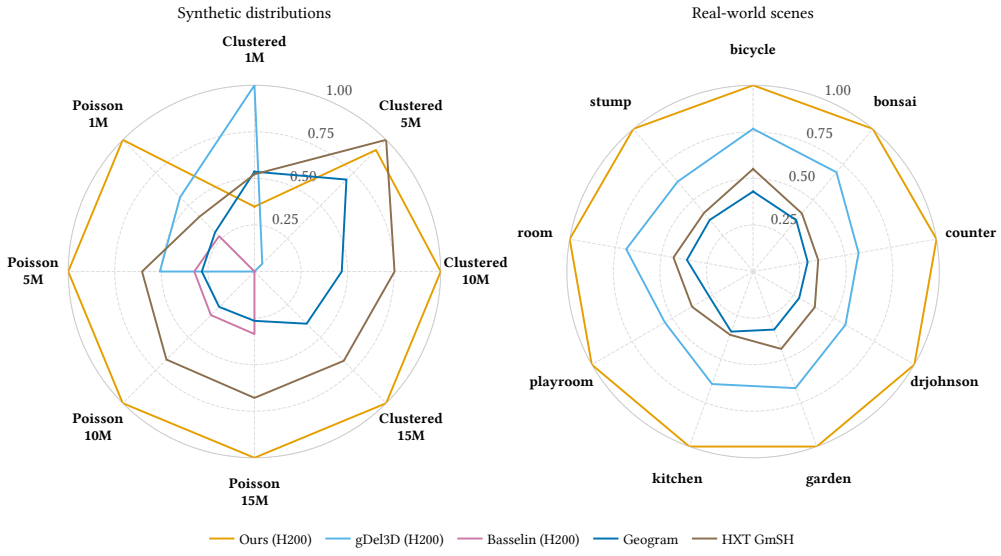


Fig. 5. Runtime comparison on synthetic and real-world point sets. The outer ring corresponds to the fastest method on that dataset (score = 1). A score of zero indicates that the method failed to complete a valid result. *Poisson* represents the average of the Poisson-distributed test cases (white noise and density gradient). *Clustered* represents the average of the clustered test cases. Our method consistently achieves scores near the outer ring across all distributions, while competitors degrade at higher point counts and non-uniform distributions.

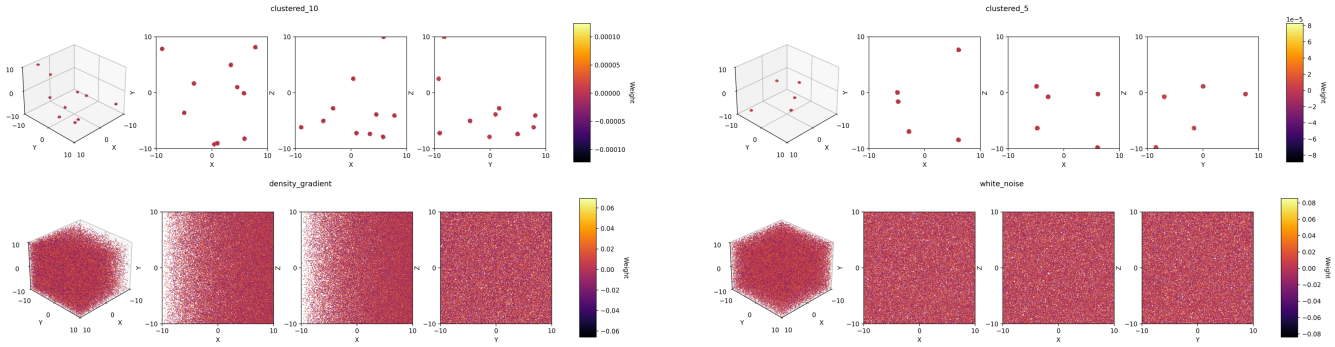


Fig. 6. Visualization of synthetic test cases.

Fig. 7. Images from the real-world test cases (described in Sec. 5.1), with point sets projected to illustrate the density and spatial spread of the data. The scenes span a range of indoor and outdoor environments with complex spatial distributions and point counts ranging from 2M to 4.2M.