J Math Imaging Vis (2013) 46:211–237
DOI 10.1007/s10851-012-0379-2

Lattice-Based High-Dimensional Gaussian Filtering and the Permutohedral Lattice

Jongmin Baek · Andrew Adams · Jennifer Dolson

Published online: 8 September 2012

Abstract High-dimensional Gaussian filtering is a popular technique in image processing, geometry processing and computer graphics for smoothing data while preserving important features. For instance, the bilateral filter, cross bilateral filter and non-local means filter fall under the broad umbrella of high-dimensional Gaussian filters. Recent algorithmic advances therein have demonstrated that by relying on a sampled representation of the underlying space, one can obtain speed-ups of orders of magnitude over the naïve approach. The simplest such sampled representation is a *lattice*, and it has been used successfully in the bilateral grid and the permutohedral lattice algorithms. In this paper, we analyze these lattice-based algorithms, developing a general theory of lattice-based high-dimensional Gaussian filtering. We consider the set of criteria for an optimal lattice for filtering, as it offers a good tradeoff of quality for computational efficiency, and evaluate the existing lattices under the criteria. In particular, we give a rigorous exposition of the properties of the permutohedral lattice and argue that it is the optimal lattice for Gaussian filtering. Lastly, we explore further uses of the permutohedral-lattice-based Gaussian filtering framework, showing that it can be easily adapted to perform mean shift filtering and yield improvement over the traditional approach based on a Cartesian grid.

J. Baek (✉) · J. Dolson
Stanford University, Gates Computer Science Building, 353 Serra Mall, Rm. 360, Stanford, CA 94305, USA
e-mail: jbaek@cs.stanford.edu

J. Dolson
e-mail: jldolson@stanford.edu

A. Adams
Massachusetts Institute of Technology, 32 Vassar St., D414, Cambridge, MA 02139, USA
e-mail: abadams@csail.mit.edu

Keywords Bilateral filtering · High-dimensional filtering · Non-local means · Lattices · Gaussian filtering · Permutohedral lattice

1 Introduction

High-dimensional Gaussian filtering smooths a low-dimensional dataset embedded within a high-dimensional metric space, using a Gaussian kernel. Its usefulness arises from the elevation into the higher-dimensional space, which allows one to express an otherwise non-linear filter as a linear operation. Examples include some very popular non-linear filters, such as the bilateral filter, cross bilateral filter, and non-local means filter. Outside of image processing, this operation is also known as a Gauss transform.

The bilateral filter is an averaging filter that blurs images while preserving strong edges. When expressed in its original non-linear form, it is slow to compute. As such, there have been many attempts to accelerate the bilateral filter and its relatives, beginning with Durand and Dorsey [16]. The most helpful observation has been that by embedding the image in a higher dimensional space, the filter can be expressed as a linear operation, as noted by Paris and Durand [30].

The subsequent challenge then is to determine how to represent the data in this high-dimensional space. One intuitive approach is to use a lattice to sample the space, as a lattice has a predictable, repetitive structure that facilitates indexing. This approach was taken by Chen et al. [9] and Adams et al. [2]. These represent the state of the art in fast high-dimensional Gaussian filtering. Other approaches do exist, however; for example, Adams et al. [1] represents the space with a KD-tree. Tree-based approaches, such as FGT [20] or dual tree [46], are used in the context

of numerical simulation or machine learning. Note that all these Gaussian-filtering methods can be combined with a dimensionality-reduction techniques that project into a low-dimensional space, such as random Fourier features [36] or principal component analysis, for tasks like learning kernels or analyzing large image patches.

Lattices have been studied extensively in the field of both pure and applied mathematics, and have appeared in computer graphics as well. The aim of this paper is to rigorously analyze the use of a lattice as the underlying data structure for high dimensional Gaussian filtering, using the two existing examples by Chen et al. [9] and Adams et al. [2] as case studies and drawing from the existing body of mathematics literature.

Overview This article is an extension of the work of Adams, Baek, and Davis [2], which introduced the permutohedral lattice algorithm for Gaussian filtering and showcased its applications. In contrast to the previous work, this article focuses on the mathematical theory underlying the technique, rather than its applications.

The rest of this article is organized as follows: Sect. 2 details the existing literature on the bilateral filter, introducing some common formulations thereof. In Sect. 3, we formulate a general lattice-based Gaussian filtering framework, noting connections to the splines literature. In Sect. 4, we define the permutohedral lattice and illustrate its most useful properties. In Sect. 5, the aforementioned framework is applied to the permutohedral lattice, and we provide a full description of the algorithm for that lattice. In Sect. 6, we analyze the accuracy of our algorithm by comparing it to a true Gaussian filter. In Sect. 7, we consider other possible lattices and argue that the permutohedral lattice is optimal under a reasonable set of criteria, for the purpose of maximizing computational efficiency. Lastly, in Sect. 8, we further demonstrate the utility of the permutohedral lattice by implementing mean shift filtering using the lattice.

2 Related Work

There has been a wealth of literature on the bilateral filter since its inception, covering its theoretical foundations, applications, extensions, and accelerations. In this section we discuss the existing body of research on the bilateral filter, and make the connection to high-dimensional Gaussian filtering. We also briefly address the study of lattices in general, and the study of the permutohedral lattice in particular.

2.1 Bilateral Filter and Its Relatives

The bilateral filter first appeared in the works of Aurich and Weule [4], Smith and Brady [43], and was named by Tomasi and Manduchi [44]. To calculate the filtered value of a given pixel, it computes a weighted average of nearby pixels with weights that depend not only on their distance to the pixel in question (as with any convolution with a radially symmetric kernel, e.g. a Gaussian), but also on the difference in the pixel values. That is, given an input image I, the value of the output image I' at pixel location x is given as follows:

$$I'(\mathbf{x}) = \frac{1}{W_{\mathbf{x}}} \sum_{\mathbf{y}} I(\mathbf{y}) \cdot N(\mathbf{y}-\mathbf{x}; \sigma_s) \cdot N\big(I(\mathbf{y}) - I(\mathbf{x}); \sigma_r\big), \tag{2.1}$$

where $W_{\mathbf{x}} = \sum_{\mathbf{y}} N(\mathbf{y}-\mathbf{x}; \sigma_s) \cdot N(I(\mathbf{y}) - I(\mathbf{x}); \sigma_r)$ is a normalization factor. Here $N(\cdot; \sigma)$ denotes a Gaussian kernel with standard deviation σ. While a Gaussian kernel is not mandatory, it has been a dominant choice; the pillbox function has also been used ([52] and [47]). By taking into account the difference in pixel values, the bilateral filter prevents neighboring pixels that are substantially different from influencing one another, thereby respecting strong edges.

The cross bilateral filter, also called the joint bilateral filter, is a variation of the bilateral filter that filters an input image I in a manner that respects edges in another reference image J. It was introduced by Eisemann and Durand [17], Petschnigg et al. [35], Kopf et al. [24], and is formulated as follows:

$$I'(\mathbf{x}) = \frac{1}{W_{\mathbf{x}}} \sum_{\mathbf{y}} I(\mathbf{y}) \cdot N(\mathbf{y}-\mathbf{x}; \sigma_s) \cdot N\big(J(\mathbf{y}) - J(\mathbf{x}); \sigma_r\big), \tag{2.2}$$

where $W_{\mathbf{x}} = \sum_{\mathbf{y}} N(\mathbf{y}-\mathbf{x}; \sigma_s) \cdot N(J(\mathbf{y}) - J(\mathbf{x}); \sigma_r)$ this time. Contrast Eq. (2.2) with Eq. (2.1) and note the difference in the argument to the Gaussian. Smoothing I with respect to another source, namely J, allows not only fusing information from two images, as in the case of the flash-no-flash photography [17, 35], but also filtering a dataset with respect to a distinct source of different resolution, like an image and a depth map [15, 51]. Note that in this second case the sample $J(\mathbf{y})$ must be prefiltered to match the sampling rate of I, whereas the sample $J(\mathbf{x})$ is taken at J's native resolution.

The non-local means filter by Buades et al. [8] redefines the distance between the two pixel values as the distance between the two image patches centered at the two pixels. The consequence is that pixels with similar local neighborhoods are averaged together, which is an effective way to denoise.

For a more comprehensive exposition on the history of the bilateral filter and applications, see [32].

2.2 Bilateral Filtering as Gaussian Filtering

The bilateral filter in its original form (Eq. (2.1)) is nonlinear. However, the bilateral filter can be reformulated as

a linear convolution with a Gaussian kernel in a higher-dimensional Euclidean space, as noted by Paris and Durand [30] and Adams et al. [1]. Let us briefly review this formulation. This so-called Gaussian filter operates on N data points, each of which has a position $\mathbf{p}_i$, and a value $\mathbf{v}_i$. The value vectors are then averaged with respect to the distance between the position vectors, using a standard Gaussian kernel in the space of position vectors:

$$\mathbf{v}'_i = \sum_{j=1}^{N} e^{-\frac{1}{2}|\mathbf{p}_i - \mathbf{p}_j|^2} \mathbf{v}_j. \tag{2.3}$$

This equation as a generic mathematical operator is known as a Gauss transform. Accurate evaluation of Gauss transform has been studied by Yang et al. [50].

It is straightforward to express the grayscale bilateral filter using Eq. (2.3): for each pixel, define $\mathbf{p}_i = (x_i, y_i, c_i)$ where x_i, y_i are the pixel location and c_i is its grayscale intensity; define $\mathbf{v}_i = (c_i, 1)$. Note that we haved added a homogeneous coordinate to the value vector. Then, applying Eq. (2.3), one obtains

$$\mathbf{v}'_i = \left(\sum_{j=1}^{N} e^{-\frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2 + (c_i - c_j)^2)} c_j, \right.$$
$$\left. \sum_{j=1}^{N} e^{-\frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2 + (c_i - c_j)^2)} \right). \tag{2.4}$$

The final pixel value is obtained by removing the homogeneous coordinate, i.e. dividing the first coordinate by the second. Note that the parameters σ_s and σ_r from Eq. (2.1) are absent. They can be accounted for by scaling the position vectors appropriately prior to filtering; for instance,

$$\forall i, \quad \mathbf{p}_i \longleftarrow \theta \mathbf{p}_i$$

would effectively decrease the blur size by a factor of θ.

Extending this framework onto color bilateral filter is trivial; simply replace c_i with the (r_i, g_i, b_i) triple representing the color of pixel i. For the cross bilateral filter, the color values in the position vector should be drawn from the reference image; for the non-local means filter, the position vector should be replaced by a patch descriptor [1].

2.3 Accelerating High-Dimensional Gaussian Filtering

Several authors have exploited the fact that the bilateral filter can be expressed as a linear convolution in order to accelerate its evaluation; rather than treating the input image as a point set, or a discrete collection of pixels, it can be seen as a continuous signal that has been sampled at the locations of the position vectors. Paris and Durand [30] voxelized (discretized) the space of position vectors to represent this signal, which is perhaps the most intuitive approach. Because the Gaussian kernel is separable along the dimensions of the position vector, the convolution can be performed very quickly. However, while linear in the size of the input, this approach scales exponentially with the dimensionality of the position vector. Adams et al. [1] represented the space of position vectors *sparsely* using a KD-tree with nodes placed at the location of the input samples, obtaining a runtime that grows linearly with the dimensionality and loglinearly with the input size. Sparsity is crucial in this acceleration scheme, as the image describes a low-dimensional manifold inside the higher-dimensional space and therefore is necessarily sparse. The authors noted, however, that KD-tree construction and querying is difficult to parallelize efficiently. Adams et al. [2] relied on a different lattice, called the permutohedral lattice, to sample the high-dimensional space, and obtained considerable speed-up over the previous work that used a Cartesian grid for higher-dimensional cases.

2.4 Lattices and the Permutohedral Lattice

Lattices are of general interest to mathematicians as they are the machinery one uses to tile or represent an arbitrary-dimensional Euclidean space. One example is the familiar Cartesian grid $\mathbb{Z}^d$, consisting of all d-dimensional points of integer coordinates.

The *covering problem*, as defined by Rogers [37], is concerned with placement of unit spheres in an Euclidean space so that every point in the space belongs to at least one unit sphere. A proof is given by Kershner [22] that the hexagonal lattice generates the least dense covering in two-dimensional Euclidean space. In higher dimensions, the optimal covering is not known, but much literature exists on the optimal *lattice* covering, as the algebraic properties of lattices avail themselves of more mathematical machinery. In particular, the permutohedral lattice is provably the most efficient lattice up to dimension 5 [5, 13, 39], and the most efficient *known* lattice up to dimension 22 [12].

The permutohedral lattice is denoted in mathematics as A_d^* and belongs to the family of *integral* lattices. While any lattice can be used to represent high-dimensional data, A_d^* has received much attention because of its low covering density. Petersen and Middleton [34] noted the relationship between covering density and efficient sampling of functions that have isotropic support in the Fourier domain. In other words, since sampling a band-limited function effectively tiles the space with its support, a lattice with low covering density would require fewer samples to avoid aliasing. In this context, a low covering density would indicate an *efficient* tradeoff between faithfulness and computational cost; for a fixed level of faithfulness, low covering density leads to lower resource consumption; conversely, for a fixed resource budget, low covering density leads to a more accurate representation of the data.

The two- and three-dimensional analogues of A_d^* are the hexagonal lattice and the body-centered cubic lattice (BCC), which are well known. A_d^* is affinely equivalent to the Kuhn triangulation [25], which partitions a unit cube into tetrahedra. Sampling and reconstructions on the BCC lattice have been studied previously by Entezari et al. [18, 19]. In the computer graphics community, A_d^* has been used by Perlin [33] for generating high-dimensional procedural noise, and by Kim [23] for interpolating high-dimensional splines.

2.5 Contributions

The main contributions of this article are as follows:

- We present a formulation of high-dimensional Gaussian filtering with respect to an arbitrary lattice. This general formulation is consistent with the existing algorithms based on either the Cartesian grid or the permutohedral lattice.
- We give a thorough treatment of the permutohedral lattice and its properties in the context of filtering, and establish connections to the splines literature.
- Using the general formulation stated above, we develop a set of criteria for determining the optimal lattice with which to perform Gaussian filtering. Moreover, we argue with mathematical justification that the permutohedral lattice is the optimal lattice.
- We give a numerical analysis of the Cartesian grid and the permutohedral lattice on how closely they approximate the true Gaussian kernel.
- We extend mean shift filtering onto the permutohedral lattice by minimally altering our framework.

3 Lattice-Based Filtering

We propose a general formulation of high-dimensional Gaussian filtering in which an arbitrary lattice L is used to represent the underlying space, building upon the scheme of Paris and Durand [32] that operates on a Cartesian grid. In the terminology of Adams et al. [1], this scheme is comprised of three stages named *splatting*, *blurring* and *slicing*. In the splatting stage, the signal contained in the input data is resampled onto the vertices of the given lattice; in the blurring stage, the lattice undergoes linear convolution in order to simulate a Gaussian blur; in the slicing stage, the blurred signal represented in the lattice is resampled at the original sample locations, producing the output. In essence, the entire algorithm is a blurring operation flanked by two resampling operations. The purpose of the resampling operations is to modify the representation of the signal in a way that can accommodate fast blurring.

The overall process of high-dimensional Gaussian filtering is summarized in Fig. 1. The following sections describe each of the stages in detail.

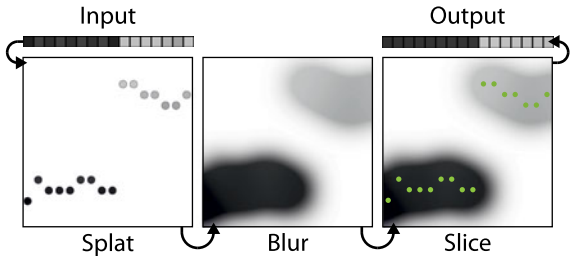

Fig. 1 High-dimensional Gaussian filtering is a blurring operation flanked by two resampling operations. The signal to be filtered is resampled onto a lattice at the splatting stage, is then blurred, and then is sliced out at the appropriate location to yield the output

Table 1 Common notation used in the paper

d	The dimensionality of position vectors.
m	The dimensionality of value vectors.
I	the function $I : \mathbb{R}^d \to \mathbb{R}^m$ to be filtered.
I'	the resulting function after I is filtered.
$\mathbf{x}$	A vector of dimensionality $d+1$, unless specified otherwise: $\{x_0, x_1, \ldots, x_d\}$.
$\mathbf{1}$	The $d+1$-dimensional vector $\{1, 1, \ldots, 1\}$.
H_d	The d-dimensional hyperplane $\{\mathbf{x} \mid \mathbf{x} \cdot \mathbf{1} = 0\} \subset \mathbb{R}^{d+1}$.
A_d^*	The permutohedral lattice of dimensionality d, lying inside H_d.
$T(\cdot)$	The projection mapping $\mathbb{R}^{d+1}$ onto H_d along the vector $\mathbf{1}$. (See Proposition 4.3.)
$N(\mathbf{x}; \theta)$	Multi-variate Gaussian of covariance matrix θI, evaluated at $\mathbf{x}$.
$\mathbf{u}_k$	The k-th standard basis vector, indexed from zero, i.e. $\mathbf{u}_0 = \{1, 0, 0, \ldots, 0\}$.

3.1 Preliminaries

Table 1 summarizes the notations used throughout the paper. Other standard mathematical notations, such as $\mathbb{R}$ and $\mathbb{Z}$ apply as well.

We now review the basic definitions that will be useful in the subsequent sections. Our first definition is, not surprisingly, the standard definition of a lattice:

Definition 3.1 A *lattice* L is a discrete additive subgroup of a Euclidean space.

An additive subgroup of a Euclidean space is closed under addition and subtraction, and contains the origin. Hence, one can alternatively characterize a lattice as all linear combinations of a set of vectors called the *frame* (akin to the basis of a vector space) with integer weights. For example, the Cartesian grid $\mathbb{Z}^d$ corresponds to the frame consisting of standard d-dimensional basis vectors $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{d-1}$.

The fact that a lattice is an additive group immediately gives rise to many useful properties, including translational symmetry; the local structure of a lattice around each lattice

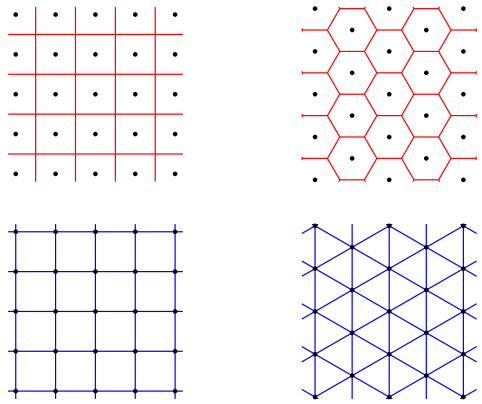

Fig. 2 Voronoi and Delaunay tessellations of some two-dimensional lattices. *Top*: the Voronoi cells of $\mathbb{Z}^2$ and A_2^* are shown, along with lattice points. The two lattices have equal density. *Bottom*: the Delaunay cells induced by the Voronoi cells above. All cells of $\mathbb{Z}^2$ are squares, whereas A_2^* has hexagonal Voronoi cells and triangular Delaunay cells

point is invariant. This ensures the simplicity of any algorithm operating on the lattice, since each lattice point can be treated uniformly.

The traditional way to study the local structure of any discrete point set is to examine its Voronoi cells under the Euclidean norm:

Definition 3.2 Let $\mathbf{x}$ be a point in a lattice $L \subset \mathbb{R}^d$. Then the *Voronoi cell* of $\mathbf{x}$ is the set of points in $\mathbb{R}^d$ nearer to $\mathbf{x}$ than to any other lattice point, i.e.

$$\left\{\mathbf{y} \in \mathbb{R}^d \;\middle|\; \forall \mathbf{x}' \in L,\ \|\mathbf{x}-\mathbf{y}\|^2 \leq \|\mathbf{x}'-\mathbf{y}\|^2\right\}.$$

It is clear that the Voronoi cells of a lattice are uniform and related to one another by translation. They also tessellate the underlying space $\mathbb{R}^d$, since any point in $\mathbb{R}^d$ can be associated with a lattice point nearest to it. A polytope that gives rise to a face-to-face tessellation of the Euclidean space by translation is called a *parallelohedron*. Thus, the Voronoi cell of a lattice is a parallelohedron. (We will revisit this point later in Sect. 7.)

Definition 3.3 A *Delaunay cell* of a lattice is the convex hull of all lattice points whose Voronoi cells share a common vertex.

The Delaunay cells are the dual of the Voronoi cells, and yield a natural notion of "nearby" lattice points for an arbitrary point in space. They also tessellate the space, but they need not be uniform. Figure 2 demonstrates the Voronoi and Delaunay cells for a pair of two-dimensional lattices.

Note that $\mathbb{R}^d$ in the above definitions can be replaced with any set isometric to $\mathbb{R}^d$. For instance, H_d, as defined in Table 1, is a d-dimensional hyperplane in $\mathbb{R}^{d+1}$ and is therefore isometric to $\mathbb{R}^d$. Any discrete additive subgroup of H_d is a lattice, and its Voronoi and Delaunay cells reside in H_d. This is the case with the definition of the permutohedral lattice, as we shall see shortly in Sect. 4. We refer the readers to Conway and Sloane [12] for a more extensive treatment of lattices and their uses.

3.2 Splatting

In high-dimensional Gaussian filtering, the input is a set of d-dimensional vectors $\{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ with associated m-dimensional vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$. We call the former position vectors, and the latter value vectors. The aim is to compute the weighted sum of the value vectors at each position vector, for which the weight decays as a Gaussian of the distance in the position vector space, as shown in Eq. (2.3). In the signal-processing approach of Paris and Durand [30], it is presumed that the input represents sampling of a latent signal $I : \mathbb{R}^d \longrightarrow \mathbb{R}^m$:

$$I(\mathbf{p}_1) = \mathbf{v}_1,$$

$$\vdots$$

$$I(\mathbf{p}_N) = \mathbf{v}_N,$$

where N is the total number of input points. In typical image-processing tasks, N is the number of pixels.

The first step in high-dimensional Gaussian filtering is to *splat*, or embed, this input signal onto the lattice L. For each known sample of the input signal, its value is accumulated at one or more spatially nearby lattice points belonging to L. This process essentially resamples the input signal at the lattice points.

In case only one vertex in L is chosen, the process is equivalent to quantization, or nearest-neighbor interpolation. Alternatively, choosing multiple points with nonnegative weights that sum to 1 improves accuracy. For instance, one of the most common interpolation scheme on $\mathbb{Z}^d$ in computer graphics is the use of the standard multi-linear weights, as done by Paris and Durand [30]. For other lattices, one could splat the sample onto the vertices of the Delaunay cell containing it. An example is shown in Fig. 3.

While there are many ways to assign weights for resampling, corresponding to different *basis functions*, the ideal scheme should balance computational cost with faithful representation of the embedded signal. For instance, one could precisely band-limit a signal that is to be represented on $\mathbb{Z}^d$ using sinc weights, but the support of a sinc function is unbounded, making this particular suggestion impractical.

3.3 Blurring

At the conclusion of the splatting stage, the input signal is now represented by the discrete samples at the vertices of

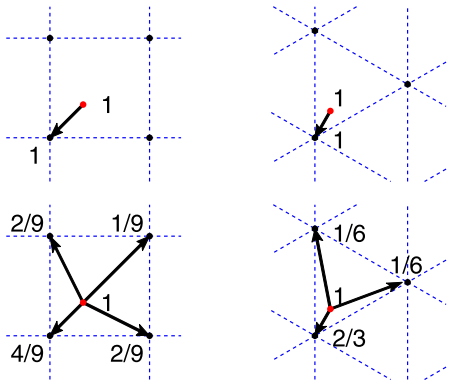

Fig. 3 The splatting process on $\mathbb{Z}^2$ and A_2^*. The Delaunay tessellations are shown in dotted guidelines. *Top*: a sample point is rounded to the nearest lattice point. *Bottom*: a sample point is spread onto the vertices of the Delaunay cell containing it, using bilinear coordinates for $\mathbb{Z}^2$ and barycentric coordinates for A_2^*

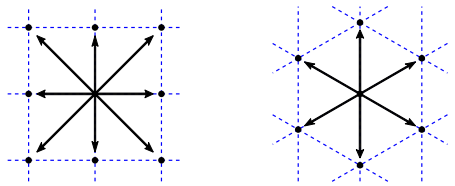

Fig. 4 The blurring process on $\mathbb{Z}^2$ and A_2^*. The Delaunay tessellations are shown in the dotted guidelines. The *arrows* mark the vertices that share at least one Delaunay cell with the central vertex, which define a notion of the set of neighbors. The value associated with the central vertex should propagate to these neighbors with decaying weights

the lattice L. In the blurring stage, the value stored at each location is blurred with those at nearby locations in L, with weights that decay as the Gaussian of their L_2-distance. For an arbitrary lattice, we consider as neighbors all vertices that share at least one Delaunay cell with the given vertex, as in Fig. 4. Having a larger support will achieve a more accurate convolution, but will be computationally expensive.

Note that, in case of $\mathbb{Z}^d$, its separable nature enables us to blur each dimension successively, requiring access only to neighbors along each axis. For example, one can propagate the value in the center vertex of the Fig. 4 onto eight neighbors via only two axis-aligned blurs. Lattices that permit separable blurs will have computational complexity that scales more gracefully with dimension than those that do not.

3.4 Slicing

Slicing is the process of resampling from the lattice structure and is exactly analogous to splatting: each sample of the output signal is reconstructed from nearby positions, often with the same set of weights as in splatting. In certain applications such as super-resolution and upsampling [15, 24, 51], the data structure may be sampled at locations other than the original position vectors.

4 The permutohedral lattice

The permutohedral lattice has found applications in many fields ranging from crystallography to communication. It arises ubiquitously in nature; the commonly found hexagonal grid and the body-centered cubic lattice, seen in Fig. 2, are its two- and three-dimensional analogues, respectively. We give definitions of the permutohedral lattice and derive its structural properties.

4.1 Definition

The permutohedral lattice is the dual of the root lattice A_d. Both A_d and the permutohedral lattice are typically embedded in $H_d \subset \mathbb{R}^{d+1}$ for ease of manipulation. Recall that H_d is the hyperplane consisting of points whose coordinates sum to zero. See Kim [23] for the non-standard representation within $\mathbb{R}^d$ instead.

Definition 4.1 The root lattice A_d is

$$\left\{\mathbf{x} = (x_0, \ldots, x_d) \in \mathbb{Z}^{d+1} \;\middle|\; \mathbf{x} \cdot \mathbf{1} = 0\right\}.$$

In other words, $A_d = \mathbb{Z}^{d+1} \cap H_d$. Clearly, $A_d \subset H_d$.

Definition 4.2 [12] The *permutohedral lattice*, denoted by A_d^*, is the dual (or reciprocal) lattice of A_d inside H_d:

$$A_d^* := \{\mathbf{x} \in H_d \mid \forall \mathbf{y} \in A_d,\ \mathbf{x} \cdot \mathbf{y} \in \mathbb{Z}\}. \tag{4.1}$$

For example, A_2 is spanned by the following vectors

$$(1, 1, -2), \quad (1, -2, 1),$$

and in turn, A_2^* is spanned by $(\frac{1}{3}, \frac{1}{3}, -\frac{2}{3})$ and $(\frac{1}{3}, -\frac{2}{3}, \frac{1}{3})$. It is easy to verify that these two vectors indeed have integral dot products with vertices of A_2.

We shall now state two equivalent definitions of A_d^* uniformly scaled by $d + 1$. The scale factor ensures that all coordinates will be integers. These new definitions better elucidate the properties of the lattice.

Proposition 4.3 *The following two definitions of* A_d^* *are equivalent to* (4.1) *scaled up by* $d + 1$:

$$A_d^* := \left\{T(\mathbf{x}) \;\middle|\; \mathbf{x} \in (d+1)\mathbb{Z}^{d+1}\right\}, \tag{4.2}$$

where T *is the projection of* $\mathbb{R}^{d+1}$ *onto* H_d, *namely*

$$T : \mathbf{x} \mapsto \mathbf{x} - \left(\frac{\mathbf{x} \cdot \mathbf{1}}{\mathbf{1} \cdot \mathbf{1}}\right)\mathbf{1}.$$

$$A_d^* := \bigcup_{k=0}^{d} \{\mathbf{x} \in H_d \mid \mathbf{x} \textit{ is a remainder-}k \textit{ point}\}, \tag{4.3}$$

where we call $\mathbf{x} \in H_d$ *a remainder-*k *point for some* $k \in \{0, \ldots, d\}$ *iff all coordinates are congruent to* k *modulo* $d+1$.

Recall that $\mathbf{1}$ is a $d+1$-dimensional vector whose components are all 1's.

Proof (4.2)⊆(4.3): Let $\mathbf{x} \in (d+1)\mathbb{Z}^{d+1}$. Then we can write $\mathbf{x} = (d+1)\mathbf{y}$ for some $\mathbf{y} \in \mathbb{Z}^{d+1}$. This yields,

$$\begin{aligned} T(\mathbf{x}) &= (d+1)\mathbf{y} - \frac{(d+1)(\mathbf{y}\cdot\mathbf{1})}{d+1}\mathbf{1} \\ &= (d+1)\mathbf{y} - (\mathbf{y}\cdot\mathbf{1})\mathbf{1}. \end{aligned}$$

We see that each component of $T(\mathbf{x})$ is an integer and has a consistent remainder modulo $d+1$, namely $-(\mathbf{y}\cdot\mathbf{1})$.

(4.3)⊆(4.1): Let $\mathbf{x}$ be a remainder-k point for some $k \in \{0, \ldots, d\}$. Then we can write $\mathbf{x} = (d+1)\mathbf{y} + k\mathbf{1}$ for some $\mathbf{y} \in \mathbb{Z}^{d+1}$. To show that $\mathbf{x}/(d+1)$ is in the dual of A_d, note that for all $\mathbf{z} \in A_d$,

$$\begin{aligned} \frac{\mathbf{x}}{d+1}\cdot\mathbf{z} &= \left(\mathbf{y} + \frac{k}{d+1}\mathbf{1}\right)\cdot\mathbf{z} \\ &= \mathbf{y}\cdot\mathbf{z} \in \mathbb{Z}, \quad \text{since } \mathbf{z}\cdot\mathbf{1} = 0. \end{aligned}$$

(4.1)⊆(4.2): Let $\mathbf{x}/(d+1)$ be in the dual lattice of A_d. Note that for all $i \neq j$, the vector $\mathbf{u}_i - \mathbf{u}_j$ belongs to A_d, because $(\mathbf{u}_i - \mathbf{u}_j)\cdot\mathbf{1} = 0$. Hence,

$$\frac{\mathbf{x}}{d+1}\cdot(\mathbf{u}_i - \mathbf{u}_j) = \frac{x_i - x_j}{d+1} \in \mathbb{Z}.$$

This implies that all pairs of coordinates x_i, x_j differ by multiples of $d+1$. Writing $x_i = (d+1)y_i + k$ for some $y_i \in \mathbb{Z}, k \in \mathbb{R}$, we see that $\mathbf{x}$ is the projection of $(d+1)\mathbf{y}$ obtained by subtracting a multiple of $(1, \ldots, 1)$. Because $\mathbf{x} \in H_d$, this projection is precisely T. □

According to Proposition 4.3, the simplest way to characterize A_d^* is to project $\mathbb{Z}^{d+1}$ along the long diagonal vector $\mathbf{1}$, with a scalar factor of $d+1$ to keep the coordinates in integers, as seen in Fig. 5. From now on, we will assume the scaled version of A_d^*, i.e. (4.2) or (4.3), rather than the original (4.1).

4.2 Structural Properties

As with other lattices, A_d^* induces a tessellation of H_d with its Voronoi cells. In this section, we study the Voronoi and Delaunay cells of A_d^*, as their properties are essential to the high-dimensional Gaussian filtering framework we established in Sect. 3.

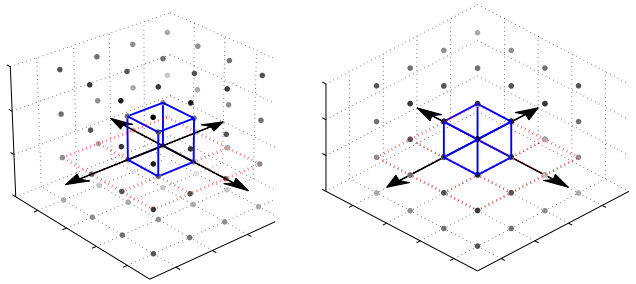

Fig. 5 Construction of A_d^* in case $d = 2$. *Left*: a Cartesian grid. The xy-plane is shown in *red*, along with the unit cube in *blue*. *Right*: the same figure in orthographic projection, seen along the vector $\mathbf{1}$. Note that many of the points line up, indicating that they will project to the same point on H_d. The set of projected points form A_d^*, after scaling (Color figure online)

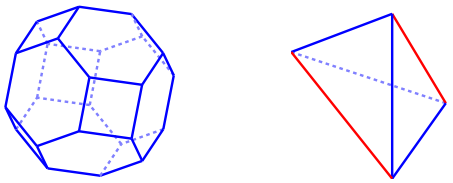

Fig. 6 *Left*: the Voronoi cell of A_3^* (a permutohedron in $\mathbb{R}^3$), which is also the uniform truncated octahedron. *Right*: the Delaunay cell of A_3^*, which is an isosceles tetrahedron. The *red sides* are longer than the *blue sides* (Color figure online)

The Voronoi cells of A_d^* are polytopes called *permutohedra* because they are obtained by permuting the coordinates of any single vertex. The name of these polytopes also lends itself to this lattice.

Proposition 4.4 [12] *The Voronoi cell of the origin in* A_d^* *is a permutohedron with*

$$\left\{\rho\left(\frac{d}{2}, \frac{d}{2} - 1, \ldots, \frac{-d}{2} + 1, \frac{-d}{2}\right) \,\middle|\, \rho \in S_{d+1}\right\},$$

as its vertices, where S_{d+1} *is the symmetric group acting on* $d+1$ *elements, i.e. the set of all permutations.*

The permutohedra in the first three dimensions are a line segment, a hexagon (Fig. 2) and a uniform truncated octahedron (Fig. 6), respectively.

Theorem 4.5 *The Delaunay cells of* A_d^* *are* d*-simplices, and are related via permutation and translation to the canonical simplex whose vertices are the following*:

$$(0, 0, \ldots, 0, 0, 0),$$
$$(1, 1, \ldots, 1, 1, -d),$$
$$\vdots$$

$$(\underbrace{k,\ldots,k}_{d+1-k},\underbrace{k-(d+1),\ldots,k-(d+1)}_{k}),$$

$$\vdots$$

$$(d,-1,\ldots,-1,-1,-1).$$

Proof Because of the translational symmetry in A_d^*, it suffices to characterize only the Delaunay cells containing the origin. Let $\mathbf{x}$ be the vertex shared by a set of Voronoi cells, one of which is the Voronoi cell of the origin. By Proposition 4.4, $\mathbf{x}$ is some permutation of $(\frac{d}{2},\ldots,\frac{-d}{2})$. Without loss of generality, let us assume $\mathbf{x}=(\frac{d}{2},\ldots,-\frac{d}{2})$.

Since $\mathbf{x}$ is a vertex of a Voronoi cell, the Delaunay cell containing it consists of the lattice points closest to $\mathbf{x}$. By Proposition 4.3, lattice points in A_d^* are remainder-k points, having the form $\mathbf{y}=(d+1)\mathbf{z}+k\mathbf{1}$ where $\mathbf{z}\in\mathbb{Z}^{d+1}$.

We may attempt to minimize the distance between $\mathbf{x}$ and $\mathbf{y}$ by choosing $\mathbf{z}$ carefully. Assuming k fixed,

$$\begin{aligned}
&\operatorname{argmin}_{\mathbf{z}}\|\mathbf{y}-\mathbf{x}\|^2\\
&\quad=\operatorname{argmin}_{\mathbf{z}}\sum\left|(d+1)z_i+k-x_i\right|^2\\
&\quad=\operatorname{argmin}_{\mathbf{z}}\sum(d+1)^2z_i^2+2(d+1)z_i(k-x_i)\\
&\quad=\operatorname{argmin}_{\mathbf{z}}\sum(d+1)^2z_i^2-2(d+1)z_ix_i\\
&\qquad+\sum2(d+1)z_ik\\
&\quad=\operatorname{argmin}_{\mathbf{z}}\sum(d+1)^2z_i^2-2(d+1)z_ix_i\\
&\qquad+2k\big[(d+1)\mathbf{z}\cdot\mathbf{1}\big]\\
&\quad=\operatorname{argmin}_{\mathbf{z}}\sum(d+1)^2z_i^2-2(d+1)z_ix_i \qquad (4.4)\\
&\quad=\operatorname{argmin}_{\mathbf{z}}\big\|(d+1)\mathbf{z}-(d/2,\ldots,-d/2)\big\|^2\\
&\quad=(\underbrace{0,\ldots,0}_{d+1-k},\underbrace{-1,\ldots,-1}_{k}). \qquad (4.5)
\end{aligned}$$

Note that (4.4) follows directly from the observation that $(d+1)\mathbf{z}\cdot\mathbf{1}=(\mathbf{y}-k\mathbf{1})\cdot\mathbf{1}=-k(d+1)$, which is independent of $\mathbf{z}$.

Substituting (4.5) back into $\mathbf{y}=(d+1)\mathbf{z}+k\mathbf{1}$, we obtain $\mathbf{y}=(k,\ldots,k,k-(d+1)\ldots,k-(d+1))$, showing that there is a unique nearest lattice point of remainder k. Next, we can check in a straightforward manner that the nearest remainder-k points for all k are all equidistant from $\mathbf{x}$. Hence, varying k over $\{0,\ldots,d\}$ yields the canonical simplex as stated above.

If $\mathbf{x}$ is some permutation ρ of $(\frac{d}{2},\ldots,\frac{-d}{2})$, the above derivation commutes fully with ρ, yielding a simplex that is related to the canonical simplex via ρ. □

Corollary 4.6 *Each Delaunay cell contains exactly one lattice point of remainder k, for each $k\in\{0,\ldots,d\}$.*

Proof By construction, the canonical simplex contains exactly one lattice point of remainder k. Because other simplices are obtained by permuting the components of its vertices and translating them, the same holds. □

In fact, we can significantly strengthen Corollary 4.6:

Corollary 4.7 *For every $\mathbf{x}\in H_d$, the Delaunay cell containing it is the set of the closest remainder-k point for each $k\in\{0,\ldots,d\}$.*

Corollary 4.7 is of main interest to us because it suggests a computationally efficient way of calculating the vertices of the Delaunay cell containing an arbitrarily specified point.

To demonstrate this corollary, we first show a lemma characterizing points near an arbitrary lattice point:

Lemma 4.8 (The Range Lemma) *Let $\mathbf{x}\in H_d$, and let $\mathbf{y}\in A_d^*$ be a remainder-k point. Then, $\mathbf{y}$ is the closest remainder-k point to $\mathbf{x}$ iff*

$$\max_i(x_i-y_i)-\min_i(x_i-y_i)\le d+1.$$

Proof Because of translational symmetry, it suffices to prove the claim for $\mathbf{y}=(0,\ldots,0)$. By inspection, all vertices of the canonical simplex obey the following inequalities:

$$x_0\ge x_1\ge x_2\ge\cdots\ge x_d,\quad\text{and}\quad x_0-x_d\le d+1. \qquad (4.6)$$

Since any point in the simplex is a convex combination of the vertices, the above inequality must also hold for any point inside. In particular, for any of the $d+1$ inequalities in (4.6), all but one vertex of the canonical simplex satisfy it with equality. Since a linear inequality is a $(d-1)$-dimensional hyperplane, which is completely determined by d points, it must be that the face formed by the d vertices corresponds to the hyperplane. Therefore, the region bounded by the faces (i.e. the simplex) corresponds exactly to the set of points satisfying (4.6).

Hence, for any $\mathbf{x}$ in the canonical simplex, $\max_i x_i-\min_i x_i=x_0-x_d\le d+1$. Likewise, for any simplex that is a permutation of the canonical simplex, we have $x_j-x_k\le d+1$ for some j,k where $x_j=\max_i x_i$ and $x_k=\min_i x_i$. □

Corollary 4.7 then follows: if $\mathbf{x}$ is in the canonical simplex, we can easily check that the vertices given in Theorem 4.5 satisfy the premise of the Range Lemma. Therefore, they must be the closest remainder-k points to $\mathbf{x}$ as desired. The expression $\max_i(x_i-y_i)-\min_i(x_i-y_i)$ is invariant under permutation of the components or translation of $\mathbf{x}$ and $\mathbf{y}$, so the argument applies to other simplices in A_d^*.

Lemma 4.9 [12] *Given $\mathbf{x}\in H_d$, the closest remainder-0 points in A_d^* can be found with the following procedure*:

1. *Define* $\mathbf{y} = \{y_0, \ldots, y_d\}$ *where* y_i *is given by rounding* x_i *to the nearest multiple of* $d+1$.
2. *Compute the difference* $\boldsymbol{\Delta} = \mathbf{x} - \mathbf{y}$, *and* $h = \frac{\sum_{i=0}^{d} y_i}{d+1}$.
3. *If* $h < 0$, *add* $d+1$ *to each* y_j *where* Δ_j *is one of the* $|h|$*-largest differences. If* $h > 0$, *subtract* $d+1$ *from each* y_j *where* Δ_j *is one of the* h*-smallest differences.*

Proof Note that the set of remainder-0 points in A_d^* is equivalent to A_d scaled up by $d+1$. Conway and Sloane [12] give an analogous algorithm for finding the nearest lattice point in A_d with proof. □

Theorem 4.10 *Given* $\mathbf{x} \in H_d$, *the Delaunay cell containing it can be found by the following procedure in* $O(d^2)$ *time*:

1. *Find the nearest remainder-*0 *point* $\mathbf{y}$.
2. *Compute the differential* $\boldsymbol{\Delta} = \mathbf{x} - \mathbf{y}$.
3. *Find the permutation* ρ *that sorts* $\boldsymbol{\Delta}$ *in decreasing order.*
4. *For each* k, *the nearest remainder-*k *point is given by* $\rho^{-1}(\mathbf{c}_k) + \mathbf{y}$ *where* $\mathbf{c}_k$ *is the remainder-*k *point in the canonical simplex.*

Proof The theorem follows directly from Lemma 4.9 and Theorem 4.5. In essence, the theorem identifies the transform that maps $\mathbf{x}$ into the interior of the canonical simplex, and then applies the inverse transform to the vertices of the canonical simplex in order to retrieve the Delaunay cell. As for runtime, each of the four steps run in $O(d)$, $O(d)$, $O(d \log d)$ and $O(d^2)$, respectively, so the overall time complexity is $O(d^2)$. See Fig. 7 for a visualization of the theorem. □

In summary, we have explored the structure of the permutohedral lattice in terms of its Voronoi and Delaunay cells, along with useful results on how to quickly locate nearby lattice points or verify them.

We end the section by briefly discussing the volumes of the Voronoi and Delaunay cells, as they will be necessarily in the subsequent sections:

Proposition 4.11 *The volume of a permutohedron is given by* $(d+1)^{d-1/2}$. *The volume of the canonical simplex is* $(d+1)^{d-1/2}/d!$.

Proof By inspection, A_d^* is spanned by any d of the permutations of $(1, 1, \ldots, -d)$. The volume of the parallelopiped constructed by a basis is given by the square root of its Gram determinant, which can be easily computed to be $(d+1)^{d-1/2}$. The volume of the permutohedron must equal this number, because both the parallelopiped and the permutohedron tessellate the space with equal density, namely the density of the lattice points in space. Lastly, a simple counting argument shows that there are $d!$ equally-sized simplices for every permutohedron. □

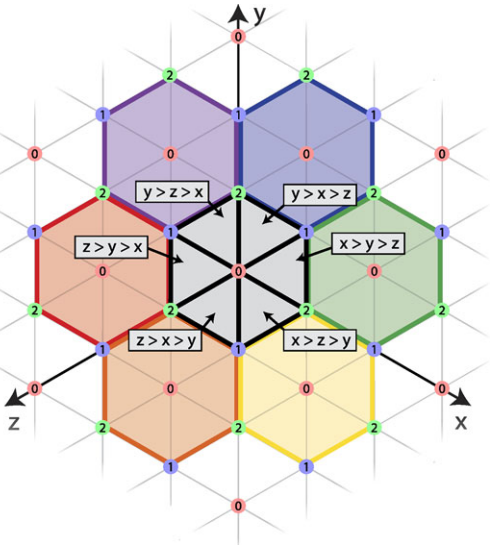

Fig. 7 Visualization of H_2. Each vertex of the lattice is marked with its remainder. Note that each of the Delaunay cells bordering the origin is uniquely specified by the ordering of the three coordinates, and that the complex formed by the union of these cells tessellate the space. Hence, the Delaunay cell containing any point can be uniquely identified by the complex that contains it, and the ordering of the coordinates relative to the remainder-0 point in the center

5 Algorithms

Now we proceed to describe in full how to perform Gaussian filtering using the permutohedral lattice A_d^*. Each of splatting, blurring, slicing steps is discussed in detail, with pseudocode given when applicable.

5.1 Splatting

Splatting a point involves computing its neighbors in A_d^*, namely the vertices of the Delaunay cell containing it, and accumulating the value at the vertices with appropriate weights. For simplices, the most common interpolation scheme is that of barycentric coordinates. Barycentric coordinates for $\mathbf{x}$ inside the canonical simplex $\{\mathbf{c}_0, \ldots, \mathbf{c}_d\}$ is given by the *barycentric coordinate function* de Boor [6]:

$$\mathbf{b} := \begin{bmatrix} \mathbf{c}_0 \cdots \mathbf{c}_d \\ 1 \cdots 1 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}.$$

Note that because in our definitions $\mathbf{c}_0, \ldots, \mathbf{c}_d$ reside in the $d+1$-dimensional coordinate system, it is necessary to remove a row from the linear system above to achieve linear independence.

Lemma 5.1 *Denote by* $\mathbf{c}_k$ *the remainder-*k *vertex of the canonical simplex. Let* $\mathbf{x}$ *be an arbitrary point in the canonical simplex, and let* $\mathbf{b}$ *be its barycentric coordinates in the*

simplex. *Then*,

$$b_k = \begin{cases} \frac{x_{d-k} - x_{d+1-k}}{d+1}, & k \neq 0, \\ 1 - \frac{x_0 - x_d}{d+1}, & k = 0. \end{cases}$$

Proof It suffices to show that the given weights do yield $\mathbf{x}$. Let $\mathbf{y} = [\mathbf{c}_0 \cdots \mathbf{c}_d]\mathbf{b}$. Then,

$$\begin{aligned} y_j &= \sum_{k=0}^{d} b_k \cdot (\mathbf{c}_k)_j \\ &= \left\lfloor \sum_{k=0}^{d-j} b_k k \right\rfloor + \left\lfloor \sum_{k=d-j+1}^{d} b_k \big(k - (d+1)\big) \right\rfloor \\ &= \left\lfloor \sum_{k=0}^{d} b_k k \right\rfloor - \left\lfloor (d+1) \sum_{k=d-j+1}^{d} b_k \right\rfloor \\ &= \left\lfloor \left(\frac{x_{d-1} - x_d}{d+1} \right) + 2\left(\frac{x_{d-2} - x_{d-1}}{d+1} \right) + \cdots \right. \\ &\quad \left. + d\left(\frac{x_0 - x_1}{d+1} \right) \right\rfloor - \left\lfloor (d+1) \sum_{k=d-j+1}^{d} b_k \right\rfloor \\ &= \frac{-x_d - x_{d-1} - \cdots - x_1 + d x_0}{d+1} - (x_0 - x_j) \\ &= \frac{-x_d - x_{d-1} - \cdots - x_1 - x_0}{d+1} + x_j \\ &= x_j \quad \text{as desired.} \end{aligned}$$

Lastly, $b_0, \ldots, b_d$ sum to 1 by construction. □

Lemma 5.1 easily generalizes to arbitrary points in H_d by utilizing the fact that other simplices are given by permutation and translation of the canonical simplex.

Proposition 5.2 *Let* $\mathbf{x}$ *be an arbitrary point in* H_d, *and let* $\mathbf{y}$ *be the closest remainder-*0 *point*. *Also*, *let* $\rho \in S_{d+1}$ *be the permutation that sorts the components of* $\mathbf{\Delta} = \mathbf{x} - \mathbf{y}$ *in decreasing order*. *Then*, *the barycentric coordinates for* $\mathbf{x}$ *is given by* $\mathbf{b}$ *where*

$$b_k = \begin{cases} \frac{\rho(\mathbf{\Delta})_{d-k} - \rho(\mathbf{\Delta})_{d+1-k}}{d+1}, & k \neq 0, \\ 1 - \frac{\rho(\mathbf{\Delta})_0 - \rho(\mathbf{\Delta})_d}{d+1}, & k = 0. \end{cases}$$

Proof The claim follows from Lemma 5.1 and Theorem 4.10. □

Hence the barycentric coordinates of a point in H_d can be computed in $O(d \log d)$, and if ρ and $\mathbf{y}$ are already available, in $O(d)$.

The splatting step is complete once every sample from the input signal has been splatted. Algorithm 1 summarizes the process. Note that at each lattice point, nearby samples from the input are accumulated with weights that depend on the offset vector $\mathbf{\Delta}$. So the splatting process is a convolution with a kernel in H_d (followed by sampling at the lattice points).

Algorithm 1 The splatting algorithm. Given the input signal $I : X \to \mathbb{R}^m$, where X is the discrete subset of H_d, it embeds the signal in A_d^* and returns the result

Require: $I : X \to \mathbb{R}^m$.
  $V(\mathbf{y}) \leftarrow 0, \quad \forall \mathbf{y} \in A_d^*$
  for all $\mathbf{x} \in X$ **do**
    $\mathbf{y} \leftarrow$ the closest remainder-0 point to $\mathbf{x}$ (Lemma 4.9)
    $\mathbf{\Delta} \leftarrow \mathbf{x} - \mathbf{y}$
    $\rho \leftarrow$ the permutation that sorts $\mathbf{\Delta}$ in decreasing order
    for $k = 0$ to d **do**
      $\mathbf{v_k} \leftarrow$ the closest remainder-k point (Lemma 4.10)
      $b_k \leftarrow$ the barycentric coordinate (Proposition 5.2)
      $V(\mathbf{y}) \leftarrow V(\mathbf{y}) + b_k I(\mathbf{x})$
    end for
  end for
  return $V : A_d^* \to \mathbb{R}^m$

Proposition 5.3 *The splatting process is equivalent to convolution with the following kernel* $K_s : H_d \to \mathbb{R}$:

$$K_s : \mathbf{x} \mapsto 1 - \frac{\max_i x_i - \min_i x_i}{d+1}.$$

Proof It suffices to examine the values accumulated at the origin, because of the translational symmetry of the lattice. For each Delaunay cell containing the origin, which is obtained by permuting the coordinates of the canonical simplex, the weight associated to the origin is $1 - \frac{\rho(\mathbf{x})_0 - \rho(\mathbf{x})_d}{d+1}$, where ρ sorts $\mathbf{x}$ in decreasing order. (See Proposition 5.2.) Hence $K_s(\mathbf{x}) = 1 - \frac{\max_i x_i - \min_i x_i}{d+1}$ as desired. □

This kernel is known in the splines literature as an instance of a linear *box spline*, obtained by projecting a higher-dimensional hypercube onto a subspace (de Boor et al. [7]), in this case H_d. Its direction matrix Ξ is the operator T. Entezari et al. [18] previously derived this kernel for $d = 3$.

Proposition 5.4 *Let* $T : \mathbb{R}^{d+1} \to H_d$ *be the projection onto* H_d *as before*. *Then*

$$K_s(\mathbf{x}) = \frac{1}{(d+1)^{3/2}} \lambda\big(\{\mathbf{y} \in [0, d+1]^{d+1} \mid T(\mathbf{y}) = \mathbf{x}\}\big),$$

where λ *is the* 1*-dimensional Lebesgue measure*, *i.e. the length of the set that projects onto* $\mathbf{x}$. *Equivalently*, K_s *is the density of a uniform cube* $[0, d+1]^{d+1}$ *after it is flattened onto* H_d *along* $\mathbf{1}$, *up to a multiplicative factor.*

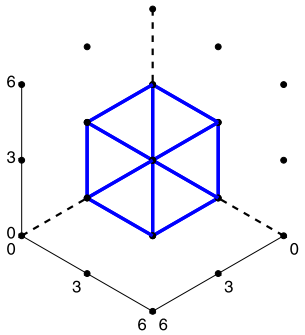

Fig. 8 A plot of the 3-dimensional cube $[0, 3]^3$, along with points in $3\mathbb{Z}^3$. The plot is oriented such that $(1, 1, 1)$ is normal to the plane of the figure. Note that the flattening of the cube along $(1, 1, 1)$ yields the union of six simplices, which equals the support for K_s for $d = 2$

Proof The set $\{\mathbf{y} \in [0, d+1]^{d+1} \mid T(\mathbf{y}) = \mathbf{x}\}$ is the set of vectors that project onto $\mathbf{x}$ via T. Because T removes the component of a vector parallel to $\mathbf{1}$, the set should contain elements of the form $\mathbf{y} = \mathbf{x} + q \cdot \mathbf{1}$ for some $q \in \mathbb{R}$. Since $\mathbf{y} \in [0, d+1]^{d+1}$ as well, it must be that $0 \le x_i + q \le d + 1$ for all i. Equivalently, we have

$$-\min_i x_i \le q \le d + 1 - \max_i x_i. \tag{5.1}$$

The set of $\mathbf{y}$ satisfying the above inequalities is a linear segment parallel to $\mathbf{1}$, so its measure is given by its projected length along any of standard axis multipled by $\|\mathbf{1}\| = \sqrt{d+1}$. Its projected length is simply the extent of q, which is $d + 1 - (\max_i x_i - \min_i x_i)$ according to (5.1). Therefore,

$$\begin{aligned}
&\frac{1}{(d+1)^{3/2}} \lambda\left(\left\{\mathbf{y} \in [0, d+1]^{d+1} \;\middle|\; T(\mathbf{y}) = \mathbf{x}\right\}\right) \\
&= \frac{1}{(d+1)^{3/2}} \sqrt{d+1}\left(d + 1 - \left(\max_i x_i - \min_i x_i\right)\right) \\
&= 1 - \frac{\max_i x_i - \min_i x_i}{d+1} \\
&= K_s(\mathbf{x}) \quad \text{by Proposition 5.3,}
\end{aligned}$$

as desired. □

See Fig. 8 for illustration of Proposition 5.4. This "unflattened" representation facilitates numerical analysis of the splatting kernel, as seen below.

Corollary 5.5

$$\int_{H_d} K_s(\mathbf{x}) d\mathbf{x} = (d+1)^{d-\frac{1}{2}}.$$

Proof We can "unproject" K_s back into a cube with uniform density, and the total mass would remain constant. It follows from Proposition 5.4 that the total mass is $(d+1)^{-3/2} \times \mathrm{Vol}([0, d+1]^{d+1}) = (d+1)^{d-\frac{1}{2}}$. □

Proposition 5.6 *The variance, or the second moment, of K_s as a distribution is $d(d+1)^2/12$.*

Proof By definition,

$$\mathrm{Var}(K_s) = \frac{\int_{H_d} \|\mathbf{x}\|^2 K_s(\mathbf{x}) d\mathbf{x}}{\int_{H_d} K_s(\mathbf{x}) d\mathbf{x}}.$$

The denominator is given by Corollary 5.5. For the numerator, the sum of $\|\mathbf{x}\|^2$ over H_d weighted by the density of K_s is equivalent to an unweighted sum of $\|T(\mathbf{y})\|^2$ over $[0, d+1]^{d+1}$. Hence,

$$\begin{aligned}
&\mathrm{Var}(K_s) \\
&= \frac{\int_{[0,d+1]^{d+1}} \|T(\mathbf{y})\|^2 \frac{1}{(d+1)^{3/2}} d\mathbf{y}}{(d+1)^{d-1/2}} \\
&= \frac{1}{(d+1)^{d+1}} \int_{[0,d+1]^{d+1}} \left\|T(\mathbf{y})\right\|^2 d\mathbf{y} \\
&= \frac{1}{(d+1)^{d+1}} \int_{[0,d+1]^{d+1}} \left\|\mathbf{y} - \frac{\sum y_i}{d+1}\mathbf{1}\right\|^2 d\mathbf{y} \\
&= \frac{1}{(d+1)^{d+1}} \cdot \int_{[0,d+1]^{d+1}} \sum y_i^2 \\
&\quad + \frac{(\sum y_i)^2}{d+1} - \frac{2(\sum y_i)^2}{d+1} d\mathbf{y} \\
&= \frac{1}{(d+1)^{d+1}} \int_{[0,d+1]^{d+1}} \frac{d\sum y_i^2}{d+1} - \frac{\sum_{i \ne j} y_i y_j}{d+1} d\mathbf{y} \\
&= \frac{1}{(d+1)^{d+1}} \sum_i \int_0^{d+1} (d+1)^d \frac{d \cdot y_i^2}{d+1} dy_i \\
&\quad - \frac{1}{(d+1)^{d+1}} \sum_{i \ne j} \iint_0^{d+1} (d+1)^{d-1} \frac{y_i y_j}{d+1} dy_i dy_j \\
&= d(d+1)^2/3 - d(d+1)^2/4 \\
&= d(d+1)^2/12
\end{aligned}$$

as desired. □

5.2 Blurring

A standard Gaussian blur in a d-dimensional space is separable into d Gaussian blurs along each of the d independent axes. The advantage of using $\mathbb{Z}^d$ as the lattice is that the choice of the d axes is straightforward, and that each of the d directional blurs is simple: for all $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{x}$ can be blurred with points of the form $\mathbf{x} \pm \mathbf{u}_k$. Meanwhile, it has been observed that box splines on the body-centered cubic lattice A_3^* can be expressed using four projected axes of the 4D hypercube [19]. We generalize this observation below.

To this end, we use the fact that A_d^* is obtained by applying the projection T on $(d+1)\mathbb{Z}^{d+1} \in \mathbb{R}^{d+1}$. The original space is spanned by the vectors in the set $\{(d+1)\mathbf{u}_k\}$. Because T is linear, we can equivalently state that A_d^* is spanned by their unit-length projections $\mathbf{w}_k$:

$$\mathbf{w}_k := \frac{T(\mathbf{u}_k)}{\|T(\mathbf{u}_k)\|} = \frac{(d+1)\mathbf{u}_k - \mathbf{1}}{\sqrt{d(d+1)}} = \frac{1}{\sqrt{d(d+1)}}(\underbrace{-1,\ldots,-1}_{k}, d, \underbrace{-1,\ldots,-1}_{d-k}). \tag{5.2}$$

Note that $\{\mathbf{w}_k\}$ is therefore a frame of A_d^*. Hence, $\{\mathbf{w}_k\}$ represents a natural choice of $d+1$ axes along which to blur a signal represented in A_d^*.

Of course, $\{\mathbf{w}_k\}$ is not linearly independent, since its cardinality exceeds the rank of the space. Nevertheless, $d+1$ directional blurs along these axes compose to form a standard multi-variate Gaussian blur in H_d, as shown in the following proposition.

Proposition 5.7 *Define $\mathcal{G}_k$ to be the operator that performs Gaussian blur along $\mathbf{w}_k$ with variance θ:*

$$\mathcal{G}_k(f) : \mathbf{x} \mapsto \int_{\mathbb{R}} f(\mathbf{x} - t\cdot\mathbf{w}_k)N(t;\theta)dt,$$

where $f : H_d \to \mathbb{R}$. Then, $\mathcal{G} = \mathcal{G}_0 \circ \cdots \circ \mathcal{G}_d$ performs the standard multi-variate Gaussian blur in H_d with variance $\theta\frac{d+1}{d}$.

Proof Composing $\mathcal{G}_0, \ldots, \mathcal{G}_k$ yields

$$\mathcal{G}(f) : \mathbf{x} \mapsto \int_{\mathbb{R}^{d+1}} f\Big(\mathbf{x} - \sum t_k\cdot\mathbf{w}_k\Big)N(\mathbf{t};\theta)d\mathbf{t}.$$

Then,

$$\begin{aligned}
&\mathcal{G}(f)(x)\\
&= \int_{\mathbb{R}^{d+1}} f\Big(\mathbf{x} - \sum t_k\cdot\mathbf{w}_k\Big)N(\mathbf{t};\theta)d\mathbf{t}\\
&\quad \text{Decompose } \mathbf{t} \text{ into } \mathbf{s} + q\cdot\frac{\mathbf{1}}{\|\mathbf{1}\|},\\
&\quad \text{where } \mathbf{s} \in H_d \text{ and } q \in \mathbb{R}\\
&= \int_{\mathbb{R}}\int_{H_d} f\left(\mathbf{x} - \sum\left(\mathbf{s}_k + \frac{q}{\|\mathbf{1}\|}\right)\cdot\mathbf{w}_k\right)N(\mathbf{t};\theta)d\mathbf{t}\\
&= \int_{\mathbb{R}}\int_{H_d} f\Big(\mathbf{x} - \sum\mathbf{s}_k\cdot\mathbf{w}_k\Big)N(\mathbf{s};\theta)N(q;\theta)d\mathbf{s}dq\\
&\quad \text{as } \mathbf{w_k} \in H_d \text{ so } \sum\frac{q}{\|\mathbf{1}\|}\mathbf{w_k} = 0,\\
&= \int_{H_d} f\Big(\mathbf{x} - \sum\mathbf{s}_k\cdot\mathbf{w}_k\Big)N(\mathbf{s};\theta)d\mathbf{s}\\
&= \int_{H_d} f\left(\mathbf{x} - \alpha\sum\mathbf{s}_k\cdot\left(\mathbf{u}_k - \frac{\mathbf{1}}{d+1}\right)\right)N(\mathbf{s};\theta)d\mathbf{s}\\
&\quad \text{where } \alpha = \sqrt{\frac{d+1}{d}}, \text{ via Eq. (5.2)}\\
&= \int_{H_d} f\left(\mathbf{x} - \alpha\mathbf{s} + \alpha\sum\mathbf{s}_k\cdot\frac{\mathbf{1}}{d+1}\right)N(\mathbf{s};\theta)d\mathbf{s}\\
&= \int_{H_d} f(\mathbf{x} - \alpha\mathbf{s})N(\mathbf{s};\theta)d\mathbf{s}, \quad \text{since } \sum s_k = 0,\\
&= \int_{H_d} f(\mathbf{x} - \mathbf{z})N\left(\frac{\mathbf{z}}{\alpha};\theta\right)\frac{1}{\alpha^d}d\mathbf{z}, \quad \text{letting } z = \alpha s,\\
&= \int_{H_d} f(\mathbf{x} - \mathbf{z})N\big(\mathbf{z};\alpha^2\theta\big)d\mathbf{z}.
\end{aligned}$$

It is clear that $\mathcal{G}(f)$ must be a Gaussian with variance $\theta\alpha^2 = \theta\frac{d+1}{d}$ as claimed. □

Proposition 5.7 demonstrates that a Gaussian blur can be performed on H_d in a "separable" fashion. To adapt this approach onto A_d^*, simply define $K_b^k : H_d \to \mathbb{R}$ as the discretization of $\mathcal{G}_k$:

$$K_b^k : \mathbf{x} \mapsto \sum_{q=-Q}^{Q} N\left(\mathbf{x};\theta\frac{d+1}{d}\right)\cdot\delta(\mathbf{x} - q\cdot\mathbf{t}),$$

where $\mathbf{t} = (\underbrace{-1,\ldots,-1}_{k}, d, -1, \ldots, -1)$. Note that K_b^k is nonzero only at lattice points along the projection of the k-th standard axis. See Algorithm 2 for the exact description of the steps.

Algorithm 2 The blurring algorithm. Data stored in the permutohedral lattice in the form $V : A_d^* \to \mathbb{R}^m$ is blurred with variance θ and stored in $W : A_d^* \to \mathbb{R}^m$. In practice, the domain of V and W is restricted to a finite subset of A_d^*

Require: $V : A_d^* \to \mathbb{R}^m$
Require: $Q \in \mathbb{N}$
Require: $\theta \in \mathbb{R}_+$
  for $k = 0$ to d **do**
    $\mathbf{t} \leftarrow (\underbrace{1,\ldots,1}_{k}, -d, \underbrace{1,\ldots,1}_{d-k})$
    for all $\mathbf{x} \in A_d^*$ **do**
      $W(\mathbf{x}) \leftarrow \sum_{q=-Q}^{Q} V(\mathbf{x} + q\cdot\mathbf{t})\cdot N(q\cdot\mathbf{t};\theta d/(d+1))$
    end for
    $V \leftarrow W$
  end for
  return $I : X \to \mathbb{R}^m$

It is important to realize that the composition of $K_b^0, \ldots, K_b^d$ does not equal K_b, a true Gaussian blur, if it is performed on a lattice instead of a continuous space. This is

Algorithm 3 The slicing algorithm. Given data stored in the permutohedral lattice in the form $W : A_d^* \to \mathbb{R}^m$, the output $I' : X \subset H_d \to \mathbb{R}^m$ is reconstructed

Require: $W : A_d^* \to \mathbb{R}^m$.
  $I'(\mathbf{x}) \leftarrow 0, \quad \forall \mathbf{x} \in X$
  for all $\mathbf{x} \in X$ **do**
    $\mathbf{y} \leftarrow$ the closest remainder-0 point (Lemma 4.9)
    $\mathbf{\Delta} \leftarrow \mathbf{x} - \mathbf{y}$
    $\rho \leftarrow$ the permutation sorting $\mathbf{\Delta}$ in decreasing order
    for $k = 0$ to d **do**
      $\mathbf{v_k} \leftarrow$ the closest remainder-k point (Lemma 4.10)
      $b_k \leftarrow$ the barycentric coordinate (Proposition 5.2)
      $I'(\mathbf{x}) \leftarrow I'(\mathbf{x}) + b_k W(\mathbf{y})$
    end for
  end for
  return $I' : X \to \mathbb{R}^m$

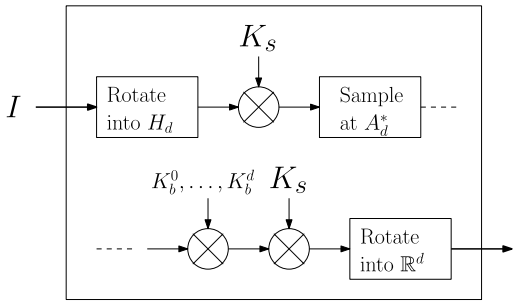

Fig. 9 Summary of Gaussian filtering with A_d^*. The input signal I resides in $\mathbb{R}^d$, and is reparametrized into H_d via an appropriate rotation, yielding a map $I' : H_d \to \mathbb{R}^m$. Each component of I' is then convolved with K_s in the splatting step, is sampled at positions in A_d^*, is convolved with $K_b^0, \ldots, K_b^d$ and K_s in the blurring step and the slicing step, respectively, and rotated back into $\mathbb{R}^d$ to yield an output signal $I'' : \mathbb{R}^d \to \mathbb{R}^m$

the discretization error inherent to any discrete spatial data structures such as lattices. $K_b^0 \circ \cdots \circ K_b^d$ is akin to approximating a multi-variate Gaussian kernel by sampling it at points in A_d^*, up to some discretization error.

In practice, one can use a simpler and more compact set of weights for the directional blur, e.g. for every lattice point $\mathbf{x} \in A_d^*$,

$$W(\mathbf{x}) \leftarrow \frac{V(\mathbf{x}-\mathbf{t})}{4} + \frac{V(\mathbf{x})}{2} + \frac{V(\mathbf{x}+\mathbf{t})}{4}. \tag{5.3}$$

Equation (5.3) amounts to setting $Q = 1$ and $\theta = \frac{\ln\sqrt{2}}{(d+1)^2}$ in Algorithm 2. If a different standard deviation is desired, the input signal may be scaled appropriately instead, as discussed in Sect. 2.2. We used the weights in Eq. (5.3) in the implementation presented in [2], which facilitated computation and proved to be sufficiently close to a Gaussian kernel in practice. The exact implication of these simplified weights are explored in Sect. 6.

5.3 Slicing

The output signal can be reconstructed from the permutohedral lattice by sampling from nearby vertices, as in splatting. The splatting algorithm, given in Algorithm 1, can be adapted with very little change, as shown in Algorithm 3. The slicing process is again equivalent to a convolution with resampling kernel K_s.

Note that for all $\mathbf{x} \in X$, the closest remainder-0 point $\mathbf{y}$ and the permutation ρ were previously computed in the splatting algorithm. By storing these results during the splatting step, we can reduce the runtime of the slicing step.

5.4 Summary

All three steps—splatting, blurring, slicing—are linear in the input, so their composition can be described as a spatially-varying convolution, or alternatively, a chain of convolutions demarcated by sampling as shown in Fig. 9. The relevant kernels for $d = 2$ are visualized in Fig. 10. While the kernels have fixed variance, one can effectively simulate other values of variance by scaling the input signal appropriately before feeding it into the pipeline. Note that the block diagram in Fig. 9 is applicable to the general framework discussed in Sect. 3 based on any lattice, provided that K_s, K_b are replaced with lattice-specific kernels.

5.5 Note on Performance

We implemented the high-dimensional Gaussian filtering algorithm on the permutohedral lattice [2]. This implementation differed from the analogous algorithm based on a Cartesian grid by Paris and Durand [30], in that the vertices of the lattice were stored sparsely using a hash table. Because the dominant use case of high-dimensional Gaussian filtering is to operate on a lower-dimensional signal, a dense data structure like a multi-dimensional array is prohibitively expensive at high dimensionality. Theorems 5.8 and 5.9 demonstrate the runtime and storage requirements of our algorithm.

Theorem 5.8 *Let N be the size of the input. The Gaussian filtering algorithm on A_d^* has a worst-case runtime of $O(d^3 N)$.*

Proof Because each input position vector is splatted onto $d+1$ vertices in A_d^*, there can be at most $(d+1)N$ vertices with nonzero value. The splatting and slicing stage (Algorithm 1) involve identifying the nearest remainder-0 point to each input and sorting its differential, which require $O(d \log d N)$. The blurring stage consists of $O(d)$ separable blurs in a sparse lattice. Each separable blur requires looking up the neighboring vertices, which costs $O(d)$ to compute and $O(d)$ to look up. Hence the total cost of the

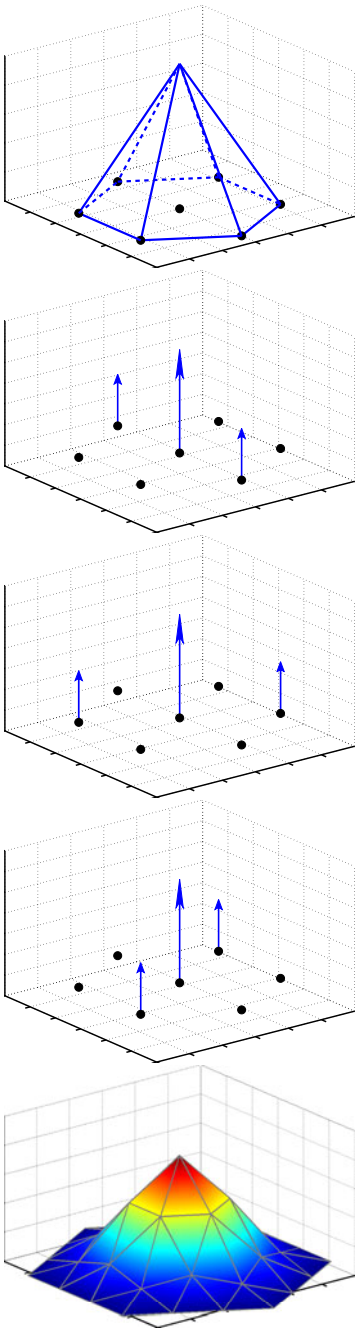

Fig. 10 Visualization of convolution kernels for each step ($d = 2$). *Top*: the splatting or slicing kernel K_s. *Middle (3)*: the three blur kernels K_b^0, K_b^1, K_b^2. *Bottom*: the overall kernel for $\mathbf{x} = 0$

blurring stage is $O(d^3N)$, which dominates the rest of the algorithm. □

While the worst-case runtime is $O(d^3N)$, for most filtering tasks, the number of unique lattice points created in the splatting process is $O(N)$, rather than $O(dN)$, as observed by Adams et al. [2]. This leads to an overall runtime of $O(d^2N)$.

Theorem 5.9 *The Gaussian filtering algorithm on A_d^* has storage requirement of $O(dN)$.*

Proof Recall from the proof of Theorem 5.8 that there are $O(dN)$ vertices of A_d^* that are created during the splatting stage. The näive approach is to store all the vertices in the hash table, keyed by their coordinates. Since each coordinate is of size $O(d)$, this would incur a total storage of $O(d^2N)$.

However, each vertex can be uniquely identified by the index of the input position vector used to create it, along with its remainder. Hence, this pair of information can function as the key in the hash table for the vertex. Each key is of size $O(1)$, adding up to $O(dN)$ over all points.

Using the shortened key raises one possible concern: we have avoided storing the full coordinates of the vertices, but they are required for the common hash-table operations, such as collision checking. To solve this, for each input point $\mathbf{x}$, we store the nearest remainder-0 point $\mathbf{y}$ and the permutation ρ that sorts $\Delta = \mathbf{x} - \mathbf{y}$. These values are computed in the splatting stage (see Algorithm 1) and incur $O(dN)$ total memory. By Theorem 4.10, this allows us to compute any remainder-k point in the Delaunay cell of $\mathbf{x}$ in $O(d)$. Because any operation that involves the explicit coordinate will be $O(d)$ at least, performing $O(d)$ computations to calculate the explicit coordinate does not increase the asymptotic cost of such operations. □

An empirical study of the runtime and storage requirement of the permutohedral lattice, in comparison to the Cartesian grid and other Gaussian filtering algorithms are given in [2].

6 Kernel Analysis

The framework developed in Sect. 3 and the lattice-specific algorithm in Sect. 5 are meant to be fast approximations of a true high-dimensional Gaussian filter. Ideally, the net effect of such a filter should closely resemble that of a true Gaussian. In this section, we test this hypothesis by comparing with true Gaussian the effective kernels that arise from applying the framework to $\mathbb{Z}^d$ and A_d^*.

Formally, let I be a d-dimensional signal $I : \mathbb{R}^d \to \mathbb{R}^m$, and let I' be the resulting signal when we apply our Gaussian filtering algorithm (based on A_d^*). Note that any linear framework can be modeled with a spatially varying convolution kernel. Hence, there exists some family of kernels $\{K_{\mathbf{x}} : H_d \to \mathbb{R}\}_{\mathbf{x}}$ such that

$$I'(\mathbf{x}) = \int_{H_d} K_{\mathbf{x}}(\mathbf{y} - \mathbf{x}) I(\mathbf{y}) d\mathbf{y}.$$

Analogously, let $Z_{\mathbf{x}}$ be the kernel at $\mathbf{x}$ applicable when we apply the same algorithm based on the Cartesian grid $\mathbb{Z}^d$.

For convenience, we presume that the blur kernel in each dimension is $[\frac{1}{4}\frac{1}{2}\frac{1}{4}]$, as in Eq. (5.3).

Proposition 6.1

$$Z_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) = \prod_{i=1}^{d} \sum_{p,q\in\mathbb{Z}} B_p(x_i)B_q(y_i)b(p-q),$$

where

$$b : \mathbf{x} \mapsto \begin{cases} \frac{1}{2}, & x = 0, \\ \frac{1}{4}, & x = \pm 1, \\ 0, & \textit{otherwise}. \end{cases} \qquad B_p : x \mapsto \max(1-|p-x|, 0).$$

Proof The input signal at $\mathbf{y}$ is splatted onto each lattice point $\mathbf{q}$ with multi-linear weight $\prod_{i=1}^{d} B_{q_i}(y_i)$, which is then blurred onto every other lattice point $\mathbf{p}$ with weight $\prod_{i=1}^{d} b(p_i - q_i)$. Then the output at $\mathbf{x}$ is reconstructed by summing over each lattice point $\mathbf{p}$ with weight $\prod_{i=1}^{d} B_{p_i}(x_i)$. The claim in the proposition then follows once we rewrite the sum of product terms as the product of sums. □

Proposition 6.2

$$K_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) = \sum_{\mathbf{p},\mathbf{q}\in A_d^*} B_{\mathbf{p}}(\mathbf{x})B_{\mathbf{q}}(\mathbf{y})b(\mathbf{p}-\mathbf{q}),$$

where $B_{\mathbf{p}}(\mathbf{x})$ is the barycentric weight assigned to point $\mathbf{p}$ for the interpolation of $\mathbf{x}$, and $b(\cdot)$ is the appropriate blurring kernel given by convolving $K_b^0, \ldots, K_b^d$.

Proof The proof is fully analogous to that of Proposition 6.1. □

6.1 Kernel Distance

One metric for evaluating the fidelity of our filtering algorithm is the squared L_2-norm between $K_{\mathbf{x}}$ (or $Z_{\mathbf{x}}$) and an isotropic Gaussian kernel, where the Gaussian kernel has the same total variance as $K_{\mathbf{x}}$ (or $Z_{\mathbf{x}}$). This measure is the continuous equivalent of the Frobenius norm for matrices, and bounds the relative error on arbitrary input signals. As the kernels are spatially varying, i.e. they depend on $\mathbf{x}$, the L_2-norm should be power-averaged over all possible $\mathbf{x}$. It should also be normalized by the power of the Gaussian kernel. We can consider this to be the inverse of the "signal-to-noise ratio" (SNR) where $K_{\mathbf{x}}$ is the observed signal and the Gaussian is the ground truth. Recall that SNR is conventionally defined as,

$$\frac{\text{Power of the signal}}{\text{Power of the noise} = \text{observation} - \text{signal}}.$$

Our definition is fully analogous, and is formalized below:

Definition 6.3 Given a family of kernels $f_{\mathbf{x}} : \mathbb{R}^d \to \mathbb{R}$ parametrized by $\mathbf{x} \in \mathbb{R}^d$ that approximates a Gaussian blur, its *signal-to-noise* ratio (SNR) is

$$\eta := \frac{\int_{\mathbb{R}^d} N(\mathbf{y};\theta)^2 d\mathbf{y}}{E_{\mathbf{x}}[\int_{\mathbb{R}^d} (f_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) - N(\mathbf{y}-\mathbf{x};\theta))^2 d\mathbf{y}]},$$

where θ is the expected variance of $f_{\mathbf{x}}$ over all $\mathbf{x}$. Also, it is assumed that $f_{\mathbf{x}}$ is normalized to sum to 1 over $\mathbb{R}^d$.

Lemma 6.4 *The expected variance of $Z_{\mathbf{x}}$ where $\mathbf{x}$ is chosen randomly in $\mathbb{R}^d$ is $5d/6$. The expected variance of $K_{\mathbf{x}}$ where $\mathbf{x}$ is chosen randomly in H_d is $2d(d+1)^2/3$.*

Proof For $\mathbb{Z}^d$, the multi-linear splatting and slicing are simply the d-dimensional tent functions with variance $d/6$. The blurring stage has variance

$$d \cdot \left[\frac{1}{4}(-1)^2 + \frac{1}{2}(0)^2 + \frac{1}{4}(1^2)\right] = \frac{d}{2}.$$

Hence the total variance is $(d/6)\cdot 2 + (d/2) = 5d/6$.

For A_d^*, Proposition 5.6 tells us that the variance of the splatting and slicing step is $d(d+1)^2/12$ each. The variance of the blurring step is,

$$(d+1)\cdot\left[\frac{1}{4}\left(-\sqrt{d(d+1)}\right)^2 + \frac{1}{2}(0)^2 + \frac{1}{4}\left(\sqrt{d(d+1)}\right)^2\right],$$

which equals $d(d+1)^2/2$. Together, the total expected variance is $2d(d+1)^2/3$. The term *expected* variance is used because the actual variance of a particular kernel $Z_{\mathbf{x}}$ or $K_{\mathbf{x}}$ depend on $\mathbf{x}$, but the additive nature of the variances of each step is clear from the proofs of Propositions 6.1 and 6.2. □

Note that the expected variance is the sum of variance over all dimensions, so it should be divided by d to yield the variance of a comparable isotropic multi-variate Gaussian.

The remaining steps for the calculation of SNR for $Z_{\mathbf{x}}$ and $K_{\mathbf{x}}$ are given in Sects. A.2 and A.3, respectively.

6.2 SNR Comparison

In the Appendix, we derive a closed form of the SNR for $Z_{\mathbf{x}}$; for $K_{\mathbf{x}}$, a recipe for numerically estimating the SNR is provided. The SNRs for the two kernels are plotted in Fig. 11. The SNRs of the two lattices are comparable, with $K_{\mathbf{x}}$ outperforming $Z_{\mathbf{x}}$ at low dimensionality and $Z_{\mathbf{x}}$ outperforming $K_{\mathbf{x}}$ at higher dimensionality. This is explained by the fact that a sample in $\mathbb{Z}^d$ is splatted to exponentially many lattice points as d increases, whereas in A_d^* it is splatted only to few points $(d+1)$. We also compare the SNR with that of true Gaussian kernels that are truncated after certain numbers of standard deviations. Both $Z_{\mathbf{x}}$ and $K_{\mathbf{x}}$ are reasonably

good approximations of the true Gaussian kernel, compared to truncated Gaussians.

Table 2 shows the difference in the SNR and the computational cost of the two kernels. Note that the reduction in computational cost required for filtering using $K_{\mathbf{x}}$ clearly outpaces the SNR loss, which grows very slowly with the dimensionality. This observation is especially significant in light of the fact that the existing implementation of bilateral filtering based on the Cartesian grid becomes prohibitively expensive past $d = 5$ [1, 2].

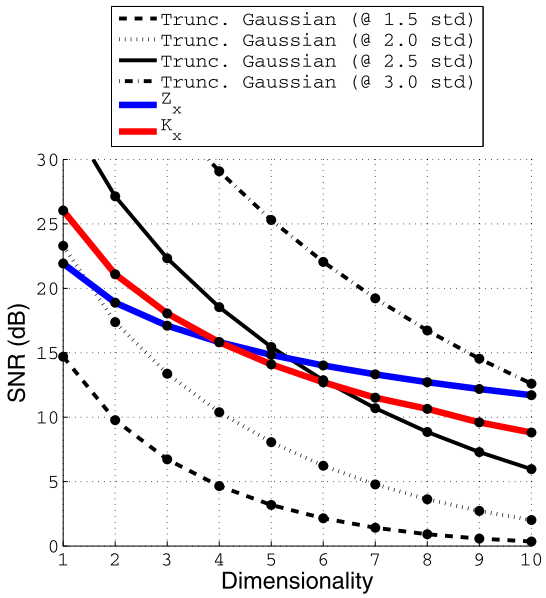

Fig. 11 The signal-to-noise ratio of the kernels $K_{\mathbf{x}}$ and $Z_{\mathbf{x}}$ for $d = 1, \ldots, 10$, along with truncated Gaussian kernels. Note that $K_{\mathbf{x}}$ exhibits SNR that is similar to that of $Z_{\mathbf{x}}$, despite touching only polynomially-many vertices; $Z_{\mathbf{x}}$ touches exponentially many vertices. Because of this difference, the SNR for $K_{\mathbf{x}}$ suffers slightly more when d grows high

6.3 Filtering Comparison

We have thus far established theoretically that K_x reasonably emulates a true Gaussian kernel, albeit not quite as well as Z_x, at a fraction of the computational cost. To confirm this claim empirically, we processed the datasets in Figs. 12, 13, 14 and 15, considering the runtime of the filters and the fidelity of the results. Table 3 shows the performance benchmarks of the two lattices on these two datasets. As expected, the permutohedral lattice offers speed-up of 1–2 orders of magnitude over the Cartesian lattice, at the cost of lower SNR. The only exception occurs with the dataset with the lowest dimensionality ($d = 3$), at which the little gain in runtime complexity is dwarfed by the overhead of computing vertices. Both lattices generally outperform a naïve im-

Table 2 Relative performance and SNR of $K_{\mathbf{x}}$ compared to that of $Z_{\mathbf{x}}$. The cost is measured by the number of lattice points accessed through the splat-blur-slice pipeline ("footprint"), and reported in two columns. The first column assumes that each splat creates the maximum number of new lattice points; the second assumes that the total number of new lattice points created is equal to the number of the input points, which is empirically true in a number of applications

dim.	Change in SNR (dB)	Change in cost	
1	+4.11	+40.0 %	+28.6 %
2	+2.20	−3.6 %	0.0 %
3	+0.95	−38.9 %	−26.1 %
4	+0.00	−63.1 %	−48.8 %
5	−0.73	−78.4 %	−66.7 %
6	−1.30	−87.6 %	−79.4 %
7	−1.81	−93.0 %	−87.8 %
8	−2.08	−96.1 %	−93.0 %
9	−2.59	−97.9 %	−96.1 %
10	−2.90	−98.8 %	−97.8 %

Fig. 12 Bilateral filtering ($d = 5$) result on an 800×532 image "Mountain." The two rows correspond to a spatial kernel of size 5 and 15 pixels, respectively. For the color dimensions, a standard deviation of 0.1 is used. Note that both the Cartesian grid and the permutohedral lattice approximate the ground truth well, removing fine textures while preserving hard edges

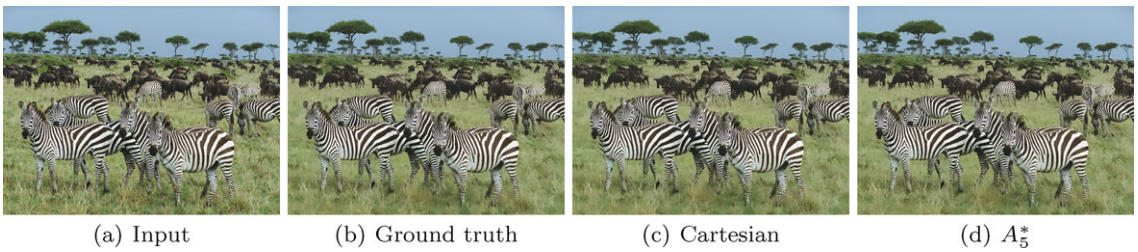

Fig. 13 Bilateral filtering ($d = 5$) result on a 1600×1200 image "Animals." We used spatial standard deviation of 15, and color standard deviation of 0.1

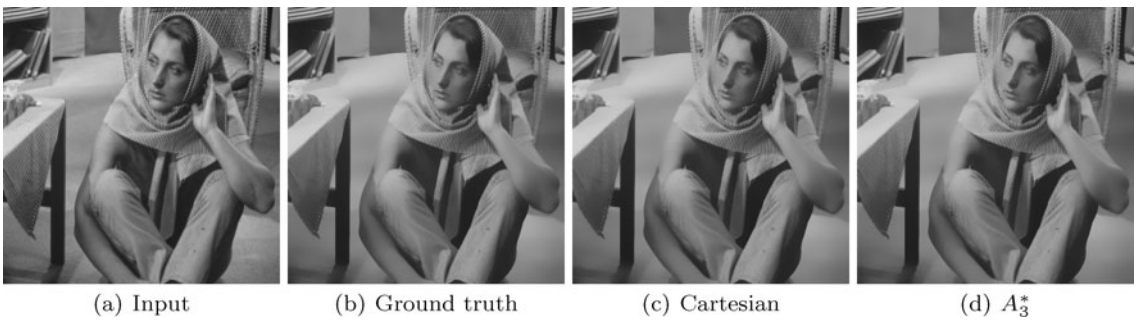

Fig. 14 Bilateral filtering ($d = 3$) result on a 512×512 "Barbara." We used spatial standard deviation of 10, and color standard deviation of 0.08

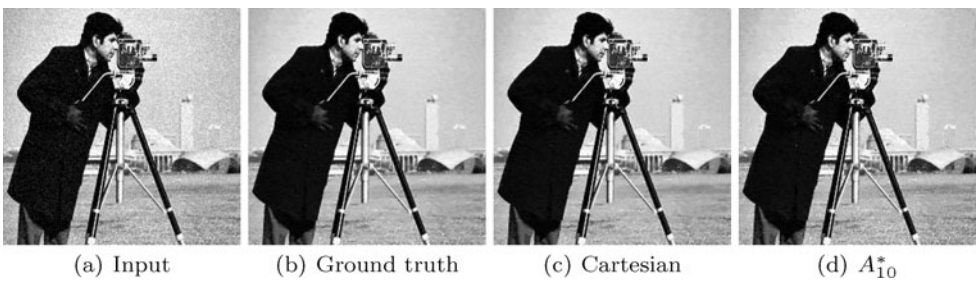

Fig. 15 Non-local means denoising ($d = 10$) result on a 256×256 noisy image "Cameraman." For each pixel, its neighboring 13×13 patch is distilled into an 8-dimensional descriptor using PCA. The spatial dimensions are appended to the descriptor, forming a 10-dimensional position vector. The standard deviation for the spatial dimensions is 15

plementation, although at high dimensionality the Cartesian lattice breaks down from the exponential runtime. Do note that a naïve implementation can only process unstructured data.

We caution that a strictly controlled comparison between the Cartesian grid and the permutohedral lattice algorithm is difficult, as there are many confounding variables, e.g. the choice of reconstruction filter for each lattice, computational complexity, prior on the input, et cetera. The readers should keep in mind that the benchmark in Table 3 was conducted for a particular choice of filters (Z_x and K_x). A related study was done by Mirzargar and Entezari [27], showing that the reconstruction error on A_3^* is less than that on $\mathbb{Z}^3$ or A_3, for first- and second-order Voronoi splines. Following this, we conjecture that it should be possible to design a filter on A_d^* that has equal asymptotic time complexity as on $\mathbb{Z}^d$ and better SNR for Gaussian filtering. However, it remains unclear if subexponential time complexity is possible for a Gaussian filtering algorithm that generalizes across both A_d^* and $\mathbb{Z}^d$. (The Voronoi spline is general, but is exponential; K_x is subexponential as intended, but only exists on A_d^*.)

7 Lattice Optimality

Both the Cartesian grid $\mathbb{Z}^d$ and the permutohedral lattice A_d^* have been used in the literature as the underlying data structure for high-dimensional Gaussian filtering. The choice of $\mathbb{Z}^d$ was the most intuitive one; that of A_d^* was motivated by its low covering density [12]. However, many other lattices exist, and it is certainly possible to fashion a filtering algorithm based on any lattice following the framework laid out in Sect. 3. To explore this possibility further, we

Table 3 Gaussian filtering performance, for examples shown in Figs. 12 and 15. The runtime includes file I/O. As expected, bilateral filtering is considerably faster on the permutohedral lattice, with increasing gains at high dimensionality. While the RMSE is larger as well, its magnitude for $d = 5$ is comparable to the quantization error for 8-bit images. Furthermore, the output images demonstrate that the perceptual difference is nonexistent for either $d = 5$ or $d = 10$. For reference, we include the runtime of an "almost naïve" implementation that brute-forces the answer with a filter truncated at $\pm 2\sigma$

Dataset	Dim.	Spatial Gaussian	Runtime (s)			RMSE		Image SNR (dB)	
			Cartesian	A_5^*	Naïve	Cartesian	A_5^*	Cartesian	A_5^*
"Mountain"	$d = 5$	$\sigma = 5$	10.18	1.16	6.93	0.556	0.695	26.61	25.65
		$\sigma = 10$	3.09	0.65	26.24	0.547	0.622	26.68	26.13
		$\sigma = 15$	1.70	0.54	58.12	0.546	0.623	26.69	26.12
		$\sigma = 20$	1.24	0.49	102.02	0.543	0.628	26.72	26.09
"Animals"	$d = 5$	$\sigma = 15$	8.88	3.58	311.83	0.649	0.777	25.94	25.16
"Barbara"	$d = 3$	$\sigma = 10$	0.10	0.14	11.61	0.839	1.03	24.83	23.94
Dataset	Dim.	Spatial Gaussian	Runtime (s)			RMSE		Image SNR (dB)	
			Cartesian	A_{10}^*	Naïve	Cartesian	A_{10}^*	Cartesian	A_{10}^*
"Cameraman"	$d = 10$	$\sigma = 15$	255.7	2.87	12.81	0.748	2.260	25.33	20.52

begin by listing properties of a lattice that is desirable for high-dimensional Gaussian filtering, given our algorithmic framework, and investigate whether known lattices satisfy these criteria.

7.1 Criteria for the Ideal Lattice

The framework in the previous section guides the designation of appropriate criteria for selecting the ideal lattice for high-dimensional Gaussian filtering. In forming the criteria, we concern ourselves with numerical accuracy, computational efficiency and generality, and also draw from the properties of $\mathbb{Z}^d$ and A_d^* that have proven useful.

Generality There should exist an analogous construction of the lattice for each dimensionality d. For instance, a specialized construction for a particular dimensionality alone has limited utility. Existing applications span a wide range of dimensionality, ranging from two-dimensional cases (e.g. Gaussian blur of an image) to dimensionalities well over 16, such as non-local means filtering of a video [1].

Isotropy The distribution of the points in space should be as even and isotropic as possible. This requirement ensures that the kernels will sufficiently resemble a true Gaussian. A useful measure of isotropy is the covering density described in Sect. 2.4, or the *packing density*, which refers to the portion of space covered when equal spheres are placed at each lattice point so that no spheres overlap [12]. Efficient covering or packing property also indicates that the underlying space can be represented with a small number of points.

Fast Splat and Slice Given $\mathbf{x} \in \mathbb{R}^d$, one must be able to quickly and systematically locate nearby lattice points, i.e. find the Delaunay cell containing $\mathbf{x}$. The number of vertices in the Delaunay cell should scale slowly with d. In general, the number of vertices is bounded below by $d + 1$, since any d-dimensional polytope must have at least $d + 1$ vertices.

Fast Blur Given an arbitrary lattice point, one must be able to quickly and systematically locate nearby lattice points with which to blur, and be able to propagate information efficiently. Several issues affect the performance, such as the number of nearby lattice points to sample and the separability of the blur.

7.2 Identifying the Ideal Lattice

The classification of all possible lattices in an arbitrary dimensionality is an open research problem in mathematics, and is beyond the scope of this paper. However, it does happen that a large class of well-studied lattices can be constructed from a rather simple set of lattices called *root lattices*.

Definition 7.1 A *root system* is a set of nonzero vectors satisfying the following conditions:

1. For every $\mathbf{x}$ in the root system, the only scalar multiples of $\mathbf{x}$ in the root system is $\pm\mathbf{x}$.
2. It is closed under reflection; that is, for every $\mathbf{x}, \mathbf{y}$ in the root system, the reflection of $\mathbf{x}$ over $\mathbf{y}$ is also in the root system.
3. For every $\mathbf{x}, \mathbf{y}$, the projection of $\mathbf{x}$ onto $\mathbf{y}$ is an integral multiple of $\mathbf{y}/2$.

Table 4 Classification of indecomposable root lattices in finite dimensions. The dual lattices are also given for ones that generalize across dimensions, and are denoted by an additional asterisk. The right-hand column gives the number of vertices in the Delaunay cells; some lattices may have more than one type of Delaunay cells that occur. Note that A_d^* is the only family of root lattices that generalizes across d, tessellates the space with a uniform Delaunay cell, and features a small number of vertices in each cell

Lattice	# of vertices in Delaunay cell
A_d	$\binom{d+1}{k}$ for $k \in \{1, \ldots, d\}$
A_d^*	$d+1$
$B_d = B_d^* = \mathbb{Z}^d$	2^d
D_d	$2d$ or $2^{d-1}+1$
D_d^*	$4^{\lfloor d/2 \rfloor}$
E_6	27
E_7	8 or 56
E_8	9 or 16
F_4	8
G_2	3

A root lattice is a lattice generated from using a root system as the frame.

Theorem 7.2 [49] *Let L be an integral lattice, meaning that $\forall \mathbf{x}, \mathbf{y} \in L$, $\mathbf{x} \cdot \mathbf{y} \in \mathbb{Z}$. Then, all sublattices of L generated by vectors of norm* 1 *and* 2 *are direct sums of root lattices.*

Integral lattices are of interest to crystallographers because the generating vectors in their frames will be integrally dependent [41]. For instance, all 14 significant lattices in 3D crystallograpy, called the Bravais lattices, are integral lattices and can be decomposed into root lattices and their duals up to affine transform. Moreover, Theorem 7.2 holds true for all unimodular lattices, as they can be generated by a frame of vectors of norm 1 and 2 [12].

Therefore, root systems generate the most common lattices and more. Theorem 7.2 does leave out non-integral lattices, but we will further defend this shortly. Fortunately, indecomposable root systems in finite dimensions have been fully enumerated:

Theorem 7.3 [40, 42] *There exists exactly the following indecomposable root systems*:

$$A_d,\ B_d,\ D_d,\ E_6,\ E_7,\ E_8,\ F_4 \text{ and } G_2.$$

Note that B_d is the Cartesian lattice $\mathbb{Z}^d$.

Theorems 7.2 and 7.3 together imply that in our exploration of alternate lattices, we may safely restrict our search to these irreducible root lattices and their duals, and this would adequately cover the most studied lattices.

The Voronoi and Delaunay cells of irreducible root lattices are enumerated by Mood and Patera [28]. Those of the dual lattices can be found without much difficulty. Table 4 lists the root lattices along with their duals, and the number of vertices in each of their Delaunay cells. This number must be small in order to support fast splatting and slicing, and the lattice should generalize across dimensions. As shown, only A_d^*, the permutohedral lattice, meets these criteria. In fact, A_d^* achieves the lower bound for the number of vertices possible for any d-dimensional Delaunay cell.

Let us further remark on this criterion: ideally, the Delaunay cell is a simplex, in order to minimize the number of vertices. This condition is equivalent to requiring that the Voronoi cell of the lattice is a primitive parallelohedron, which we discussed in Sect. 3:

Definition 7.4 A *primitive* parallelohedron is a parallelohedron for which the Delaunay cells of the induced tiling are simplices.

Theorem 7.5 [14, 38, 45] *Any primitive parallelohedron is affinely equivalent to the Voronoi cell of some integral lattice.*

Theorem 7.5 justifies our concentration on integral lattice and the use of Theorem 7.2.

Revisiting the criteria in Sect. 7.1, we note that A_d^* is the only generalizable lattice that admits the lower bound for the number of vertices in a Delaunay cell. Moreover, A_d^* exhibits the most efficient known covering for lattices up to $d = 22$, satisfying the criterion on isotropy. A_d^* also provides fast blurring, as it allows for a separable Gaussian kernel similar to that for $\mathbb{Z}^d$ (Sect. 5.2).

In summary, the permutohedral lattice A_d^* is the only root lattice or the dual of a root lattice that meets the criteria spelled out in Sect. 7.1. It is computationally efficient (fast splatting, blurring and slicing), exhibits isotropic distribution, and generalizes across dimensions. While this does not completely rule out integral lattices not covered by Theorem 7.2 from being superior, it does strongly suggest the optimality of A_d^*.

8 Mean Shift Filtering

In addition to Gaussian filtering, another natural application of the permutohedral lattice is mean shift filtering. We show that we can easily adapt the *splatting*, *blurring* and *slicing* process described in the context of Gaussian filtering to create a mean shift filter.

In mean shift filtering, each point is assigned a color based on the value of its nearest mode in feature space. Comaniciu and Meer [11] and Cheng [10] evaluate the properties of mean shift, showing that for images it is equivalent to an ascent toward the modes of the probability density function computed in the joint color-spatial domain. We build

upon the work of Paris and Durand [31] in which they use a grid to improve the efficiency of density estimation and mode finding, and utilize the permutohedral lattice to accelerate the mean shift filtering.

Note that while Gaussian filtering is useful in tasks like de-noising, mean shift filtering is useful in creating piecewise-constant images similar to those in [48]. These cartoon-like images can be used for segmentation, abstraction, or decomposition. While the grids used in [29, 31] restricted the dimensionality of the descriptor space, using the permutohedral lattice, we can efficiently scale mean shift filtering to descriptors with higher dimensionality.

The same principles of steepest ascent toward modes also enable mean shift to be applied to clustering. Instead of replacing pixel colors, one could group pixels initially by their proximity to modes. In the case of images, one could then efficiently calculate a hierarchical segmentation by successively merging different clusters based on a similarity threshold, following the method of Paris and Durand [31] and similar to Grundmann et al. [21] and Arbelaez [3]. A complete discussion and evaluation of hierarchical image segmentation is outside the scope of this work, although we demonstrate the usefulness of the permutohedral lattice as a first stage in a hierarchical clustering algorithm.

The mean shift filtering algorithm can be decomposed into three major parts: density estimation, mode extraction, and cluster assignment. We detail each part below in the context of mean-shift filtering of images using the permutohedral lattice.

8.1 Density Estimation

The first two steps of the Gaussian filtering algorithm, *splatting* and *blurring*, also serve as an efficient means for density estimation of the underlying descriptor distribution in feature space.

In the splatting stage, for each sample of the input signal, its value is accumulated at one or more spatially proximate lattice points. We maintain the same formulation as discussed above, where the splatting kernel K_s can be described as a tent function. After splatting, the homogeneous weights available at each lattice point represent a coarse estimate of the underlying density function. We then refine this density estimate in the blur stage, where we perform a discrete approximation of a Gaussian blur in each dimension of the lattice. After splatting and blurring, the weights at each lattice point now approximate a density estimate $D : A_d^* \to \mathbb{R}_+$ with

$$D : \mathbf{x} \mapsto \sum_i N\big(\mathbf{x} - \mathbf{p}_i; \sqrt{7d(d+1)^2/12}\big),$$

representing the density estimate at $\mathbf{x}$. The standard deviation follows easily from the proof of Lemma 6.4.

8.2 Cluster Assignment

After the *splatting* and *blurring* stages, each node in the lattice now contains a density estimate in the value of its homogeneous coordinate. As discussed by Paris [29], nodes that are local maxima in our density estimate contain the values that we wish to propagate to neighboring non-maximal nodes.

As described in Algorithm 4, we perform an iterative mode-finding process where each node in the lattice considers both itself and its neighbors. Here neighbors of a lattice point $\mathbf{x}$ are defined to be the vertices connected to $\mathbf{x}$ in the Delaunay triangulation of A_d^*; these are the same vertices with which $\mathbf{x}$ is averaged in the blurring stage. In Theorem 8.1 we show that the number of iterations of this algorithm grows logarithmically with the number of vertices in the lattice.

Theorem 8.1 *The mean-shift filtering algorithm in Algorithm 4 has time complexity $O(d^2N + N \log N)$ for N vertices in d dimensions.*

Proof The algorithm described has two steps. In the first step, all local maxima are identified and marked as done, and all other points are assigned its nearest neighbor with the

Algorithm 4 The cluster assignment algorithm. Given an initial density estimate $D(\mathbf{x})$ at each lattice point $\mathbf{x}$, it assigns each lattice point to a nearby lattice point whose density estimate is the local maxima. This mapping is denoted by $M : A_d^* \to A_d^*$ below. An auxiliary boolean function `done` tracks whether $M(\mathbf{x})$ is correct

Require: $D : A_d^* \to \mathbb{R}_+$.
(Step 1):
for all $\mathbf{x} \in A_d^*$ **do**
  if $D(\mathbf{x}) \geq$ density of neighbors of $\mathbf{x}$ **then**
    done($\mathbf{x}$) $\leftarrow$ true
    $M(\mathbf{x}) \leftarrow \mathbf{x}$
  else
    done($\mathbf{x}$) $\leftarrow$ false
    $M(\mathbf{x}) \leftarrow \mathbf{y}$, where $\mathbf{y}$ is the neighbor of $\mathbf{x}$ with the highest value of D.
  end if
end for
(Step 2):
while $\exists \mathbf{x} \in A_d^*$ where done($\mathbf{x}$) = false **do**
  for all $\mathbf{x} \in A_d^*$ where done($\mathbf{x}$) = false **do**
    $M(\mathbf{x}) \leftarrow M(M(\mathbf{x}))$
    done($\mathbf{x}$) = done($M(\mathbf{x})$)
  end for
end while
return $M : A_d^* \to A_d^*$.

Fig. 16 Progression of the cluster assignment algorithm. The *leftmost image* is the input image. *Each image to the right* is the result of 1, 2, 3, 4, 5 and 6 iterations of mode finding on the lattice, respectively. For this image, cluster assignment converges after the 6th iteration

largest density. The cost of this step is $O(d^2N)$ for N lattice points in d dimensions, as each lattice point must perform an $O(d)$ hash-table lookup on each of its d neighbors.

The second step consists of a number of iterations in which the mode estimate is updated. To bound the number of iterations, first fix a vertex $\mathbf{x}$, and let $\{\mathbf{x}_0 = \mathbf{x}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ be the sequence of neighboring vertices of increasing density, such that $M(\mathbf{x}_j) = \mathbf{x}_{j+1}$ for all $1 \leq j < k$ after the first loop is concluded. $\mathbf{x}_k$ is a local maximum. Clearly, the length of such a sequence is bounded by the number of total vertices.

In each iteration, the mapping M is updated, such that $\mathbf{x}_0$ is now assigned a vertex further down in this chain. A simple recursive argument tells us that after l iterations, $M(\mathbf{x}_0)$ is at least $\mathbf{x}_{2^l}$, depending on evaluation order. Hence, $M(\mathbf{x})$ must hold the value $\mathbf{x}_k$ within a logarithmic number of iterations. The value of "done" propagates downward at the same time in a similar fashion; after l iterations, all the vertices within at least 2^l hops from the local maximum will be marked as being done.

Therefore, the number of iterations of the while loop is logarithmic in the number of vertices, whereas the inner loop is linear. In this second stage there is no use of the positions of the lattice nodes, and so the complexity does not depend on d. Overall, the runtime of this algorithm is $O(d^2N + N \log N)$. □

We can visualize the output as the mode estimate M is being updated, by replacing each pixel $\mathbf{x}$ with the color of $M(\mathbf{x})$. (See Sect. 8.3 for details.) Figure 16 illustrates this progression of value assignment, for 1, 2, 3, 4, 5, and 6 iterations, respectively, where convergence is reached at iteration 6. For most natural images, we find that this process converges within 4 to 6 iterations, as the number of iterations is bounded by the log of the number of lattice points within the largest cluster, and clusters tend to be small relative to the image. d^2 is typically much larger than the number of iterations, and so in practice the runtime is dominated by the first stage of Algorithm 4 and by the earlier blurring stage, which both take $O(d^2N)$ time.

8.3 Pixel Color Assignment

We slightly alter the original *slicing* stage discussed in Sect. 5.3 in the context of Gaussian filtering to complete our mean-shift filter.

Algorithm 5 The modified slicing algorithm. Given the data stored in the lattice, along with density and mode assignment, the output $I' : X \subset H_d \to \mathbb{R}^m$ is reconstructed

Require: $D : A_d^* \to \mathbb{R}_+$.
Require: $W : A_d^* \to \mathbb{R}^m$ (from Algorithm 2).
Require: $M : A_d^* \to A_d^*$ (from Algorithm 4).
  for all $\mathbf{x} \in X$ **do**
    Identify the simplex $\{\mathbf{x}_0, \ldots, \mathbf{x}_d\}$ enclosing $\mathbf{x}$.
    $j \leftarrow \max_i D(\mathbf{x}_i)$
    $I'(\mathbf{p}) \leftarrow W(M(\mathbf{x}_j))$ to $\mathbf{p}$
  end for
  return $I' : X \to \mathbb{R}^m$.

As in slicing, the output can be constructed from the permutohedral lattice by sampling from nearby lattice points. We assign each pixel to have the value of the mode of the nearest lattice point to the pixel. The nearest lattice point to each pixel can be easily identified and stored during the splatting stage; it is the one with the largest barycentric weight. This assignment leads to a consistent segmentation in which pixels assigned to the same mode adopt the same color. Algorithm 5 formalizes this description.

8.4 Discussion

As with the Gaussian filter, all three steps are linear in the input size or linear in the number of lattice vertices, which is bounded by the input size in non-degenerate cases. The additional step of propagating local extrema in density has the same complexity as the blurring step, so it does not increase the total runtime of the algorithm. Therefore, the time complexity of our mean-shift filtering algorithm is $O(d^2N)$.

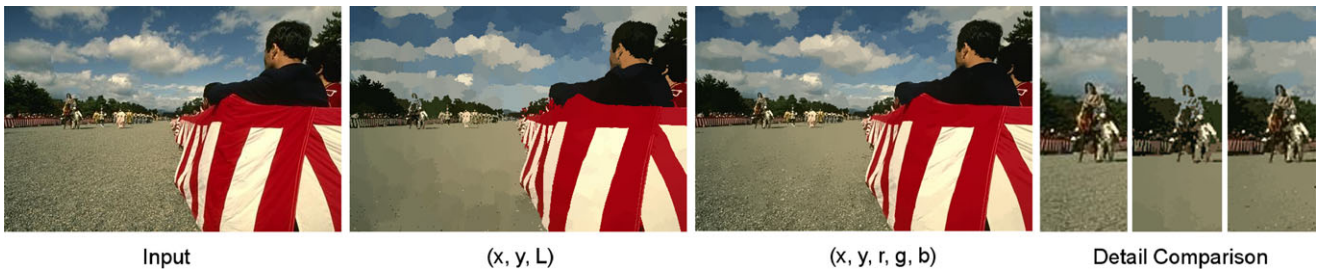

Fig. 17 Stability of the mean-shift filters with various constraints. The first filter is constrained only by $(x, y,$ Luminance) position vectors, while the second has five-dimensional (x, y, r, g, b) position vectors. We show the original input, followed by the output from the (x, y, L) and (x, y, r, g, b) filters respectively. In the comparison, we see that areas of similar luminance but not similar chorminance values may be grouped together in the (x, y, L) filter, where the five-dimensional filter better preserves the oringal input object

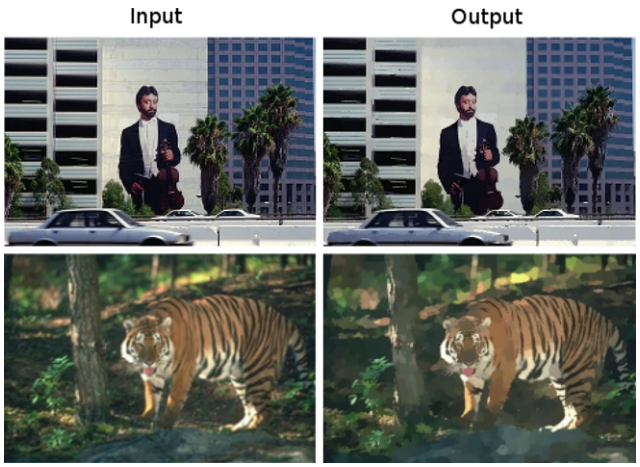

Fig. 18 Example output of mean shift filters in (x, y, r, g, b) applied to images from the Berkeley database [26]

To control the variance of the Gaussian blur, again one may scale the input signal appropriately before feeding it into the pipeline.

Using our lattice, we can efficiently scale the dimensionality of the descriptor used to filter, as discussed by Adams et al. [2] and [1]. This efficiency of scaling in terms of both storage and runtime complexity enables a higher level of accuracy in filtering without being limited in dimensionality or using aggressive descriptor compression (e.g. using PCA to reduce the descriptor to a small number of dimensions). In Fig. 17 we compare the results from a three-dimensional filter constrained by pixel position and luminance value, and a five-dimensional filter with pixel position and r, g, b constraints. For both filters, the spatial standard deviation was set at 5 pixels, and the color standard deviation was set at 0.03, where L, r, g, and, b $\in [0, 1]$. Figure 18 contains additional filtering results using (x, y, r, g, b) descriptors. All images in Figs. 16, 17, and 18 are from the Berkeley segmentation data set [26].

The algorithm described above is straightforward to implement within our existing filtering framework, only requiring an extra pointer and boolean value per lattice node to perform the iterative updates in Algorithm 4. It therefore has the same space complexity as the Gaussian filter: $O(dN)$. We empirically compare the runtime of our permutohedral-lattice-based mean shift algorithm against that of the Cartesian-lattice-based analogue by Paris and Durand [31], in Table 5.

Table 5 Mean-shift filtering performance for the permutohedral lattice and the Cartesian grid. The input dataset "Animals" (also used in Fig. 13, 1600×1200) is shown in the top left corner. The output is shown in the top right corner. A color standard deviation of 0.1 was used in both algorithms. In both algorithms, the splatting step accounts for the largest fraction of the runtime (roughly 60 %), followed by the labeling step. The permutohedral-lattice-based algorithm converged after 4 iterations of mode-finding

Dim.	Spatial Gaussian	Runtime (s)	
		Cartesian	A_5^*
$d = 5$	$\sigma = 15$	18.58	6.30

9 Conclusions

We have presented a framework for performing high-dimensional Gaussian filtering on a lattice, along with a particular implementation thereof using the permutohedral lattice. We further provide a justification on the choice of the permutohedral lattice both theoretically and empirically; on one hand the study of root lattices points strongly towards the complexity-theoretic optimality of the permutohedral lattice, and on the other hand the calculation of the expected deviation from the true Gaussian kernel assures

us of the fidelity of our filtering algorithm, and the filtering results confirm our claims. Lastly, we showed that the high-dimensional Gaussian filtering framework can easily be modified or extended to achieve other interesting effects, such as the mean-shift filtering, and hope that it will inspire more clever uses of this lattice-based framework.

Acknowledgements We would like to thank Marc Levoy for his advice and support, as well as Nokia Research.

Jongmin Baek acknowledges support from Nokia Research, as well as Lucent Technologies Stanford Graduate Fellowship; Andrew Adams is supported by a Reed-Hodgson Stanford Graduate Fellowship; Jennifer Dolson acknowledges support from an NDSEG Graduate Fellowship from the United States Department of Defense.

Appendix

A.1 Useful Lemmas

This section outlines a number of useful lemmas required for computing the SNR terms for K_x and Z_x. Most of them concern the permutohedral lattice—more machinery is required for the derivations on A_d^*. They are included here so that Sect. A.2 and A.3 can remain succinct and parallel.

Lemma A.1 *The power of a d-dimensional Gaussian kernel with covariance θI is $(4\pi\theta)^{-d/2}$.*

Proof

$$\begin{aligned}\int_{\mathbb{R}^d} N(\mathbf{y};\theta)^2 dy &= \left[\int_{-\infty}^{\infty} N(y;\theta)^2 dy\right]^d \\ &= \left[\int_{-\infty}^{\infty} \frac{\exp(-y^2/(2\theta))^2}{2\pi\theta} dy\right]^d \\ &= \left[\int_{-\infty}^{\infty} \frac{\exp(-y^2/(2\cdot\frac{\theta}{2}))}{\sqrt{2\pi\frac{\theta}{2}}\sqrt{4\pi\theta}} dy\right]^d \\ &= \left[\frac{1}{\sqrt{4\pi\theta}}\int_{-\infty}^{\infty} N(y;\theta/2) dy\right]^d \\ &= (4\pi\theta)^{-d/2},\end{aligned}$$

as desired. □

Lemma A.2 *Let $\mathcal{S}$ be a d-dimensional simplex defined by vertices $\mathbf{s_0},\ldots,\mathbf{s_d}$, and let $B_0(\mathbf{x}), B_1(\mathbf{x}),\ldots,B_d(\mathbf{x})$ be the barycentric coordinates of $\mathbf{x}$ for the vertices in the given order. Then, $\forall k, j \in \{0,\ldots,d\}$,*

$$\frac{\int_{\mathcal{S}} B_k(\mathbf{x})B_j(\mathbf{x})d\mathbf{x}}{\int_{\mathcal{S}} d\mathbf{x}} = \begin{cases}\frac{2}{(d+1)(d+2)}, & k=j,\\ \frac{1}{(d+1)(d+2)}, & k\neq j.\end{cases}$$

Proof Let P be the face defined by all vertices except $\mathbf{s_k}$. Then $\mathcal{S}$ is a pyramid with base P and apex $\mathbf{s_k}$. Let h be the height of the altitude and let $\mathbf{y}$ be the foot of the altitude. Consider slicing $\mathcal{S}$ along a hyperplane parallel to P that contains $(1-t)\cdot\mathbf{s_k}+t\cdot\mathbf{y}$, for some $t\in[0,1]$. The slice will be similar to P with a scale factor of t, and by definition of barycentric coordinates, $B_k(\cdot)$ will evaluate to a constant, namely $1-t$, across the whole slice. Therefore,

$$\begin{aligned}\int_{\mathcal{S}} B_k(\mathbf{x})^2 d\mathbf{x} &= \int_0^1 \int_{(1-t)\mathbf{s_k}+tP} (1-t)^2 d\mathbf{x}(h\cdot dt) \\ &= \int_0^1 (1-t)^2\left[\int_{tP} d\mathbf{x}\right](h\cdot dt) \\ &= \int_0^1 (1-t)^2 t^{d-1}\mathrm{Vol(P)}h\cdot dt \\ &= \mathrm{Vol}(P)\cdot h\cdot\int_0^1 (1-t)^2 t^{d-1} dt \\ &= \mathrm{Vol}(P)\cdot h\cdot\frac{2}{d(d+1)(d+2)}. \qquad (A.1)\end{aligned}$$

Note that the same integral without the weight term $(1-t)^2$ will simply yield the volume of $\mathcal{S}$:

$$\begin{aligned}\int_0^1\int_{tP} d\mathbf{x}(h\cdot dt) &= \mathrm{Vol}(P)\cdot h\cdot\int_0^1 t^{d-1} dt \\ &= \mathrm{Vol}(P)\cdot h\cdot\frac{1}{d}. \qquad (A.2)\end{aligned}$$

Dividing (A.1) by (A.2) yields the first of the two cases. When $k\neq j$, we consider applying the same decomposition to P: let P' be the face of P spanned by all vertices of $\mathcal{S}$ except $\mathbf{s_k}$ and $\mathbf{s_j}$. Then P is again a pyramid with base P' and height h', and y' the foot of the altitude. Each slice of P parallel to P' cutting the altitude at $(1-r)\cdot\mathbf{s_j}+r\cdot\mathbf{y}'$ is again a copy of P' scaled by r, and the barycentric weight $B_j(\cdot)$ is uniformly $1-r$. Therefore,

$$\begin{aligned}&\int_{\mathcal{S}} B_k(\mathbf{x})B_j(\mathbf{x})d\mathbf{x} \\ &= \int_0^1\int_0^1 (1-t)\cdot t(1-r) \\ &\quad\cdot\left(\mathrm{Vol}(P')t^{d-1}r^{d-2}\right)(h'\cdot dr)(h\cdot dt) \\ &= hh'\,\mathrm{Vol}(P')\int_0^1 t^d - t^{d+1} dt\int_0^1 r^{d-2} - r^{d-1} dr \\ &= hh'\,\mathrm{Vol}(P')\frac{1}{(d+1)(d+2)(d-1)d}. \qquad (A.3)\end{aligned}$$

Without the barycentric weights,

$$\int_{\mathcal{S}} d\mathbf{x} = \int_0^1 \int_0^1 \left(\mathrm{Vol}(P')t^{d-1}r^{d-2}\right)(h' \cdot dr)(h \cdot dt)$$
$$= hh' \,\mathrm{Vol}(P') \int_0^1 t^{d-1}dt \int_0^1 r^{d-2}dr$$
$$= hh' \,\mathrm{Vol}(P') \frac{1}{d(d-1)}. \tag{A.4}$$

Dividing (A.3) by (A.4) yields the second of the two cases, as desired. □

Lemma A.3 *Let* $\mathbf{x}, \mathbf{y} \in A_d^*$. *Then the number of Delaunay cells containing both* $\mathbf{x}$ *and* $\mathbf{y}$ *is given by*

$$\begin{cases} (d+1-k)!k!, & \text{if } \mathbf{x}-\mathbf{y} \text{ is some permutation of the remainder-}k \text{ vertex of the canonical simplex,} \\ 0, & \text{otherwise.} \end{cases}$$

Proof Since every Delaunay cell is given by permuting the coordinates of the canonical simplex and translating the vertices, if $\mathbf{x}-\mathbf{y}$ cannot be written as a permutation of a vertex of the canonical simplex, then they must not belong to any common Delaunay cell. So we can assume that $\mathbf{x}-\mathbf{y}$ is a permutation of some vertex of the canonical simplex.

Without loss of generality, let $\mathbf{y} = 0$. Then any simplex containing $\mathbf{y}$ has a corresponding permutation in S_{d+1}. Among those, the number of simplices also containing $\mathbf{x}$ is the number of permutations that sends the remainder-k vertex of the canonical simplex to $\mathbf{x}$.

Because components of $\mathbf{x}$ have two possible values, namely k and $k-(d+1)$, and the number of these two values are $d+1-k$ and k respectively, the number of possible permutations is $(d+1-k)!k!$. □

The next pair of lemmas illustrate how to integrate arbitrary functions over the canonical simplex with particular weighting schemes.

Lemma A.4 *Let* X *be the union of Delaunay cells containing the origin in* A_d^*, *and let* $f : X \to \mathbb{R}$ *be a function. Then,*

$$\int_X f(\mathbf{x}) \cdot K_s(\mathbf{x}) d\mathbf{x} = \frac{1}{(d+1)^{3/2}} \int_{[0,1]^{d+1}} f\big(T\big((d+1)\mathbf{x}\big)\big) d\mathbf{x}.$$

Proof This is a direct consequence of Proposition 5.4. In short, weighted sampling corresponding to K_s can be achieved by uniformly sampling in the unit cube and projecting the sample onto H_d. □

Lemma A.5 *Let* $f : A_d^* \to \mathbb{R}$ *be a function on* A_d^*. *Let* $b : A_d^* \to \mathbb{R}$ *and the vectors* $\mathbf{w_k}$ *be as in Sect.* 5.2. *Then,*

$$\sum_{\mathbf{q} \in A_d^*} f(\mathbf{q}) b(\mathbf{q}) = \sum_{\mathbf{t} \in \{-1,0,1\}^{d+1}} f\Big(\sum_k t_k \cdot \mathbf{w_k}\Big) \prod_k \frac{1}{2^{1+|t_k|}}.$$

Proof The blur kernel $b(\mathbf{x})$ is given by the convolution of $K_b^0, \ldots, K_b^d$. Therefore, it is the accumulation of the weights corresponding to offsets in the $d+1$ axes that sum to $\mathbf{x}$. The weight equals $\frac{1}{2}$ if the offset t_k in a given axis is zero, and $\frac{1}{4}$ if it is $t_k = \pm 1$. Hence we can write it succinctly as $\frac{1}{2^{1+|t_k|}}$. The claim follows. □

A.2 Evaluation of SNR for $\mathbb{Z}^d$

The kernel $Z_{\mathbf{x}}$ repeats periodically because of the translational symmetry of $\mathbb{Z}^d$. In order to evaluate the expected value of an expression involving $Z_{\mathbf{x}}$, it suffices to consider $\mathbf{x} \in [0,1]^d$. In other words,

$$\eta = \frac{\int_{\mathbb{R}^d} N(\mathbf{y}; 5/6)^2 d\mathbf{y}}{\int_{[0,1]^d} \int_{\mathbb{R}^d} \left(Z_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) - N(\mathbf{y}-\mathbf{x}; 5/6)\right)^2 d\mathbf{y} d\mathbf{x}}.$$

Expanding the quadratic term in the integrand, we obtain $\eta = \frac{(A.6)}{(A.5)+(A.6)-2(A.7)}$, containing the following three expressions:

$$\int_{[0,1]^d} \int_{\mathbb{R}^d} Z_{\mathbf{x}}(\mathbf{y}-\mathbf{x})^2 d\mathbf{y} d\mathbf{x}, \tag{A.5}$$

$$\int_{\mathbb{R}^d} N(\mathbf{y}; 5/6)^2 d\mathbf{y}, \tag{A.6}$$

$$\int_{[0,1]^d} \int_{\mathbb{R}^d} Z_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) N(\mathbf{y}-\mathbf{x}; 5/6) d\mathbf{y} d\mathbf{x}. \tag{A.7}$$

First of all, Lemma A.1 tells us that (A.6) is $(10\pi/3)^{-d/2}$. (A.5) is similarly separable:

$$\int_{[0,1]^d} \int_{\mathbb{R}^d} Z_{\mathbf{x}}(\mathbf{y}-\mathbf{x})^2 d\mathbf{y} d\mathbf{x}$$
$$= \int_{[0,1]^d} \int_{\mathbb{R}^d} \left[\prod_{i=1}^d \sum_{p,q \in \mathbb{Z}} B_p(x_i) B_q(y_i) b(p-q)\right]^2 d\mathbf{y} d\mathbf{x}$$
$$= \left[\int_{[0,1]} \int_{\mathbb{R}} \left[\sum_{p,q \in \mathbb{Z}} B_p(x) B_q(y) b(p-q)\right]^2 dy dx\right]^d$$
$$= \left[\int_{[0,1]} \int_{\mathbb{R}} \sum_{p,p',q,q' \in \mathbb{Z}} B_p(x) B_{p'}(x) B_q(y) B_{q'}(y) \times b(p-q) b(p'-q') dy dx\right]^d$$

$$= \Bigg[\sum_{p,p',q,q' \in \mathbb{Z}} b(p-q)b(p'-q') \times \int_{\mathbb{R}} B_q(y)B_{q'}(y)dy \int_{[0,1]} B_p(x)B_{p'}(x)dx \Bigg]^d.$$

Note that $B_p(x)$ and $B_{p'}(x)$ are both nonzero only if $p = p'$ or $|p - p'| = 1$. In the former case, the integral of $B_p(x)B_{p'}(x)$ with respect to x over $\mathbb{R}$ is $2/3$; in the latter case, $1/6$. Similarly, $B_q(y)$ and $B_{q'}(y)$ are both nonzero over $y \in [0,1]$ iff $q, q' = 0$ or 1. Setting $q = q'$ yields $1/3$ for the value of the integral, and $q \neq q'$ yields $1/6$. Summing over all possible values of p', q, q' with nonzero summands, we obtain that (A.5) equals

$$\Bigg[\sum_{p \in \mathbb{Z}} \frac{b(p)^2}{2} + \frac{4b(p)b(p-1)}{9} + \frac{b(p)b(p-2)}{18} \Bigg]^d = \Bigg[\frac{29}{96} \Bigg]^d.$$

The cross term (A.7) remains. We can similarly expand the expression and collect terms into the product of sums, and exploit the fact that only few values of p, q yield nonzero summands. Unfortunately, because of the Gaussian term, there is no succinct analytic expression.

$$\begin{aligned}
(\text{A.7}) &= \int_{[0,1]^d} \int_{\mathbb{R}^d} Z_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) N(\mathbf{y}-\mathbf{x}; 5/6) d\mathbf{y} d\mathbf{x} \\
&= \int_0^1 \int_{\mathbb{R}^d} \prod_{i=1}^{d} \sum_{p,q \in \mathbb{Z}} B_p(x_i)B_q(y_i) \\
&\quad \times b(p-q)N(y_i - x_i; 5/6) d\mathbf{y} d\mathbf{x} \\
&= \Bigg[\sum_{p,q \in \mathbb{Z}} \int_0^1 \int_{\mathbb{R}} B_p(x)B_q(y) \\
&\quad \times b(p-q)N(y-x; 5/6) dy dx \Bigg]^d \\
&= \Bigg[\sum_{q \in \{-1,0,1,2\}} \int_0^1 \int_{\mathbb{R}} \big\{ (1-x)b(q) + xb(q-1) \big\} \\
&\quad \times B_q(y) N(y-x; 5/6) dy dx \Bigg]^d \\
&\simeq 0.30456.
\end{aligned}$$

Corollary A.6 *The SNR term for $\mathbb{Z}^d$ is*

$$\eta = \frac{\sqrt{3/(10\pi)}^d}{(29/96)^d + \sqrt{3/(10\pi)}^d - 2 \cdot 0.30456^d}.$$

A.3 Evaluation of SNR for A_d^*

The error analysis for A_d^* is analogous to that of $\mathbb{Z}^d$, with a few subtle changes. First, $K_{\mathbf{x}}$ does not sum to 1 over its domain and should be normalized first before being compared to a Gaussian.

Lemma A.7

$$\int_{H_d} K_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) d\mathbf{y} = (d+1)^{d-1/2}.$$

Proof

$$\begin{aligned}
&\int_{H_d} K_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) d\mathbf{y} \\
&= \int_{H_d} \sum_{\mathbf{p},\mathbf{q} \in A_d^*} B_{\mathbf{p}}(\mathbf{x})B_{\mathbf{q}}(\mathbf{y})b(\mathbf{p}-\mathbf{q}) d\mathbf{y} \\
&\quad \text{by Proposition 6.1,} \\
&= \sum_{\mathbf{p},\mathbf{q} \in A_d^*} B_{\mathbf{p}}(\mathbf{x}) \Bigg[\int_{H_d} B_{\mathbf{q}}(\mathbf{y}) d\mathbf{y} \Bigg] b(\mathbf{p}-\mathbf{q}) \\
&= \sum_{\mathbf{p},\mathbf{q} \in A_d^*} B_{\mathbf{p}}(\mathbf{x})(d+1)^{d-1/2} b(\mathbf{p}-\mathbf{q}) \\
&\quad \text{by Corollary 5.5,} \\
&= \sum_{\mathbf{p} \in A_d^*} B_{\mathbf{p}}(\mathbf{x})(d+1)^{d-1/2} \quad \text{since} \sum_{\mathbf{q} \in A_d^*} b(\mathbf{p}-\mathbf{q}) = 1, \\
&= (d+1)^{d-1/2}.
\end{aligned}$$

The last line follows as the sum of splatting weights for $\mathbf{x}$ over all lattice points is also 1. □

Since $K_{\mathbf{x}}$ is translation-invariant (for translations by a lattice point) and symmetric with respect to permutation of the axes, it suffices to iterate $\mathbf{x}$ over the canonical simplex C^*, in order to compute the expected value of the L_2 distance. Recall from Proposition 4.11 that the volume of C^* is given by $(d+1)^{d-1/2}/d!$. Therefore,

$$\eta = \frac{(d+1)^{d-1/2}/d! \int_{H_d} N(\mathbf{y}; 2(d+1)^2/3)^2 d\mathbf{y}}{\int_{C^*} \int_{H_d} [\frac{K_{\mathbf{x}}(\mathbf{y}-\mathbf{x})}{(d+1)^{d-1/2}} - N(\mathbf{y}-\mathbf{x}; 2(d+1)^2/3)]^2 d\mathbf{y} d\mathbf{x}}.$$

η can be written as $\frac{(\text{A.9})}{(\text{A.8})+(\text{A.9})-2(\text{A.10})}$, where the components are as follows:

$$\frac{d!}{(d+1)^{3d-3/2}} \int_{C^*} \int_{H_d} K_{\mathbf{x}}(\mathbf{y}-\mathbf{x})^2 d\mathbf{y} d\mathbf{x}, \tag{A.8}$$

$$\int_{H_d} N\big(\mathbf{y}; 2(d+1)^2/3\big)^2 d\mathbf{y}, \tag{A.9}$$

$$\frac{d!}{(d+1)^{2d-1}} \int_{C^*} \int_{H_d} K_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) \times N\big(\mathbf{y}-\mathbf{x}; 2(d+1)^2/3\big) d\mathbf{y} d\mathbf{x}. \tag{A.10}$$

By Lemma A.1, (A.9) equals $(\frac{8(d+1)^2\pi}{3})^{-d/2}$.

Let us compute (A.8) without the constant term. The integrand may be expanded as follows:

$$\begin{aligned}
&\int_{C^*} \int_{H_d} K_{\mathbf{x}}(\mathbf{y}-\mathbf{x})^2 d\mathbf{y} d\mathbf{x} \\
&= \int_{C^*} \int_{H_d} \Bigg[\sum_{\mathbf{p},\mathbf{q} \in A_d^*} B_{\mathbf{p}}(\mathbf{x}) B_{\mathbf{q}}(\mathbf{y}) b(\mathbf{p}-\mathbf{q}) \Bigg]^2 d\mathbf{y} d\mathbf{x} \\
&= \int_{C^*} \int_{H_d} \sum_{\substack{\mathbf{p_1},\mathbf{p_2},\\ \mathbf{q_1},\mathbf{q_2} \in A_d^*}} B_{\mathbf{p_1}}(\mathbf{x}) B_{\mathbf{p_2}}(\mathbf{x}) B_{\mathbf{q_1}}(\mathbf{y}) B_{\mathbf{q_2}}(\mathbf{y}) \\
&\quad \times b(\mathbf{p_1}-\mathbf{q_1}) b(\mathbf{p_2}-\mathbf{q_2}) d\mathbf{y} d\mathbf{x} \\
&= \sum_{\dots} b(\mathbf{q_1}) b(\mathbf{q_2}) \Bigg[\int_{C^*} B_{\mathbf{p_1}}(\mathbf{x}) B_{\mathbf{p_2}}(\mathbf{x}) d\mathbf{x} \Bigg] \\
&\quad \times \Bigg[\int_{H_d} B_{\mathbf{p_1}-\mathbf{q_1}}(\mathbf{y}) B_{\mathbf{p_2}-\mathbf{q_2}}(\mathbf{y}) d\mathbf{y} \Bigg] d\mathbf{y} d\mathbf{x},
\end{aligned}$$

via change of variable $\mathbf{q_i} \leftarrow \mathbf{p_i} - \mathbf{q_i}$.

Our expression is a weighted sum in which the weights are $b(\mathbf{q_1}) b(\mathbf{q_2})$. By Lemma A.5, it is equivalent to

$$\sum_{\substack{\mathbf{p_1} \in C^* \\ \mathbf{p_2} \in C^*}} \sum_{\substack{\mathbf{t} \in \{-1,0,1\}^{d+1} \\ \mathbf{s} \in \{-1,0,1\}^{d+1}}} [\cdots][\cdots] \prod_k \frac{1}{2^{1+|t_k|}} \frac{1}{2^{1+|s_k|}}, \tag{A.11}$$

where $\mathbf{q_1} = \sum t_k \cdot \mathbf{w_k}$ and $\mathbf{q_2} = \sum s_k \cdot \mathbf{w_k}$.

Note that the first bracketed term is effectively given by Lemma A.2. The second term is similar, but the integral is over H_d, so we must additively consider each simplex whose vertex contains both $\mathbf{p_1} - \mathbf{q_1}$ and $\mathbf{p_2} - \mathbf{q_2}$. The number of such simplices is given by Lemma A.3.

(A.11) is more attractive than its original form because its summands are now simple numerical quantities. There is no succinct analytic expression for the sum, but its numerical computation is straightforward. The number of summands is $(d+1)^2 \cdot 3^{2(d+1)}$.

That leaves us with (A.10). Once again, the integral is expanded and manipulated:

$$\begin{aligned}
&\int_{C^*} \int_{H_d} K_{\mathbf{x}}(\mathbf{y}-\mathbf{x}) N(\mathbf{y}-\mathbf{x}; 2(d+1)^2/3) d\mathbf{y} d\mathbf{x} \\
&= \sum_{\mathbf{p},\mathbf{q} \in A_d^*} \int_{C^*} \int_{H_d} B_{\mathbf{p}}(\mathbf{x}) B_{\mathbf{q}}(\mathbf{y}) \\
&\quad \times b(\mathbf{p}-\mathbf{q}) N(\mathbf{y}-\mathbf{x}; \dots) d\mathbf{y} d\mathbf{x}
\end{aligned}$$

Using Lemma A.5, we obtain

$$\sum_{\mathbf{p} \in C^*} \sum_{\mathbf{t} \in \{-1,0,1\}^{d+1}} \int_{C^*} \int_{H_d} B_{\mathbf{p}}(\mathbf{x}) B_{\mathbf{p}+\mathbf{q}}(\mathbf{y}) \times \prod_k \frac{1}{2^{1+|t_k|}} N(\mathbf{y}-\mathbf{x}; \dots) d\mathbf{y} d\mathbf{x},$$

where $\mathbf{q} = \sum t_k \cdot \mathbf{w_k}$. By symmetry, we can swap out C^* with the union of Delaunay cells containing the origin. Then we have an integral weighted by $B_{\mathbf{p}}(\mathbf{x})$ and $B_{\mathbf{p}+\mathbf{q}}(\mathbf{y})$, which can be computed by Monte-Carlo sampling the $(d+1)$-dimensional unit cube, by Lemma A.4.

In summary, we iterate over $\mathbf{t} \in \{-1,0,1\}^{d+1}$ and $\mathbf{p} \in C^*$, and for each pair of vectors, sample $\mathbf{x}$ and $\mathbf{y}$ via Lemma A.4, and evaluate the Gaussian of $\mathbf{y}-\mathbf{x}$, weighted by $\prod_k \frac{1}{2^{1+|t_k|}}$. Multiplying the result by an appropriate constant yields (A.11).

References

1. Adams, A., Gelfand, N., Dolson, J., Levoy, M.: Gaussian kd-trees for fast high-dimensional filtering. In: ACM Transactions on Graphics (Proc SIGGRAPH '09), pp. 1–12 (2009)
2. Adams, A., Baek, J., Davis, A.: High-dimensional filtering with the permutohedral lattice. In: Proceedings of EUROGRAPHICS, pp. 753–762 (2010)
3. Arbelaez, P., Maire, M., Fowlkes, C.C., Malik, J.: From contours to regions: an empirical evaluation. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 2294–2301 (2009)
4. Aurich, V., Weule, J.: Non-linear Gaussian filters performing edge preserving diffusion. In: Mustererkennung 1995, 17. DAGM-Symposium, pp. 538–545 (1995)
5. Bambah, R.P., Sloane, N.J.A.: On a problem of Ryskov concerning lattice coverings. Acta Arith. **42**, 107–109 (1982)
6. de Boor, C.: B-form basics. In: Geometric Modeling, pp. 131–148 (1987)
7. de Boor, C., Höllig, J., Riemenschneider, S.: Box splines, vol. 98. Springer, Berlin (1993)
8. Buades, A., Coll, B., Morel, J.M.: A non-local algorithm for image denoising. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 60–65 (2005)
9. Chen, J., Paris, S., Durand, F.: Real-time edge-aware image processing with the bilateral grid. In: ACM Transactions on Graphics (Proc SIGGRAPH '07), p. 103 (2007)
10. Cheng, Y.: Mean shift, mode seeking, and clustering. IEEE Trans. Pattern Anal. Mach. Intell. **17**(8), 790–799 (1995)
11. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. IEEE Trans. Pattern Anal. Mach. Intell. **24**(5), 603–619 (2002)
12. Conway, J.H., Sloane, N.J.A.: Sphere Packings, Lattices and Groups. Springer, Berlin (1998)
13. Delaunay, B.N., Ryskov, S.S.: Solution of the problem of least dense lattice covering of a four-dimensional space by equal spheres. Sov. Math. Dokl. **4**, 1014–1016 (1963)
14. Dolbilin, N.P.: Minkowski's theorems on parallelohedra and their generalizations. Russ. Math. Surv. **62**, 793–795 (2007)
15. Dolson, J., Baek, J., Plagemann, C., Thrun, S.: Upsampling range data in dynamic environments. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1141–1148 (2010)

16. Durand, F., Dorsey, J.: Fast bilateral filtering for the display of high-dynamic-range images. In: Proc. SIGGRAPH '02, pp. 257–266 (2002)
17. Eisemann, E., Durand, F.: Flash photography enhancement via intrinsic relighting. In: ACM Transactions on Graphics (Proc. SIGGRAPH 04), pp. 673–678 (2004)
18. Entezari, A., Dyer, R., Möller, T.: Linear and cubic box splines for the body centered cubic lattice. In: IEEE Visualization, pp. 11–18 (2004)
19. Entezari, A., Ville, D.V.D., Möller, T.: Practical box splines for reconstruction on the body centered cubic lattice. IEEE Trans. Vis. Comput. Graph. **14**, 313–328 (2008)
20. Greengard, L.F., Strain, J.A.: The fast gauss transform. SIAM J. Sci. Stat. Comput. **12**, 79–94 (1991)
21. Grundmann, M., Kwatra, V., Han, M., Essa, I.: Efficient hierarchical graph-based video segmentation. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 2141–2148 (2010)
22. Kershner, R.: The number of circles covering a set. Am. J. Math. **61**, 665–671 (1939)
23. Kim, M.: Symmetric box-splines on root lattice. PhD thesis, University of Florida, Gainesville, FL (2008)
24. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. In: ACM Transactions on Graphics (Proc. SIGGRAPH 07), p. 96 (2007)
25. Kuhn, H.W.: Some combinatorial lemmas in topology. IBM J. Res. Dev. **45**, 518–524 (1960)
26. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: Proc. 8th Int. Conf. Computer Vision, vol. 2, pp. 416–423 (2001)
27. Mirzargar, M., Entezari, A.: Voronoi splines. IEEE Trans. Signal Process. **58**, 4572–4582 (2010)
28. Mood, R.V., Patera, J.: Voronoi and Delaunay cells of root lattices: classification of their faces and facets by Coxeter-Dynkin diagrams. J. Phys. A, Math. Gen. **25**, 5089–5134 (1992)
29. Paris, S.: Edge-preserving smoothing and mean-shift segmentation of video streams. In: Proc. European Conference on Computer Vision, pp. 460–473 (2008)
30. Paris, S., Durand, F.: A fast approximation of the bilateral filter using a signal processing approach. In: Proc. European Conference on Computer Vision, pp. 568–580 (2006)
31. Paris, S., Durand, F.: A topological approach to hierarchical segmentation using mean shift. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2007)
32. Paris, S., Durand, F.: A fast approximation of the bilateral filter using a signal processing approach. Int. J. Comput. Vis. **81**(1), 24–52 (2009)
33. Perlin, K.: Noise. In: ACM SIGGRAPH '02 Course Notes (2002)
34. Petersen, D.P., Middleton, D.: Sampling and reconstruction of wave-number-limited functions in N-dimensional Euclidean spaces. Inf. Control **5**, 279–323 (1962)
35. Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., Toyama, K.: Digital photography with flash and no-flash image pairs. In: ACM Transactions on Graphics (Proc. SIGGRAPH 04), pp. 664–672 (2004)
36. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. Adv. Neural Inf. Process. Syst. **20**, 1177–1184 (2008)
37. Rogers, C.A.: Packing and Covering. Cambridge University Press, Cambridge (1964)
38. Ryshkov, S.S.: The structure of primitive parallelohedra and Voronoi's last problem. Russ. Math. Surv. **53**, 403–405 (1998)
39. Ryskov, S.S., Baranovskii, E.P.: Solution of the problem of least dense lattice covering of five-dimensional space by equal spheres. Sov. Math. Dokl. **16**, 586–590 (1975)
40. Samelson, H.: Notes on Lie Algebras. Van Nostrand Reinhold Company, Princeton (1969)
41. Senechal, M.: Quasicrystals and Geometry. Cambridge University Press, Cambridge (1995)
42. Serre, J.P.: Complex Semisimple Lie Algebras. Springer, Berlin (1987)
43. Smith, S., Brady, J.M.: SUSAN: a new approach to low level image processing. Int. J. Comput. Vis. **23**, 45–78 (1997)
44. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: Proc. International Conference on Computer Vision, pp. 836–846 (1998)
45. Voronoi, G.F.: Studies of Primitive Parallelotopes, vol. 2, pp. 239–368. Ukrainian Academy of Sciences Press, Kiev (1952)
46. Wang, P., Lee, D., Gray, A., Rehg, J.M.: Fast mean shift with accurate and stable convergence. In: Melia, M., Shen, X. (eds.) Artificial Intelligence and Statistics (2007)
47. Weiss, B.: Fast median and bilateral filtering. In: ACM Transactions on Graphics (Proc. SIGGRAPH '06), pp. 519–526 (2006)
48. Winnemöller, H., Olsen, S.C., Gooch, B.: Real-time video abstraction. ACM Trans. Graph. **25**(3), 1221–1226 (2006)
49. Witsenhausen, H.S.: Spiegelungsgruppen und aufzählung halbeinfacher liescher Ringe. Abh. Math. Sem. Univ. Hamburg **14** (1941)
50. Yang, C., Duraiswami, R., Gumerov, N.A., Davis, L.: Improved fast gauss transform and efficient kernel density estimation. In: Proc. International Conference on Computer Vision, vol. 1, pp. 664–671 (2003)
51. Yang, Q., Yang, R., Davis, J., Nistér, D.: Spatial-depth super resolution for range images. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2007)
52. Yaroslavsky, L.P.: Digital Picture Processing. An Introduction. Springer, Berlin (1985)

Jongmin Baek received a B.S. in Mathematics and an M.Eng. in Electric Engineering and Computer Science from MIT in 2008, and is currently a doctoral student at Stanford University under Marc Levoy. He has previously worked on edge-aware image processing and computational cameras, and his current research interest includes real-time computational photography on mobile devices.

Andrew Adams is a postdoctoral associate at MIT. He received his PhD from Stanford University in 2011, where he worked on programmable cameras (the "Frankencamera"), and on accelerating common image processing algorithms used in computational photography. He is now interested in languages and compilers for fast image processing.

Jennifer Dolson is a PhD student in the Computer Science Department at Stanford University, co-advised by Marc Levoy and Sebastian Thrun. Her research interests include image/signal processing, sensors, computer vision, data mining, and machine learning. She received a M.S. Computer Science, Stanford in May 2011 and graduated with a B.S. with high distinction, Computer Science from the University of Virginia in 2008.