Voronoi Graph Traversal in High Dimensions with Applications to Topological Data Analysis and Piecewise Linear Interpolation

Vladislav Polianskii

vpol@kth.se

KTH Royal Institute of Technology
Stockholm, Sweden

Florian T. Pokorny

fpokorny@kth.se

KTH Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

Voronoi diagrams and their dual, the Delaunay complex, are two fundamental geometric concepts that lie at the foundation of many machine learning algorithms and play a role in particular in classical piecewise linear interpolation and regression methods. More recently, they are also crucial for the construction of a common class of simplicial complexes such as Alpha and Delaunay-Čech complexes in topological data analysis. We propose a randomized approximation approach that mitigates the prohibitive cost of exact computation of Voronoi diagrams in high dimensions for machine learning applications. In experiments with data in up to 50 dimensions, we show that this allows us to significantly extend the use of Voronoi-based simplicial complexes in Topological Data Analysis (TDA) to higher dimensions. We confirm prior TDA results on image patches that previously had to rely on sub-sampled data with increased resolution and demonstrate the scalability of our approach by performing a TDA analysis on synthetic data as well as on filters of a ResNet neural network architecture. Secondly, we propose an application of our approach to piecewise linear interpolation of high dimensional data that avoids explicit complete computation of an associated Delaunay triangulation.

CCS CONCEPTS

• **Theory of computation** → Random walks and Markov chains; Computational geometry; • **Mathematics of computing** → Approximation algorithms; Interpolation.

KEYWORDS

Voronoi diagram, Delaunay complex, topological data analysis, linear interpolation

ACM Reference Format:

Vladislav Polianskii and Florian T. Pokorny. 2020. Voronoi Graph Traversal in High Dimensions with Applications to Topological Data Analysis and Piecewise Linear Interpolation. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, August 23–27, 2020, Virtual Event, CA, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3394486.3403266>

This work is licensed under a Creative Commons Attribution International 4.0 License.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7998-4/20/08.
<https://doi.org/10.1145/3394486.3403266>

Figure 1: A set of points $P \subset \mathbb{R}^2$ in black and the associated Voronoi diagram $\text{Vor}(P)$ in blue and Delaunay triangulation $\text{Del}(P)$ in brown.

1 INTRODUCTION

Machine learning applications commonly require the analysis of large amounts of high-dimensional data, oftentimes represented as a set $P = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ in Euclidean space. The Voronoi diagram $\text{Vor}(P)$ consisting of cells formed by those points nearest to any given $x_i \in P$, naturally describes the geometric structure of such a dataset P (see fig. 1) and as such is an important conceptual ingredient to a variety of geometry-based machine learning algorithms such as nearest neighbor methods. The connectivity structure of the Voronoi cells in $\text{Vor}(P)$ is furthermore captured by the associated Delaunay triangulation $\text{Del}(P)$ dual to $\text{Vor}(P)$. Important classical piecewise linear interpolation and regression methods rely on $\text{Del}(P)$ and it has been shown that $\text{Del}(P)$ is in some sense the optimal triangulation of data for this purpose [3], [35]. Similarly, many constructions in topological data analysis, such as Alpha and Delaunay-Čech complexes require the computation of $\text{Del}(P)$ [16], further justifying the importance of $\text{Vor}(P)$ and $\text{Del}(P)$ for data analysis and machine learning applications.

Our contributions: We address one of the key difficulties with applying $\text{Vor}(P)$ and $\text{Del}(P)$ based constructions in higher dimensions, which is due to the worst case $O(n^{\lceil \frac{d}{2} \rceil})$ [27] complexity of exact computation of the complete Voronoi diagram and Delaunay complex respectively. To illustrate this point empirically: on a modern desktop machine, it is easily possible to compute Delaunay triangulations and Voronoi cell decompositions of millions of points in 2D or 3D with 64 GB of RAM and state of the art implementations like qhull [4], while the same calculation for just 1000 data points in 10 dimensions is currently practically impossible to obtain within a day using this implementation (for reference, a lower bound on the expected number of just the top-dimensional simplices in $\text{Del}(X)$ in 10 dimensions can be estimated as $9 \cdot 10^6 n$ for large n [14]).

To mitigate this challenge, we introduce two classes of algorithms based on randomized ray casting that enable us to perform a walk in the Voronoi graph $G(P) \subset \text{Vor}(P)$ without explicit complete

construction of $G(P)$. The first performs a random walk on $G(P)$ to iteratively determine increasingly larger subsets S of the set of all Voronoi cell intersections of dimension k for all $0 \leq k \leq n$, while the second implements an implicit visibility walk that enables us to compute piecewise linear interpolations over high dimensional Delaunay triangulations - without complete construction of $\text{Del}(P)$. For the first algorithm, we show that, with an increasing number of iterations, S converges to the complete Voronoi cell structure and we validate in experiments that this convergence is rapid in applications with up to 50-dimensional data. Our approach dually allows us to explore the $n - k$ dimensional simplices of the Delaunay triangulation and results in a new randomized approach to extract increasingly larger subset approximations of low dimensional skeletons of Delaunay triangulations without having to compute the complete set of top dimensional Delaunay simplices. In experiments, we evaluate our approach in applications to topological data analysis problems on synthetic data as well as on a set of image patches and show rapid convergence of associated persistence diagrams with respect to the bottleneck distance as the number of iterations in our algorithms are increased. Finally, we show that our approach enables us to implicitly compute a piecewise linear interpolation over a Delaunay triangulations in high dimensions, where the underlying triangulation is only utilized implicitly - and the algorithm's output, in this case, provides an exact answer, despite relying on a randomized ray-casting approach.

Related Work. Algorithms towards the computation of Voronoi diagrams and Delaunay triangulations have a long history within computational geometry [3], where most of the focus has been on exact and complete computation of these structures with close to optimal asymptotic run-time. One of the most well-known implementations of the computation of a d -dimensional Voronoi diagram construction is available in the `qhull` package[4] using a generalization of a convex-hull construction algorithm in a lifted $(d + 1)$ -dimensional space. While this algorithm does not scale well to high dimensional data, we use it as a baseline for our lower dimensional validation experiments, when feasible. A way to approximate Voronoi diagrams was considered in [1], where the authors provide a way to compute an approximate Voronoi diagram for approximate-nearest-neighbor requests. However, to our knowledge, the authors do not provide an approach to utilize this idea for the construction of approximate Delaunay triangulations. Instead of utilizing approximate nearest neighbor queries, our work instead focusses on an incremental approach to determining an increasingly larger subset of Voronoi cell intersections, and dually the corresponding Delaunay skeleton. A significant property of Delaunay triangulations with importance for data analysis lies in their optimality for multivariate piecewise linear interpolations. A representative result in this direction is provided by [35] who show that in a class of piecewise-linear interpolation functions over triangulations, the most optimal solution under suitable constraints is given by a piecewise linear interpolation over $\text{Del}(P)$, where the function is linearly interpolated over each d -dimensional simplex in $\text{Del}(P)$ (a convex hull of $d + 1$ points in d dimensions). Due to similar optimality results, Delaunay triangulations also find a multitude of applications in areas such as finite element analysis [9].

Topological data analysis (TDA) [16] is an emerging area in machine learning that describes multiscale topological properties of data. In its most classical form, TDA computes the topological properties of the set $P_r = \bigcup_{i=1}^n B_r(x_i)$ of balls $B_r(x_i) = \{y : \|y - x_i\|^2 \leq r^2\}$ of radius r around the data points. In TDA, the computation of persistent homology groups enables one to summarize the evolution of topological features called the associated persistent homology groups which describe the number of connected components, tunnels and enclosed voids in P_r for all $r \geq 0$. This computation however crucially relies on the computation of an appropriate discrete data structure called a filtration of simplicial complexes. To provably compute the persistent homology groups for P_r , $r \geq 0$, Delaunay-Čech and Alpha complexes have been proposed [16], and both of these rely on the computation of $\text{Del}(P)$. Due to the aforementioned computational challenges, approximate constructions, in particular of Vietoris-Rips complexes, are commonly used in applications to date [29]. These are able to recover some *but not all* of the information about the persistent homology groups of the sets P_r , $r \geq 0$. Alternatively, TDA researchers have relied on repeated subsampling of P . For instance, in [17] confidence bands for persistence landscapes were proposed. In that case, one may argue that in order to properly represent the topological features of the space with high confidence, the number of points in a subsample however still has to be rather large and the approach to determining an appropriate size of subsample is largely empirical. A particular way of subsampling is provided by the use of witness complexes[11], which requires a selection of a set of landmark points. An extensive analysis of existing techniques of applying topological data analysis in high dimensions can be found in Chapter 10.5 of [3]. Yet another approach to avoid the curse of dimensionality has been based on applications of the Johnson-Lindenstrauss lemma[10]. For example, the authors of [30] utilize random projections to perform a TDA analysis in lower dimensions. While mitigating the complexity challenges, these techniques may however accidentally destroy crucial topological information, for example by introducing self-intersections of P_r after projection to lower dimensions. Our approach instead relies on constructing an increasingly larger subset of $\text{Del}(P)$ directly in the high-dimensional space itself, which converges to the full set $\text{Del}(P)$ with increasing numbers of iterations.

2 BACKGROUND AND DEFINITIONS

Voronoi diagrams. We suppose our data is provided as a set $P = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ and we consider the standard Euclidean metric on $\mathbb{R}^d$. We furthermore make the technical assumption that P is in *general position*, meaning that no $(k + 3)$ points of P lie on a k -dimensional sphere and no $(k + 2)$ points of P lie on a k -dimensional plane. Note that this is a weak condition, since a random perturbation of P by an arbitrarily small uniform or Gaussian noise has this property with probability 1 [16]. The Voronoi diagram $\text{Vor}(P)$ of P is the partition of $\mathbb{R}^d$ into n regions called *Voronoi cells*, V_p for $p \in P$, where $V_p = \{y \in \mathbb{R}^d \mid \|y - p\| \leq \|y - x_i\| \forall x_i \in P\}$. Each cell is a (possibly unbounded) d -dimensional polyhedron. We call the k -dimensional faces of a cell k -faces for $k = 0, 1, \dots, d$. Each k -face can be represented as an intersection of $(d - k + 1)$ Voronoi cells. The boundary of a k -face is a collection of its $(k - 1)$ -faces for

$k \geq 1$. We will refer to 0-faces as *Voronoi vertices* and to 1-faces as *Voronoi edges*. The set of Voronoi vertices and Voronoi edges form a graph called the Voronoi Graph $G(P)$. Similarly, the union of all faces up to dimension k is called the k -skeleton of $\text{Vor}(P)$.

A Delaunay complex $\text{Del}(P)$ (also called Delaunay triangulation or tessellation), is a dual to a Voronoi diagram as follows. Consider a k -face as an intersection of Voronoi cells, generated by points of $Q \subseteq P$, $|Q| = d - k + 1$. Then, the set of the $d - k + 1$ points in Q is called a *simplex* of dimension $d - k$ (or codimension k). Geometrically, we can identify a k -simplex formed by $k + 1$ elements with a convex hull of the associated elements so that all 0-simplices (i.e. points) correspond to generators P , 1-simplices are the edges between generators whose Voronoi cells have a non-empty intersection, 2-simplices are triangles joining three generators each, etc. The Delaunay complex consists of the set of all these simplices of all dimensions from 0 to d and there is a bijection between Voronoi faces and simplices. Formally, we have $\text{Del}(P) = \{\sigma \subset P : \cap_{p \in \sigma} V_p \neq \emptyset\}$, where σ ranges over subsets of at most $d + 1$ points of P . The k -skeleton $\text{Del}_k(P)$ of the Delaunay complex consists of all simplices of the Delaunay complex up to dimension at most k . So, in particular, the 1-skeleton is a graph, the 2-skeleton additionally contains triangles, and similarly in higher dimensions. Note that if a simplex belongs to the complex, all of its faces also belong to it, and the intersection of two simplices in the Delaunay complex are either empty or another simplex in the complex. Formally, the Delaunay complex hence forms a structure called a *simplicial complex* in computational topology.

It is known, that the total number of simplices in a Delaunay can be $O(n^{\lceil \frac{d}{2} \rceil})$ in the worst case, and the number of k -simplices is bound by $O(n^{\min(d-k, \lceil \frac{d}{2} \rceil)})$ [27]. However, in simple cases of uniformly distributed points, the number of k -simplices grows linearly with the number of points, but with a constant exponentially depending on dimension [14]. Nevertheless, the complete computation of $\text{Del}(P)$ and $\text{Vor}(P)$ for high-dimensional data quickly becomes infeasible with a large number of data points in dimensions higher than 3.

Voronoi graphs. An example of the Voronoi Graph $G(P)$ in 2D is provided in Figure 1. While the Voronoi graph indeed forms an embedded graph for arbitrary d , note that in contrast the k -faces of a Voronoi diagram are in general k -dimensional polyhedra and not necessarily k -simplices. Recall also that each vertex in $G(P)$ corresponds to a Delaunay d -simplex, and each edge in $G(P)$ corresponds to a $(d - 1)$ -dimensional common border between two d -simplices in $\text{Del}(P)$ - illustrating the fundamental duality between $\text{Del}(P)$ and $\text{Vor}(P)$. $G(P)$ has several useful properties. For example, due to the *acyclicity theorem* [15], it is always possible to find a d -simplex in $\text{Del}(P)$ that contains a given point by performing a “visibility walk”. The work [13] studies the efficiency of different strategies for point location. In this work, we will show in particular for our application to piecewise linear interpolation how a visibility walk can be performed implicitly without a costly computation of the complete $\text{Del}(P)$.

Another important property is that a Voronoi graph can potentially be characterized as an *expander graph* [23] - a sparse graph with high connectivity. While this is not necessarily always the

case as shown in [31], who demonstrate that it possible to construct a Delaunay triangulation and a line that would pass through $O(n^{\lceil d/2 \rceil})$ d -simplices, the authors of [7] show that for a case of Poisson distributed data, the expected number of simplices passed by such a line in a 2D case is in fact only $O(\sqrt{n})$. As a corollary, the visibility walk is expected to have $O(\sqrt{n})$ steps in the worst case [12] in this case. However, we are not aware of similar results in higher dimensions.

In our setting, the expected number of vertices M of the Voronoi graph $G(P)$ depends linearly on the number of data points n , but has a potentially large constant exponential of d [14]. Thus, for large enough n and d , $n \ll M$ and it is practically impossible to store the whole graph directly. Each vertex, however, has degree at most $(d + 1)$, since each vertex corresponds to a d -dimensional simplex (the degree is smaller if the corresponding simplex intersects with the border of the global convex hull, in this case some Voronoi edges go to infinity); since we only consider cases with $d \ll n$, we observe that $G(P)$ is sparse.

Topological Data Analysis. In this work we will demonstrate our approach in applications to Topological Data Analysis (TDA) [16], and in particular in the TDA setting where we consider the topological features of the sets $P_r = \bigcup_{i=1}^n B_r(x_i)$ of unions of balls $B_r(x_i) = \{y : \|y - x_i\|^2 \leq r^2\}$ of radius $r > 0$ centered at the data points. We observe that $P_r \subset P_s$ for any $r \leq s$, inducing a so-called filtration of topological spaces. Topological data analysis enables us to compute the dimensions of the i -th persistent homology groups associated with this filtration P_r , $r \geq 0$, for $0 \leq i \leq d$. While it is not possible to fully detail all aspects of Persistent Homology groups here, we will attempt to provide the core intuition to the reader unfamiliar with the subject. The book [16] provides a complete introduction. A key concept in TDA is that the output of a persistent homology analysis of the dataset P is a sequence of so-called i -th persistence diagrams in dimension $0 \leq i \leq d$, where the i -th persistence diagram summarizes the evolution of the dimensions of the i -th homology groups - a concept from algebraic topology. Each point (x, y) in the i -th persistence diagram corresponds to the appearance of a topological feature in P_r at $r = x$ and the later disappearance at $r = y$, where y may be finite or infinite and $x \leq y$ in all cases. In particular, in the zeroth persistence diagram, each point in the diagram corresponds to a connected component of P_r that is eventually merged hierarchically with other components as r is increased. On the other hand, the 1-st persistence diagram indicates the appearance and disappearance of ‘tunnels’ in P_r at various r thresholds and a similar intuition can be formulated in higher dimensions. The vertical distance $y - x$ of a point (x, y) is called the persistence of the feature in the persistence diagram and corresponds to the r -interval length during which the topological feature exists in the filtration as r is increased. Points at large distance from the diagonal are hence considered significant ‘stable’ topological features, while points close to the diagonal roughly speaking correspond to topological noise. For example, when sampling noisy data on a unit circle in n dimensions, the first persistence diagram has a single point far from the diagonal corresponding to the ‘tunnel’ around which the circle wraps. To compute persistence diagrams of P_r , $r \geq 0$, we utilize the filtration of Delaunay-Čech complexes $DC_r(P)$ for $r \geq 0$ defined as a subset of the Delaunay

Complex as follows: $DC_r(P) = \{\sigma \in \text{Del}(P) : \cap_{x \in \sigma} B_r(x) \neq \emptyset\}$, where the condition $\cap_{x \in \sigma} B_r(x) \neq \emptyset$ can be easily checked using the Miniball algorithm[36] and it has been shown that $DC_r(P)$ is homotopy equivalent to P_r , $r \geq 0$ enables the computation of the persistent homology groups associated to P_r , $r \geq 0$ using $DC_r(P)$ [5, 16]. Note that for sufficiently large r , we have $DC_r(P) = \text{Del}(P)$ and the sets $DC_r(P)$ for all $r \geq 0$ is hence finite can be computed in one step by identifying the smallest r -threshold at which any given simplex from $\text{Del}(P)$ appears in $DC_r(P)$.

3 METHODOLOGY

In the following, we describe a two-step procedure which first computes a subset of the Voronoi vertices of the Voronoi graph $G(P)$ and then performs an implicit random walk on $G(P)$ starting at these vertices to iteratively discover an increasing subset of Voronoi vertices. From the set of Voronoi vertices, we are then able to enumerate an increasing subset of k -dimensional intersections of cells in $\text{Vor}(P)$ as well as simplices in $\text{Del}(P)$. A pseudocode for all algorithms is provided in the Appendix. Secondly, we show how this enables a visibility walk in $\text{Del}(P)$ that allows us to compute piecewise linear interpolations over $\text{Del}(P)$ in high dimensions.

Computation of a Subset of Voronoi Vertices

Every $d-k$ -face σ of $\text{Vor}(P)$ is an intersection of Voronoi cells of the form $\sigma = \cap_{x \in P_\sigma} V_x$ for some subset $P_\sigma = \{p_1, \dots, p_{d-k+1}\} \subset P$ of $d-k+1$ elements of P . Note that, by definition, all elements of P_σ are equidistant from every point in σ . For each σ , we will furthermore denote by $r_\sigma \in \sigma$ a particular choice of reference point in σ that will be determined by our algorithm.

Algorithm 4 describes how to obtain a subset of Voronoi vertices $S_0 \subset G(P)$ starting from a subset $P' \subseteq P$ of generators. Starting with a set of generators (i.e. d -faces of $\text{Vor}(P)$), the algorithm determines a set of $(d-1)$ -faces τ that belong to the boundary of a given Voronoi cell σ from P' and then iterates over lower and lower dimensional faces of these respectively and at each point a reference point r_τ for the newly discovered face is stored.

Initially, and trivially, we know the set of all d -faces which are just the Voronoi cells themselves, and where for each cell $\sigma = V_{x_i}$ for $x_i \in P$, we have $P_\sigma = \{x_i\}$ and we define $r_\sigma = x_i$. A ray starting from r_σ is then uniformly sampled in a random direction $u \in \mathbb{R}^d$ within σ using algorithm 1. The generation of the uniform direction vector is described in Alg.1 and just consists of an orthonormal projection onto the smallest subspace containing σ followed by random sampling of a unit norm vector using the standard Gaussian normalization algorithm.

The value $t^* \in \mathbb{R}_+$ at which the ray $r_\sigma + tu$ intersects the boundary of σ is then determined by either one of two algorithms. The first, Alg.2, is based on brute force search while the second, Alg.3 utilizes an intersection search strategy which we will explain shortly. Once t^* is determined, we set $r_\tau := r_\sigma + t^*u$ to be the point on the boundary of the current face σ intersected by the ray. Note that in case the ray is cast along the infinite part of an unbounded Voronoi cell, we invert the direction of the ray, which is guaranteed to yield an intersection with probability 1 for points P in general position. r_τ determines a new lower-dimensional face τ of σ of points equidistant to a strictly larger number of generators of P

Figure 2: Algorithm visualization

than σ . Finally, we set $P_\tau = P_\sigma \sqcup \{x\}$, where $x \in P \setminus P_\sigma$ is a new generator that produces the boundary and then repeat the approach for the new face σ by casting a ray from the reference point r_τ until we reach a Voronoi vertex. The full procedure is described in Alg.4 and visualized in the 2D case in Fig.2(a). Our approach is inspired by a similar technique that we developed for Monte Carlo integration in [28] for the case when σ is d -dimensional. Next, let us discuss the two algorithms for determining t^* .

Determining t^ via brute force.* For the current σ , consider any generator $x_0 \in P_\sigma$ and pick a generator $x \in P \setminus P_\sigma$. If the ray intersects the boundary component generated by x , then this happens when $\|r_\sigma + tu - x_0\| = \|r_\sigma + tu - x\|$ for some t , which has a solution of the form $t = \frac{\|r_\sigma - x\|^2 - \|r_\sigma - x_0\|^2}{2\langle u, x - x_0 \rangle}$. Thus, the smallest such non-negative t over all x is given by t^* , and finding t^* by brute force testing of all possible x requires $O(n \times d)$ operations. See Algorithm 2 for details. In case of non-existence of such t^* , the boundary is infinite along the selected direction, and we invert the ray's direction.

Determining t^ via intersection search.* Instead of going through all $x \in P \setminus P_\sigma$ one by one, we perform intersection search over the t parameter of the ray. Assume the intersection occurs at some t^* . To begin with, consider some $t > 0$ and the corresponding point $p = r_\sigma + tu$. We can understand whether t is greater or smaller than t^* by checking the nearest neighbor of x_t among points in P . If the nearest neighbor is one of the generators of σ , then $p \in \sigma$ and $t \leq t^*$, otherwise, we already have crossed the boundary and $t \geq t^*$ (in case of several nearest neighbors, we can pick any). Let us define an indicator function in on $\mathbb{R}_+$ which evaluates t as $\mathbb{1}[t < t^*]$. This function is monotone (it only changes from 1 to 0 at a point t^*), and we can perform the binary search on it to find the point at which the function makes a step from 1 to 0. If the maximum t that we consider as "infinite" is equal to T_{max} and we want to find t with absolute precision ϵ , then the complexity of the whole search would be $O(\log \frac{T_{max}}{\epsilon} \cdot \text{near}(N, d))$, where $\text{near}(N, d)$ is the cost of the nearest-neighbor lookup. The latter can be performed by a KD-Tree, which provides logarithmic complexity when the data dimensionality is not too large compared to the number of points: $\text{near}(N, d) = O(\log N \cdot d)$ [2]. This approach to determining t^* is described in detail in Alg.3.

Note that a few optimizations are possible, such as an early stopping of the nearest-neighbor search and an optimal movement of the "rightmost bound" in the boundary search with an early

stop of the binary search. The former is already included in the pseudocode in the $t_r \leftarrow \dots$ assignment: while the classical binary search would set $t_r \leftarrow t_m$ in that case, we immediately move the rightmost bound to the point where the ray would have hit (or does hit) the boundary related to a currently considered generator. The latter means that in $O(\text{near}(N, d))$ time one can check if t_r is the true distance to the boundary (considering that we apply the previous t_r optimization): if there are no points closer then the current one, then one can already stop the algorithm, otherwise t_r can be moved again according to a newly found generator. This approach can also be used as a necessary stopping criterion for the search, which guarantees that we find the right intersection.

Implicit traversal of the $G(P)$

By using the above approach initialized at some subset $P' \subset P$, we obtain a set of Voronoi vertices. Each vertex v has a generator set P_v of size $(d + 1)$. Each subset provides a description of one Voronoi face and we hence implicitly have information about 2^{n+1} Voronoi faces per vertex, so that finding just one vertex, in fact, provides a lot of new data about $\text{Vor}(P)$ and even a small set of Voronoi vertices may encode a large subset of $\text{Vor}(P)$.

Note that one approach to generating a larger and larger subset of $\text{Vor}(P)$ is to increase the numbers of rays cast per generator. For any finite P such an approach converges to the complete Voronoi diagram with increasing numbers of rays m per Voronoi cell.

However, it is possible to construct a sequence of adversarial examples of sets P in fixed dimension and with a fixed number of points n such that the required number of rays m tends to infinity. See, for example, the diagram in Figure 3. It is possible to position the generators in blue on the left in a such way that the probability of a random ray from any of the generators hitting any of blue edges is below any given threshold by squeezing them to lie closer and closer in the vertical direction. The expected number of rays cast to detect the dotted blue edge then increases the closer the generators are positioned in the vertical direction.

Figure 3: A 2D example with low probability of finding blue boundaries with just raycasting descent.

To mitigate this problem, our approach is to instead utilize the initial set of Voronoi vertices as seeds for a random walk on $G(P)$ that iteratively explores additional vertices. Consider a Voronoi vertex v with $P_v = \{x_0, x_1, \dots, x_d\} \subset P$ and reference point r_v obtained from our initial algorithm and which we note is, in this case, the unique point that is the geometric intersection of the

generators in P_v , so $v = r_v$ and r_v can also be computed directly as the smallest ball containing P_v (the miniball of P_v).

Each Voronoi vertex of points in general position has $(d + 1)$ edges (possibly with an infinite end) emanating from it. An i^{th} edge e_i from v is described by $P_{e_i} = P_v \setminus \{x_i\}$. Consider the task of following the edge e_i in the Voronoi graph to determine the other Voronoi vertex connected via this edge.

By knowing the set P_{e_i} , we can obtain a random ray u collinear with the line, containing e_i . The randomness of the ray only lies in its direction, and if we want to obtain the same direction as the edge e_i leaving v , we can consider the sign of $\langle u, x_i - x_j \rangle$ where x_j is any generator from P_{e_i} , and if it is positive – replace u by $-u$. We follow the ray $r_v + tu$, $t \in \mathbb{R}_+$ until we hit the second boundary vertex of e_i . We can use the same raycasting algorithms, described in Section 3, with the only difference being to exclude the point x_i from the search. In order to perform the random walk, at the current vertex, we select an edge randomly. If it happens that the selected edge is infinite in the second direction, then another edge is picked. As mentioned in the previous subsection, we perform the random walk not from one Voronoi vertex, but from multiple vertices found by the previous algorithm. This provides us with a simple approach to perform a random walk over $G(P)$ – which we recall is a graph with a single connected component – and which incrementally adds any newly traversed Voronoi vertices to the set of discovered Voronoi vertices as described in Algorithm 5 and visualized in 2D in Fig.2(b).

In Section 2 we have shortly discussed a possibility of characterizing Voronoi graphs in terms of expanders. It has been proven[19] that performing a random walk in an expander graph approaches a uniform sampling over the set of vertices with increasing numbers of steps, guaranteeing that all vertices are eventually discovered. Since the approach to picking the next edge to traverse is purely combinatorial, geometric difficulties in situations such as the one described in Fig.3 are however avoided.

While the theoretical properties of Voronoi graphs are to the best of our knowledge not fully explored yet, our intuition is that Voronoi graphs for points in regular position are close to being regular – almost all vertex degrees are close to $d + 1$ with the sufficient number of points ($d + 1$ when a Delaunay tetrahedron is located strictly inside the triangulation, and less than $d + 1$ when it touches the convex hull over the generators). Therefore we are motivated to conjecture that the random walk's stable distribution approaches close to uniform sampling as this would in this case follow from [26].

Obtaining $\text{Del}(P)$ and Stopping Criteria

As already discussed in Section 3, each Voronoi vertex automatically provides us with a number of Voronoi faces, containing the vertex. If we are only interested in the connectivity graph, we need to obtain 1-simplices of the Delaunay complex, which correspond to $(d - 1)$ -faces, and if we are interested in H_{k-1} homology, k -simplices and $(k - 1)$ -simplices are needed. Each vertex in $\mathbb{R}^d$ covers $\binom{d+1}{k+1}$ k -simplices – $(d - k)$ -faces. Thus, from every visited vertex v we can extract all the simplices we are interested in by going through the subsets of P_v of a certain size and adding them to the collection.

With the random walk, we are guaranteed to extract complete Delaunay skeleta with sufficient iterations, but currently we have to estimate the stopping criterion empirically. While we do not know how many k -simplices there are in total, we in practice instead monitor when the random walk stops finding new k -simplices for some fixed set of iterations. In our experiments, we further study this convergence and additional theoretical work will be necessary in the future to fully describe the convergence rates and bounds precisely. If the random walk, in particular, can be proven to provide close to uniform random sampling of vertices, this could lead to useful confidence bounds on the proportion of vertices discovered so far, in particular.

Implicit Visibility Walk in $\text{Del}(P)$

Consider the following problem: given a query point $q \in \mathbb{R}^d$, determine a d -simplex $\sigma \in \text{Del}(P)$ containing it if q lies in the convex hull of P that is triangulated by $\text{Del}(P)$. Alternatively, report that p does not lie on the convex hull.

We start by determining the nearest neighbor of q in P to obtain a Voronoi cell containing q . Then, either from that neighbor or from the query point directly, use the raycasting algorithm, to descend into one of the vertices of the Voronoi diagram. Now, we can perform a classical visibility walk towards the query point: select any generator x_i that is part of the current simplex σ such that the hyperplane that corresponds to $\tau_i = \sigma \setminus \{x_i\}$ separates x_i and q , and follow the Voronoi edge related to τ_i into a new Voronoi vertex. Repeat it until q belongs to a current d -simplex, or until we can not make such a step which would mean that our query point lies outside the convex hull of the given generators. The process is also described in Algorithm 6. As already mentioned in the introduction, this algorithm is guaranteed to converge[15] as it is just an implementation of a visibility walk in $\text{Del}(P)$, but crucially we do not have to compute the complete $\text{Del}(P)$ a priori. The selection of the nearest neighbor is only needed to find an initial vertex "close" to the query point and, in theory, one could start the walk from any known simplex. Once the d dimensional simplex σ containing q is obtained, we can compute barycentric coordinates of q with respect to the vertices of σ . If each vertex p_i of σ is associated with a function value f_i , this enables us to compute a piecewise linear interpolation of the function over $\text{Del}(P)$, as we shall explore in our experiments.

4 EXPERIMENTS

In the following we provide initial experiments of an implementation of our approach focussing on the feasibility of our approach in settings where classical algorithms currently fail and on showcasing convergence of persistence diagrams in the TDA setting. As we focus on the algorithm introduction here, we defer a complete runtime analysis to future work, however. The code for all algorithms was written in C++, with the use of the Eigen[21] Library for matrix inverses and the `cnpy` library to read and write numpy data files. Additionally, `gudhi`[20] was used to compute the bottleneck distances between persistence diagrams and a modified version of `phat`[6] for homology computation. Data manipulations for the experiments were done with standard Python libraries.

Validation On Synthetic Data

Consider the following annulus embedded in $\mathbb{R}^d$ defined by

$$\text{Ring}^d = \left\{ x \in \mathbb{R}^d \mid \exists \alpha : \|x - (\cos \alpha, \sin \alpha, 0, \dots, 0)\| < 0.25 \right\}$$

Let us denote by $\text{Ring}^d(N)$, a dataset with N points sampled uniformly from the annulus as visualized in Fig.4.

Figure 4: Illustrations of 2D projections of $\text{Ring}^d(10^3)$.

Here, to investigate our algorithms, we obtain the zeroth and first persistence diagrams computed from the Delaunay-Čech complex computed from our subset of $\text{Del}(P)$ and show convergence of the resulting persistence diagrams. We computed the diagrams for different dimensions up to 50 with slightly different parameters, some of which are presented in Figure 5, where w refers to the number of random $G(P)$ exploration steps made from each starting point.

Figure 5: Obtained persistence diagrams for Ring^d data after $w = 10^3$ walks.

We use two methods to validate the diagrams, different for low-dimensional and high-dimensional data. In a low-dimensional case, we can explicitly construct the Voronoi diagram and therefore build the full Delaunay-Čech complex and compute its persistence. Afterwards, we investigate the bottleneck distances between the approximated diagrams and the true diagrams to check how close the approximation is to the ground truth. As can be seen in Fig.6a, both the first (H_0) and second (H_1) persistence diagrams rapidly converge to the ground truth in the case of $N = 10000$ data points and as the number of random walk steps is increased. The plot shows the average distance and its standard deviation over 50 runs with different random seeds, which were computed only when at least 25% of the distances were finite (infinite distances can occur when infinite points occur in the persistence diagram). Additionally, the gray line shows an example of distance in H_1 of one run, which also shows how there are several infinite bars present in H_1 during the first iterations. In the case, when it is not feasible to construct the full diagram, we can use our knowledge about the topology of the underlying space Ring^d . We can expect exactly one persistent

1-dimensional cycle with a death parameter above 0.75 - the radius of the tunnel passing through the ring. Additionally, we test for convergence by running two algorithms simultaneously with different random seeds and periodically compute the bottleneck distance between the diagrams. The evolution of the distances over the random walk steps for the 10-dimensional case with $N = 5000$ is presented in Figure 6b, with mean and standard deviation estimated from 15 different runs and all 105 pairwise comparisons between them. Additionally, Figure 7 shows how the persistence of one $Ring^{10}$ changes during the random walk with topological noise corresponding to points close to the diagonal quickly dissipating as the number of random walk steps increases. While the initial determination of Voronoi Vertices for $Ring^{10}(10000)$ took less than a minute to compute, 1000 steps of the random walk over $G(P)$ took less than 30 minutes - with explicit computation of $Vor(P)$ being infeasible with classical implementations. Two resulting diagrams from two runs with different random seeds on $Ring^{10}$ can be directly compared in Fig.8, showing that the main ring structure is recognized stably (a point vertically far from the diagonal), while small topological fluctuations (points close to the diagonal that differ between the diagrams) can be observed.

Figure 6: Bottleneck distance evolution.

Figure 7: $Ring^{10}(10000)$ evolution over number of steps.

Figure 8: Diagrams for a fixed dataset $Ring^{10}(5000)$, $w = 10^4$.

We also run a simple experiment on the recovery of H_2 homology of a noisy 2-sphere, embedded in $\mathbb{R}^{10}$. In particular, the dataset of 500 points is generated uniformly from a unit sphere in the

first three coordinates of $\mathbb{R}^{10}$ with radius 1 with Gaussian noise $N(\mu = 0, \sigma = 0.1)$ added in all 10 dimensions. Again, computing a Delaunay-Čech complex from this data with classical methods is practically infeasible on a desktop machine. We run the algorithm for $w = 5000$ iterations and obtain the persistence diagram in Figure 9. We observe that, as expected, a single point in the H_2 diagram is far from the diagonal, signaling the one-dimensional H_2 structure of the underlying 2-sphere in 10 dimensions.

Figure 9: Persistence of a noisy S^2 in $\mathbb{R}^{10}$.

TDA analysis of Natural Image Patches

To showcase further potential future use cases enabled by our algorithm, we apply TDA to the study of the topology of spaces of some 3×3 pixel image datasets. We, in particular, reproduce the experiment from [8] that considered a set of 3×3 patches of natural grayscale images. The data consists of pictures from van Hateren's image collection[34]. A detailed description of the data preprocessing is described in Sections 2 and 3 of [25]. Here, we will reiterate the key ideas. 5000 random patches are extracted from each image in the dataset. Then, the D -norm is defined as a measure of a contrast level of a patch, and 80% of image patches with the lowest contrast are removed as those which do not yield important information about the underlying space. All image patches are also normalized to have zero mean in pixel values and unit D -norm, forming a submanifold of S^7 . Furthermore, the authors of [8] continue the preprocessing for their experiments as follows. Firstly, 50000 patches are randomly selected among all available. Next, a value $\delta_k(x)$ is computed for each patch, where $\delta_k(x)$ is defined as the distance between x and the k^{th} nearest neighbor of x and is inversely related to a density function. Lastly, $T\%$ of patches in the highest density (with the lowest δ_k) are selected and others are removed. In the current article, we repeat the particular experiment with parameters $k = 15$ and $T = 30$, with the only change of adding a small noise to the data in order to avoid degenerate cases of Voronoi diagrams. In the original article, the authors use witness complexes[11] to estimate the topology of the space of images by selecting 50 landmark points in multiple repeated experiments.

Here, we instead run our algorithm on all $50000 \cdot 30\%$ patches with a number of $w = 10^4$ random walk steps to obtain a much larger Delaunay-Čech complex approximation of the data and the resulting the persistence diagrams presented in Fig.10b(b). We also revalidate the experiment from the original paper by constructing a witness complex using the Javaplex package[33]. By comparing the diagrams between the witness complex and our approach we can see that both algorithms report 5 persistent cycles in H_1 far from

the diagonal. The scaling in coordinates is however not directly comparable to [8] due to the differing numbers of landmarks. In our diagram, the 5 persistent points are approximately at x -coordinates around 0.08 with one darker shaded pair of points at almost the same (x, y) position. Note that due to a much larger number of points, an additional structure of the points closer to the diagonal is visible in the persistence diagram computed with our method that may warrant additional exploration. There are also further persistent features that have larger birth values and that may describe additional structure of the space of image patches.

Figure 10: Diagrams for van Hateren patches dataset

TDA analysis of Convolutional Filters

Next we consider a dataset of filters of convolutional networks. A related analysis has been performed in [18] with the use of Mapper[32] algorithm. We train 100 ResNet[22] networks with different random initializations on the image dataset CIFAR-10[24]. We pick the default structure of ResNet, described in Section 4.2 of [22] with parameter $n = 2$, obtaining a total of $3 \cdot 2 \cdot 2 = 12$ convolutional layers within the residual blocks. Then, we randomly select 10^4 filters from all the trained networks separately from each corresponding convolutional layer and, for each layer, we apply density filtering as in the case of image patches, introduced above, with parameters $k = 5$ and $T = 10$. We do not perform the removal of filters based on their contrast level in this task. Due to possibly different numerical scales of filters, we project them all on a hypersphere according to the Euclidean metric and add a small amount of Gaussian noise to avoid degenerate diagrams. The experiments are repeated several times with different filter sampling and random seeds for the algorithm. We then apply our algorithm to compute a 2-skeleton of the Delaunay-Čech complex of the data and compute the resulting persistence diagrams. Some of the diagrams for the convolutional layers are presented in Figure 11. We observe that some of the layers appear to contain the first persistence signature with points far from the diagonal (see, layer2.2.2 and layer3.2.1) while others do not present such structure with the selected filtering parameters (layer1.1.2 and layer1.2.1).

Piecewise Linear Interpolation

Figure 12 provides 2D and 10D examples of the traversed simplices during a visibility walk starting from a random simplex and walking towards the indicated red query point. Note that in 10D, the projection of a 10-simplex to 2D is a polyhedron with 11 vertices,

Figure 11: Diagrams for spaces of ResNet20 filters.

as illustrated in the right figure. We utilized 10^2 points in the 2D example and 10^6 in the 10D example, but only visualize a small subset of points in the 10D case in order to not clutter the figure. Figure 13 displays a toy dataset (blue) which is a union of $Ring^d$ and a small d -dimensional ball, $Ball^d$, of radius 0.25 centered at the origin. We uniformly sample points in proportion 10 : 1 from the ring and ball respectively. The figure also displays how the simplices containing various query points (red) look like in a 2D projection after the visibility walk has terminated.

Figure 12: Visibility walk, random initialization

Figure 13: Examples of detected simplices

Table 1 describes the average lengths and their standard deviations for visibility walks in a fixed dataset $Ball^d(10^4)$ of 10^4 uniformly sampled points in a d -dimensional ball, for $d = 2, 5, 10$ with uniformly sampled points from the same distribution. The column *random* denotes walks in which the descent has been initialized from a random generator, and the column *close* represents the walks descended from the nearest neighbor for each query. Rows *in* and *out* separate query points in two halves, points inside and outside the convex hull respectively. The table in particular illustrates the benefit of starting the visibility walk at the nearest neighbor, however with increasing dimensionality this benefit is reduced.

Table 1: Visibility walk in $B^d(10^4)$, number of steps

		random	close
2D	in	107.7 ± 50.5	2.8 ± 1.2
	out	109.2 ± 71.3	3.6 ± 1.9
5D	in	44.6 ± 12.7	10.8 ± 4.1
	out	50.5 ± 14.3	11.5 ± 4.7
10D	in	65.4 ± 14.2	34.6 ± 9.3
	out	65.1 ± 17.0	25.3 ± 11.7

In order to showcase the application to piecewise linear interpolation, we use the following dataset. Both train and query data are located on *Ring*⁶. While the query points are sampled uniformly from the annulus, the training points have a modification that the points' angles are sampled from a triangular distribution instead, see Figure 14a for a visual representation. This has been done to simulate the uneven distribution of training data. The size of the training set is 10^5 points, which makes it practically infeasible on a desktop computer to construct the full Delaunay complex in the 6-dim space; the size of the query set is 10^4 . With these values, the run time of the interpolation algorithm on Intel(R) Core(TM) i7-4930K is slightly below 2 minutes. For the interpolation task, we pick the function $f(x_1, \dots, x_n) = \cos 2x_1$, depicted in Figure 14b. Also, we do not interpolate points that are located inside of simplices with minball radius greater than 0.25; we assume that in this case the query points are located inside the hole and outside the convex hull of training data points. The approximation values are shown in Figure 14c. We also compare the MSE of piecewise linear interpolation (PLI) with KNN regression and RBF regression with a Gaussian kernel in Table 2. The parameters for the latter two algorithms were found via grid search over k and σ .

Figure 14: Interpolation example**Table 2: Mean square error on the described data**

PLI	KNN($k = 51$)	RBF($\sigma = 13.5$)
0.00007	0.00019	0.00011

5 FUTURE WORK

In this work, we have focussed on introducing a novel algorithm for the construction of $\text{Vor}(P)$ and $\text{Del}(P)$ in higher dimensions with an initial empirical evaluation in applications to TDA and interpolation. In future work, we would like to further investigate open theoretical questions, particularly about convergence, and error bounds. Another area that warrants detailed exploration is a study of run-times and the implementation of a parallelized version of our approach.

6 ACKNOWLEDGEMENTS

This work was supported by the Knut and Alice Wallenberg Foundation IPSYS grant.

REFERENCES

- [1] Sunil Arya, Theodoris Malamatos, and David M Mount. 2002. Space-efficient approximate Voronoi diagrams. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. 721–730.
- [2] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* 45, 6 (1998), 891–923.
- [3] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. 2013. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company.
- [4] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1996. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)* 22, 4 (1996), 469–483.
- [5] Ulrich Bauer and Herbert Edelsbrunner. 2017. The Morse theory of Čech and Delaunay complexes. *Trans. Amer. Math. Soc.* 369, 5 (2017), 3741–3762.
- [6] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. 2017. Phat-persistent homology algorithms toolbox. *Journal of symbolic computation* 78 (2017), 76–90.
- [7] Prosenjit Bose and Luc Devroye. 2007. On the stabbing number of a random Delaunay triangulation. *Computational Geometry* 36, 2 (2007), 89–105.
- [8] Gunnar Carlsson, Tigran Ishkhyanov, Vin De Silva, and Afra Zomorodian. 2008. On the local behavior of spaces of natural images. *International journal of computer vision* 76, 1 (2008), 1–12.
- [9] ZJ Cendes, D Shenton, and H Shahinasser. 1983. Magnetic field computation using Delaunay triangulation and complementary finite element methods. *IEEE Transactions on Magnetics* 19, 6 (1983), 2551–2554.
- [10] Sanjoy Dasgupta and Anupam Gupta. 1999. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report* 22, 1 (1999), 1–5.
- [11] Vin De Silva and Gunnar E Carlsson. 2004. Topological estimation using witness complexes. *SPBG* 4 (2004), 157–166.
- [12] Olivier Devillers and Ross Hemsley. 2016. The worst visibility walk in a random Delaunay triangulation is $O(\sqrt{n})$. (2016).
- [13] Olivier Devillers, Sylvain Pion, and Monique Teillaud. 2001. Walking in a triangulation. In *Proceedings of the seventeenth annual symposium on Computational geometry*. 106–114.
- [14] Rex A Dwyer. 1993. The expected number of k-faces of a Voronoi diagram. *Computers & Mathematics with Applications* 26, 5 (1993), 13–19.
- [15] Herbert Edelsbrunner. 1990. An acyclicity theorem for cell complexes in d dimension. *Combinatorica* 10, 3 (1990), 251–260.
- [16] Herbert Edelsbrunner and John Harer. 2010. *Computational topology: an introduction*. American Mathematical Soc.
- [17] Brittany Terese Fasy, Jisu Kim, Fabrizio Lecci, and Clément Maria. 2014. Introduction to the R package TDA. *arXiv preprint arXiv:1411.1830* (2014).
- [18] Rickard Brühl Gabrielsson and Gunnar Carlsson. 2018. Exposition and interpretation of the topology of neural networks. *arXiv preprint arXiv:1810.03234* (2018).
- [19] David Gillman. 1998. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.* 27, 4 (1998), 1203–1220.
- [20] François Godi. 2020. Bottleneck distance. In *GUDHI User and Reference Manual* (3.1.1 ed.). GUDHI Editorial Board. https://gudhi.inria.fr/doc/3.1.1/group__bottleneck__distance.html
- [21] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. <http://eigen.tuxfamily.org>.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [23] Shlomo Hoory, Nathan Linial, and Avi Wigderson. 2006. Expander graphs and their applications. *Bull. Amer. Math. Soc.* 43, 4 (2006), 439–561.
- [24] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
- [25] Ann B Lee, Kim S Pedersen, and David Mumford. 2003. The nonlinear statistics of high-contrast patches in natural images. *International Journal of Computer Vision* 54, 1-3 (2003), 83–103.
- [26] László Lovász et al. [n.d.]. Random walks on graphs: A survey. ([n. d.]).
- [27] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. 2009. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Vol. 501. John Wiley & Sons.
- [28] Vladislav Polianskii and Florian T Pokorny. 2019. Voronoi Boundary Classification: A High-Dimensional Geometric Approach via Weighted Monte Carlo Integration. In *International Conference on Machine Learning*. 5162–5170.
- [29] Donald R Sheehy. 2013. Linear-size approximations to the Vietoris–Rips filtration. *Discrete & Computational Geometry* 49, 4 (2013), 778–796.
- [30] Donald R Sheehy. 2014. The persistent homology of distance functions under random projection. In *Proceedings of the thirtieth annual symposium on Computational geometry*. 328–334.
- [31] Jonathan Richard Shewchuk. 2004. Stabbing delaunay tetrahedralizations. *Discrete & Computational Geometry* 32, 3 (2004), 339–343.
- [32] Gurjeet Singh, Facundo Mémoli, and Gunnar E Carlsson. 2007. Topological methods for the analysis of high dimensional data sets and 3d object recognition.. In *SPBG*. 91–100.
- [33] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. 2014. JavaPlex: A research software package for persistent (co)homology. In *Proceedings of ICMS 2014 (Lecture Notes in Computer Science 8592)*, Han Hong and Chee Yap (Eds.). 129–136. Software available at <http://appliedtopology.github.io/javaplex/>.
- [34] J Hans Van Hateren and Arjen van der Schaaf. 1998. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings of the Royal Society of London. Series B: Biological Sciences* 265, 1394 (1998), 359–366.
- [35] Shayne Waldron. 1998. The error in linear interpolation at the vertices of a simplex. *SIAM J. Numer. Anal.* 35, 3 (1998), 1191–1200.
- [36] Emo Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*. Springer, 359–370.

A RESOURCES

An up-to-date source code of the described algorithm can be found at <https://github.com/vlpolyansky/vgt>.

B ALGORITHMS

In this section, we include the pseudo-code for all core algorithms that are proposed in the article.

Algorithm 1 Random ray sampling

Require: $\{x_0, \dots, x_k\}$ set of generators of the Voronoi face

Ensure: u vector

```

function RANDRAY( $\{x_0, \dots, x_k\}$ )
  for  $i = 1$  to  $k$  do
     $v_i \leftarrow x_i - x_0$ 
    for  $j = 1$  to  $k - 1$  do
       $v_i \leftarrow v_i - \langle v_i, v_j \rangle v_j$ 
    end for
     $v_i \leftarrow v_i / \|v_i\|$ 
  end for
   $u \sim N_d(0, I)$ 
  for  $i = 1$  to  $k$  do
     $u \leftarrow u - \langle u, v_i \rangle v_i$ 
  end for
   $u \leftarrow u / \|u\|$ 
  return  $u$ 
end function

```

Algorithm 2 Ray casting (brute force)

Require: σ a Voronoi face

r a reference point within σ

u a direction vector

P the global set of all generators

Ensure: τ Voronoi face on the intersection

t length of the vector until the intersection

```

function RAYGEN( $\sigma, r, u$ )
   $(\tau^*, t^*) \leftarrow (\emptyset, \infty)$ 
  for all  $x$  in  $P$  do
    if  $x \in \sigma$  then
      continue
    end if
     $t \leftarrow \frac{\|r-x\|^2 - \|r-\bar{x}\|^2}{2\langle u, x-\bar{x} \rangle}$ 
    if  $0 \geq t < t^*$  then
       $(\tau^*, t^*) \leftarrow (\sigma \cup \{x\}, t)$ 
    end if
  end for
  return  $(\tau^*, t^*)$ 
end function

```

$\triangleright \bar{x}$ is any point from σ

Algorithm 3 Ray casting (intersection search)

Require: σ a Voronoi face

r a reference point within σ

u a direction vector

P the global set of all generators

T_{max} the maximum ray distance

ε binary search precision

$nn(\cdot)$ nearest neighbor search

Ensure: τ Voronoi face on the intersection

t length of the vector until the intersection

```

function RAYINT( $\sigma, r, u$ )
   $\tau = \emptyset$ 
   $(t_l, t_r) \leftarrow (0, T_{max})$ 
  while  $t_r - t_l > \varepsilon$  do
     $t_m \leftarrow \frac{t_l + t_r}{2}$ 
     $x \leftarrow nn(r + t_m u)$ 
    if  $x \in \sigma$  then
       $t_l \leftarrow t_m$ 
    else
       $t_r \leftarrow \frac{\|r-x\|^2 - \|r-\bar{x}\|^2}{2\langle u, x-\bar{x} \rangle}$ 
       $\tau \leftarrow \sigma \cup \{x\}$ 
    end if
  end while
  return  $(\tau, t_r)$ 
end function

```

Algorithm 4 Raycasting descent

Require: $P' = \{x_1, \dots, x_m\}$ (sub)set of generators

$randray(\cdot)$ uniform ray sampling within a Vor. face (Alg 1)

$ray(\cdot)$ raycast function (see Algorithm 2 & 3)

Ensure: S_0 subset of Voronoi vertices

```

function DESCENT( $P'$ )
   $S_d \leftarrow \{(\{x_i\}, x_i) \mid x_i \in P'\}$ 
  for  $k = d$  downto 1 do
    for all  $(\sigma, r)$  in  $S_d$  do
       $u \leftarrow randray(\sigma)$ 
       $(\tau, t) \leftarrow ray(\sigma, r, u)$ 
      if  $t = \infty$  then  $\triangleright$  in the direction is infinite
         $(\tau, t) \leftarrow ray(\sigma, r, -u)$ 
      end if
      if  $(\tau, \cdot) \notin S_{d-1}$  then
        Insert  $(\tau, r + tu)$  into  $S_{d-1}$ 
      end if
    end for
  end for
  return  $S_0$ 
end function

```

Algorithm 5 Random walk

Require: S_0 initial set of Voronoi vertices
 $nsteps$ number of steps
 $randray(\cdot)$ uniform ray sampling within a Vor. face (Alg 1)
 $ray(\cdot)$ raycast function (either Algorithm 2 & 3)
Ensure: updated S_0^*

```

function WALK( $S_0, nsteps$ )
   $S_0^* \leftarrow S_0$ 
  for all  $(v, r)$  in  $S_0$  do
    for  $s = 1$  to  $nsteps$  do
       $i \sim Uni(d + 1)$ 
       $e \leftarrow v \setminus \{x_i\}$ 
       $u \leftarrow randray(e)$ 
      if  $\langle u, x_i - x_j \rangle > 0$  then  $\triangleright$  if the direction is wrong
         $u \leftarrow -u$ 
      end if
       $(\tilde{v}, t) \leftarrow ray(e, r, u)$ 
      if  $t < \infty$  then
         $v \leftarrow \tilde{v}$ 
         $r \leftarrow r + tu$ 
        Insert  $(v, r)$  into  $S_0^*$ 
      end if
    end for
  end for
  return  $S_0^*$ 
end function

```

Algorithm 6 Visibility walk

Require: (v, r) starting Voronoi vertex with its coordinates
 $q \in \mathbb{R}^d$ a query point in the convex hull
 $barycentric(\cdot)$ barycentric coordinates
 $ray(\cdot)$ raycast function (either Algorithm 2 & 3)
Ensure: v final Voronoi vertex
 $\alpha = (\alpha_0, \dots, \alpha_d)$ interpolation coefficients

```

function VISWALK( $v, r, q$ )
  loop
     $\alpha \leftarrow barycentric(r, v)$ 
    if  $\alpha_i \geq 0$  for all  $0 \leq i \leq d$  then
      return  $v, \alpha$ 
    else
      repeat
        Remove the smallest (negative)  $\alpha_i$  from  $\alpha$ 
         $e \leftarrow v \setminus x_i$ 
         $u \leftarrow randray(e)$ 
        if  $\langle u, x_i - x_j \rangle > 0$  then
           $u \leftarrow -u$ 
        end if
         $(\tilde{v}, t) \leftarrow ray(e, r, u)$ 
      until  $t < \infty$ 
       $(v, r) \leftarrow (\tilde{v}, r + tu)$ 
    end if
  end loop
end function

```