HIGHER-DIMENSIONAL POWER DIAGRAMS FOR SEMI-DISCRETE OPTIMAL TRANSPORT

Philip Claude Caplan

Middlebury College Department of Computer Science, Middlebury, VT, U.S.A., pcaplan@middlebury.edu

ABSTRACT

Efficient algorithms for solving optimal transport problems are important for measuring and optimizing distances between functions. In the L^2 semi-discrete context, this problem consists of finding a map from a continuous density function to a discrete set of points so as to minimize the transport cost, using the squared Euclidean distance as the cost function. This has important applications in image stippling, clustering, resource allocation and in generating blue noise point distributions for rendering. Recent algorithms have been developed for solving the semi-discrete problem in $2d$ and $3d$, however, algorithms in higher dimensions have yet to be demonstrated, which rely on the efficient calculation of the power diagram (Laguerre diagram) in higher dimensions. Here, we introduce an algorithm for computing power diagrams, which extends to any topological dimension. We first evaluate the performance of the algorithm in $2d - 6d$. We then restrict our attention to four-dimensional settings, demonstrating that our power diagrams can be used to solve optimal quantization and semi-discrete optimal transport problems, whereby a prescribed mass of each power cell is achieved by computing an optimized power diagram.

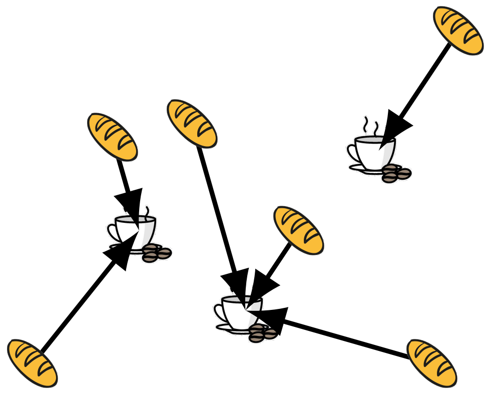
Keywords: Voronoi diagram, power diagram, Laguerre diagram, semi-discrete optimal transport, high dimensions, quantization.

1. INTRODUCTION

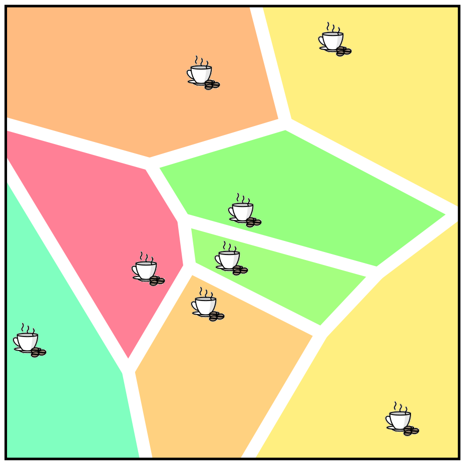
The problem of transporting mass from one location to another so as to minimize total cost was studied by Gaspard Monge in 1781 [1]. More generally, this problem consists of finding a map between two measures, and can be applied to problems in image processing [2, 3, 4, 5, 6, 7], machine learning [8] geometric processing [9], rendering [10, 11], resource allocation [12] and particle-based simulations [13, 14]. This problem also appears in the solution to partial differential equations, specifically when the two measures are continuous - see for example, the seminal work of Benamou and Brenier [15].

Another (possibly more important) application is Villani's example of transporting bread from bakeries to cafés [16]. There are a discrete number of bakeries, each with a specific baking capacity, and there are a discrete number of cafés – see Fig. 1. One may wish to distribute bread from bakeries to cafés so as to minimize the transport cost (Fig. 1a). This is a fully discrete optimal transport problem, since both the input and output measures are discrete. This type of problem is important in shape- and image-matching problems, supply chain management and clustering [17]. The revived interest in solving these types of optimal transport problems is due to the introduction of efficient algorithms that enable the computation of an optimal transport map for *large* data sets. These methods are mostly based on the idea of entropic regularization, a relaxation of the dual Kantorovich problem [18]. A noteworthy algorithm for solving this discrete problem is known as the Sinkhorn-Knopp algorithm [19].

If, instead, we can construct a model for a continuous population density across the city, we may wish to partition the city so as to equidistribute the bread de-



(a) Discrete optimal transport problem. Some cafés may be a in a busy part of town and require a lot of bread to be delivered.



(b) Semi-discrete optimal transport problem. The city is divided such that each café serves an equal population (integral of the population density over region).

Figure 1: Motivation for studying optimal transport: (a) transporting bread from bakeries to cafés and (b) assigning a population (with an assumed continuous density model) to cafés based on distance.

mand using the distance from the population to each café (Fig. 1b). This would be a semi-discrete optimal transport problem. If the cost function is the squared Euclidean distance, then we have an L^2 semi-discrete optimal transport problem. The optimal transport problem then consists of finding this partition such that every café has the same bread demand.

The solution to the latter problem can be obtained with a power diagram (or Laguerre diagram). The power diagram is a generalization of the Voronoi diagram in which sites are equipped with *weights* that control the power distance from a point to a particular site. The weights on these Voronoi sites can be tuned such that each region (a power cell) has an equal mass under the input density measure. In 1987, Aurenhammer observed that a power diagram can be computed via the restriction of a Voronoi diagram in a higher dimensional space [20, 21]. That is, the power diagram is the intersection of a higher-dimensional Voronoi diagram with our domain (in the bakery example, the city) embedded to a higher dimensional space. This concept of embedding has also been studied to achieve anisotropic Voronoi diagrams and meshes [22, 23, 24, 25, 26, 27]. In particular, Lévy and Bonneel introduced the *security radius theorem* which enabled the fast computation of the restricted Voronoi diagram [23]. In 2015, this technique was used for semi-discrete optimal transport in three dimensions, and has since been used in fluid simulations [13, 14] and astrophysics applications [28]. Power diagrams have also proven useful for computer graphics applications such as image stippling [2, 3, 4, 5], in which a blue noise sampling distribution is desirable to reduce aliasing effects that are produced by a regular sampling distribution, but produces better results than a white noise distribution. Optimal transport has also recently been used to obtain blue noise sampling distributions for rendering applications [10, 11].

Software implementations for computing power diagrams are available in `geogram` [29], `Voro++` [30] and `CGAL` [31, 32] in two and three dimensions. These implementations often rely on clipping Voronoi polygons or polyhedra using algorithms such as Sutherland-Hodgman re-entrant clipping [33], which are difficult to extend to a higher-dimensional setting. A demonstration of the power diagram calculation in four or higher dimensions has yet to be demonstrated. We make use of a simple result from polytope theory to convert between a facet-based and vertex-based representation of the power cells [34, 35]. It is well known that the number of vertices in a power cell grows exponentially with dimension, but the computation of the power diagram for semi-discrete optimal transport is nonetheless useful for higher-dimensional applications, particularly spatio-temporal simulations of partial differential equations, which would require four-dimensional power diagrams. Furthermore, Lévy and Bonneel's security radius theorem enables an efficient, emabarassingly parallel computation of the power cells, which makes the computation tractable in four dimensions.

The goal of this paper is to develop the theory and describe an algorithm for computing higher-dimensional power diagrams for semi-discrete optimal transport. To our knowledge, this is the first demonstration of an algorithm for higher-dimensional power diagrams (in terms of topological dimension) in the literature. We begin by reviewing the necessary background on semi-discrete optimal transport and power diagrams (Section 2) and then present our algorithm for computing power diagrams in a dimension-independent manner (Section 3). We evaluate the performance of the algorithm in $2d-6d$ (Section 4) and then apply the algorithm to quantization and L^2 semi-discrete optimal transport in four dimensions (Section 5), successfully demonstrating how a uniform target mass can be achieved under a prescribed density function.

2. BACKGROUND

Let us briefly review the optimal transport problem. For a more complete review, we encourage the interested reader to see the works of Lévy, [13, 36], Peyré [8], Santambrogio [37] and Villani [16, 38]. The optimal transport problem consists of finding a map that transports a probability measure μ supported on a d-dimensional domain $\mathcal{X} \subset \mathbb{R}^d$ onto another probability measure ν supported on another domain $\mathcal{Y} \subset \mathbb{R}^d$. The Monge formulation seeks the transport map $T : \mathcal{X} \to \mathcal{Y}$ from the following minimization statement:

$$\begin{aligned} T = \arg\min_T & \int_{\mathcal{X}} c(x, T(x))\,\mathrm{d}\mu(\mathbf{x}), \\ & \text{subject to :} \\ & \forall B \subset \mathcal{Y},\ \mu(T^{-1}(B)) = \nu(B), \end{aligned} \tag{1}$$

where $c(\cdot,\cdot) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is the *cost* of transporting mass from $\mathcal{X}$ to $\mathcal{Y}$. The constraint on T is a statement about conservation of mass for any Borel set $B \subset \mathcal{Y}$.

Kantorovich introduced a relaxation of the original Monge problem of Eq. 1, in which the *transport map* is replaced by a *transport plan* [39], thereby reformulating Monge's problem as a linear programming problem with convex constraints.

2.1 Semi-discrete optimal transport

In the semi-discrete setting, the source μ is a continuous measure, and the target measure ν is discrete, i.e. it is a sum of N Dirac masses: $\nu = \sum_{i=1}^N \nu_i \delta_{y_i}$. These Dirac masses are each located at a site $\mathbf{y}_i \in \mathbf{Y}$ where $\mathbf{Y} \in \mathbb{R}^{d\times N}$ can be interpreted as a matrix in which each column contains the coordinates of a site $\mathbf{y}_i$. The power cell $P_i(\mathbf{Y},\mathbf{w})$ for the Dirac mass at $\mathbf{y}_i$ is defined as

$$P_i(\mathbf{Y},\mathbf{w}) = \left\{\mathbf{x} \in \mathcal{X} \mid \forall j, c(\mathbf{x},\mathbf{y}_i) - w_i \le c(\mathbf{x},\mathbf{y}_j) - w_j\right\}. \tag{2}$$

The *Power Diagram* (or Laguerre Diagram) is then the union of all power cells, which is a partition of $\mathcal{X}$.

We now restrict our attention to the case in which the transport cost is the squared Euclidean distance, so $c(\mathbf{x},\mathbf{y}_i) = ||\mathbf{x}-\mathbf{y}_i||^2$. The dual Kantorovich formulation can be stated as a maximization of the following energy functional:

$$E(\mathbf{Y},\mathbf{w}) = \sum_{i=1}^N \int_{P_i(\mathbf{Y},\mathbf{w})} \rho(\mathbf{x})(||\mathbf{x}-\mathbf{y}_i||^2 - w_i)\,\mathrm{d}\mathbf{x} + \sum_{i=1}^N \nu_i w_i. \tag{3}$$

where we have replaced the mass $\mathrm{d}\mu(\mathbf{x}) = \rho(\mathbf{x})\mathrm{d}\mathbf{x}$ in terms of the prescribed density $\rho(\mathbf{x})$.

2.2 Optimizing the transport map

In this paper, we will consider two settings: (1) we want to optimize the sites $\mathbf{Y}$ to minimize E and (2) we want to optimize the weights $\mathbf{w}$ so as to maximize E. The former problem is known as *quantization*; a notable algorithm is *Lloyd relaxation* [40, 41] in which sites are iteratively moved to the centroids of their associated cells. That is, at each iteration of Lloyd relaxation, each site $\mathbf{y}_i$ is updated to it's centroid $\mathbf{c}_i$:

$$\mathbf{c}_i = \frac{\int_{V_i(\mathbf{Y})} \rho(\mathbf{x})\mathbf{x}\,\mathrm{d}\mathbf{x}}{\int_{V_i(\mathbf{Y})} \rho(\mathbf{x})\,\mathrm{d}\mathbf{x}}. \tag{4}$$

Note that we have replaced $P_i(\mathbf{Y},\mathbf{w}) = V_i(\mathbf{Y}) = P_i(\mathbf{Y},\mathbf{0})$ - i.e. we recover the Voronoi diagram when the weights are all zero. Lloyd relaxation is slow to converge, thus a gradient-based method is often employed, which requires the gradient of E with respect to the sites $\mathbf{y}_i$:

$$\frac{\mathrm{d}E}{\mathrm{d}\mathbf{y}_i} = 2m_i(\mathbf{y}_i - \mathbf{c}_i), \tag{5}$$

where m_i is the mass of the cell (the denominator in Eq. 4). Some Newton-based methods have been proposed, which requires the computation of the Hessian $\mathrm{d}^2E/\mathrm{d}\mathbf{y}_i\mathrm{d}\mathbf{y}_j$ [13]. For simplicity, we use a quasi-Newton approach, specifically the L-BFGS method [42, 43].

In the second setting, we wish to maximize Eq. 3 by optimizing the N weights $\mathbf{w}$. We can treat this as a minimization of $-E(\mathbf{Y},\mathbf{w})$, and use the derivatives:

$$\frac{\mathrm{d}E}{\mathrm{d}w_i} = \nu_i - \int_{P_i(\mathbf{Y},\mathbf{w})} \rho(\mathbf{x})\,\mathrm{d}\mathbf{x} = \nu_i - m_i, \tag{6}$$

where ν_i is the target (prescribed) mass of the power cell $P_i(\mathbf{Y},\mathbf{w})$. Since the energy is concave, it admits a unique maximizer [13]. Previous works have analyzed Newton-based approaches [44] for optimizing the weights, as well as multi-scale methods for large scale transport problems [45]. These approaches have been restricted to two- and three-dimensions due to existing software implementations for computing power diagrams [29, 30, 31, 32]. Here we strive to compute power diagrams in any dimension to support the solution of higher-dimensional semi-discrete optimal transport problems.

3. POWER DIAGRAMS IN HIGHER DIMENSIONS

Given N input sites $\mathbf{Y} = \{\mathbf{y}_i \mid \mathbf{y}_i \in \mathbb{R}^d\}_{i=1}^N$ and some weights $\mathbf{w} \in \mathbb{R}^N$ (a scalar weight is associated with each site), our goal is to compute the power diagram (with cells defined by Eq. 2) *restricted* to the domain $\mathcal{X} \subseteq \mathbb{R}^d$. This domain can either be specified as a mesh of d-polytopes or d-simplices, therefore, the term "restricted" means that we intersect $\mathcal{X}$ with $P_i(\mathbf{Y}, \mathbf{w})$ for each power cell.

Existing software implementations have been restricted to two- and three-dimensions and use geometric algorithms such as Sutherland-Hodgman re-entrant clipping [33] to compute the intersection of a polygon or polyhedron with a Voronoi bisector. The latter is needed in order to compute the vertex coordinates of a clipped polyhedron so as to apply the security radius theorem [23] and determine if clipping should be terminated. Here, we introduce an alternative view of the clipping procedure, which extends to any dimensional domain $\mathcal{X}$. Before describing our algorithm, we need a "simple" result from polytope theory.

3.1 Voronoi polytopes are simple

Our algorithm relies on the fact that Voronoi polytopes are *simple*, since they are the duals of Delaunay simplices. This enables the conversion between a facet-based and vertex-based representation of a Voronoi polytope. A convex d-dimensional polytope P can be described as the intersection of m halfspaces. Thus each point in the polytope can be described as the bounded solution set of m linear inequalities [34]

$$\mathcal{H}(P) = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{A}^T(\mathbf{x} - \mathbf{x}_0) \leq \mathbf{0}\}, \tag{7}$$

where each column of $\mathbf{A} \in \mathbb{R}^{d\times m}$ represents the normal to a facet. This is known as the *H-Representation* or *HRep* for short, denoted by $\mathcal{H}$. The *vertex enumeration problem* consists of converting the HRep to the *V-Representation* (*VRep*, denoted by $\mathcal{V}$), which is a description of the polytope as the convex hull of its n vertices:

$$\mathcal{V}(P) = \mathrm{conv}(\mathbf{V}), \tag{8}$$

where $\mathbf{V} \in \mathbb{R}^{d\times n}$ is a matrix with each vertex $\mathbf{v}_i \in \mathbb{R}^d$ stacked columnwise.

In the following, assume we have information regarding which facets are incident to every vertex, which is stored in the *vertex-facet-incidence matrix*, and will be denoted as $\mathbf{F}(\cdot) : \mathbb{N} \to \mathbb{Z}^k$, for some $k \geq 0$. Observe that a vertex v is identified using a nonnegative integer, whereas a facet b is labeled as an integer. This enables the distinction between a Voronoi bisector ($b \geq 0$) between two sites, and a mesh facet ($b < 0$), which is a $(d-1)$-dimensional facet of the input mesh. In order to compute the intersection of a d-polytope with a halfspace, we need the following definition from Henk [34].

Definition 1 (*simple polytope*)**.** *A d-polytope P is said to be* simple *if every vertex is incident to exactly d facets. A property of simple polytopes is that their dual polytopes are simplices.*

Thus, for Voronoi polytopes, $\mathbf{F}(\cdot) : \mathbb{N} \to \mathbb{Z}^d$ (i.e. $k = d$). We then have the following result, which is critical for our intersection algorithm.

Corollary 1. *The edges $\mathcal{E}(P)$ can be identified from the following relation on $\mathcal{V}(P)$:*

$$\mathcal{E}(P) = \left\{e = (v_0, v_1) \;\middle|\; |\mathbf{F}(v_0) \cap \mathbf{F}(v_1)| = d - 1\right\}. \tag{9}$$

The result is certainly true for $d \leq 4$ [34], but is important to consider for higher-dimensions. For *simple* polytopes, the dual of P is a simplex (the Delaunay simplex) from which the entire set of facets is trivially constructed. Furthermore, the hierarchy of the facets of a polytope can be obtained from the corresponding facet hierarchy of its dual [34], which contains the edges. Therefore, the edges $\mathcal{E}(P)$ of a simple d-polytope P can be derived purely from the vertex-facet-incidence matrix of P by traversing the facet-hierarchy of the dual simplex in reverse order. Thus we can determine if there is an edge between two vertices in the VRep of P if they share $d-1$ common facets.

For example, in Fig. 2, $\mathbf{F}(v_{i,1}) = \{b_1, b_2\}$ and $\mathbf{F}(v_{i,2}) = \{b_2, b_3\}$. Since vertices $v_{i,1}$ and $v_{i,2}$ share a single common facet (b_2), then they also share an edge. However, vertices $v_{j,3}$ and $v_{j,1}$ do not share an edge because $\mathbf{F}(v_{j,3}) \cap \mathbf{F}(v_{j,1}) = \emptyset$. Furthermore, we can identify that vertices $v_{i,2}$, $v_{j,5}$ and $v_{k,0}$ are all symbolically equivalent because they share a common bisector, thus enabling these vertices to be merged as a single vertex if necessary. In fact, these vertices represent the Delaunay simplex between sites $\mathbf{y}_i$, $\mathbf{y}_j$ and $\mathbf{y}_k$.

Each Voronoi cell will be computed from the intersection of a finite number of halfspaces, thus it is important to ensure that each cell remains a simple polytope after intersecting it with a halfspace, so that we can iteratively apply Eq. 9 for each bisector.

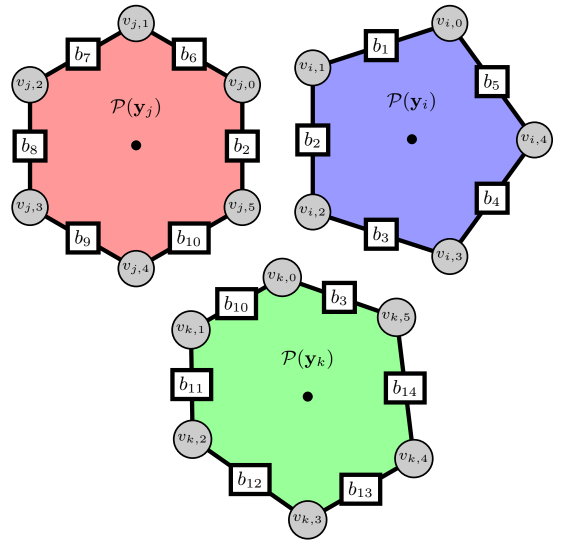


Figure 2: Computing the edges of a polytope from the vertex-facet-incidence relations. Three Voronoi cells are shown in the different colors and bisectors (possibly shared) are labeled in the squares.

Proposition 1. *A simple d-polytope $P \subset \mathbb{R}^d$ intersected with a halfspace $\mathcal{H}^+$ produces a simple polytope.*

Proof. Assuming the intersection of P with $\mathcal{H}^+$ is non-empty, it suffices to show that every vertex of the new polytope will be incident to exactly d facets. One way of describing the halfspace $\mathcal{H}^+$ is by using a unique point $\mathbf{x}_0$ and normal to the dividing hyperplane $\mathbf{n} \in \mathbb{R}^d$. Since P is simple, we can determine its edges $\mathcal{E}(P)$. Every vertex of the original polytope $v \in \mathcal{V}(P)$ can then be classified as to whether it is in the halfspace as

$$\mathcal{V}^+(P) = \left\{ v \in \mathcal{V}(P) \mid (\mathbf{x}(v) - \mathbf{x}_0) \cdot \mathbf{n} > 0 \right\}, \tag{10}$$

where $\mathbf{x}(v) \in \mathbb{R}^d$ are the coordinates of vertex v. The vertices created from the intersection are then computed by intersecting each edge with the dividing hyperplane (note we can filter which edges are intersected by finding edges with one vertex in $\mathcal{V}^+(P)$ and one that is not). Denote the set of intersection vertices as $\mathcal{S}$, therefore, the new vertices of the polytope Q are $\mathcal{V}(Q) = \mathcal{V}^+(P) \cup \mathcal{S}$. There are two cases to consider. First, the vertices $\mathcal{V}^+(P)$ are clearly adjacent to d facets since they were not affected by the intersection. Second, the vertices in $\mathcal{S}$ were each created from the intersection of an edge with the hyperplane. Since edges are adjacent to $d-1$ facets (Eq. 9) and the dividing hyperplane, itself, defines a facet of Q, then each intersection vertex is adjacent to d facets. □

We are now ready to describe our algorithm for computing power cells.

3.2 Computing the power cells

In fact, we simply compute the Voronoi cells! As noted earlier, Aurenhammer observed that the power diagram is the intersection of a higher-dimensional Voronoi diagram with our d-dimensional space lifted to the same ambient dimension of this Voronoi diagram [20, 21]. We use the result of Lévy [36] to lift the Voronoi sites (equipped with weights) to a $(d+1)$-dimensional space, thereby obtaining $\mathbf{Z} \in \mathbb{R}^{(d+1)\times N}$ in which

$$\mathbf{z}_i = \left(\mathbf{y}_i^T, \ \sqrt{\max(\mathbf{w}) - w_i} \right)^T, \quad \forall i = 1, 2, \ldots, N. \tag{11}$$

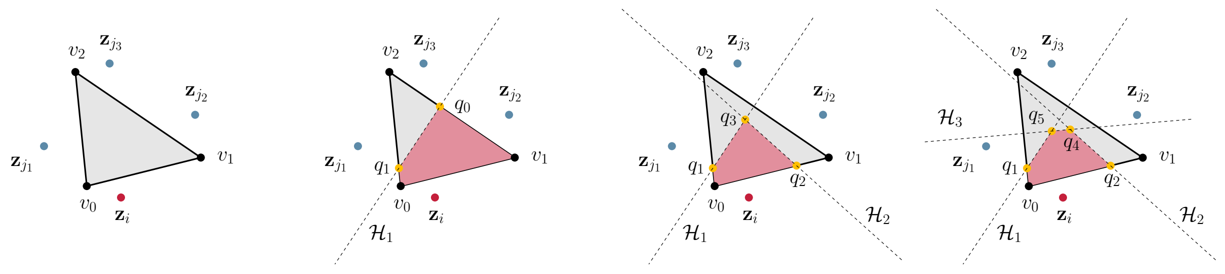
We additionally need to lift our domain of interest to $\mathbb{R}^{d+1}$, which is done by simply appending a zero to the coordinates of the domain (the vertices of the mesh). As a result, we only need to compute the Voronoi diagram in $\mathbb{R}^{d+1}$. That is, the power cells are $P_i(\mathbf{Y}, \mathbf{w}) = V_i(\mathbf{Z})$.

Although the domains we study in this paper can be entirely described with a single polytope (a d-cube), we will consider the general case in which the domain to be clipped against is represented as a simplicial mesh. The mechanics of the algorithm are the same, but this description lends well to future work for the interested reader.

We begin with a single element, here represented by a d-simplex κ. The vertex-facet incidence relations for each vertex of κ are the *mesh facets* (the $(d-1)$-simplices) incident to a particular vertex. As mentioned earlier, these mesh facets are labeled with negative integers so as to distinguish them from Voronoi bisectors. Our goal is to compute the Voronoi cell associated with site $\mathbf{z}_i$, clipped with the element κ. In the following description, please follow along with Fig. 3.

We begin by clipping κ with the Voronoi bisector $\mathcal{H}_1$ defined by $\mathbf{z}_i$ and it's nearest neighbor, $\mathbf{z}_{j_1}$. In order to determine the clipped polytope (in red), we first identify the edges of κ using Eq. 9. Next, we determine which edges have vertices on either side of the Voronoi bisector $\mathcal{H}_1$. Those with vertices on either side of $\mathcal{H}_1$ are then intersected with $\mathcal{H}_1$, creating new vertices q_0 and q_1. We then set $\mathbf{F}(q_0) = \mathbf{F}(v_1) \cap \mathbf{F}(v_2) \cup \{b_{i,j_1}\}$, where b_{i,j_1} is the unique integer label assigned to the Voronoi bisector between sites i and j_1. Furthermore, $\mathbf{F}(q_1) = \mathbf{F}(v_0) \cap \mathbf{F}(v_2) \cup \{b_{i,j_1}\}$. The geometric coordinates of $\mathbf{q}_0$ and $\mathbf{q}_1$ are computed and used to check if clipping should be terminated, using the radius of security theorem [23].

If clipping should continue, we then proceed to the next nearest neighbor of site $\mathbf{z}_i$, which is $\mathbf{z}_{j_2}$. A similar procedure ensues, this time, starting with the polytope defined by vertices $\{v_1, q_0, q_1, v_2\}$. Extracting the edges (Eq. 9) and intersecting those with ver-



(a) Initial simplex κ. (b) Clipping with $\mathcal{H}_1$. (c) Clipping with $\mathcal{H}_2$. (d) Clipping with $\mathcal{H}_3$.

Figure 3: Computing the intersection of a mesh element (here, a simplex) with a Voronoi cell defined by the site $\mathbf{z}_i$. Clipping starts with the Voronoi bisector $\mathcal{H}_1$ defined by the nearest neighbor to site $\mathbf{z}_i$ ($\mathbf{z}_{j_1}$) and proceeds to the next neighbors, thus clipping by $\mathcal{H}_2$ and then $\mathcal{H}_3$ from left to right.

tices on either side of $\mathcal{H}_2$ reveals that q_0 is no longer in the Voronoi cell. New vertices q_2 and q_3 are introduced from the intersection of the edges with $\mathcal{H}_2$. Then $\mathbf{F}(q_2) = \mathbf{F}(v_0) \cap \mathbf{F}(v_1) \cup \{b_{i,j_2}\}$ and $\mathbf{F}(q_3) = \mathbf{F}(q_0) \cap \mathbf{F}(q_1) \cup \{b_{i,j_2}\}$.

Proceeding in a similar fashion to the next nearest neighbor $\mathbf{z}_{j_3}$ clips the current polytope defined by vertices $\{v_0, q_2, q_3, q_1\}$ (no need for them to be ordered counterclockwise). We then end up with the polytope $\{v_0, q_2, q_4, q_5, q_1\}$ where $\mathbf{F}(q_4) = \mathbf{F}(q_2) \cap \mathbf{F}(q_3) \cup \{b_{i,j_3}\}$ and $\mathbf{F}(q_5) = \mathbf{F}(q_1) \cap \mathbf{F}(q_3) \cup \{b_{i,j_3}\}$. We suppose the clipping terminates after checking the coordinates of this polytope with the radius of security theorem.

This procedure is described in Algorithm 1. The inputs to the algorithm are a Voronoi site $\mathbf{z}_i$, a mesh element κ and a nearest neighbor structure $\mathcal{N}$ that computes the nearest sites upon request. This structure is initialized to 50 nearest neighbors at the start of clipping - we expect well distributed mesh vertices to have a simplex valency of 6 in $2d$, $15-20$ in $3d$ and about 120 in $4d$ [46]. Whenever the bounds of the nearest neighbors are reached, an additional 10 neighbors are appended to the current neighbors. As we will examine in the results section, the cost of computing the nearest neighbors is overshadowed by clipping, simplex decomposition and numerical integration.

Clipping continues until the security radius is reached, which is equal to twice the maximum distance from any vertex in P to $\mathbf{z}_i$ (we absorb the factor of 2 into the definition of the radius of security - a slight modification of Lévy's definition). Each edge is then intersected with the bisector defined by sites $\mathbf{z}_i$ and $\mathbf{z}_j$ - the j^{th} nearest neighbor to $\mathbf{z}_i$. When an intersection occurs, only one of e_0 or e_1 (the edge vertices) is in the halfspace defined by $\mathcal{H}$ which includes $\mathbf{z}_i$ (i.e. $\mathcal{H}^+$). This vertex, along with the intersection vertex is then appended to the set of vertices defining the clipped polytope Q, and the vertex-facet incidence relations are updated.

In the current work, we parallelize the clipping procedure over the Voronoi sites in contrast to a parallelization over the mesh elements. The reason is because we keep our domains are defined by a single d-cube but we seek power diagrams defined by sometimes millions of Voronoi sites. Furthermore, we parallelize the computation on the CPU, specifically with `OpenMP`, however the computation is well suited for parallelization on the GPU, which will be explored as future work.

Eq. 9 suggests that extracting the polytope edges is quadratic in the number of vertices in the current clipped polytope. Since the number of vertices in a Voronoi polytope is exponential in the dimension of the polytope, this could incur a significant computational cost - especially since this extraction is repeatedly performed until the radius of security is reached. Therefore, we reduce the computational cost by (1) retaining any edges that lie entirely within $\mathcal{H}^+$ (the side of the hyperplane containing the site $\mathbf{z}_i$), (2) updating the endpoints of any intersected edges with indices of the new intersection vertices and (3) computing new edges that lie exactly on $\mathcal{H}$ using Eq. 9. Step (3) is still quadratic in the number of vertices that lie on $\mathcal{H}$, but since these define a $(d-1)$-dimensional polytope, the cost remains reasonable for our applications ($d \leq 6$).

A note regarding exactness In this paper, we are purely interested in the *geometry* of the power cells and do not extract any *topological* information corresponding to the dual Delaunay mesh. Although this topological information could be extracted by merging vertices that have identical symbolic information (e.g. vertices $v_{i,2}$, $v_{j,5}$ and $v_{k,0}$ in Fig. 2), this computation is susceptible to numerical precision issues. Exact geometric predicates, such as an `orientnd` function, similar to Shewchuk's `orient2d` and `orient3d` functions could be used to detect cosphericities [47]. In the con-

```
computeVoronoiCell

    input: site $\mathbf{z}_i$, element $\kappa$, neighbors $\mathcal{N}$
    output: Voronoi cell as a $d$-polytope $P$

 1  $P \leftarrow \kappa$ // initialize to domain element
 2  $j = 1$ // start with nearest neighbor
 3  $\mathbf{z}_j = \mathcal{N}(\mathbf{z}_i, j)$
 4  while $||\mathbf{z}_i - \mathbf{z}_j|| <$ securityRadius($\mathbf{z}_i, P$)
 5      $\mathcal{H} \leftarrow \mathcal{H}(\mathbf{z}_i, \mathbf{z}_j)$ // define bisector
 6      $Q \leftarrow \emptyset$ // initial clipped polytope
 7      $\mathcal{E} \leftarrow \mathcal{E}(P)$ // from Eq. 9
 8      for $e = (e_0, e_1) \in \mathcal{E}$
 9          $s_0 =$ side($e_0, \mathcal{H}$)
10          $s_1 =$ side($e_1, \mathcal{H}$)
11          if $s_0 \equiv s_1$ // no intersection
12              continue
13          $p \leftarrow e_0$ or $e_1$ (whichever has $s > 0$)
14          $\mathbf{q} \leftarrow e \cap \mathcal{H}$ and set $q \leftarrow |Q|$
15          $\mathbf{F}(q) \leftarrow \mathbf{F}(e_0) \cap \mathbf{F}(e_1) \cup b(\mathcal{H})$
16          $Q \leftarrow Q \cup \{p, q\}$
        end
17      $P \leftarrow Q$ // update current polytope
18      $j = j + 1$ // proceed to next neighbor
19      $\mathbf{z}_j = \mathcal{N}(\mathbf{z}_i, j)$
    end
```

Algorithm 1: Computing the intersection of a mesh element κ with a Voronoi cell defined by a site $\mathbf{z}_i$.

text of our algorithm, the calls to the `side` function on Lines 9 and 10 should be replaced with a `side_rd` function [48] which computes the intersection of a r-dimensional simplex with the bisector $\mathcal{H}$. In order to determine which r-simplex should be used, we simply need to track which simplex facets of the original mesh are incident to each vertex. This can be done by tracking the "simplex vertices" of every clipped vertex: $\mathbf{S}(v)$. Any time an intersection point is introduced, the new vertex inherits all the simplex vertices of the endpoint edge vertices. For example, in the rightmost diagram of Fig. 3, $\mathbf{S}(q_5) = \mathbf{S}(q_3) \cup \mathbf{S}(q_1) = \mathbf{S}(q_0) \cup \mathbf{S}(q_1) = \{v_0, v_1, v_2\}$, meaning q_5 is the intersection of the d-dimensional simplex $\kappa = (v_0, v_1, v_2)$ with the bisectors $\mathcal{H}_1 \cap \mathcal{H}_2 \cap \mathcal{H}_3$.

3.3 Numerical integration

Computing the objective function and gradients of Eqs. 3, 5 and 6 requires the calculation of integrals over the resulting Voronoi polytopes. To perform the integration, we decompose the d-polytopes into a set of d-simplices by introducing a vertex at the centroid of the polytope and then recurse through the $(d-1)$-facets, continuously introducing vertices at these facet centroids until the edges are reached (at which point the triangulation is trivial). The simplicial decomposition of the d-polytope is obtained by reconnecting the simplicial facets with the centroids. This mesh of d-simplices is then used to perform the integration with numerical quadrature rules of Stroud [49]. Although this evaluation is also performed in parallel over the integration simplices, the integral evaluation is a bottleneck in our algorithm, which will be discussed in the results section.

3.4 Visualizing the power diagram

In the results section, we will present visualizations of four-dimensional power diagrams, thus it is necessary to briefly describe our procedure for doing so. Some previous work in $4d$ mesh visualization includes the work of Caplan [46] and Belda-Ferrín [50].

Our method for visualizing a d-dimensional mesh first consists of identifying which polytopes of this mesh are cut by some input viewing volume, represented as a $(d-1)$-dimensional hyperplane with point $\mathbf{x}_0$ and normal $\mathbf{n}$. Next, each polytope P must be intersected with the hyperplane. We cull polytopes that are not clipped if all their vertices lie on the same side of the hyperplane, using the predicate in Eq. 10. Otherwise, the polytope is clipped by the hyperplane. We then identify which edges have vertices on either side of the hyperplane, and then compute the intersection point of the hyperplane with the edge. Each intersection point is then appended to a list of vertices that lie *exactly* on the hyperplane, which defines a $(d-1)$-dimensional polytope that can be directly visualized if a suitable visualization dimension (such as $2d$ or $3d$) is reached.

For high-dimensional ($d > 4$) applications, a recursive procedure can be applied to this $(d-1)$-dimensional mesh along with a $(d-2)$-dimensional hyperplane. In the results section, we only visualize $4d$ meshes through (1) the intersection of a $3d$ viewing volume or (2) a second intersection with a $2d$ clipping plane. In order to view the polyhedra resulting from the intersection, we tetrahedralize the convex polyhedra using a Delaunay triangulator [51] and visualize the mesh edges by applying Eq. 9 to extract the edges of each polyhedron clipped to the $3d$ viewing volume.

4. PERFORMANCE

In this section we evaluate the performance of our algorithm for computing d-dimensional power diagrams (of d-polytopes) for $d = 2$ to $d = 6$. All tests are performed with an Intel Xeon W-2145 CPU at 3.70GHz with 8 cores (16 threads).

Our first performance test consists of distributing N points within a unit d-cube and measuring the time to compute the power diagram. We consider two point distributions: (1) a completely random white noise distribution and (2) a blue noise distribution. The reason we consider the former is because the nearest neighbor calculation is more costly for a white noise distribution of points, whereas the number of nearest neighbors is more uniform in the case of a blue noise distribution. The blue noise sampling distributions were obtained with the `SpokeDarts` software of Mitchell et al. [52]. An example of the resulting Voronoi diagrams for four-dimensional white and blue noise point distributions, sliced along the fourth dimension, are shown in Fig. 4.

Performing the calculation in 2- through 6-dimensional domains (where the ambient dimension is equal to the topological dimension of the domain) reveals a good scaling of the algorithm in low dimensions, particular in $2d$ through $4d$. Fig. 5 reveals that our calculation is more costly for random point distributions than for blue noise distributions. Also observe that the cost of computing the Voronoi diagram in $5d$ and $6d$ is significantly higher than in low dimensions, which is due to the fact that the number of vertices in a Voronoi polytope grows exponentially with the dimension. Fig. 6 shows how the number of vertices and facets grows with the input number of points (Voronoi sites) for a white noise distribution.

In our second performance study, we focus on dimensions 2-4 and compute (1) the time to clip the Voronoi cells (as in the previous study), (2) the total time devoted to compute nearest neighbors, (3) the time devoted to decomposing the polytopes into integration simplices and (4) the time to perform the numerical integration for various quadrature orders q. The results are shown in Table 1. In $2d$ (polygons) and $3d$ (polyhedra), the cost of the Voronoi diagram calculation outweighs any other calculation, but is still relatively fast for one million points in $2d$ (25 seconds). The cost of decomposing the Voronoi cells into simplices is roughly one fifth the cost of computing the Voronoi diagram in $3d$ and $4d$. In higher dimensions, the cost of integration is limiting and begins to exceed the cost of computing the Voronoi diagram, even for low quadrature orders ($q > 2$). In general, the cost of computing the Voronoi diagram is even lower than the results shown in Table 1 (which is for white noise distributions) once the point distributions exhibit more structure.

5. APPLICATIONS

We now use our power diagrams to solve quantization and semi-discrete optimal transport problems in four-dimensional domains. The difference between the two problems is that (1) in quantization, we optimize Voronoi site coordinates and (2) in optimal transport, we optimize the weights on Voronoi sites.

Table 1: Performance statistics for white noise in $2d-4d$ for a varying number of Voronoi sites (N). The timing results (in seconds) are broken into (1) the time to compute the Voronoi diagram (t_{vor}), (2) the time devoted to computing nearest neighbors (t_{knn}), (3) the time to decompose the d-polytopes into d-simplices (t_{tri}) and (4) the time to perform the numerical integration a particular quadrature order q (t_q).

d	N	t_{vor}	t_{knn}	t_{tri}	$t_{q=2}$	$t_{q=3}$	$t_{q=4}$
2	10k	0.037	0.085	0.065	0.029	0.039	0.049
2	100k	2.7	0.71	0.46	0.17	0.26	0.40
2	1M	25	12	4.2	1.3	2.2	3.5
3	10k	2.8	0.10	0.41	0.35	1.0	2.2
3	100k	29	1.5	4.4	3.4	10	23
3	250k	71	5.2	11	8.6	25	59
4	1k	4.0	0.027	0.82	1.6	7.3	23
4	10k	50	0.40	9.8	18	87	270
4	15k	78	0.68	15	28	130	420

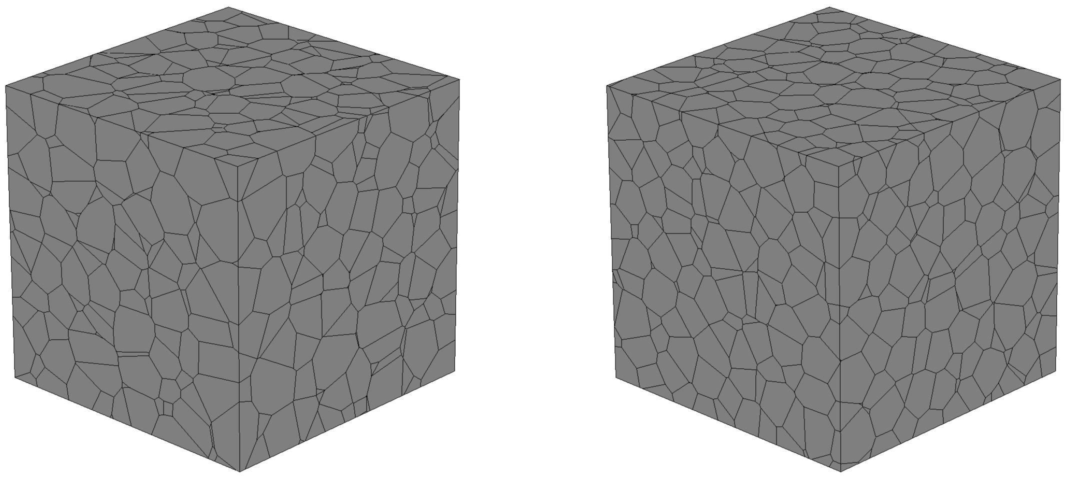
5.1 Quantization

Here, our goal is to compute an optimal point distribution for some prescribed density function. As we observed in the performance study, the cost of evaluating the integrals is prohibitive for a large number of points. As a result, we design the densities to be integrated more accurately with lower quadrature orders.

```
optimizePoints($N, \rho(\mathbf{x})$)

    input: number of sites $N$, density $\rho(\mathbf{x})$
    output: optimized point distribution $\mathbf{Y}$
 1  $\mathbf{Y} \leftarrow$ randomly sample $N$ points in domain $\Omega$
 2  iter $\leftarrow 0$
 3  while iter $<$ nb_iter
 4      compute RVD: $\mathrm{Vor}(\mathbf{Y}) \cap \Omega$ (Section 3.2)
 5      compute mass and centroids in Eq. 4
 6      if lloyd // use Lloyd relaxation
 7          update $\mathbf{x}$ to current centroids
 8      else // use L-BFGS update
 9          compute gradients $\mathrm{d}E/\mathrm{d}\mathbf{y}_i$ using Eq. 5
10          perform L-BFGS update on $\mathbf{x}$
        end
    end
```

Algorithm 2: Optimizing a point distribution according to an input density measure, using either Lloyd relaxation or an L-BFGS update. In our applications, the input domain Ω is the unit d-cube represented as a polytopal mesh with a single element.



(a) White noise. (b) Blue noise.

Figure 4: Four-dimensional Voronoi diagrams for white and blue noise point distributions for 16,332 sites embedded in $4d$. The images represent the slices of the Voronoi diagram along a hyperplane at $t = 0$ where t is the fourth dimension.

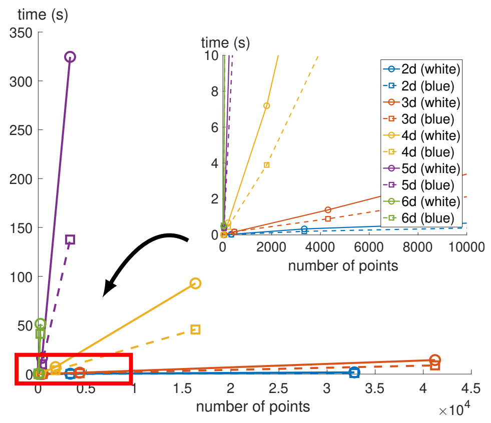


Figure 5: Time to compute the Voronoi diagram in $2d$-$6d$ for white and blue noise distributions with a varying number of points (Voronoi sites).

We consider three density measures. The first is a uniform distribution $\rho_u(\mathbf{x}) = 1$ (the optimized Voronoi cells should appear uniform). The second density measure is a Gaussian

$$\rho_g(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^4 \det \Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \tag{12}$$

where $\boldsymbol{\mu} = (0.5, 0.5, 0.5, 0.5)^T$ and $\Sigma = \text{diag}(0.02, 0.02, 0.02, 0.02)$. In this case, we expect smaller Voronoi polytopes near $\boldsymbol{\mu}$, which then increase in distance around this mean point. The third density measure we consider is that which describes an expanding sphere, which traces the

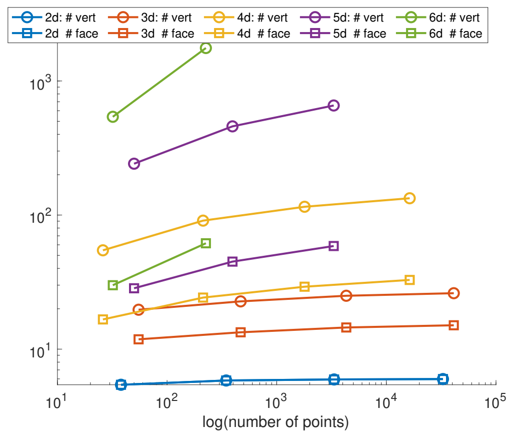


Figure 6: Total number of vertices (circles) and facets (squares) for Voronoi diagrams computed from a random point distribution in $2d-6d$ for a varying number of input points (Voronoi sites).

geometry of a hypercone in $4d$. In particular, this density measure is

$$\rho_c(\mathbf{x}) = 100/(h^2 + 0.001) \tag{13}$$

where h is the distance to the cone defined by $\mathbf{r}(t) = r_0 + t \cdot \tan(\alpha)$ where t represents the fourth coordinate in the domain (to be interpreted as time). Please refer to Fig. 7 for the geometry of the cone - note that $r_0 = 0.4$ and $r_1 = 0.7$. Note that this density measure is effectively two-dimensional in an $r - t$ coordinate system. Rotational symmetry about the t axis leads to the cone geometry. In fact, when slicing the

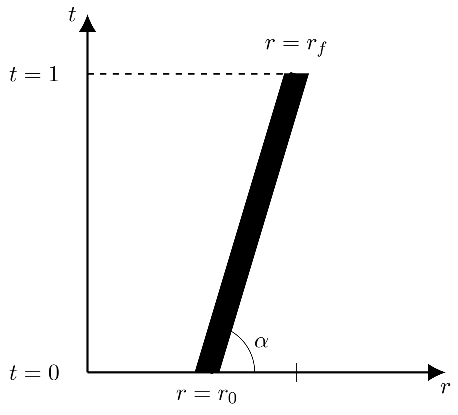


Figure 7: Definition of the hypercone in a radial-temporal coordinate system. The sphere starts with a radius of $r_0 = 0.4$ at $t = 0$ and expands to a radius of $r_1 = 0.7$ at $t = 1$.

resulting Voronoi diagram (with points uniformly distributed according to $\rho_c(\mathbf{x})$), we should expect to see smaller cells clustering around the geometry of a $3d$ cone when slicing along a non-constant t hyperplane, but should expect to see smaller cells clustering around a $3d$ sphere at constant t hyperplanes.

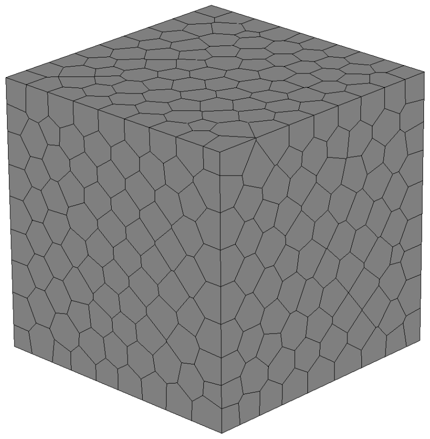
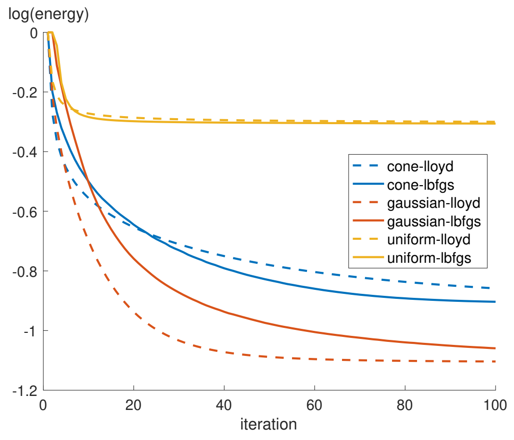
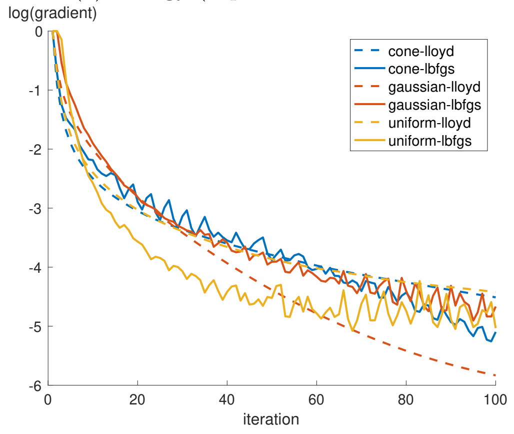


Figure 8: Optimized four-dimensional Voronoi diagram sliced at $t = 0$ (where t is the fourth dimension) for the case of a uniform density, $\rho_u(\mathbf{x})$.

We optimize four-dimensional point distributions with $N = 10,000$ points using Lloyd relaxation [40] the gradient-based L-BFGS optimizer [43] - this procedure is outlined in Algorithm 2. After 100 iterations (function calls in the `nlopt` optimizer), the point distributions indeed exhibit a uniform distribution under the prescribed density measure. Fig. 8 shows uniformly distributed polytopes sliced along the $t = 0$ hyperplane. Fig. 10 shows the interior of a slice along the $t = 0.5$ hyperplane for the Gaussian density. The clustering of Voronoi cells around the mean is clear and the cell sizes increases with distance from the mean. Furthermore, the expected clustering around the hypercone is visible in Fig. 11. Specifically, when the four-dimensional Voronoi diagram is sliced along a hyperplane with non-constant t ($x = 0$), we can see clustering near a $3d$ cone. We can also see smaller polytopes clustered around a sphere when the optimized Voronoi diagram is sliced at $t = 0$ (smaller sphere with radius r_0) and $t = 1$ (larger sphere with radius r_1). Fig. 9 demonstrates the convergence of the en-



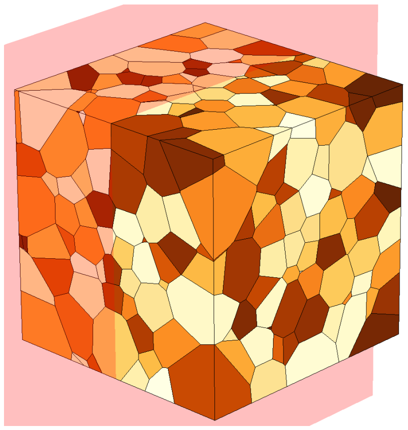
(a) Energy (Eq. 3 versus iteration.



(b) Gradient norm (of Eq. 5) versus iteration.

Figure 9: Convergence of the energy and its gradient for Lloyd relaxation and LBFGS optimizer for all three density functions (uniform, Gaussian, cone) considered in the quantization application.

ergy functional and gradient norm during the optimization for both Lloyd relaxation (even though it is not driven by the gradient) and the L-BFGS optimizer. The data in each curve is normalized by the initial energy and gradient norm at the onset of the optimization. The L-BFGS optimizer achieves a lower energy and gradient norm for the uniform and cone density measures, but performs slightly worse than Lloyd relaxation for the Gaussian density. Nonetheless, the results demonstrate the ability to obtain point distributions that are uniform with respect to a prescribed density function in $4d$ for a relatively large number of points ($N = 10,000$).


(a) Optimized Voronoi diagram sliced at $t = 0.5$ with clipping plane shown.

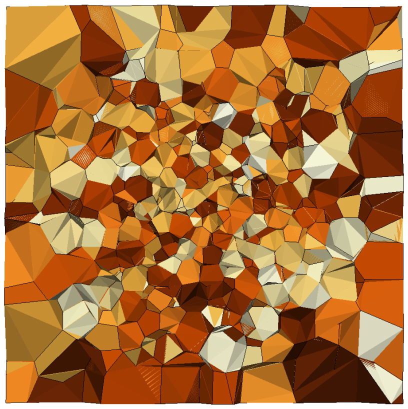
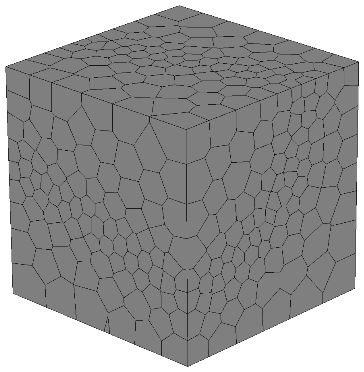
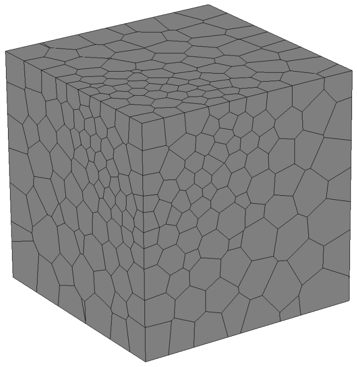

(b) Optimized Voronoi diagram sliced at $t = 0.5$, clipped at $x = 0.5$ (transparent clipping plane shown in pink above).

Figure 10: Optimized four-dimensional Voronoi diagram sliced at $t = 0.5$ (where t is the fourth dimension) for the case of a Gaussian density, $\rho_g(\mathbf{x})$. The bottom figure shows a slice (at $x = 0.5$) of the top figure, where the clipping plane is defined by the transparent plane shaded in pink.


(a) Voronoi diagram slice at $x = 0$.


(b) Voronoi diagram slice at $t = 0$.

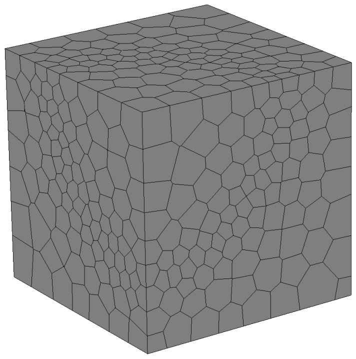

(c) Voronoi diagram slice at $t = 1$.

Figure 11: Optimized four-dimensional Voronoi diagram sliced at $t = 0$, $t = 1$ (where t is the fourth dimension) and $x = 0$ for the case of a cone-like density, $\rho_c(\mathbf{x})$. A three-dimensional cone is visible at $x = 0$ (where the smaller cells cluster), whereas the sphere with initial radius r_0 and final radius r_1 is shown at $t = 0$ and $t = 1$, respectively. Only one quarter of the cone and spheres are visible within this domain.

5.2 Semi-discrete optimal transport

We now turn our attention to the problem of assigning an equal mass (measured under some input density function) to each Voronoi cell. We will study smaller problem sizes because we do not currently implement a multiscale algorithm to initialize the weights for the next "level," although this would certainly accelerate the convergence of the optimization as pointed out by

```
optimizeWeights($N, \rho(\mathbf{x})$)

   input: number of sites $N$, density $\rho(\mathbf{x})$
   output: power cells with uniform mass
1  $\mathbf{Y} \leftarrow$ optimizePoints($N, \rho$)
2  $\mathbf{w} \leftarrow \mathbf{0}$
3  $\nu_t \leftarrow m_t/N$ // $m_t$ is the total mass
4  for iter $= 1 :$ nb_iter
5      $\mathbf{Z} \leftarrow$lift $\mathbf{Y}$ to $\mathbb{R}^{d+1}$ (Eq. 11)
6      compute Vor($\mathbf{Z}$) $\cap\, \Omega$ (Section 3.2)
7      compute energy and mass in Eq. 4
8      compute gradients $\mathrm{d}E/\mathrm{d}w_i$ using Eq. 6
9      perform L-BFGS update on $\mathbf{w}$
           with $-E$ and $-\mathrm{d}E/\mathrm{d}\mathbf{w}$
   end
```

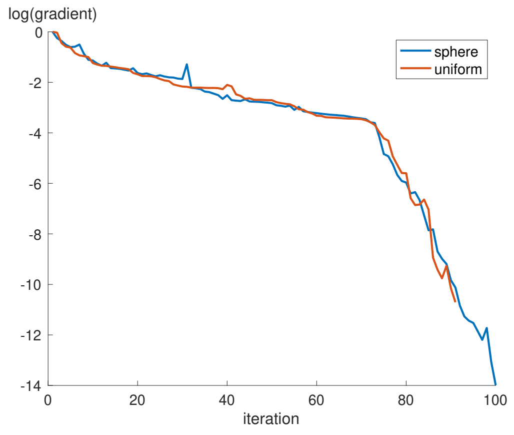
Algorithm 3: Optimizing the weights to achieve a target mass according to an input density measure. In our applications, the input domain Ω is the unit d-cube represented as a polytopal mesh with a single element.

Mérigot [45, 18] and Lévy [36]. We thus use $N = 1000$ points distributed within a four-dimensional domain.

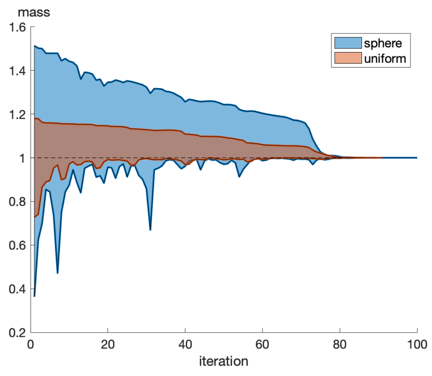
The two density measures we consider here are (1) a uniform density $\rho_u(\mathbf{x}) = 1$ and (2) a spherical density $\rho_s(\mathbf{x}) = 1 + 100\,||\mathbf{x} - \boldsymbol{\mu}||^2$ where $\boldsymbol{\mu} = (0.5, 0.5, 0.5, 0.5)^T$. In contrast to the previous section which employed a Gaussian as the density measure, this spherical density is more accurately integrated with lower quadrature orders, thus keeping the computational cost reasonable for this demonstration.

Before optimizing the weights, we first optimize the point distributions with the L-BFGS method (of the previous section) to achieve a uniform distribution with respect to the input density measure - see Algorithm 3. We then obtain the total mass m_t by integrating the input density over the entire domain and set the target mass $\nu_t = m_t/N$ for every site. Finally, we optimize the weights by iteratively computing the power diagram and integrating the quantities in Eq. 6 which are then passed to the L-BFGS optimizer to determine the next set of weights.

The convergence of the energy functional and gradient norm (Eq. 6) is shown in Fig. 12a. Observe that the optimization procedure exhibits a very fast convergence near the 75th iteration. Furthermore, Fig. 12b shows the distribution of mass at each iteration of the optimization, normalized by the target mass ν_t - thus we strive for a normalized mass of 1 for each cell. Initially, the cell masses are distributed across a wide range and converge to the target mass at about the 75th iteration. This demonstrates the ability of our algorithm to achieve a target mass for each Voronoi cell.



(a) Gradient norm (of Eq. 6) versus iteration.



(b) Distribution of power cell mass versus iteration.

Figure 12: Convergence of the energy gradient and normalized mass for all the uniform and sphere-like density functions considered in the semi-discrete optimal transport application. The dashed line in Fig. 12b represents the target mass.

6. CONCLUSIONS & FUTURE WORK

In this paper, we have presented the theory and implementation details for computing higher-dimensional power diagrams, and demonstrated the first implementation for solving four-dimensional quantization and L^2 semi-discrete optimal transport problems which seeks a uniform target mass of each Voronoi cell, measured under some input density function.

The performance of the algorithm was also evaluated in up to six dimensions (the topological dimension of the polytopes), which demonstrated a reasonable scaling with the number of input points in dimensions $2-4$. The cost of computing the Voronoi diagram with a randomly distributed point set was higher than that to compute the Voronoi diagram with a point distribution that exhibited blue noise properties. Also, the cost of performing the numerical integration to compute gradients significantly outweighed the cost of computing the power diagram, especially for four-dimensional polytopes. Future work may consist of deriving less expensive quadrature schemes in higher dimensions, which is under way in the works of Williams [53] and Frontin [54].

Furthermore, the cost of computing the power diagram is higher than to compute the Voronoi diagram because the introduction of weights lifts the points to a unit codimensional Euclidean space, thereby increasing the distance between sites and requiring several more (possibly non-contributing) bisectors to be clipped against. This is a failure case of the security radius theorem, as pointed out by Sainlot et al. [55], therefore, future work could consist of extending their corner validation algorithm to the higher-dimensional setting.

Our algorithm is also very well-suited for an implementation on the GPU, similar to the work of Ray et al. [56] which demonstrated an efficient $3d$ restricted Voronoi diagram calculation on the GPU. Our algorithm could also be made more efficient by employing a multiscale approach [45, 18] in the optimization of the weights. It would also be useful to implement a more robust optimization which avoids weights that cause power cells to vanish.

Finally, our four-dimensional power diagrams could be used to compute the transport distance between a data set and a continuous density measure in both space and time - Janati et al. recently consider a fully discrete version of this problem [57]. Other applications could include performing coupled space-time numerical simulations of physical phenomena.

References

[1] Monge G. "Mémoire sur la théorie des déblais et des remblais."

[2] Balzer M., Schlömer T., Deussen O. "Capacity-constrained point distributions: a variant of Lloyd's method." *ACM Transasctions on Graphics*, vol. 28, no. 3, 2009

[3] de Goes F., Breeden K., Ostromoukhov V., Desbrun M. "Blue noise through optimal transport." *ACM Transactions on Graphics*, vol. 31, no. 6, 1–11, 2012

[4] Ma L., Chen Y., Qian Y., Sun H. "Incremental Voronoi sets for instant stippling." *The Visual Computer*, vol. 34, 863–873, 2018

[5] Ma L., Guo J., Yan D.M., Sun H., Chen Y. "Instant stippling on $3d$ scenes." *Pacific Graphics*, vol. 37. 2018

[6] Meyron J. *Semi-discrete optimal transport and applications in non-imaging optics.* Ph.D. thesis, 2018

[7] Meyron J. "Initialization procedures for discrete and semi-discrete optimal transport." *Computer-Aided Design*, vol. 115, 13–22, 2019

[8] Peyré G., Cuturi M. "Computational Optimal Transport: With Applications to Data Science." *Foundations and Trends® in Machine Learning*, vol. 11, no. 5-6, 355–607, 2019

[9] de Goes F., Cohen-Steiner D., Allez P., Desbrun M. "An optimal transport approach to robust reconstruction and simplification of $2d$ shapes." *Computer Graphics Forum*, vol. 30, no. 5, 1593–1602, 2011

[10] Bonneel N., Coeurjolly D. "SPOT: sliced partial optimal transport." *ACM Transactions on Graphics*, vol. 38, no. 4, 1–13, 2019

[11] Paulin L., Bonneel N., Coeurjolly D., Iehl J.C., Webanck A., Desbrun M., Ostromoukhov V. "Sliced optimal transport sampling." *ACM Transactions on Graphics*, vol. 39, no. 4, 2019

[12] Hartmann V., Schuhmacher D. "Semi-discrete optimal transport: a solution procedure for the unsquared Euclidean distance." *Mathematical Methods of Operations Research*, vol. 92, 133–163, 2020

[13] Lévy B., Schmidt E.L. "Notions of optimal transport and how to implement them on a computer." *Computers and Graphics*, vol. 72, 135–148, 2018

[14] de Goes F., Wallez C., Huang J., Pavlov D., Desbrun M. "Power particles: an incompressible fluid solver based on power diagrams." *ACM Transactions on Graphics*, vol. 34, no. 4, 1–11, 2015

[15] Benamou J.D., Brenier Y. "A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem." *Numerische Mathematik*, vol. 84, no. 3, 375–393, 2000

[16] Villani C. *Optimal transport: old and new.* Springer, 2009

[17] Solomon J. "Computational optimal transport." *Snapshots of Modern Mathematics*, vol. 8, 2017

[18] Mérigot Q., Meyron J., Thibert B. "An algorithm for optimal transport between a simplex soup and a point cloud." *SIAM Journal on Imaging Sciences*, vol. 11, 2017

[19] Cuturi M. "Sinkhorn Distances: Lightspeed Computation of Optimal Transport." C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, vol. 26, pp. 2292–2300. Curran Associates, Inc., 2013

[20] Aurenhammer F. "Power diagrams: properties, algorithms and applications." *SIAM Journal on Computing*, vol. 16, no. 1, 78–96, 1987

[21] Aurenhammer F. "Voronoi diagrams: a survey of a fundaental geometric data structure." *ACM Computing Surveys*, vol. 23, no. 3, 1991

[22] Cañas G.D., Gortler S.J. "Surface Remeshing in Arbitrary Codimensions." *The Visual Computer*, vol. 22, no. 9, 885–895, Sep. 2006

[23] Lévy B., Bonneel N. "Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration." *Proceedings of the 21st International Meshing Roundtable*. 2012

[24] Dassi F., Si H., Perotto S., Streckenbach T. "Anisotropic Finite Element Mesh Adaptation via Higher Dimensional Embedding." *Procedia Engineering*, vol. 124, 265 – 277, 2015. 24th International Meshing Roundtable

[25] Dassi F., Farrell P., Si H. "An Anisoptropic Surface Remeshing Strategy Combining Higher Dimensional Embedding with Radial Basis Functions." *Procedia Engineering*, vol. 163, 72 – 83, 2016. 25th International Meshing Roundtable

[26] Dassi F., Perotto S., Si H., Streckenbach T. "A Priori Anisotropic Mesh Adaptation Driven by a Higher Dimensional Embedding." *Computer-Aided Design*, vol. 85, 111 – 122, 2017

[27] Nivoliers V., Lévy B., Geuzaine C. "Anisotropic and Feature Sensitive Triangular Remeshing Using Normal Lifting." *Journal of Computational and Applied Mathematics*, vol. 289, 225–240, Dec. 2015

[28] Lévy B., Mohayaee R., von Hausegger S. "A fast semi-discrete optimal transport algorithm for a unique reconstruction of the early Universe." 2020

[29] Lévy B. "Geogram: A Programming Library of Geometric Algorithms.", 2016

[30] Rycroft C.H. "Voro++: A three-dimensional Voronoi cell library in C++." *Chaos*, vol. 19, no. 4, 2009

[31] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2 edn., 2020. URL `https://doc.cgal.org/5.2/Manual/packages.html`

[32] Karavelas M. "2D Voronoi Diagram Adaptor." *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2 edn., 2020

[33] Sutherland I.E., Hodgman G.W. "Reentrant polygon clipping." *Communications of the ACM*, vol. 17, no. 1, 1974

[34] Henk M., Richter-Gebert J., Ziegler G.M. "Basic Properties of Convex Polytopes." *Handbook of Discrete and Computational Geometry, 2nd Ed.* 2004

[35] Ziegler G.M. *Lectures on Polytopes*, vol. 152 of *Graduate Texts in Mathematics*. Springer, 1995

[36] Lévy B. "A numerical algorithm for L^2 semi-discrete optimal transport." *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 49, 1693–1715, 2015

[37] Santambrogio F. *Optimal transport for applied mathematicians*. Birkhauser Basel, 2015

[38] Villani C. *Topics in optimal transportation*, vol. 58. American Mathematical Society, 2003

[39] Kantorovitch L. "On the translocation of masses." *Management Science*, vol. 5, no. 1, 1–4, 1958

[40] Lloyd S.P. "Least Squares Quantization in PCM." *IEEE Transaction on Information Theory*, vol. 28, no. 2, 129–137, 1982

[41] Du Q., Faber V., Gunzburger M. "Centroidal Voronoi Tessellations: Applications and Algorithms." *SIAM Review*, vol. 41, 637–676, 1999

[42] Liu D.C., Nocedal J. "On the limited memory BFGS method for large scale optimiation." *Mathematical Programming*, vol. 45, 503–528, 1989

[43] Johnson S.G. "The NLopt nonlinear-optimization package, available at: `http://ab-initio.mit.edu/nlopt`."

[44] Kitagawa J., Mérigot Q., Thibert B. "Convergence of a Newton algorithm for semi-discrete optimal transport." 2016

[45] Mérigot Q. "A multiscale approach to optimal transport." *Computer Graphics Forum*, vol. 30, 1583–1592, 2011

[46] Caplan P.C. *Four-dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations*. PhD thesis, Massachusetts Institute of Technology, 2019

[47] Shewchuk J.R. “Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates.” *Discrete & Computational Geometry*, vol. 18, no. 3, 305–363, 1996

[48] Lévy B. “Robustness and Efficiency of Geometric Programs: The Predicate Construction Kit.” *Computer-Aided Design*, vol. 72, 3–12, 2016

[49] Stroud A. *Approximate Calculation of Multiple Integrals*. Prentice-Hall Inc., 1971

[50] Ferrín G.B., Ruiz-Gironés E., Roca X. “Visualization of pentatopic meshes.” Tech. rep., 2020

[51] Si H. “TetGen: A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator.” Weierstrass Institute for Applied Analysis and Stochastics, 2005. `http://tetgen.berlios.de`

[52] Mitchell S.A. “Spoke-Darts for high-dimensional blue-noise sampling.” *ACM Transactions on Graphics*, vol. 37, no. 2, 2018

[53] Williams D.M., Frontin C.V., Miller E.A., Darmofal D.L. “A family of symmetric, optimized quadrature rule for pentatopes.” *Computers & Mathematics with Applications*, vol. 80, no. 5, 1405–1420, 2020

[54] Frontin C.V., Walters G.S., Witherden F.D., Lee C.W., Williams D.M., Darmofal D.L. “Foundations of space-time finite element methods: polytopes, interpolation and integration.” *arXiV preprint*, 2020

[55] Sainlot M., Nivoliers V., Attali D. “Restricting Voronoi diagrams to meshes using corner validation.” *Eurographics Symposium on Geometry Processing*, vol. 36, no. 5, 2017

[56] Ray N., Sokolov D., Lefebvre S., Lévy B. “Meshless Voronoi on the GPU.” *ACM Transactions on Graphics*, vol. 37, no. 6, 2018

[57] Janati H., Cuturi M., Gramfort A. “Spatio-temporal alignments: optimal transport through space and time.” S. Chiappa, R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, vol. 108, pp. 1695–1704. 26–28 Aug 2020