In Search of Empty Spheres: 3D Apollonius Diagrams on GPU

CYPRIEN PLATEAU–HOLLEVILLE, Université de Limoges, XLIM, France
BENJAMIN STAMM, Universität Stuttgart, Institute of Applied Analysis and Numerical Simulation, Germany
VINCENT NIVOLIERS, Université Claude Bernard Lyon 1, LIRIS, France
MAXIME MARIA, Université de Limoges, XLIM, France
STÉPHANE MÉRILLOU, Université de Limoges, XLIM, France

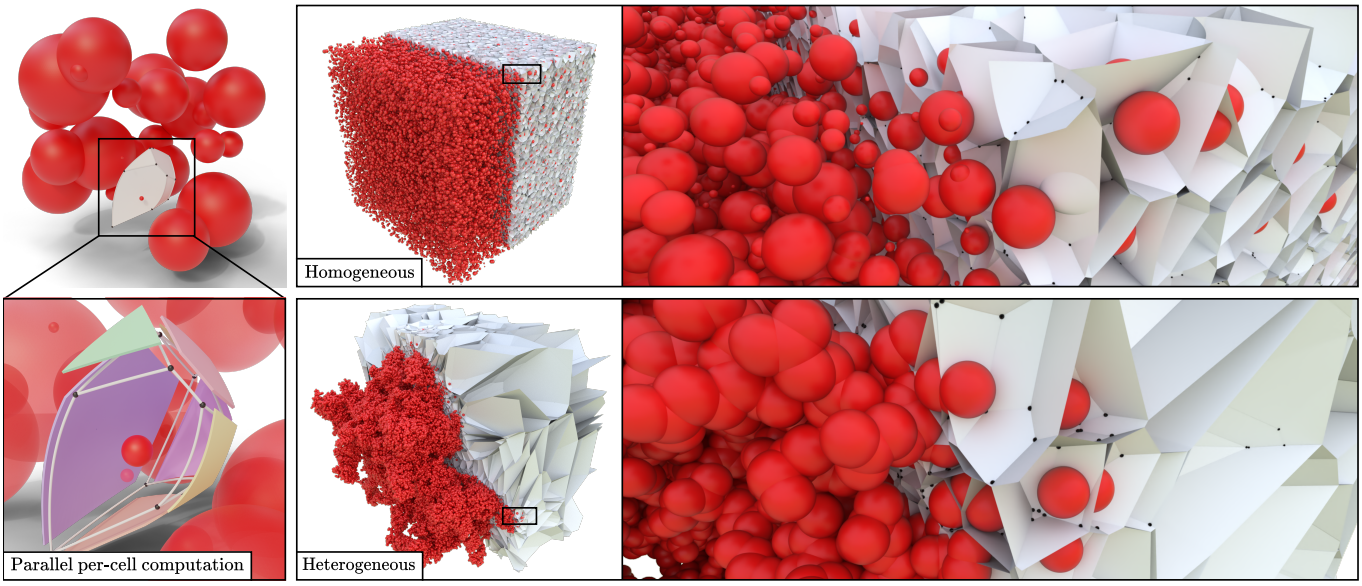


Fig. 1. We propose a method to compute an Apollonius diagram on GPU. Left: The computation is performed from the geometry of its cells to allow high parallelism level. Our method handles both homogeneous and heterogeneous spatial distributions supporting a wide range of applications. Top right: Homogeneous distribution of 100000 sites with radii between 0.1 and 10.1 computed in 8.1s. Bottom right: Heterogeneous distribution of 211834 sites with radii between 1.5 and 2.2 computed in 1.5s (PDB Id: 6RXU). Performances are given for an NVIDIA RTX 4090.

We present a novel comprehensive construction algorithm of Apollonius diagrams designed for GPUs. Efficient and robust algorithms have been proposed for the computation of Voronoi diagrams or Power diagrams. In contrast, Apollonius cells are neither convex nor bounded by straight boundaries, making their computation complex, especially in more than two dimensions. Their parallel computation also represents a challenge because of the sequential nature of state-of-the-art algorithms. In this article, we tackle the computation of these diagrams from the geometry of their cells. Our strategy is based on a core cell topology update allowing the iterative insertion of new sites found through nearest neighbor queries. To benefit from the highly parallel environment of modern GPUs and fit their memory restriction, we define a lightweight data structure allowing the representation of the complex topology of Apollonius cells. Additionally, we provide several space exploration procedures for their efficient construction under both homogeneous and heterogeneous spatial distributions. Our method outperforms the fastest state-of-the-art CPU implementation while computing the complete geometry. As a possible use case, we show an application for molecular illustration.

CCS Concepts: • **Computing methodologies** → **Massively parallel algorithms**; • **Theory of computation** → **Computational geometry**.

Additional Key Words and Phrases: Apollonius Diagram, Voronoi Diagram, GPU

ACM Reference Format:
Cyprien Plateau–Holleville, Benjamin Stamm, Vincent Nivoliers, Maxime Maria, and Stéphane Mérillou. 2025. In Search of Empty Spheres: 3D Apollonius Diagrams on GPU. *ACM Trans. Graph.* 44, 4 (August 2025), 15 pages. https://doi.org/10.1145/3730868

Authors' addresses: Cyprien Plateau–Holleville, cyprien.plateauholleville@unilim.fr, Université de Limoges, XLIM, Limoges, France; Benjamin Stamm, benjamin.stamm@ians.uni-stuttgart.de, Universität Stuttgart, Institute of Applied Analysis and Numerical Simulation, Stuttgart, Germany; Vincent Nivoliers, vincent.nivoliers@univ-lyon1.fr, Université Claude Bernard Lyon 1, LIRIS, Lyon, France; Maxime Maria, maxime.maria@unilim.fr, Université de Limoges, XLIM, Limoges, France; Stéphane Mérillou, stephane.merillou@unilim.fr, Université de Limoges, XLIM, Limoges, France.

1 INTRODUCTION

Space partitions are fundamental structures at the core of various data analysis algorithms. Once data is numerically encoded, querying attributes quickly turns into a geometric problem. Voronoi

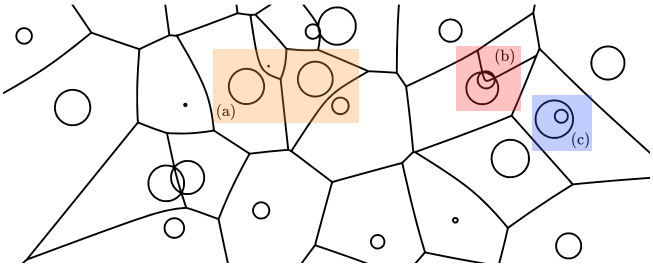


Fig. 2. Apollonius diagram in $\mathbb{R}^2$. The curvature of bisectors depends on the difference between the sites radii (a) and their distance (b). When a site is contained into another one, it does not contribute to the diagram (c).

diagrams and generalizations [Aurenhammer 1991] with arbitrary distances classically appear when dealing with discretized continuous domains as a set of sites. These diagrams define a simple mapping between the domain and the sites based on proximity according to a specific distance. They naturally appear as the solution of optimal transport problems [Bonneel and Digne 2023; Geiß et al. 2012], but can also be used for fluid simulations [Brochu et al. 2010] or chemistry [Manak et al. 2016].

The Apollonius diagram, or additively weighted Voronoi diagram, is defined by the additively weighted Euclidean distance

$$\delta(s_i, x) = ||p_i - x|| - r_i, \tag{1}$$

with $s_i = (p_i, r_i) \in \mathbb{R}^d \times \mathbb{R}$ a weighted site represented by a position p_i and a weight r_i. Apollonius diagrams describe the Euclidean distance between weighted sites and give a representation of the empty space within the input set. These characteristics come at the cost of a harder construction than affine diagrams, such as Voronoi and Power diagrams, due to curved facets (Fig. 2) and disconnected skeletons in $\mathbb{R}^3$.

Apollonius diagrams are used in a wide range of applications in various fields of research. Notably, they are fundamental when an actual distance to the sphere is required which cannot be replaced with another distance leading to an affine diagram. In $\mathbb{R}^2$, they have been used in material science [Pivovarov et al. 2018], route planning [Sharifzadeh and Shahabi 2007], geography [Moreno-Regidor et al. 2012], economy [Lanzara and Santacesaria 2023], or music theory [McLean et al. 2007]. In $\mathbb{R}^3$, several applications are relying on their properties, for sphere packing [Weller and Zachmann 2010] or studies where spheres are used as simple proxies approximating more complex shapes, notably in biochemistry [Krone et al. 2016; Lindow et al. 2013; Manak 2019]. Such works challenge available methods in terms of computation times due to the increasing input size which can reach millions of spheres. Moreover, biochemical structures as proteins are often characterized by heterogeneous spatial distributions (Fig. 1) involving more complex construction due to anisotropic cell shapes and bisectors between distant sites.

Due to their intrinsic complexity, Apollonius diagrams have received less attention than Voronoi and Power diagrams. Even if some two-dimensional robust algorithms [Karavelas and Yvinec 2002] and implementations [The CGAL Project 2023] are available, we are not aware of implementation in higher dimensions offering similar robustness.

We present in this article a comprehensive construction of the three-dimensional Apollonius diagram through a *meshless* approach [Ray et al. 2019] allowing an implementation on GPU by processing each cell in parallel. Our method is based on nearest neighbors queries with the following contributions:

- we present a method for the *update of the topology* of an Apollonius cell allowing its iterative construction;
- we provide several search procedures for the construction of *spatially homogenous and heterogenous distributions* of sites;
- we propose a *data structure with a low memory footprint* fitting GPU restrictions.

Despite mainly targeting comprehensive computation, our algorithm features faster execution than existing methods in $\mathbb{R}^3$. It also remains efficient with heterogeneous spatial distributions, difficult to handle in a cell-oriented setting [Basselin et al. 2021; Ray et al. 2019]. In terms of robustness, our study does not include exact predicates and can be subject to numerical inaccuracies. Therefore, we use an explicit random perturbation of the input set to ensure general position (*i.e.*, no more than four cospherical sites, etc.), as classically performed [Barber et al. 1996]. In contrast with the topology of the diagram, which requires a robust computation, the handling of general positions only have a minor impact on the accuracy of its geometry. Nevertheless, it is at the basis of various applications, targeting structural protein analysis, optimization or numerical simulations. Our reference implementation can be found at https://github.com/PlathC/apo.git.

2 BACKGROUND

First, we recall the characteristics of Apollonius diagrams in $\mathbb{R}^3$. Table 1 provides the nomenclature and Fig. 3 gives an illustration of such a diagram.

Let $S = \{s_1, s_2, \ldots, s_n\}$, be a set of weighted sites where each $s_i = (p_i, r_i)$ can be seen as a sphere of radius r_i centered at p_i. Given any point $x \in \mathbb{R}^3$, the distance between x and the sphere characterized by s_i is given by $\delta(s_i, x)$ (1). We define *nearest*, yielding

Symbol	Signification
δ	the additively weighted Euclidean distance.
x	a point.
p_i, r_i, s_i, S	a site, a weight, a weighted site and a set of weighted sites.
$\mathcal{A}(S), \mathcal{A}(s), \mathcal{A}^t(s)$	an Apollonius diagram, a cell, and a cell built from a subset of t neighbors of S.
$\mathcal{A}(\sigma)$	a set of points whose set of nearest neighbors sites is σ.
H_{ij}	a bisector or facet.
$e_{ijk}, \mathcal{E}$	a trisector or an edge and a set of edges.
$v_{ijkl}, \mathcal{V}$	a quadrisector or a vertex and a set of vertices.
$\tau(x)$	the largest open ball centered at x with an empty intersection with the set of sites.

Table 1. Nomenclature

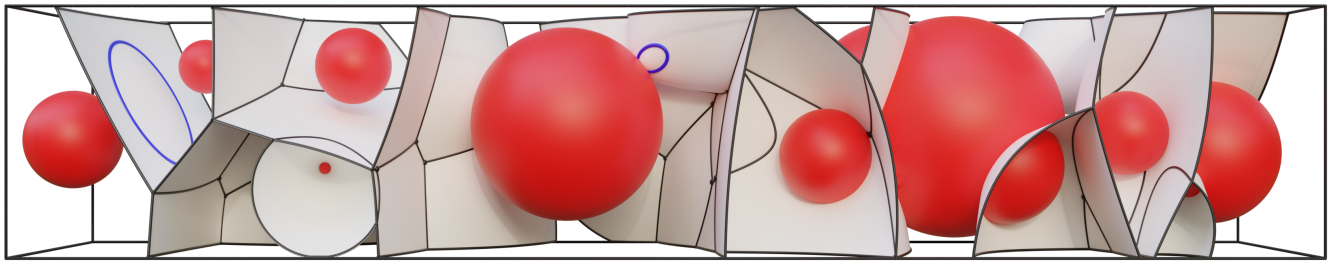


Fig. 3. A set of weighted sites (in red) and their corresponding Apollonius diagram in $\mathbb{R}^3$. Facets can be curved and the skeleton disconnected due to the additively weighted distance. The curviness of facets strongly depends on the distance between the involved sites and their radii. Some edges are closed (in blue) and have no vertex.

the set of sites for which the distance to x is minimal

$$nearest(x) \coloneqq \arg\min_{s_i \in S} \delta(s_i, x). \tag{2}$$

The combinatorics of the Apollonius diagram can then be simply defined. Let $\sigma \subset S$, a subset of sites. Its Apollonius dual is given by

$$\mathcal{A}(\sigma) \coloneqq \left\{ x \in \mathbb{R}^3 \;\middle|\; \sigma \subseteq nearest(x) \right\}.$$

In particular, $\mathcal{A}(\{s_i\})$ characterizes the entire cell of s_i, $\mathcal{A}(\{s_i, s_j\})$ a bisector between s_i and s_j and so on. In this article, we refer to the cell $\mathcal{A}(\{s_i\})$ with $\mathcal{A}(s_i)$ for conciseness. In contrast with Voronoi and Power cells, Apollonius cells have curved boundaries. They are therefore more difficult to compute than affine cells. Notably, they are not always convex but only star-shaped around their site and their edges are not even guaranteed to be connected (Fig. 3).

Bisector. An Apollonius bisector H_{ij} defined by $\mathcal{A}(\{s_i, s_j\})$, is a single sheet of a two-sheeted hyperboloid of revolution [Wang et al. 2020] when radii are different: its foci being the two sites centers and the sheet being the one curved around the site with the smaller radius. With equal radii, both sheets degenerate into a plane which is the regular Euclidean bisector of the sites centers.

Edge. An Apollonius edge e_{ijk}, characterized by $\mathcal{A}(\{s_i, s_j, s_k\})$, is the intersection of three bisectors. Only two of these bisectors are needed to define the edge. The result is a subset of a conic lying on a plane, which is a hyperbola when none of the sites is strictly in the convex hull of $\{s_i, s_j, s_k\}$, or an ellipse otherwise [Will, H.-M. 1999]. The conic degenerates into a line when sites radii are equal, or into a parabola as a limit case between hyperbola and ellipse. The subset of this conic corresponding to the trisector is not guaranteed to be connected. While hyperbolas are unbounded without limiting vertices, ellipses are always closed and bounded. In the following parts of this paper, we refer to edges shaped as hyperbolas and ellipses respectively as hyperbolic and elliptic edges.

Vertex. An Apollonius vertex v_{ijkl}, defined by $\mathcal{A}(\{s_i, s_j, s_k, s_l\})$, is the intersection of an edge e_{ijk} and a bisector H_{il} (or any other permutation of i, j, k and l). When not empty, this intersection is composed of one or two points [Gavrilova and Rokne 2003].

The set $nearest(x)$ (2) can be interpreted in terms of $\tau(x)$, the largest open ball centered at x whose intersection with S is empty. Its boundary sphere is then tangential to its nearest sites. This yields to a characterization of Apollonius diagrams similar to classical Voronoi diagrams: $\mathcal{A}(\sigma)$ is a part of the diagram if and only if there exists a sphere tangent to σ and centered at $x \in \mathcal{A}(\sigma)$ whose corresponding open ball $\tau(x \in \mathcal{A}(\sigma))$ does not intersect S.

3 PREVIOUS WORKS

In this section, we present the related works targeting Voronoi diagrams and generalizations. We start with Apollonius diagrams fundamental studies which are followed by their main construction algorithms in $\mathbb{R}^3$. Finally, we present several related works on parallel computation methods, not restricted to Apollonius diagrams.

Apollonius diagrams fundamental studies. Several theoretical studies have laid the foundation of Apollonius diagrams computation algorithms. In a survey on Power diagrams, Aurenhammer has presented several links between Apollonius diagrams in $\mathbb{R}^d$ and Power diagrams in $\mathbb{R}^{d+1}$ [Aurenhammer 1987]. Another mathematical framework for the study of Apollonius diagrams components have been proposed by Gavrilova and Rokne in their work on the update of a Euclidean Delaunay tessellation [Gavrilova and Rokne 2003]. Following the developments made for the robust computation of $\mathbb{R}^2$ Apollonius diagrams [Emiris and Karavelas 2006], Kamarianakis has presented a mathematical analysis of the predicates required for Apollonius diagrams in $\mathbb{R}^3$ [Kamarianakis 2020].

Construction of Apollonius diagrams in $\mathbb{R}^3$. Deriving the results presented by Aurenhammer [Aurenhammer 1987], Boissonnat et al. construct a single Apollonius cell in $\mathbb{R}^d$ from a corresponding Power cell in $\mathbb{R}^{d+1}$ [Boissonnat et al. 2006]. This linearization allows an elegant construction of Apollonius diagrams in $\mathbb{R}^3$ which is, however, limited. Notably, the construction of a Power diagram in $\mathbb{R}^4$ is itself computationally expensive due to the exponential number of vertices. Edge-Tracing [Kim et al. 2004] is a popular algorithm, notably for its simplicity, targeting the connected Apollonius skeleton. It starts with a valid edge and then iteratively computes the vertices from which other edges may be found. This algorithm offers an interesting ease of implementation but is considered non-robust, notably under disconnected cases, which occur with high radius

change. This method has been implemented in Voronota [Olechnovič and Venclovas 2014], a fast library targeting the computation of Apollonius diagrams in $\mathbb{R}^3$. In particular, it offers the best performance for the construction of connected skeletons of Apollonius diagrams. Edge-tracing has also been extended to support disconnected skeletons [Manak and Kolingerova 2016]. Apollonius diagrams can be constructed from the update of the Voronoi diagram of the sites centers, which is however sequential [Kim and Kim 2006]. Another method proposed a space refinement construction [Wang et al. 2020] which handles various configurations thanks to its sweeping scheme. However, it comes at the cost of significant computation times. Recently, Lee et al. have introduced a method focused on the topological robust construction [Lee et al. 2022]. Song et al. have provided a dataset for the evaluation of the robustness of Apollonius diagrams construction [Song et al. 2022]. Du et al. have presented an algorithm robustly handling the computation of implicit surface networks. It can then be leveraged to extract the topological structure of Apollonius diagrams [Du et al. 2022].

Parallel construction. Multiple strategies can be adopted for parallel construction of Voronoi diagrams and generalizations. Cell-oriented computations of Voronoi diagrams can be achieved by iteratively clipping cells with bisector planes [Rycroft 2009]. Additionally, it has also been used for Voronoi diagrams generalizations [Basselin et al. 2021; Bukenberger et al. 2022]. In contrast with other sequential algorithms [Bowyer 1981; Watson 1981], this allows a trivial parallelization through the individual processing of each cell. GPU implementations have been proposed by following this strategy for Voronoi diagrams [Ray et al. 2019] and Power diagrams [Basselin et al. 2021]. They allow the definition of an integration method well suited for optimal transport and simulation purposes. These strategies rely on the assumption that the set of cell neighbors is similar to the set of the nearest neighbors which is not compatible with spatially heterogeneous distribution. Other strategies have been proposed based on the sampling of Apollonius bisectors to individually build the cells structures. This is achieved by tracing rays on bisectors, which allows reconstructing edges and vertices. These algorithms can handle non-general position [Hu et al. 2017] or robust computation of non-intersecting sites [Mukundan et al. 2022]. However, they offer a lower level of performance compared to fast implementations of Edge-Tracing [Olechnovič and Venclovas 2014]. Finally, other strategies target fast computation from space discretization but their precision is strongly linked to their memory consumption [Rong and Tan 2006].

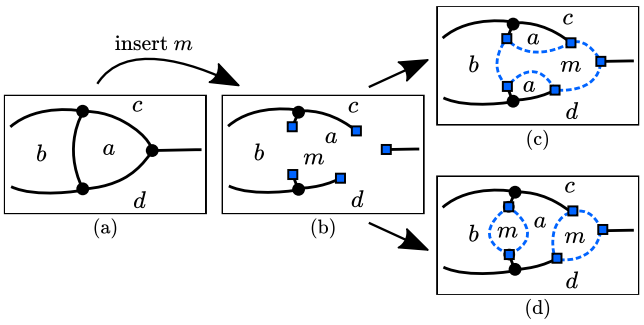


Fig. 4. An example of a topology update relatively to a cell. Facets are qualified by their corresponding neighbor index. The connection between two vertices through an edge is a geometrical information which depends on the sites configuration and cannot be recovered only from the combinatorial information. (a) Initial state of the cell. (b) Suppression of invalidated vertices and computation of new vertices due to the addition of s_m (as blue square nodes). (c, d) Two possible combinations of new edges (as blue dotted links).

Considering these previous works, the construction of Apollonius diagrams without assumption on their skeleton connectivity remains challenging, notably in terms of performance. To address this issue, we propose a novel GPU method handling these cases while offering completely parallel cells computation.

4 APOLLONIUS DIAGRAMS CONSTRUCTION

In this section, we present our comprehensive computation method of Apollonius cells. We start with an overview of the method (Section 4.1) which is followed by the presentation of our data structure (Section 4.2). Then, we present the initialization of the cells (Section 4.3), the topology update process (Section 4.4) and dedicated space exploration methods (Section 4.5). Finally, we give several implementation details for fast execution on GPU (Section 4.6).

4.1 Overview

Given a weighted site s_i and a set of weighted sites S, our method allows the construction of the cell $\mathcal{A}(s_i)$. Let s_m be a new neighbor of s_i in S. s_m is considered as *contributing* to $\mathcal{A}(s_i)$ if and only if the bisector H_{im} exists in $\mathcal{A}(s_i)$. Three main challenges must be addressed to achieve a comprehensive and efficient construction.

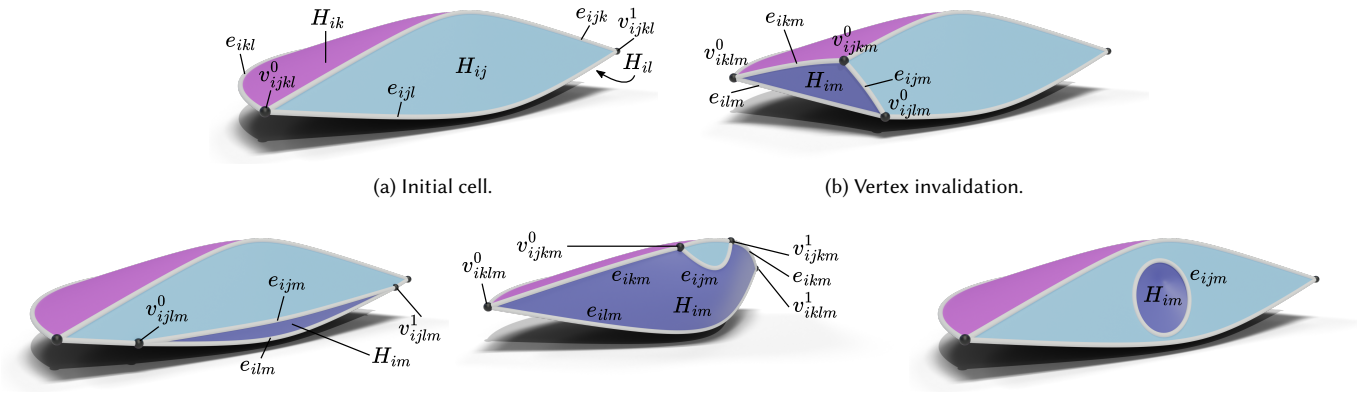
First, when a new bisector H_{im} is introduced in $\mathcal{A}(s_i)$, parts of the cell topology are *invalidated* and must be removed. Additionally, new parts must be inserted by accurately identifying the boundaries of H_{im} (Fig. 4). This is handled by our topology update method based on the systematic validation of new parts of the cell (Section 4.4).

Second, the construction of $\mathcal{A}(s_i)$ involves the search of the subset of weighted sites in S contributing to the cell. Previous works were restricted to spatially homogeneous distribution allowing the assumption that the set of contributing sites to a cell is equivalent to the set of its nearest neighbors [Basselin et al. 2021; Ray et al. 2019]. However, this assumption cannot hold under heterogeneous spatial distribution due to possible bisectors with distant sites. We then define a set of dedicated search methods to ensure that $\mathcal{A}(s_i)$ is correct without considering all possible sites in S (Section 4.5).

Finally, the computation of Apollonius diagrams requires high computational capabilities. Our method benefits from the massively parallel architecture of modern GPUs thanks to the cell-oriented construction. To fit this restricting environment, we define a dedicated data-structure encoding the diagram topology (Section 4.2) and provide several key implementation designs (Section 4.6).

4.2 Data structure

Our method relies on a compute-over-store approach to take into account a large number of sites while fitting the restricted memory of the GPU. This is achieved by storing minimal information and



(a) Initial cell. (b) Vertex invalidation.

(c) Edge intersection without invalidated vertex. (d) Edge intersection with an invalidated vertex. (e) Bisector intersection leading to a closed edge.

Fig. 5. Illustration of the cell $\mathcal{A}(s_i)$ and four configurations resulting from the insertion of a new neighbor s_m producing the bisector H_{im}. (a) The initial cell $\mathcal{A}(s_i)$ before inserting s_m. The initial structure is only named in (a) for conciseness. (b) s_m invalidates one vertex point and contributes to the three new vertices v_{iklm}, v_{ijkm} and v_{ijlm} and their corresponding edges. (c) H_{im} intersects e_{ijl} producing the new vertex v_{ijlm} composed of two points, and their corresponding edges. (d) s_m invalidates the vertex v_{ijkl} and contributes to the new vertices v_{iklm} and v_{ijkm}, both composed of two points, and their corresponding edges. (e) H_{im} intersects the bisector H_{ij} producing a new closed edge e_{ijm}. The presented topology update process, based on the detection of bisector and edge invalidation, handles all these cases.

relying on recomputation when needed. In this section, we present our data structure (Fig. 6) allowing to store the topology of a cell.

Vertices. We represent vertices by three sites indices. Benefiting from the cell-oriented computation, we do not explicitly store the first index of the quadruplet which is given by the cell index. Even if our method is compatible with the direct storage of the vertices, it requires the specific handling of the multiple solutions corresponding to a given vertex. Then, we store the points individually and differentiate them with an additional integer.

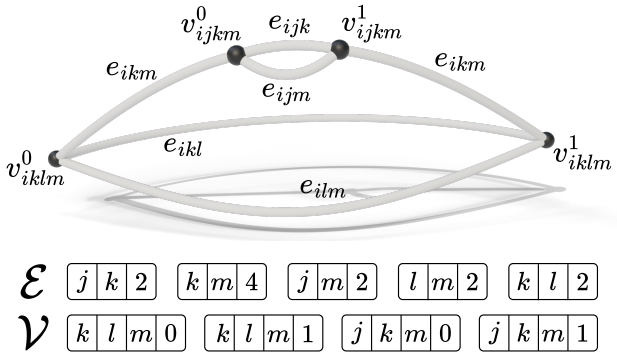


Fig. 6. Illustration of the data structure. We propose to store per-edge vertex number instead of an explicit edge vertex adjacency, both lighter and easier to maintain during the construction. For instance, the edge e_{ikl} is represented by the couple k, l, since i is implicitly given by the cell index, and 2, its number of vertices. We represent the two points v^0_{iklm} and v^1_{iklm} of the vertex v_{iklm} with the triplet k, l, m, and a solution index 0 or 1, allowing to differentiate them.

Edges. In affine diagrams, every edge of a given cell is incident to two vertices. The edges can then be directly deduced from their combinatorial structure. In Apollonius diagrams, edges may have none to an unbounded number of vertices. Therefore, we explicitly store edges using two indices representing the neighbors of the cell and an additional integer for the number of incident vertices.

Bisectors. Bisectors are always bounded by edges. Thus, they are not explicitly stored in the data structure. When required, they are fully recovered from edge data.

Our data structure is composed of a list of edges $\mathcal{E}$ and of vertices $\mathcal{V}$. It does not explicitly store their adjacency relations. Indeed, edges can be made of multiple components whose explicit storage can be complex to maintain (Fig. 4). In particular, this connectivity is not essential for our algorithm. Given the indices of an edge, the vertices can be scanned to filter those adjacent. In the following sections, we show how we can fully compute the topology of a single cell.

4.3 Initialization

The construction of any Apollonius diagrams requires the handling of unbounded edges, which would complexify our topology update and slow down the overall process. Since most applications use a limited domain, we start the construction of every cell with an initialization stage, bounding all their facets.

Previous works have shown that the set of unbounded edges is topologically equivalent to the convex hull of the input set [Manak and Kolingerova 2016]. We bound all cells by defining new artificial sites such that no original site contributes to the convex hull. In practice, we position artificial sites on the vertices of an octahedron. We first compute the radius r_{max} of the bounding sphere of the input set S and then introduce six new sites positioned on each

axis at a distance greater than $\sqrt{3}r_{max}$ from the sphere center. This ensures that the octahedron shaped by artificial sites contains all sites of the input set. We assign artificial sites radii to the minimum radius of the input set, thus reducing the curvature of their bisectors.

4.4 Topology update

Based on the presented data structure and initialization procedure, it remains to iteratively modify the cell topology. This sequential method allows the construction of each cell, in parallel.

While updating the topology of a cell due to the consideration of a new site $s_m \in S$, let $\mathcal{A}^t(s_i)$, the Apollonius cell of $s_i \in S$ built from the set of the t first considered neighbors $\mathcal{N}^t \subset S \setminus \{s_i\}$, we define $\mathcal{A}^{t+1}(s_i)$ as the cell of s_i considering $\mathcal{N}^{t+1} = \mathcal{N}^t \cup \{s_m\}$. This update represents one of the main challenges of Apollonius diagram construction since multiple cases must be taken into account (Fig. 5). These various possibilities also represent a strong performance bottleneck due to execution divergence on GPU. Thus, we present a method for the update of the cell after the insertion of a new neighbor. It is based on a key observation: a new neighbor s_m contributes to the cell $\mathcal{A}^t(s_i)$ if and only if there exists a sphere tangent to s_i and s_m whose corresponding open ball has an empty intersection with $\mathcal{N}^t$. Algorithm 1 illustrates the process.

Predicate. Our method is based on the predicate *valid* allowing to test if a part of the topology $\mathcal{A}(\sigma)$ exists according to a weighted site s_j. It is defined by

$$valid\left(\mathcal{A}(\sigma),\ s_j\right) \coloneqq \exists x \in \mathcal{A}(\sigma)\ s.t.\ \left(\tau(x) \cap s_j\right) = \emptyset,$$

with $\tau(x)$ as the largest open ball centered at x in $\mathcal{A}(\sigma)$ *s.t.* it does not intersect σ (Section 2). Through *valid*, we identify both the invalidation of existing topology and the creation of the new ones after the insertion of a new neighbor. App. A provides necessary equations for the implementation of the predicate.

Vertex invalidation. We start by testing if the new neighbor s_m invalidates a vertex (Fig. 5b). We find invalid vertices v_{ijkl} with $valid(v_{ijkl}, s_m)$ and decrement the vertex count of related edges (l.2-9 of Algorithm 1). Even if the vertex count of an edge becomes 0, they are not deleted yet since they can still contribute to the cell.

New vertices. In affine diagrams construction, edge invalidation is deduced from vertex invalidation thanks to cell convexity [Ray et al. 2019]. When a single vertex of an edge is invalidated, it leads to a new vertex and when two vertices of an edge are invalidated, it is also invalidated. In Apollonius diagrams, two other cases may occur. An edge can be intersected by the bisector of a new neighbor resulting in new vertices while its two original vertices are both valid (Fig. 5c) or invalid (Fig. 5d). Then, for all edges e_{ijk}, we start by computing the vertex v_{ijkm} resulting from their intersection with

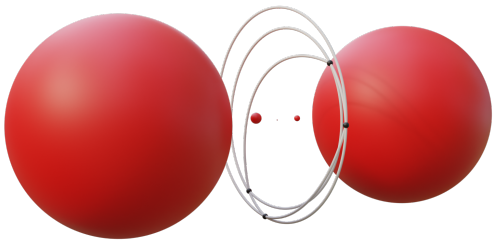


Fig. 7. A complex configuration of sites with intersections of elliptic edges.

Algorithm 1: Topology update

Data: A cell $\mathcal{A}^t(s_i)$ and a new site s_m
Result: The updated cell $\mathcal{A}^{t+1}(s_i)$

1 $\mathcal{E}, \mathcal{V} \leftarrow \mathcal{A}^t(s_i)$ ▷ Fetch cell data structure
2 **forall** $v_{ijkl} \in \mathcal{V}$ **do** ▷ **Vertex invalidation**
3 **if** $valid(v_{ijkl},\ s_m)$ **then**
4 **continue**
5 Tag vertex as invalid
6 **forall** $e_{ij'k'} \in \mathcal{E}$ **do**
  ▷ If v_{ijkl} is not on $e_{ij'k'}$
7 **if** $\{j', k'\} \notin \{j, k, l\}$ **then**
8 **continue**
9 Decrement the vertex count of $e_{ik'l'}$
10 $\mathcal{E}* \leftarrow \{\}$ ▷ Temporary pool of new edges
11 **forall** $e_{ijk} \in \mathcal{E}$ **do** ▷ **Computation of new vertices**
12 $v_{ijkm} \leftarrow \left(H_{im} \cap e_{ijk}\right)$
  ▷ Validate new vertex v_{ijkm}
13 **forall** $s_l \in \mathcal{N}^t \setminus \{s_j, s_k\}$ **do**
14 **if** $\neg valid(v_{ijkm},\ s_l)$ **then**
15 Tag v_{ijkm} as invalid
16 **break**
17 **if** v_{ijkm} is valid **then**
18 $\mathcal{E}* \leftarrow \mathcal{E}* \cup \{e_{ijm}, e_{ikm}\}$
19 $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_{ijkm}\}$
20 Increment the vertex count of e_{ijk}
  ▷ Test if the edge is fully invalidated
21 **if** e_{ijk} is not a closed elliptic trisector **then**
22 **continue**
23 **if** $H_{im} \cap e_{ijk} = \emptyset \wedge \neg valid(e_{ijk},\ s_m)$ **then**
24 $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e_{ijk}\}$
25 **forall** $s_j \in \mathcal{N}^t$ **do** ▷ **Closed edges creation**
26 $e_{ijm} \leftarrow H_{ij} \cap H_{im}$
27 **if** s_j can contribute to a vertex with s_m **then**
28 **continue**
29 **if** e_{ijm} is not an elliptic trisector **then**
30 **continue**
  ▷ Validate new edge e_{ijm}
31 **forall** $s_k \in \mathcal{N}^t \setminus \{s_j\}$ **do**
32 **if** $\neg valid(e_{ijm},\ s_k)$ **then**
33 e_{ijm} does not exist
34 **break**
35 **if** e_{ijm} exists **then**
36 Tag e_{ijm} as closed
37 $\mathcal{E}* \leftarrow \mathcal{E}* \cup \{e_{ijm}\}$
38 Emplace in $\mathcal{E}$ all unique values in $\mathcal{E}*$
39 Remove all edges with no vertex from $\mathcal{E}$

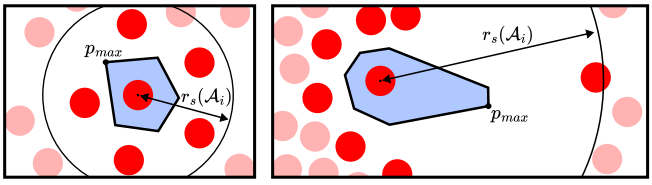
the bisector H_{im}. If the intersection of the open ball positioned on a new vertex $\tau(v_{ijkm})$ and all previously considered neighbors in $\mathcal{N}^t$ is empty, v_{ijkm} exists in $\mathcal{A}^{t+1}(s_i)$ (l.11-16 of Algorithm 1). We then append v_{ijkm} to the data structure and increment the vertex count of the edge e_{ijk} (l.17-20 of Algorithm 1). For every new vertex v_{ijkm}, two new edges e_{ijm} and e_{ikm} are also created in the cell. Since these new edges may be shared by multiple vertices, and thus created multiple times, we save them into a temporary pool $\mathcal{E}*$. When all edges are processed, we identify unique values in $\mathcal{E}*$ and count their corresponding vertices based on the number of occurrences.

Closed edges. Even if the new neighbor s_m does not produce any new vertex, it may still contribute to a new edge. One of the bisectors of the current cell can be intersected resulting in a closed edge (Fig. 5e). Such a case happens between s_i, one of its neighbors $s_j \in \mathcal{N}^t$ and s_m if and only if they satisfy two conditions: e_{ijm} is elliptic and no vertex is possible between e_{ijm} and any other site in $\mathcal{N}^t$ (l.27-30 of Algorithm 1). Then, if the intersection with an open ball $\tau(x \in e_{ijm})$ and $\mathcal{N}^t$ is empty, the edge exists, is closed, and can be added to the cell topology. In contrast, if a closed edge e_{ijk} has no possible vertices with s_m, it may be fully invalidated. Then, if an open ball $\tau(x \in e_{ijk})$ intersects s_m, the closed edge e_{ijk} can be removed from the data structure (l.21-24 of Algorithm 1).

After these steps, bounded edges without vertices are invalidated and deleted. Our method, based on the detection of edge intersections, allows a uniform processing of the Apollonius cells while handling complex structures (Fig. 7). Even if it requires a potentially high computational cost, it allows a simple formulation while limiting its memory footprint. It also makes enable the use of parallel cooperative processing, further discussed in Section 4.6.

4.5 Spatial exploration

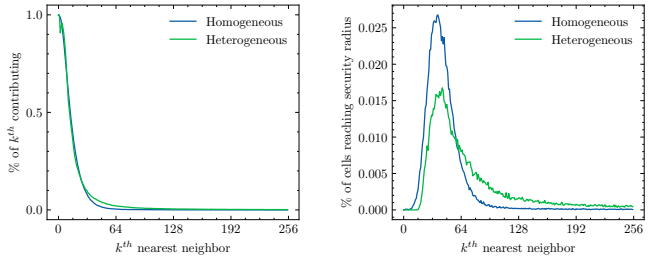
To update the topology efficiently, we need the subset of S containing all contributing neighbors. We present in this section a set of spatial exploration strategies identifying the subset of contributing neighbors. The goal is to reduce the number of non-contributing sites taken into account to limit the overall construction cost.



(a) Homogeneous distribution.

(b) Heterogeneous distribution.

Fig. 8. The security radius allows bounding the search for new contributing neighbors. (a) Under homogeneous spatial distribution, the set of the nearest neighbors corresponds to the set of contributing neighbors. (b) With heterogeneous distributions the radius becomes large and a lot of non-contributing neighbors must be considered. (Red: contributing neighbors, Faded red: noncontributing neighbors).



(a) k^{th} nearest neighbors contributing to a cell.

(b) Cells reaching security radius after the insertion of the k^{th} nearest neighbor.

Fig. 9. Study of the security radius and its assumptions. (a) A small number of the nearest neighbors actually contribute to the cell. After 32 considered nearest neighbors, few nearest neighbors are contributing, independently from the distribution. (b) Under a heterogeneous setting, cells require to consider more neighbors to satisfy the security radius than in a homogeneous setting. Some cell may need the complete input set to satisfy the security radius if their furthest neighbor is contributing.

k-Nearest Neighbors (kNN). Previous GPU cell-oriented construction methods are based on kNN queries [Basselin et al. 2021; Ray et al. 2019]. They make the assumption that the spatial distribution of sites is homogeneous, thus that the set of nearest neighbors is close to the combinatorial structure. To ensure that the cell is correct without considering all the input set, they rely on the iterative insertion of neighbors, sorted by distance from the current site. To limit the search, the security radius allows defining a local boundary from the cell geometry [Lévy and Bonneel 2013]. It can be seen as the sphere located at p_i containing all open balls $\tau(x \in \mathcal{A}^t(s_i))$. Let x^t_{max} the furthest point from s_i on the cell geometry is

$$x^t_{max} = \underset{x \in \mathcal{A}^t(s_i)}{\arg\max}\ \delta(s_i, x).$$

Then, the security radius r_s can be defined as

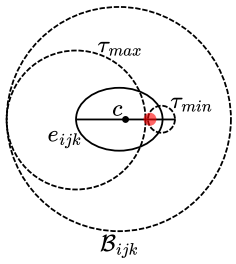
$$r_s(\mathcal{A}^t(s_i)) = 2\delta(s_i, x^t_{max}).$$

If the distance between s_i and the $t+1$ considered nearest neighbor is greater than $r_s(\mathcal{A}^t(s_i))$, all open balls positioned on the cell topology $\tau(x \in \mathcal{A}^t(s_i))$ are within the known radius. Then, the cell is valid according to the input set S and no more neighbor must be considered. In contrast with affine diagrams, the furthest point of an Apollonius cell is not always positioned on one of its vertices but can also be located on an edge. Notice that, while the maximum distance to the cell is always located on a vertex with bounded hyperbolic edges, it is not always true with elliptic edges. In such case, the theoretical maximum distance is on the intersection between the edge and the plane defined by the center of its three sites [Medvedev et al. 2006]. The security radius of an Apollonius cell can thus be estimated from its vertices and elliptic edges. This is, in practice, always considered as is to avoid this computation even if this part of the edge may not belong to the actual segment on the cell.

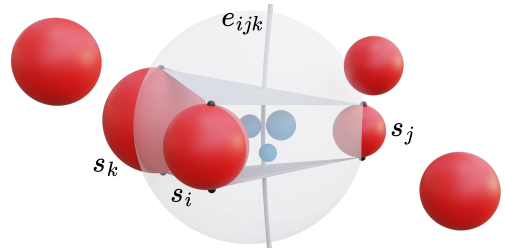
Local validation. While kNN-based strategy allows fast processing of uniform distribution of sites as well as elegant bounding

of the search from the security radius, it does not suit irregular spatial configuration (Fig. 8). The security radius then becomes very large and a lot of non-contributing sites must be taken into account (Fig. 9). Independently from the distribution, the number of the k^{th} nearest neighbors contributing to the cell is falling dramatically while k is increasing. As expected, under heterogeneous distributions of sites, cells reach the security radius less than under homogeneous distribution of sites, as expected. However, an affine cell can only be modified by a new neighbor if one of its vertices can be invalidated [Sainlot et al. 2017]. Then, we can directly validate every vertex by testing if their nearest neighbors in the input set are equivalent to their combinatorial structure, a process called corner validation. This strategy allows an efficient spatial discovery and limits the consideration of non-contributing sites based on the current state of the cell. While corner validation can directly be applied to Apollonius diagrams, it is not enough to ensure that the computed cell is valid. Notably, curved parts of the cell may fall out of the space covered by open balls centered on vertices, similarly to the security radius. Thus, we define two additional search procedures allowing the validation of other parts of the topology.

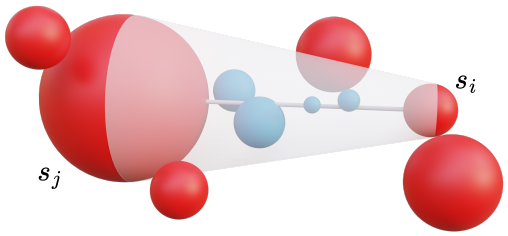
Apollonius edge validation. After vertex validation, parts of edges may not be validated. Considering first bounded hyperbolic edges (Section 4.3), the largest zone that has not been validated yet occurs when both vertices are positioned at infinity. In such case, their validated open balls are two half-spaces and the remaining zone is defined by a portion of a Dupin cyclid [Olechnovič and Venclovas 2014]. In contrast, no assumption could be made from the validation of elliptic edges without precisely identifying their set of vertices, making their validation more complex. To simplify the search, we use coarse bounding shapes. The hyperbolic portion of a Dupin cyclid is approximated from the two tangent planes to the triplet of sites, defining the half-spaces. We also use the sphere passing by four different points located on these planes and a site of the triplet (Fig. 10a). A yet unconsidered site can contribute if and only if it intersects the bounding sphere and does not intersect the half-spaces. Considering elliptic edges, we use a bounding sphere $\mathcal{B}_{ijk}$ fully including all possible open balls of the edge $\tau(x \in e_{ijk})$. Such sphere can be computed from both open balls of minimum τ_{min} and maximum τ_{max} radii on the edge, located on the intersection between the edge and the plane defined by the three sites' centers. Its center c is given by the ellipse center and its radius is the distance between c and τ_{max} plus the radius of τ_{max}. After edge validation, new vertices may have been found, thus vertex validation must be re-executed as long as new vertices are found during edge validation.



Apollonius bisector validation. Similarly to edges, bisectors may also contain non-validated part after previous validations. Again, the worst case would be an infinitely large elliptic edge bounding the current bisector. The remaining non-validated part is defined by the intersection of all half-spaces tangent to and including both



(a) Hyperbolic edge validation.



(b) Bisector validation.

Fig. 10. Illustration of the zones required for (a) hyperbolic and (b) bisector validation. In blue: Possibly unconsidered sites.

sites which is equivalent to their convex hull, a section of a cone (Fig. 10b). In practice, we use a cylinder of axis $(p_j - p_i)$ and of radius $\max(r_i, r_j)$, compatible with sites of similar radii.

Acceleration structure. Previous works use an acceleration grid to perform kNN queries on homogeneous spatial distribution [Basselin et al. 2021; Ray et al. 2019]. This kind of structure is often selected for their simple construction and fast performance, notably for fixed radius queries [Hoetzlein 2014]. To perform the validation steps, we may require large radius search leading to the processing of a substantial number of grid cells. Additionally, since sites can have very different radii, selecting a single cell size would result in inefficient processing across the input set. Thus, we use a BVH allowing precise refinement of the space while searching for contributing neighbors thanks to its hierarchy [Meister and Bittner 2022]. We benefit from its accurate bounding primitives, fitting high radii changes.

Our complete pipeline starts with the kNN search, which is followed by vertex, edge and bisector validations. All stages are based on the BVH traversal alleviating the construction cost.

4.6 Implementation details and GPU optimizations

Even if the presented process allows the computation of Apollonius diagrams, several optimizations are required to achieve fast execution. Our implementation uses CUDA but other platform supporting warp operations could be used. We rely on a set of specialized kernels using our core topology update (Section 4.4) and focusing on a single spatial exploration type (Section 4.5).

Pipeline configuration. Similarly to other pipelines [Basselin et al. 2021; Ray et al. 2019], our method relies on the maximum size reserved for the cell data structure (*e.g.*, maximum vertex or edge

number) which may significantly vary depending on the input configuration. We also allow selecting the number of nearest neighbors considered during the first step of the spatial exploration pipeline. Starting the construction of the cell with a small set of its nearest neighbors may lead to strong performance speed up (Fig. 11).

Cooperative processing. Computing each cell of the diagram requires a lot of tests on the combinatorial structure. Additionally, as previous works [Basselin et al. 2021; Ray et al. 2019], we rely on shared memory to store the cell data structure. However, Apollonius diagrams require an explicit storage of edge information inducing a higher cost per cell. This is notably emphasized by heterogeneous spatial distributions, leading to more complex cell topology than homogeneous distributions. In such context, our implementation uses a complete warp per cell. This allows to benefit from higher computation capabilities through cooperative processing and a more precise distribution of the workload. Each cell also benefits from more available shared memory, since fewer cells are stored at the same time. Additionally, their processing can rely on the use of warp intrinsics during the topology update for fast compaction and reduction. Several operations over the complete cell topology are performed by using all threads of the warp to benefit from a better parallelization of the computation, for instance the predicate *valid*.

Spatial exploration. For all spatial exploration queries, we use an LBVH [Karras 2012] for its trade-off between construction and query performance. kNN queries are performed using a max-heap structure [Ray et al. 2019]. Additionally, in order to avoid performing topology validations on noncontributing neighbors, we use multiple pre-validation steps, assuring that considered neighbors are contributing to the cell. This is notably made by approximating the bounding shapes described in Section 4.5 with bounding spheres allowing fast coarse intersection tests.

Precision. Since we target applications benefiting from the fast computation of the geometry of the diagram, we do not focus on its robustness in terms of floating point operations. However, in order to keep accurate results, all computations are performed with 64 bits floating point precision. This choice results in lower performance since GPUs commonly offer less dedicated 64 bits floating operations cores than 32 bits ones [NVIDIA 2023]. Applications willing to increase their performance at the cost of precision could thus switch to 32 bits floating point operations. In contrast, precision could be improved with robust geometric predicates or more advanced perturbation [Devillers et al. 2017]. In our experiments with our implementation, single precision numbers allowed significant gain, but led to multiple erroneous computation of validation zones.

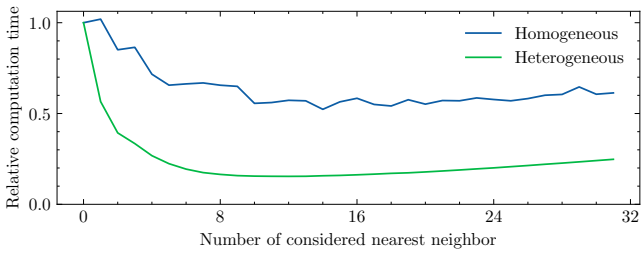


Fig. 11. Relative computation time of our method using increasing number of considered nearest neighbors before spatial validation on a site set with spatially homogeneous (10000 sites in box of size 500 and radii between 0.1 and 10) and heterogeneous (a protein of 58870 sites, PDB Id: 1AON) distributions. Performances are normalized according to the first test performed with no considered nearest neighbor.

5 EXPERIMENTS

In this section, we analyze our computation pipeline. All experiments have been executed on the same hardware: an Intel I9-13900K and an NVIDIA RTX 4090. We compare our method to the edge-tracing multi-core CPU implementation provided in Voronota [Olechnovič and Venclovas 2014] which represents the fastest method of the state-of-the-art. This implementation is optimized for biochemical structures but limited to edges with vertices. Benchmarks are

Proteins				
Name	**#Site**	**Voronota (ms)**	**Ours (ms)**	**Speedup**
5ZCK	31	0.19	1.93	0.10 x
1AGA	126	0.72	2.72	0.27 x
3DIK	219	1.38	2.81	0.49 x
101M	1413	9.64	6.52	1.48 x
1A3F	2784	26.82	10.09	2.66 x
1A2Z	7666	73.27	20.60	3.56 x
8ID8	8635	94.82	44.85	2.11 x
7DBB	17733	212.14	112.88	1.88 x
7P3W	37149	418.13	249.99	1.67 x
7O0U	55758	604.40	487.27	1.24 x
1AON	58870	628.93	154.49	4.07 x
6QZ9	71724	904.19	454.67	1.99 x
3JC8	107640	1389.16	726.89	1.91 x
4V8W	123082	1494.57	300.00	4.98 x
7LER	158430	1964.02	425.35	4.62 x
6RXU	211834	2911.25	1507.20	1.93 x
4V6X	237685	3674.12	1705.29	2.15 x
7CGO	335722	5079.39	2424.51	2.10 x
4V60	483912	7078.91	3295.94	2.15 x
6U42	1358547	29963.55	9644.29	3.11 x
Moderate white noise ($\min(r) = 1, \max(r) = 3$)				
Spreading	**#Site**	**Voronota (ms)**	**Ours (ms)**	**Speedup**
25	100	1.06	2.99	0.35 x
50	1000	18.64	13.09	1.42 x
250	10000	645.85	81.12	7.96 x
500	100000	9783.94	1877.00	5.21 x
White noise ($\min(r) = .1, \max(r) = 10.1$)				
Spreading	**#Site**	**Voronota (ms)**	**Ours (ms)**	**Speedup**
100	100	1.39	2.93	0.47 x
200	1000	31.00	16.57	1.87 x
500	10000	914.27	205.07	4.46 x
1000	100000	30393.90	8160.00	3.72 x

Table 2. Benchmarks of our method compared to Voronota [Olechnovič and Venclovas 2014]. Experiments were performed with 10 unconsidered warmup samples and with the mean of 100 iterations. We present GPU timings of our method and Voronota uses the 32 cores of the CPU.

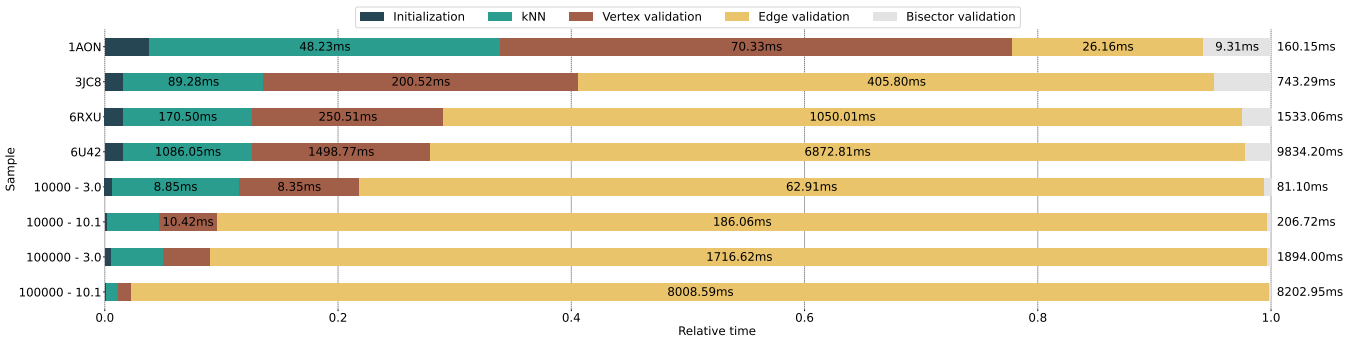


Fig. 12. Per stage computation time with proteins (first four rows) and homogeneous spatial distribution (last four rows). Timings in milliseconds represent the mean of 100 iterations after 10 warm up iterations, unconsidered.

then targeting two main use cases: uniform and heterogeneous spatial distributions of sites, from the following three datasets:

- *Proteins,* $r_i \in [1.5, 2.4]$: assumptions made with the security radius are not satisfied, few elliptic edges occur;
- *Moderate white noise,* $r_i \in [1, 3]$: assumptions made with the security radius are mostly satisfied, few elliptic edges occur;
- *White noise,* $r_i \in [0.1, 10]$: assumptions made with the security radius are mostly satisfied, more elliptic edges occur.

Protein files are obtained from the Protein Data Bank (PDB) [Berman et al. 2000]. We configure our pipeline with global empirical settings allowing the computation of all test cases without parameter change:

- a single pass considering the 16 nearest neighbors (Fig. 11);
- a maximum of 264 edges and 152 vertices allowing to handle the complete heterogeneous dataset.

We emphasize that these settings and our implementation mainly target comprehensive computation in a general use case, using all available shared memory on the target GPU. Better performance could be reached by manually tuning the parameters depending on the application and the targeted hardware. Notably, shared memory consumption can limit the number of warp running in parallel. Targeting homogeneous distribution only would allow reducing the necessary memory space for the data structure thanks to less complex cell geometry, as previous works [Basselin et al. 2021; Ray et al. 2019]. All input sets are perturbed using random values [O'Neill 2014] between $\left[-1e^{-3}, 1e^{-3}\right]$ ensuring general position.

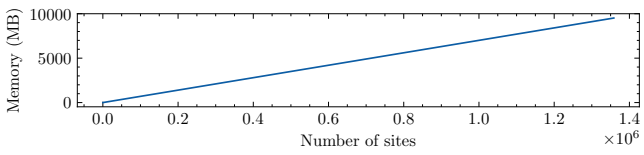


Fig. 13. Memory consumption of our method on the three datasets (Table 2). It is linearly dependent to the number of sites in the input set which directly results from the cell-oriented computation.

5.1 Performance

Table 2 presents performance benchmarks. All experiments were performed by executing both computation pipelines completely. Considering our method, this includes the complete construction of individual cells. Depending on the application, additional operations (merging to store the diagram, explicit computation of bisectors, ...) may be required and could add an additional minor cost. To extract Voronota's computed combinatorial structure, we rely on the performance setup provided in the library [Olechnovič and Venclovas 2014] which performs the complete parallel computation of the diagram's vertices with OpenMP.

Our method performs two times faster in average while allowing the comprehensive construction of the diagram. This is in contrast with Voronota which only outputs the vertices and the connected skeleton. This performance speedup is notably observed under heterogeneous spatial distribution, even if it represents the target of Voronota. We also note that both methods require more computation time with homogeneous distribution compared to the protein dataset with a similar number of sites. For instance, Voronota's performances are seven times slower under the moderate white noise configuration with 100000 sites compared to the protein 3JC8. In contrast, our method is only impacted by a factor of three. This is notably due to the increased number of elliptic edges, linked to high radii changes. This observation is confirmed at an increased range of radii, that strengthen the chances for a site to be included within the convex hull of two others.

Fig. 12 provides per stage computation time for several samples in all benchmarks datasets. We first notice that the most consuming stage is the edge validation representing at least half of the computation time for most samples. This is due to our handling of elliptic edges using their bounding sphere, a coarser approximation than other types of validation. Again, this is emphasized under homogeneous distribution. Other stages are performed in less computation time thanks to their precise bounding shapes.

Finally, Fig. 13 gives a benchmark of the memory consumption. Thanks to the cell-oriented computation, its consumption is linearly growing according to the number of sites, independently from the spatial distribution. This is especially interesting to support large sets of sites, benefiting from reduced computation time.

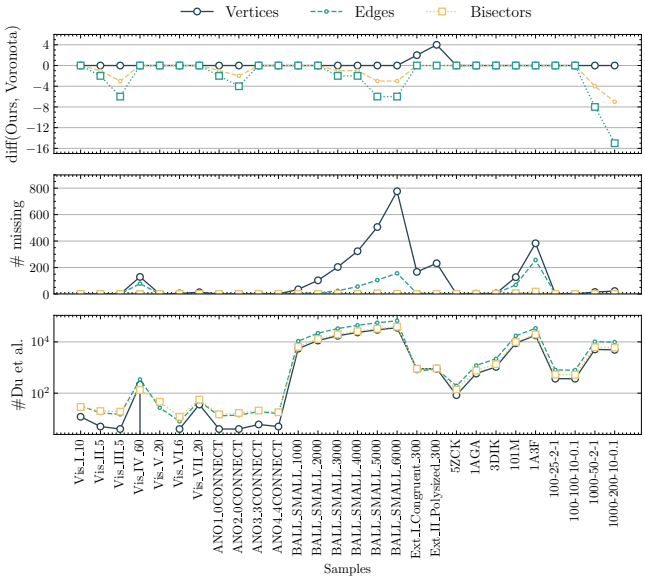
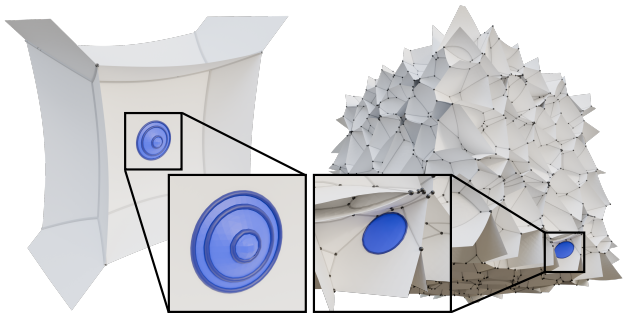
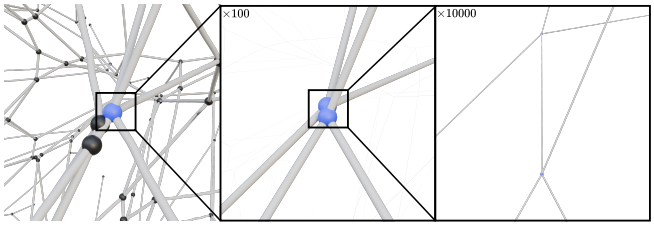


Fig. 14. Number of missing components (bisector, trisector and quadrisector) in Voronota's and our method's topology compared with a robust algorithm relying on a piecewise linear assumption [Du et al. 2022]. (Top) Difference between our method's number of missing components and Voronota's. (Middle) Number of missing components in our method's topology. (Bottom) Number of components produced by Du et al. [Du et al. 2022].



(a) Two cases (left: Vis_III_5, right: 1000-50-2-1) where Voronota do not recover the complete topology due to closed elliptic edges (in blue).



(b) Two missing vertices in Voronota's and our method's topologies (1AGA).

Fig. 16. Illustration of the mesh produced by Du et al.'s method [Du et al. 2022] and the components differing from Du et al. (in blue) in Voronota's and our method's produced topology.

Even if our implementation does not target provably robust computation, Table 4 provides performance results of our method compared to Voronota on a robustness dataset [Song et al. 2022] which confirm the observation made on our three datasets.

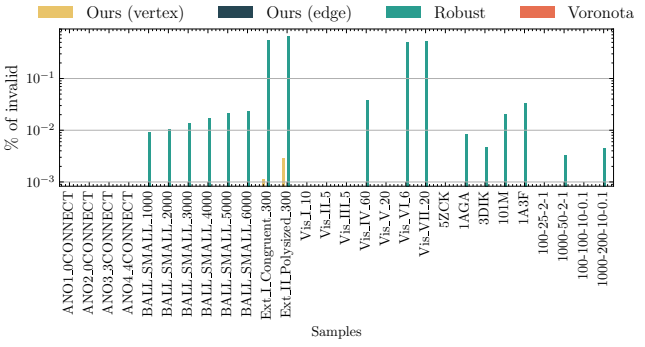


Fig. 15. Study of the compared methods output geometry (Robust [Du et al. 2022], Voronota [Olechnovič and Venclovas 2014]). The geometry of a component is said to be invalid if there exists a site, not in its combinatorial structure, which is closer to the point. Closed trisectors are validated using the minimum point of their geometry. Our method is the only one providing the information of closed trisectors.

5.2 Validation

We further evaluate our method from the quality of the produced topology and geometry. Du et al. introduced an algorithm computing implicit surface networks through discretized samples [Du et al. 2022]. It is able to build a combinatorial structure of an Apollonius diagram but is limited by the precision of the input values as well as its memory consumption. Fig. 14 provides the number of missing bisectors, trisectors and quadrisectors in the combinatorial structure computed by Voronota [Olechnovič and Venclovas 2014] and our method in comparison with the output of Du et al.'s method configured with a high resolution 100^3 grid [Du et al. 2022]. Due to memory restriction, we provide this analysis for all cases composed of less than 6000 sites in all datasets. We extract the topology computed by our method by saving a component of a cell if the index of the cell is the smallest combinatorial index of the component. In contrast, we compute Voronota's topology from the combinatorial structure of the produced vertices.

First, we notice that bisectors and trisectors computation errors of our method is constantly lower than Voronota's which is unable to compute elliptic edges without vertices (Fig. 16a). These results show that both our construction process and our data structure support a wider range of cases. Notably, this is emphasized in several complex cases in Song et al.'s dataset (Vis_III_5, ANO_2_0_CONNECT) and the random cloud samples (1000-50-2-1, 1000-200-10-0.1) in a direct dependency with the number of sites and the scale of radii. This illustrates the quality of our closed edges handling, more likely to

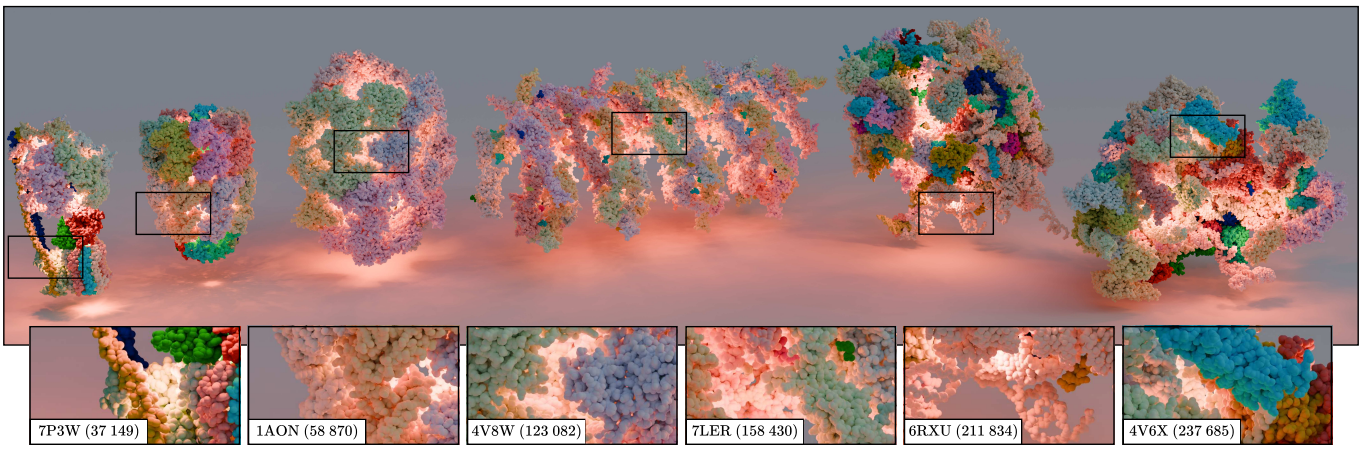


Fig. 17. Illustration of an application of Apollonius diagrams to molecular graphics. Vertices are computed with our method and filtered based on their visibility. They are then used to position lights suggesting the cavities and emphasizing the molecular structure. The understanding of the shape of the large structures is especially improved in this setup.

occur in these configurations. Both our method's and Voronota's outputs may differ strongly from the output of Du et al.'s method in complex cases (Fig. 16b) due to the piecewise linear assumption.

To precisely measure the quality of the output geometry, we finally test each method's outputs using interval arithmetic. We start by computing the distance between the output components and their combinatorial structure. Then, we test for the existence of a closer site in the input set. Fig. 15 reports percentages of invalid components. Our method is the only one to explicitly provide the closed edges of the diagram. We validate these edges by ensuring that the point τ_{min} (Section 4.5), minimizing the distance to the combinatorial structure, is not closer to any other site. As illustrated, the linear approximation performed by Du et al.'s method [Du et al. 2022] may result in invalid geometry. In contrast, no invalid vertices produced by Voronota were observed in the presented cases. Our method's outputs are also almost always valid. We notice that it can produce invalid vertices in the most complex cases of the dataset (Ext_I_Congruent_300, Ext_II_Polysized_300). More robust predicate could allow a better handling of such configurations.

6 APPLICATIONS TO MOLECULAR GRAPHICS

Molecular illustration represents a challenge to support the communication of research notably through popularization [Johnston 2023; Slater et al. 2022]. Several works have shown that Voronoi diagrams can be used to enhance molecular illustrations by highlighting interesting structures [Lindow et al. 2011]. Molecular studies are, however, intrinsically depending on the precision level of the Voronoi diagram of the atomic structure. Then, Apollonius diagrams have been shown as better suited for these applications than the Voronoi diagram of atoms centers [Goede et al. 1997; Olechnovič et al. 2010; Yaffe et al. 2008]. Notably, they allow a better estimation of atomic cavities, known for their biochemical meaning [Krone et al. 2016].

Representing vertices hidden within the structure as lights allows to suggest the potential cavities (Fig. 17). We then present in this section an application of our method for the illustration of large molecular systems, strongly benefiting from reduced computation time. We start by computing the Apollonius diagram and extract vertices. We then filter these vertices based on several parameters.

Vertex extraction. Starting from the presented cell-oriented data structure, we first extract a set of unique vertices since they are shared by multiple cells. This is made by saving the points of a vertex if its smallest index is the index of the current cell.

Filtering. Since we focus on the suggestion of hidden structures in the molecular body, we need to compute a burial level for a given vertex point. Lindow et al. identified that this can be approximated similarly to ambient occlusion [Lindow et al. 2011; Zhukov et al. 1998]. Thus, for every vertex point, we launch a set of 64 rays uniformly distributed on a sphere and compute a ratio from the number of rays hitting one of the sites. To further reduce the number of points, we additionally allow filtering them from their distance to their corresponding sites. As many points are located in a narrow location, this improves the refinement to interesting areas.

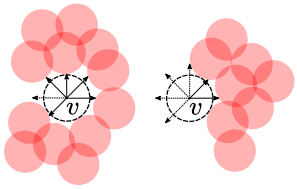


Fig. 17 provides an illustration of several molecular systems with lights positioned on the filtered vertices. We can notice that cavities are correctly hinted from their illumination and support the understanding of the molecular body. This is especially true for large systems which interesting areas are more difficult to notice. Finally, Table 3 gives the performance of every stage. It suggests that the method supports the interactive visualization of large molecular systems and that it strongly benefits from GPU execution. Such systems could be integrated within illustration software allowing to automatically highlight the shape of the studied systems.

PDB Id	#Site	#Vertex	Vertices	Hit ratio	Filtering
7P3W	37149	253431	248.58	5.84	0.13
4V8W	123082	838101	303.32	18.42	0.19
7LER	158430	1077978	429.56	23.1	0.3
6RXU	211834	1450141	1418.74	33.01	0.24
4V6X	237685	1646763	1720.43	37.53	0.24

Table 3. Performance of the cavity enhancement procedure. Vertices section includes computation of the complete diagram and compaction of vertices while Hit Ratio section includes the complete tracing of all vertices. All stage times are given in milliseconds.

7 LIMITATIONS AND FUTURE WORKS

Our pipeline targets standard applications of Apollonius diagrams and is planned to be more flexible than previous methods. However, several properties could be improved.

First, despite our light data structure (Section 4.2), specific cases could lead to complex geometry, causing potential GPU memory overflow. To address this issue, a CPU fallback is commonly considered [Basselin et al. 2021; Ray et al. 2019].

Second, even if our method allows a comprehensive construction thanks to our construction method (Section 4.4), it does not include exact predicates and relies on perturbations to ensure general position. Investigation for robust predicates on GPU could address various needs for combinatorial information of Apollonius diagrams.

Third, the elliptic edges validation rely on their bounding sphere to avoid analyzing per-edge set of vertices. Even if this coarse approximation allows fast tests, more precise bounding volumes could represent a possible improvement in terms of performance.

Fourth, numerical errors may be observed during the validation of very large bounding spheres. To ensure the termination of the algorithm, we fixed the maximum number of validation steps to 10 in our implementation. More precise restriction of the diagram and more robust predicate could allow a better handling of these cases.

Fifth, we selected several parameters for the allocation of the data structure allowing to fully handle our test dataset. However, it can lead to unoptimal settings. An automatic computation of these parameters could improve the compatibility of our method.

Finally, our method features faster computation compared to previous works. Further progress could, however, be made by targeting dedicated GPU optimizations of the meshless framework. A better distribution of the workload can notably help to uniformize the processing, while a reduction of register and shared memory usage could allow more warps to run in parallel. Such optimizations could especially benefit to the processing of heterogeneous distributions.

8 CONCLUSION

We present a comprehensive computation method of the Apollonius diagram in $\mathbb{R}^3$ adapted to the GPU. Thanks to a dedicated computation pipeline, our strategy allows to handle both homogeneous and heterogeneous spatial distributions of sites. We show that it outperforms the fastest state-of-the-art method while allowing a complete construction of the cells. This addresses a significant need in various application fields for the systematic construction of Apollonius cells. As an example, we demonstrate its usability for the illustration of molecular systems.

ACKNOWLEDGMENTS

We thank the reviewers for their comments and suggestions which supported the improvement of our article. Cyprien Plateau–Holleville is supported by institutional grants from the National Research Agency under the Investments for the future program with the reference ANR-18-EURE-0017 TACTIC.

REFERENCES

F. Aurenhammer. 1987. Power Diagrams: Properties, Algorithms and Applications. *SIAM J. Comput.* 16, 1 (feb 1987), 78–96. https://doi.org/10.1137/0216006

F. Aurenhammer. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. *Comput. Surveys* 23, 3 (sep 1991), 345–405. https://doi.org/10.1145/116873.116880

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

J. Basselin, L. Alonso, N. Ray, D. Sokolov, S. Lefebvre, and B. Lévy. 2021. Restricted Power Diagrams on the GPU. *Computer Graphics Forum* 40, 2 (may 2021), 1–12. https://doi.org/10.1111/cgf.142610

H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. 2000. The Protein Data Bank. *Nucleic Acids Research* 28, 1 (Jan. 2000), 235–242. https://doi.org/10.1093/nar/28.1.235

J-D. Boissonnat and M. I. Karavelas. 2003. On the Combinatorial Complexity of Euclidean Voronoi Cells and Convex Hulls of D-Dimensional Spheres. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Baltimore, Maryland) *(SODA '03)*. Society for Industrial and Applied Mathematics, USA, 305–312. https://hal.inria.fr/inria-00072084/document

J-D. Boissonnat, C. Wormser, and M. Yvinec. 2006. Curved Voronoi Diagrams. In *Effective Computational Geometry for Curves and Surfaces*. Springer Berlin Heidelberg, 67–116. https://doi.org/10.1007/978-3-540-33259-6_2

N. Bonneel and J. Digne. 2023. A survey of Optimal Transport for Computer Graphics and Computer Vision. *Computer Graphics Forum* 42, 2 (May 2023), 439–460. https://doi.org/10.1111/cgf.14778

A. Bowyer. 1981. Computing Dirichlet tessellations. *Comput. J.* 24, 2 (Feb. 1981), 162–166. https://doi.org/10.1093/comjnl/24.2.162

T. Brochu, C. Batty, and R. Bridson. 2010. Matching fluid simulation elements to surface geometry and topology. *ACM Transactions on Graphics* 29, 4 (July 2010), 1–9. https://doi.org/10.1145/1778765.1778784

D. R. Bukenberger, K. Buchin, and M. Botsch. 2022. Constructing L∞ Voronoi Diagrams in 2D and 3D. *Computer Graphics Forum* 41, 5 (Aug. 2022), 135–147. https://doi.org/10.1111/cgf.14609

O. Devillers, M. Karavelas, and M. Teillaud. 2017. Qualitative symbolic perturbation: two applications of a new geometry-based perturbation framework. *Journal of Computational Geometry* (2017), Vol. 8 No. 1 (2017). https://doi.org/10.20382/JOCG.V8I1A11

X. Du, Q. Zhou, N. Carr, and T. Ju. 2022. Robust computation of implicit surface networks for piecewise linear functions. *ACM Transactions on Graphics* 41, 4 (jul 2022), 1–16. https://doi.org/10.1145/3528223.3530176

I. Z. Emiris and M. I. Karavelas. 2006. The predicates of the Apollonius diagram: Algorithmic analysis and implementation. *Computational Geometry* 33, 1-2 (jan 2006), 18–57. https://doi.org/10.1016/j.comgeo.2004.02.006

M. L. Gavrilova and J. Rokne. 2003. Updating the topology of the dynamic Voronoi diagram for spheres in Euclidean d-dimensional space. *Computer Aided Geometric Design* 20, 4 (jul 2003), 231–242. https://doi.org/10.1016/s0167-8396(03)00027-x

D. Geiß, R. Klein, and R. Penninger. 2012. *Optimally Solving a Transportation Problem Using Voronoi Diagrams*. Springer Berlin Heidelberg, 264–274. https://doi.org/10.1007/978-3-642-32241-9_23

A. Goede, R. Preissner, and C. Frömmel. 1997. Voronoi cell: New method for allocation of space among atoms: Elimination of avoidable errors in calculation of atomic volume and density. *Journal of Computational Chemistry* 18, 9 (jul 1997), 1113–1123. https://doi.org/10.1002/(sici)1096-987x(19970715)18:9<1113::aid-jcc1>3.0.co;2-u

R. Hoetzlein. 2014. Fast Fixed-Radius Nearest Neighbors: Interactive Million-Particle Fluids. In *GPU Technology Conference (GTC)*.

Z. Hu, X. Li, A. Krishnamurthy, I. Hanniel, and S. McMains. 2017. Voronoi cells of non-general position spheres using the GPU. *Computer-Aided Design and Applications* 14, 5 (mar 2017), 572–581. https://doi.org/10.1080/16864360.2016.1273576

B. Johnston. 2023. BradyAJohnston/MolecularNodes: MolecularNodes v2.2.0 for Blender 3.4.X. https://doi.org/10.5281/ZENODO.7498428

Name	#Site	Voronota (ms)	Ours (ms)	Speedup
BALL_1_1_10000	10000	132.95	16.72	7.95 x
BALL_1_1_20000	20000	275.44	31.22	8.82 x
BALL_1_1_30000	30000	365.12	45.65	8.00 x
BALL_1_1_40000	40000	595.72	60.28	9.88 x
BALL_1_1_50000	50000	648.16	74.64	8.68 x
BALL_1_1_60000	60000	819.59	89.55	9.15 x
BALL_1_1_70000	70000	989.60	104.43	9.48 x
BALL_1_1_80000	80000	1194.53	118.27	10.10 x
BALL_1_1_90000	90000	1184.07	132.32	8.95 x
BALL_1_1_100000	100000	1347.72	147.13	9.16 x
BALL_1_2_10000	10000	178.49	67.47	2.65 x
BALL_1_2_20000	20000	396.58	191.85	2.07 x
BALL_1_2_30000	30000	548.95	221.52	2.48 x
BALL_1_2_40000	40000	870.26	322.65	2.70 x
BALL_1_2_50000	50000	973.19	487.62	2.00 x
BALL_1_2_60000	60000	1292.13	428.48	3.02 x
BALL_1_2_70000	70000	1542.65	557.39	2.77 x
BALL_1_2_80000	80000	1873.27	568.06	3.30 x
BALL_1_2_90000	90000	1811.82	658.68	2.75 x
BALL_1_2_100000	100000	2130.34	844.65	2.52 x
BALL_1_5_10000	10000	392.60	133.79	2.93 x
BALL_1_5_20000	20000	994.66	345.37	2.88 x
BALL_1_5_30000	30000	1609.12	468.82	3.43 x
BALL_1_5_40000	40000	2320.87	609.64	3.81 x
BALL_1_5_50000	50000	2960.44	962.32	3.08 x
BALL_1_5_60000	60000	3598.05	1296.68	2.77 x
BALL_1_5_70000	70000	4611.46	1516.61	3.04 x
BALL_1_5_80000	80000	5898.28	2083.70	2.83 x
BALL_1_5_90000	90000	5903.36	2285.13	2.58 x
BALL_1_5_100000	100000	7039.21	2591.04	2.72 x
BALL_1_10_10000	10000	759.97	144.96	5.24 x
BALL_1_10_20000	20000	2177.13	366.47	5.94 x
BALL_1_10_30000	30000	3372.43	622.71	5.42 x
BALL_1_10_40000	40000	5090.46	952.52	5.34 x
BALL_1_10_50000	50000	7449.66	1378.77	5.40 x
BALL_1_10_60000	60000	8949.87	2131.36	4.20 x
BALL_1_10_70000	70000	11060.81	2295.69	4.82 x
BALL_1_10_80000	80000	13436.33	2854.91	4.71 x
BALL_1_10_90000	90000	16038.38	3678.57	4.36 x
BALL_1_10_100000	100000	19519.90	4119.86	4.74 x
Ext_I_Congruent_300	300	18.65	10.60	1.76 x
Ext_II_Polysized_300	300	10.60	12.69	0.83 x

Name	#Site	Voronota (ms)	Ours (ms)	Speedup
BALL_SMALL_1000	1000	27.78	10.09	2.75 x
BALL_SMALL_2000	2000	58.95	19.13	3.08 x
BALL_SMALL_3000	3000	108.73	28.73	3.78 x
BALL_SMALL_4000	4000	181.82	53.08	3.43 x
BALL_SMALL_5000	5000	229.25	47.56	4.82 x
BALL_SMALL_6000	6000	303.46	76.49	3.97 x
BALL_SMALL_7000	7000	396.37	85.83	4.62 x
BALL_SMALL_8000	8000	498.33	99.33	5.02 x
BALL_SMALL_9000	9000	590.85	124.93	4.73 x
BALL_SMALL_10000	10000	767.61	145.36	5.28 x
BALL_SMALL_11000	11000	843.84	169.20	4.99 x
BALL_SMALL_12000	12000	1054.06	180.95	5.83 x
BALL_SMALL_13000	13000	1172.59	212.43	5.52 x
BALL_SMALL_14000	14000	1317.50	271.90	4.85 x
BALL_SMALL_15000	15000	1546.33	239.55	6.46 x
BALL_SMALL_16000	16000	1754.37	244.63	7.17 x
BALL_SMALL_17000	17000	1873.64	252.29	7.43 x
BALL_SMALL_18000	18000	2102.28	312.85	6.72 x
BALL_SMALL_19000	19000	2310.31	325.94	7.09 x
BALL_SMALL_20000	20000	2160.31	366.06	5.90 x
BALL_SMALL_21000	21000	2478.50	399.71	6.20 x
BALL_SMALL_22000	22000	2398.47	438.21	5.47 x
BALL_SMALL_23000	23000	2670.94	505.26	5.29 x
BALL_SMALL_24000	24000	3054.86	520.94	5.86 x
BALL_SMALL_25000	25000	3187.04	466.87	6.83 x
BALL_SMALL_26000	26000	3372.50	527.42	6.39 x
BALL_SMALL_27000	27000	3327.13	611.58	5.44 x
BALL_SMALL_28000	28000	3509.32	593.65	5.91 x
BALL_SMALL_29000	29000	3732.42	672.92	5.55 x
BALL_SMALL_30000	30000	3376.36	621.96	5.43 x
ANO1_0CONNECT	7	7.97	1.09	7.31 x
ANO2_0CONNECT	8	0.03	1.14	0.03 x
ANO3_3CONNECT	9	0.03	1.17	0.02 x
ANO4_4CONNECT	8	0.02	1.15	0.02 x
Vis_I_10	10	0.05	1.14	0.04 x
Vis_II_5	9	0.04	1.14	0.03 x
Vis_III_5	9	0.04	1.19	0.03 x
Vis_IV_60	60	0.75	2.06	0.36 x
Vis_V_20	20	0.09	1.50	0.06 x
Vis_VI_6	6	1.58	1.05	1.49 x
Vis_VII_20	20	1.74	1.73	1.01 x

Table 4. Benchmarks of our method compared to Voronota [Olechnovič and Venclovas 2014] on the robustness dataset [Song et al. 2022]. The set of proteins provided in the dataset has been omitted in favor of the larger one presented in this paper. Performances are evaluated similarly to Table 2.

M. Kamarianakis. 2020. Predicates of the 3D Apollonius Diagram. https://doi.org/10.48550/ARXIV.2007.06658

M. I. Karavelas and M. Yvinec. 2002. Dynamic Additively Weighted Voronoi Diagrams in 2D. In *Algorithms — ESA 2002*. Springer Berlin Heidelberg, 586–598. https://doi.org/10.1007/3-540-45749-6_52

T. Karras. 2012. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (Paris, France) *(EGGH-HPG'12)*. Eurographics Association, Goslar, DEU, 33–37.

D-S. Kim, Y. Cho, and D. Kim. 2004. Edge-tracing algorithm for euclidean voronoi diagram of 3d spheres. In *Proceedings of the 16th Canadian Conference on Computational Geometry, CCCG'04, Concordia University, Montréal, Québec, Canada, August 9-11, 2004*. 176–179.

D. Kim and D-S. Kim. 2006. Region-expansion for the Voronoi diagram of 3D spheres. *Computer-Aided Design* 38, 5 (may 2006), 417–430. https://doi.org/10.1016/j.cad.2005.11.007

M. Krone, B. Kozlíková, N. Lindow, M. Baaden, D. Baum, J. Parulek, H.-C. Hege, and I. Viola. 2016. Visual Analysis of Biomolecular Cavities: State of the Art. *Computer Graphics Forum* 35, 3 (jun 2016), 527–551. https://doi.org/10.1111/cgf.12928

G. Lanzara and M. Santacesaria. 2023. Market areas in general equilibrium. *Journal of Economic Theory* 211 (July 2023), 105675. https://doi.org/10.1016/j.jet.2023.105675

M. Lee, K. Sugihara, and D-S. Kim. 2022. Robust Construction of Voronoi Diagrams of Spherical Balls in Three-Dimensional Space. *Computer-Aided Design* 152 (nov 2022), 103374. https://doi.org/10.1016/j.cad.2022.103374

B. Lévy and N. Bonneel. 2013. Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration. In *Proceedings of the 21st International Meshing Roundtable*. Springer Berlin Heidelberg, 349–366. https://doi.org/10.1007/978-3-642-33573-0_21

N. Lindow, D. Baum, A-N. Bondar, and H. C. Hege. 2013. Exploring cavity dynamics in biomolecular systems. *BMC Bioinformatics* 14, S19 (Nov. 2013). https://doi.org/10.1186/1471-2105-14-s19-s5

N. Lindow, D. Baum, and H-C. Hege. 2011. Voronoi-Based Extraction and Visualization of Molecular Paths. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (dec 2011), 2025–2034. https://doi.org/10.1109/tvcg.2011.259

M. Manak. 2019. Voronoi-based detection of pockets in proteins defined by large and small probes. *Journal of Computational Chemistry* 40, 19 (apr 2019), 1758–1771. https://doi.org/10.1002/jcc.25828

M. Manak, L. Jirkovsky, and I. Kolingerova. 2016. Interactive Analysis of Connolly Surfaces for Various Probes. *Computer Graphics Forum* 36, 6 (may 2016), 160–172. https://doi.org/10.1111/cgf.12870

M. Manak and I. Kolingerova. 2016. Extension of the edge tracing algorithm to disconnected Voronoi skeletons. *Inform. Process. Lett.* 116, 2 (feb 2016), 85–92. https://doi.org/10.1016/j.ipl.2015.09.017

A. McLean, F. F. Leymarie, and G. Wiggins. 2007. Apollonius diagrams and the Representation of Sounds and Music. In *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE. https://doi.org/10.1109/isvd.2007.7

N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, and M. L. Gavrilova. 2006. An algorithm for three-dimensional Voronoi S-network. *Journal of Computational Chemistry* 27, 14 (2006), 1676–1692. https://doi.org/10.1002/jcc.20484

D. Meister and J. Bittner. 2022. Performance Comparison of Bounding Volume Hierarchies for GPU Ray Tracing. *Journal of Computer Graphics Techniques* 11, 4 (2022), 1–19.
P. Moreno-Regidor, J. García López de Lacalle, and M-Á. Manso-Callejo. 2012. Zone design of specific sizes using adaptive additively weighted Voronoi diagrams. *International Journal of Geographical Information Science* 26, 10 (Oct. 2012), 1811–1829. https://doi.org/10.1080/13658816.2012.655742
M. K. Mukundan, S. B. Thayyil, and R. Muthuganapathy. 2022. A parallel algorithm for computing Voronoi diagram of a set of spheres using restricted lower envelope approach and topology matching. *Computers & Graphics* 106 (Aug. 2022), 210–221. https://doi.org/10.1016/j.cag.2022.05.017
NVIDIA. 2023. NVIDIA ADA LOVELACE PROFESSIONAL GPU ARCHITECTURE. https://images.nvidia.com/aem-dam/en-zz/Solutions/technologies/NVIDIA-ADA-GPU-PROVIZ-Architecture-Whitepaper_1.1.pdf
K. Olechnovič, M. Margelevičius, and Č. Venclovas. 2010. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics* 27, 5 (Dec. 2010), 723–724. https://doi.org/10.1093/bioinformatics/btq720
K. Olechnovič and Č. Venclovas. 2014. Voronota: A fast and reliable tool for computing the vertices of the Voronoi diagram of atomic balls. *Journal of Computational Chemistry* 35, 8 (feb 2014), 672–681. https://doi.org/10.1002/jcc.23538
M. E. O'Neill. 2014. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation.* Technical Report HMC-CS-2014-0905. Claremont, CA.
D. Pivovarov, T. Oberleiter, K. Willner, and P. Steinmann. 2018. Fuzzy-stochastic FEM–based homogenization framework for materials with polymorphic uncertainties in the microstructure. *Internat. J. Numer. Methods Engrg.* 116, 9 (Sept. 2018), 633–660. https://doi.org/10.1002/nme.5947
N. Ray, D. Sokolov, S. Lefebvre, and B. Lévy. 2019. Meshless voronoi on the GPU. *ACM Transactions on Graphics* 37, 6 (jan 2019), 1–12. https://doi.org/10.1145/3272127.3275092
G. Rong and T.-S. Tan. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06 (SI3D '06).* ACM Press, 109. https://doi.org/10.1145/1111411.1111431
C. H. Rycroft. 2009. VORO++: A three-dimensional Voronoi cell library in C++. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 19, 4 (Dec. 2009), 041111. https://doi.org/10.1063/1.3215722
M. Sainlot, V. Nivoliers, and D. Attali. 2017. Restricting Voronoi diagrams to meshes using corner validation. *Computer Graphics Forum* 36, 5 (aug 2017), 81–91. https://doi.org/10.1111/cgf.13247
M. Sharifzadeh and C. Shahabi. 2007. Processing Optimal Sequenced Route Queries Using Voronoi Diagrams. *GeoInformatica* 12, 4 (Oct. 2007), 411–433. https://doi.org/10.1007/s10707-007-0034-z
A. Slater, N. Nair, R. Suétt, R. Mac Donnchadha, C. Bamford, S. Jasim, D. Livingstone, and E. Hutchinson. 2022. Visualising Viruses. *Journal of General Virology* 103, 1 (jan 2022). https://doi.org/10.1099/jgv.0.001730
C. Song, M. Lee, S. Choi, and D.-S. Kim. 2022. Benchmark dataset for the Voronoi diagram of 3D spherical balls. *Data in Brief* 45 (dec 2022), 108605. https://doi.org/10.1016/j.dib.2022.108605
The CGAL Project. 2023. *CGAL User and Reference Manual* (5.6 ed.). CGAL Editorial Board. https://doc.cgal.org/5.6/Manual/packages.html
P. Wang, N. Yuan, Y. Ma, S. Xin, Y. He, S. Chen, J. Xu, and W. Wang. 2020. Robust Computation of 3D Apollonius Diagrams. *Computer Graphics Forum* 39, 7 (oct 2020), 43–55. https://doi.org/10.1111/cgf.14125
D. F. Watson. 1981. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Comput. J.* 24, 2 (Feb. 1981), 167–172. https://doi.org/10.1093/comjnl/24.2.167
R. Weller and G. Zachmann. 2010. ProtoSphere: a GPU-assisted prototype guided sphere packing algorithm for arbitrary objects. In *ACM SIGGRAPH ASIA 2010 Sketches (SA '10).* ACM. https://doi.org/10.1145/1899950.1899958
Will, H.-M. 1999. *Computation of additively weighted Voronoi cells for applications in molecular biology.* Ph. D. Dissertation. https://doi.org/10.3929/ETHZ-A-00384556210.3929/ethz-a-006652878
E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov. 2008. MolAxis: Efficient and accurate identification of channels in macromolecules. *Proteins: Structure, Function, and Bioinformatics* 73, 1 (April 2008), 72–86. https://doi.org/10.1002/prot.22052
S. Zhukov, A. Iones, and G. Kronin. 1998. An ambient light illumination model. (1998), 45–55. https://doi.org/10.1007/978-3-7091-6453-2_5

A VALID PREDICATE IMPLEMENTATION

The implementation of our predicate requires two geometric information: vertex points and closed elliptic edge boundaries.

As shown by Aurenhammer [Aurenhammer 1987] and by Boissonnat and Karavelas [Boissonnat and Karavelas 2003], an Apollonius cell in $\mathbb{R}^d$ of a site $s_i = (p_i, r_i)$ can be seen as the projection on $\mathbb{R}^d \times \{\emptyset\}$ of the intersection between the cone Γ_i of axis x_{d+1} and the Power cell of the site $\tilde{s_i} = (s_i, \sqrt{2}r_i)$. Γ_i is then given by

$$\Gamma_i \coloneqq \left\{\tilde{x} = (x, x_{d+1}) \in \mathbb{R}^{d+1} \mid x_{d+1} + r_i = ||x - p_i||\right\}, \tag{3}$$

while Power bisectors Π^2_{ij} of the cell $\mathcal{P}(\tilde{s_i})$ are characterized by

$$\Pi^2_{ij} \coloneqq \left\{\tilde{x} = (x, x_{d+1}) \in \mathbb{R}^{d+1} \mid \tilde{x} \cdot \tilde{n} + c_{ij} = 0\right\},$$

with $\tilde{n} = s_j - s_i$ and $c_{ij} = \frac{1}{2}(||p_i||^2 - r_i^2 - ||p_j||^2 + r_j^2)$. These results allow characterizing Apollonius diagrams components.

Let $\tilde{x} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ and the four weighted sites s_i, s_j, s_k and s_l. The Apollonius vertex v_{ijkl} is given by the intersection between the hyperplanes Π^2_{ij}, Π^2_{ik} and Π^2_{il} with Γ_i. The intersection $\Pi^2_{ij} \cap \Pi^2_{ik} \cap \Pi^2_{il}$ must then satisfy

$$\begin{cases} x_1 n_{ij,1} + x_2 n_{ij,2} + x_3 n_{ij,3} + x_4 n_{ij,4} + c_{ij} &= 0 \\ x_1 n_{ik,1} + x_2 n_{ik,2} + x_3 n_{ik,3} + x_4 n_{ik,4} + c_{ik} &= 0 \\ x_1 n_{il,1} + x_2 n_{il,2} + x_3 n_{il,3} + x_4 n_{il,4} + c_{il} &= 0 \end{cases}$$

These equations can be expressed as a matrix resulting in

$$\begin{pmatrix} x_2 \\ x_3 \\ x_4 \end{pmatrix} = -A^{-1} \begin{pmatrix} n_{ij,1} \\ n_{ik,1} \\ n_{il,1} \end{pmatrix} x_1 - A^{-1} \begin{pmatrix} c_{ij} \\ c_{ik} \\ c_{il} \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} x_2 \\ x_3 \\ x_4 \end{pmatrix} = N x_1 + b.$$

Changing the coordinates to $\tilde{x'} = (x - p_i, x_4 + r_i)$ results in

$$\begin{pmatrix} x'_2 \\ x'_3 \\ x'_4 \end{pmatrix} = N x'_1 + e \text{ with } e = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} N_1 p_{i,1} + b_1 - p_{i,2} \\ N_2 p_{i,1} + b_2 - p_{i,3} \\ N_3 p_{i,1} + b_3 + r_i \end{pmatrix}.$$

Finally, this expression can be inserted in the equation of Γ_i (3)

$$x'^2_1 + (N_1 x'_1 + e_1)^2 + (N_2 x'_1 + e_2)^2 = (N_3 x'_1 + e_3)^2$$

$$\Leftrightarrow\ x'^2_1 E + 2M x'_1 = e_3^2 - e_1^2 - e_2^2,$$

with $E = 1 + N_1^2 + N_2^2 - N_3^2$ and $M = N_1 e_1 + N_2 e_2 - N_3 e_3$. The resulting quadratic expression has none to two solutions.

As noticed by Medvedev et al., the minimum and maximum of an Apollonius edge e_{ijk} are given by the intersection of e_{ijk} and the plane passing through the points p_i, p_j and p_k [Medvedev et al. 2006]. Similarly to vertex points, they can be obtained by solving

$$\begin{cases} \tilde{x} \cdot \tilde{n}_{ij} + c_{ij} = \tilde{x} \cdot \tilde{n}_{ik} + c_{ik} = \tilde{x} \cdot \tilde{n}_{ijk} + c_{ijk} = 0 \\ ||x - p_i|| = x_{d+1} + r_i, \end{cases}$$

with

$$\tilde{n}_{ijk} = \left(\frac{(p_j - p_i) \times (p_k - p_i)}{||(p_j - p_i) \times (p_k - p_i)||}, 0\right)^T \text{ and } c_{ijk} = -(n_{ijk} \cdot p_i).$$

Since the resulting equations give only one point if the edge is open and two points otherwise, we can also rely on the number of its valid solutions to assert the type of the edge.