Tensor Decompositions for Integral Histogram Compression and Look-Up

Rafael Ballester-Ripoll, *Student Member, IEEE,* and Renato Pajarola, *Member, IEEE*

Abstract—Histograms are a fundamental tool for multidimensional data analysis and processing, and many applications in graphics and visualization rely on computing histograms over large regions of interest (ROI). Integral histograms (IH) greatly accelerate the calculation in the case of rectangular regions, but come at a large extra storage cost. Based on the tensor train decomposition model, we propose a new compression and approximate retrieval algorithm to reduce the overall IH memory usage by several orders of magnitude at a user-defined accuracy. To this end we propose an incremental tensor decomposition algorithm that allows us to compress integral histograms of hundreds of gigabytes. We then encode the borders of any desired rectangular ROI in the IH tensor-compressed domain and reconstruct the target histogram at a high speed which is independent of the region size. We furthermore generalize the algorithm to support regions of arbitrary shape rather than only rectangles, as well as histogram field computation, i.e. recovering many histograms at once. We test our method with several multidimensional data sets and demonstrate that it radically speeds up costly histogram queries while avoiding storing massive, uncompressed IHs.

Index Terms—integral histograms, tensor decomposition, multidimensional compression

1 Introduction

Histograms are widely used in many visualization applications including segmentation, filtering, object tracking and classification, volume rendering, and more. Recent advances in data acquisition technology have given rise to increasingly large multidimensional data sets that often require histogram queries over large data regions. Developing algorithms for compact representation and fast histogram computation is thus an area of active research. The integral histogram [1], that we denote IH, is a very time-efficient way to obtain a histogram over any rectangular axis-aligned region in a Cartesian space. It extends the concept of summed area table (SAT) [2] by storing a cumulative histogram at each table entry. In exchange for its large query speed-up, this data structure is highly redundant: it entails a manifold increase of the memory footprint and may often exceed available memory resources. For instance, in the present paper we consider a 1GB micro-computer tomography (μCT) with 128 histogram bins. The resulting 512GB integral histogram does not fit into memory in current desktop environments. Furthermore, IH queries require hundreds of non-contiguous accesses. This makes compressing histograms in their integral form an important target. The goal is to reduce storage needs to a manageable amount while still allowing a faster histogram calculation than a brute-force traversal of the original data.

Our approach is based on lossy compression, and we show that excellent IH data reduction rates can be achieved when approximated results are tolerated. Tensor decompositions are a very convenient tool for this. Such decompositions generalize the 2D singular value decomposition (SVD) to more dimensions and have been shown to successfully tackle the so-called *curse of dimensionality*. We demonstrate that in particular the tensor train model (TT) can effectively compress large IHs and bring together two key advantages for the query stage, namely high compression rates and fast histogram look-up. The proposed method is flexible in the sense that: a) it allows decreasing both size and query time as desired, at the expense of a variable loss in response accuracy due to compression; and b) it can handle arbitrary query regions, as opposed to other methods that are often limited to axis-aligned query rectangles. Furthermore, while directly computing a histogram over a region of interest (ROI) requires many non-sequential memory accesses, in tensor reconstruction the compressed representation is always traversed in the same dimensional order, which allows a more memory-optimized computation.

Our system is highly asymmetric: we emphasize high data reduction rates during offline decomposition, making it very computationally intensive, in order to achieve as fast as possible online reconstruction in exchange. This paradigm is rather standard in many interactive visualization and analysis applications [3], [4], [5].

Contribution

The tensor framework has been used previously in a range of graphics and visual computing applications (see also Sec. 2). In this paper we contribute the first tensor-based compression and querying algorithm for integral histograms: We incorporate the classical SAT reconstruction formula into the compressed domain and then extend it to non-rectangular regions by expressing them also in a tensor-decomposed format. More specifically:

- We propose a TT-compressed IH as a data structure for histogram look-up over medium-to-large regions, and contribute an incremental compression algorithm that never operates on the full (and often intractable) IH.
- We exploit tensor-domain multilinearity and show how to efficiently query the proposed data structure to retrieve one or many histograms over non-rectangular regions.

Our method is able to compress IHs by several orders of magnitude and approximately reconstruct histograms at interac-

• *Rafael Ballester-Ripoll (rballester@ifi.uzh.ch) and Renato Pajarola (pajarola@ifi.uzh.ch) are with the Visualization and MultiMedia Lab, Department of Informatics, University of Zürich, Switzerland.*

tive rates, even over huge regions which can be rectangular or otherwise. It can also efficiently decompress histogram fields, a useful concept in for example feature extraction and visualization that we demonstrate later in the form of an entropy field.

Fig. 1 summarizes the main steps involved in our tensor integral histogram decomposition and look-up framework.

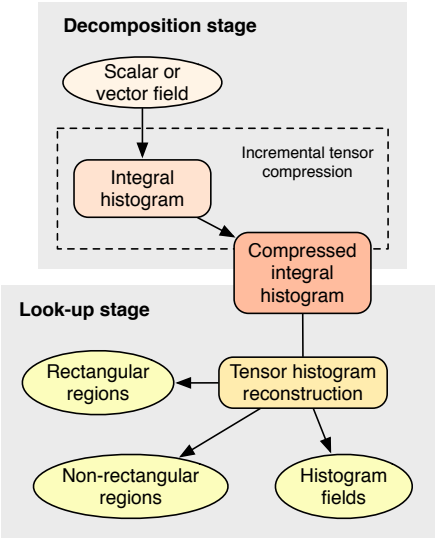


Fig. 1: Overview of the proposed tensor-based histogram decomposition and reconstruction.

The rest of this paper is organized as follows. Sec. 2 reviews the existing literature on histogram look-up algorithms, applications and compression techniques as well as tensor approximation, which is covered in more detail in Sec. 3. Sec. 4 is concerned with incremental compression of (potentially large) IHs. In Sec. 5 and 6 we show how to encode a target region into the SAT compressed domain, so that when reconstructing it we obtain the integral over that query region. Experimental evaluation is conducted in Sec. 7 and discussed in Sec. 8, and concluding remarks are given in Sec. 9.

2 Related Work

Next we overview some general techniques and motivation on computing histograms of multidimensional data, both for the exact and the approximate cases. We continue with summed area tables and their adaptation into integral histograms for the problem at hand, and finally put in context our chosen mathematical tool, namely tensor decompositions.

2.1 Efficient Multidimensional Histograms

When processing large regions of interest within multidimensional data sets, computational costs are commonly a concern in real-time visualization applications. One approach for efficient histogram generation is to avoid redundant computations, e.g. with incremental sliding windows [6], [7] or identifying overlapping regions [8]. Alternatively, other methods compute the desired histogram features directly from the data [9], [10]. In [10] and [11], pixel or voxel neighborhood information is compactly represented as a sparse sum of Gaussian probability density functions, which allows for convenient retrieval of features used in filtering and for applying a transfer function in volume rendering. On the other hand, some tools compute histograms of a transformation of the data; prominent examples are the histogram of gradients (HOG, a method to extract robust feature descriptors) and its variants [12], and the 2D histogram, which takes into account gradient information (useful for e.g. segmentation and transfer function selection).

2.2 Summed Area Tables

The *summed area table* (SAT) is a data structure for fast integral look-up over rectangular regions [2]. In 1D it is just a discretization of the Fundamental Theorem of Calculus: by storing the cumulative sum of an array $F(n) := \sum_0^{n-1} f(i)$ (with $F(0) := 0$), one can compute an arbitrary summation $\sum_a^b f(i)$ over an interval $[a, b]$ in constant time as the difference of F at the interval borders, $F(b+1) - F(a)$. For higher dimensions $N \geq 2$ the integral is given by the sum of values (with alternating signs) from the 2^N corners of the box region. SATs have been proven useful in computer graphics, especially in volume rendering, and parallel integration and filtering algorithms have been developed on GPUs for fast SAT generation [13], [14], [15].

2.3 Integral Histograms

The *integral histogram* (IH) emerged as a natural extension of a SAT for histogram look-up by adding a bin dimension [1], and SAT querying algorithms can be in principle adapted to query it. The idea behind an IH works as follows: Each bin $b = 1, ..., B$ is associated to a binary mask: an array of the same size as the input, in which each entry is set to 1 if and only if the corresponding input pixel has the value b, and 0 otherwise. Then, the SAT of each of these masks is computed; i.e. its cumulative sum along all dimensions. All resulting SATs are stacked along a new dimension of size B to yield the final IH. To retrieve a histogram from a certain target rectangle, a SAT query is performed for each bin over that region. Fig 2 illustrates the IH look-up for the 2D (image) case.

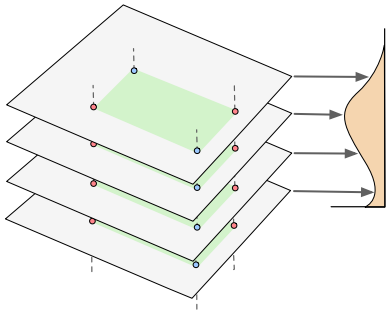


Fig. 2: Querying a 2D integral histogram over a rectangular region (highlighted in green) requires 4 look-ups per bin.

The IH data structure generalizes easily to high dimensions [16], and requires $B \cdot 2^N$ value look-ups to compute the final result. More importantly, a slice of the size of the original data must be stored for every value bin, which can render the uncompressed IH too large to be used interactively. Compression approaches have been explored, e.g. rectangle area sums can be computed efficiently over wavelet-decomposed data [17]. Lee and Shen proposed WaveletSAT [5], a lossless wavelet-based compression scheme for integral histograms. They report fast query

times and compression rates of 1:8 and above compared to the original IH size, for data up to 256^3 voxels. However, the method is still limited to axis-aligned rectangular regions, and dealing with other types of regions would require approximating them as sums of rectangles. An improvement in this direction was proposed by Heckbert [18]. Based on repeated SAT integrations, the author devised a procedure to obtain integrals across regions which are defined by polynomials and not just rectangles. Non-rectangular regions are often preferred indeed, e.g. Gaussian (to achieve rotation-invariance) or 3D cross-neighborhoods [19], or complex segmented target regions. In this context, our tensor-compressed histogram querying algorithm emphasizes fast approximate reconstruction, low storage needs and flexibility regarding target region shape.

2.4 Tensor Methods in Graphics and Visualization

Tensor decompositions express multidimensional data as sums of separable components [20]. There are several ways in which these components can interact with each other, giving rise to different *tensor models*. Pioneering decomposition models such as CANDECOMP/PARAFAC [21], Tucker [22] or the higher-order singular value decomposition (HOSVD) [23] were originally developed as compact multiway dimensionality reduction techniques. These concepts later spawned many applications in signal processing, data mining, computer vision, and more, and a variety of additional decompositions have been proposed. More recently, tensor methods have become a versatile tool, increasingly used in computer graphics and visualization [24], [25]. Data reduction is often achieved by *truncating* higher *tensor ranks*, i.e. discarding separable components that contribute little (in the L^2 sense) to the input tensor; this is also known as *low-rank reduction*. Wu et al. [26], [27] developed a hierarchical partitioning for visual data compression, outperforming the wavelet transform (WT) in terms of feature preservation. Suter et al. [4], [28] proposed fast GPU decompression methods for parallel, real-time large volume visualization. Wetzstein et al. [29] introduced tensor light field displays. Ruiters et al. [30], [31] and Ballester-Ripoll et al. [32], [33] decompose and filter bidirectional texture functions and large multiresolution volumes, respectively, while Costantini et al. [34] used HOSVD for fast video texture synthesis. A compilation of tensor applications in these and other areas can be found in [20]. Overall, tensor representations are compact, lossy in most applications, and offer multiple ways to manipulate data in its compressed format.

3 Tensor Decomposition

3.1 Notation

Throughout this paper, we understand vectors as finite 1D arrays of scalars that are represented with bold lowercase letters (e.g. $\boldsymbol{a}$). Matrices use bold uppercase (e.g. $\mathbf{U}$), and tensors (i.e. higher-order multiarrays) use calligraphic letters (e.g. $\mathcal{T}$). N is the number of dimensions, also called tensor modes, and the symbols $I_1, ..., I_N$ denote the resolution of the input data. If a distinction between dimensions is not necessary because we assume they have the same magnitude, we just use I for simplicity. We use zero-based colon notation to index individual elements, or subspaces of multiarrays. For example, $\mathbf{U}[:, i{:}j]$ are the columns i (included) to j (not included) of a matrix, and $\mathcal{T}[:, 0, :]$ denotes the first slice of a 3D tensor $\mathcal{T}$ along its 2nd dimension. B is the number of histogram bins. We use the relative error ϵ for comparing tensors: if $\mathcal{T}$ is a tensor and $\widetilde{\mathcal{T}}$ its approximation, then $\epsilon = \|\mathcal{T} - \widetilde{\mathcal{T}}\| / \|\mathcal{T}\|$ where $\|\mathcal{T}\| = \sqrt{\sum_{\boldsymbol{i}} \mathcal{T}(\boldsymbol{i})^2}$ is the Frobenius norm. This is the prevalent metric in the tensor literature [20], [23], [35], and it has a one-to-one correspondence to peak signal-to-noise ratio (PSNR) as follows:

$$\mathrm{PSNR} = 20 \cdot \log_{10} \left(\frac{M}{\epsilon \cdot \|\mathcal{T}\| \cdot \sqrt{\mathrm{size}(\mathcal{T})}} \right),$$

where M is the maximum signal intensity. We also use ϵ for all histogram query results.

3.2 The Tensor Train Model

The *tensor train* (TT) was introduced by Oseledets [35] as an alternative compact representation to earlier models, including CP and Tucker. It is particularly well suited for applications with high dimensionality, as its number of coefficients increases linearly (instead of exponentially) with respect to the dimensions N. Its coefficients are arranged in a sequence (hence the name train) of 3D tensor cores. Each core has size $R_{n-1} \times I_n \times R_n$ with $R_0 = R_N = 1$, i.e. the first and last ones have size $1 \times I_1 \times R_1$ and $R_{N-1} \times I_N \times 1$ respectively, see also a depiction in Fig. 3. We call the integers $R_1, \ldots, R_{N-1}$ *tensor ranks*, and sometimes write R to denote the highest rank for convenience. We often write the decomposition in terms of its N cores: $\mathcal{T} = [[\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}]]$.

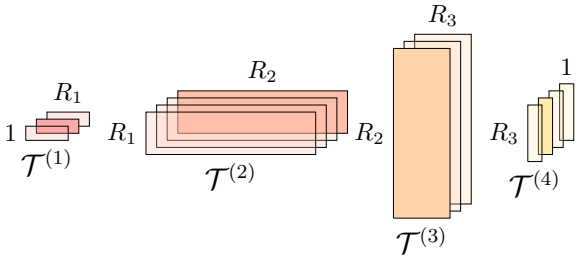


Fig. 3: A schematic illustration of a 4D tensor of size $3 \times 4 \times 3 \times 4$ decomposed in the TT format using ranks (R_1, R_2, R_3). We highlight the subspaces that are needed for reconstructing a single element $(1, 3, 0, 1)$: row $\mathcal{T}^{(1)}[0, 1, :]$ from the first core, slices $\mathcal{T}^{(2)}[:, 3, :]$ and $\mathcal{T}^{(3)}[:, 0, :]$ from the second and third cores, and column $\mathcal{T}^{(4)}[:, 1, 0]$ from the last core.

Element-wise reconstruction from the TT format corresponds to a sequence of matrix products:

$$\mathcal{T}[i_1, i_2 ..., i_N] = \mathcal{T}^{(1)}[0, i_1, :] \cdot \mathcal{T}^{(2)}[:, i_2, :] \cdots \mathcal{T}^{(N)}[:, i_N, 0],$$

where the first and last matrices consist of just one row and column, respectively.

An attractive feature of the TT model is the speed of its single element reconstruction: only certain slices of the compressed data participate in the calculation, which is a sequence of matrix-vector products requiring $O(NR^2)$ operations. This proves very useful for histogram reconstruction, as we demonstrate later in Sec. 5 and 6.

The main advantage of this tensor formulation with respect to other data reduction schemes is the ease of linear manipulations: convolution, differentiation, integration and others can be performed efficiently using only a subset of the compressed-format elements. Furthermore, editing a tensor in the TT format results in another TT tensor that can then be decompressed the same way

as the original. Such operations are much more complicated in other compression methods; for example brick-based approaches (e.g. vector quantization) are not well-suited for linear operations unless rather large bricks are used. These and similar limitations are especially prominent in non-transform based compression strategies.

4 Integral Histogram Tensor Compression

Let $\mathcal{T}$ be a multidimensional array of size $I_1 \times \cdots \times I_N$, for example an image ($N = 2$) or a volume ($N = 3$). The full IH $\mathcal{I}$ of $\mathcal{T}$ can be computed from its level-set stack $\mathcal{L}$, which has size $I_1 \times ... \times I_N \times B$ and is defined as:

$$\mathcal{L}[x_1, ..., x_N, b] = \begin{cases} 0 & \text{if } \mathcal{T}[x_1, ..., x_N] \neq b \\ 1 & \text{if } \mathcal{T}[x_1, ..., x_N] = b \end{cases}$$

To eventually get $\mathcal{I}$, one needs to calculate all *cumulative* partial sums of $\mathcal{L}$'s slices along the last dimension. The result has size $(I_1 + 1) \times \cdots \times (I_N + 1) \times B$. Element-wise, each $\mathcal{I}[x_1, ..., x_N, b]$ is defined as:

$$\begin{cases} 0 & \text{if } i_n = 0 \text{ for some } n \\ \sum_{i_1,...,i_N=1}^{x_1,...,x_N} \mathcal{L}[i_1 - 1, ..., i_N - 1, b] & \text{otherwise} \end{cases}$$

Note that whereas $\mathcal{L}$ is highly sparse, the cumulative $\mathcal{I}$ is not. While one can already use $\mathcal{L}$ to compute histograms by direct integration, we choose to compress the array $\mathcal{I}$ instead. The motivation is that we found that histograms reconstructed from a tensor-compressed $\mathcal{I}$ are much more accurate for the same number of coefficients. The columns $\mathcal{L}[x_1, ..., x_N, :]$ have little correlation between each other: any value change between two points (e.g. $\mathcal{T}[x_1, ..., x_N] \neq \mathcal{T}[y_1, ..., y_N]$) implies that $\langle \mathcal{L}[x_1, ..., x_N, :], \mathcal{L}[y_1, ..., y_N, :] \rangle = 0$. On the other hand, every column $\mathcal{I}[x_1, ..., x_N, :]$ depends on the whole region between it and the origin $\mathcal{I}[0, \ldots, 0, :]$; thus, columns close to each other have very similar contents. To support this observation, we show in Fig. 4 the histogram reconstruction accuracy from compressing $\mathcal{L}$ versus compressing $\mathcal{I}$.

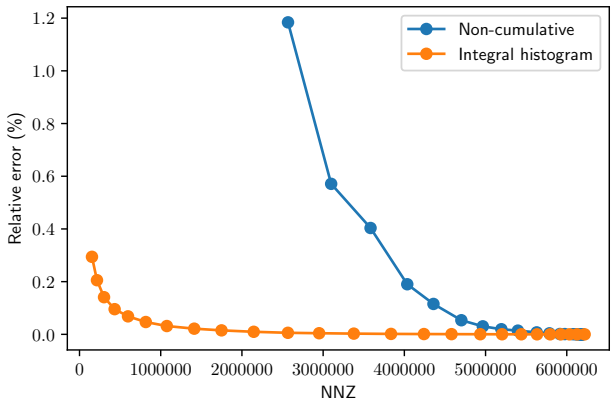


Fig. 4: Average histogram accuracy for rectangular ROIs of size 32^3 within a brick of size 64^3 (Bonsai data set). The target error during compression ϵ is decreased as we move left to right; each result dot is the average reconstruction error of 100 regions placed at random. Using the IH $\mathcal{I}$ instead of the level-set $\mathcal{L}$ leads to a much smoother degradation as well as faster histogram reconstruction over rectangular regions; see also Sec. 5.

Now, the question of how to efficiently compress $\mathcal{I}$ arises. The most straightforward approach would be to build the full $\mathcal{I}$ explicitly, and then compress it using TT-EIG. However, note that for medium or large models this quickly becomes impractical. Fortunately, thanks to the versatility of the TT format, we are able to split up any tensor and incrementally build up its compression. We namely combine two steps:

- An algorithm to compress any *slice* of the IH, i.e. on a bin-by-bin basis;
- A *sum-and-compress* procedure to combine and merge existing compressed slices into one single compressed tensor.

We detail next how we address each of these separate steps.

4.1 Slice Compression

Any full dense tensor can be decomposed in the TT format via the TT-SVD algorithm [35], which successively unfolds the data and computes the singular value decomposition (SVD) on the result, one mode at a time. The desired target error $0 \leq \epsilon$ is defined beforehand and is the only parameter. TT-SVD guarantees that the final relative error will be no larger than the target ϵ.

Let $\mathcal{S}_b$ be the b-th bin slice of the IH, i.e. $\mathcal{I}[:, \ldots, :, b]$. We compute it independently of all other bin slices: first we create a binary mask from the input data, and then we compute its SAT. We now apply the TT-SVD algorithm to compress $\mathcal{S}_b$ into a TT tensor with N cores at a prescribed accuracy ϵ. Note that this ϵ affects only the global quality of the IH and that relative errors of individually reconstructed histograms (Secs. 5 and 6) will differ, especially with respect to the region size.

The original TT-SVD uses a sequence of tensor unfoldings, interleaved with SVD decompositions that progressively remove one dimension at a time to produce one core and thus reduce the overall tensor size [35]. Such unfoldings flatten the tensor into a matrix in various orderings so as to remove redundancy along one dimension at a time. We observed that in our applications these unfoldings are often very tall or wide matrices. Therefore, we choose to obtain the necessary singular vectors via the eigenvalue decomposition of covariance matrices. We call this modified routine TT-EIG, and found it to be significantly faster than standard SVD. Although these two approaches have a different numerical behavior and their resulting singular vectors and values may differ, these differences are negligible for the compression rates and real-world multidimensional signals that we tested. The complete procedure is given in the Appendix.

4.2 Sum-and-Compress

As argued before, IHs are often too large to be manipulated as a whole. Given a multidimensional data set $\mathcal{T}$ and a number of bins B, we propose an incremental algorithm that progressively adds one slice at a time. We start with $b = 0$, and at each slice $0 \leq b < B$ we take a compressed IH of bins $[0, b-1]$ and produce an IH for bins $[0, b]$:

1) We compute the b-th slice of the IH (of size $I_1 \times \cdots \times I_N$) and compress it into a TT tensor $\mathcal{A}$ as outlined above in Sec. 4.1.
2) We add an *indexing core* at the end with B elements that encodes the value b using *one-hot encoding*, i.e. all its elements are 0, except the b-th one which is 1. As

a result, the modified TT $\mathcal{A}$ encodes now a tensor of $N+1$ dimensions with size $I_1 \times \dots I_N \times B$. It has zeros everywhere outside its b-th slice.

3) We add this TT to our partial IH. Adding two tensors is a straightforward operation [35]. The result encodes now an IH for all bins $[0, b]$.
4) We recompress the result to remove any redundant information that may have appeared. We achieve this by means of the *TT-round* procedure, which is related to TT-SVD, its full details are given in [35].

Our incremental algorithm Alg. 1 given below exploits the fact that each compressed slice is small enough to be handled separately. Although the ranks are reduced after each recompression, the overall IH compressed size tends to grow as we work our way towards the last bin. Steps 3 and 4 are computationally intensive, especially when the accumulated IH is large. In order to reduce this cost we merge the IH slices in a recursive, binary-tree fashion: slice 0 is merged with slice 1, the result is merged with the combination of 2 with 3, and so on. Each tree root $\mathcal{R}_c$ encodes the IH for 2^c consecutive slices, and we create $\mathcal{R}_{c+1}$ by joining it with the next 2^c slices. This strategy ensures that, asymptotically, most merge operations involve only small tensors.

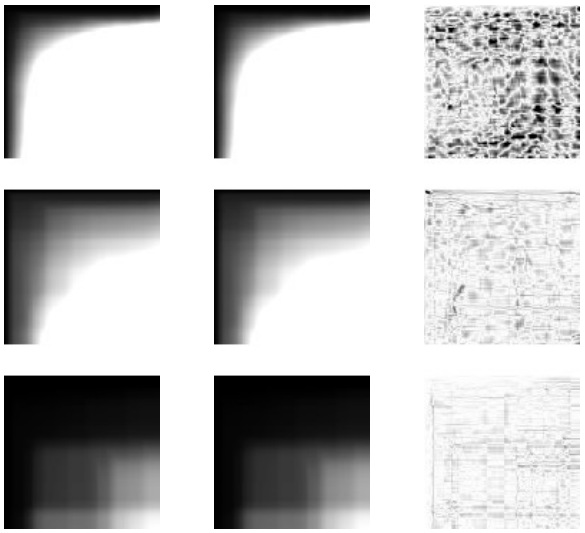
Fig. 5 shows the accuracy of the proposed IH compression over a 4096^2 grayscale image.

Algorithm 1 Build an integral histogram $\mathcal{I}$ from an input tensor $\mathcal{T}$ with B bins and error parameter ϵ, using sum-and-compress combined with TT-EIG (see Appendix).

procedure TT-IH($\mathcal{T}, B, \epsilon$)
  for $b = 0, \dots B-1$ **do**
    // Compute and compress the b-th slice of the IH of $\mathcal{T}$
    $\mathcal{A} :=$ TT-EIG($\mathcal{I}[:, \dots, :, b], \epsilon$)
    $\mathcal{B} :=$ ones$(1 \times B \times 1)$ *// This core indexes the bin*
    $\mathcal{B}[0, b, 0] := 1$ *// One-hot encoding*
    $\mathcal{A} := [[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(N)}, \mathcal{B}]]$ *// Append the indexing core*
    $c := 0$
    while $\mathcal{R}_c$ exists **do**
      $\mathcal{A} :=$ TT-ROUND$(\mathcal{A} + \mathcal{R}_c, \epsilon)$
      Delete $\mathcal{R}_c$
      $c := c + 1$
    end while
    $\mathcal{R}_c := \mathcal{A}$
  end for
  $\mathcal{I} =$ zeros$(I_1 \times \dots \times I_N \times B)$
  for all $\mathcal{R}_c$ **do** *// Gather and sum all remaining roots*
    $\mathcal{I} := \mathcal{I} + \mathcal{R}_c$
    Delete $\mathcal{R}_c$
  end for
  return $\mathcal{I}$
end procedure

5 Histogram Reconstruction

All tensor decompositions are multilinear in nature. This is a key feature that we exploit in this section and makes several compressed-domain operations very efficient.



(a) Original (b) After compression (c) Error (x100 amplified)

Fig. 5: Integral histograms are well compressible using the TT decomposition model. Column (a): three evenly-spaced IH bins for the Waterfall image (Fig. 6a) for $B = 64$. Column (b): IH bins after TT compression to $(1, 64, 51, 1)$ ranks. Column (c): absolute difference, magnified 100-fold to ease visual appreciation.

5.1 Spatial Tensor Basis Manipulation

Linear operations can be applied on a compressed IH tensor $\mathcal{I} = [[\mathcal{I}^{(1)}, \dots, \mathcal{I}^{(N)}, \mathcal{B}]]$ by slice-wise manipulating its TT cores. If instead of a core $\mathcal{I}^{(n)}$ we use a weighted sum of its slices $\mathbf{I}^{(n)} := \sum_{i=0}^{I_n} \boldsymbol{c}[i] \cdot \mathbf{I}^{(n)}[:, i, :]$ (where $\boldsymbol{c}$ is a vector with $I_n + 1$ entries), then the reconstruction produces a linear combination of the corresponding hyperslices along the n-th mode. In tensor notation, this means that

$$[[\mathcal{I}^{(1)}, \dots, \mathcal{I}^{(n)} \times_2 \boldsymbol{c}, \dots, \mathcal{I}^{(N)}, \mathcal{B}]] = \mathcal{I} \times_n \boldsymbol{c}, \qquad (1)$$

where we have used tensor-times-vector (TTV) products. An n-mode TTV computes the dot product between all tensor vectors along its n-th dimension and an external vector.

Eq. 1 allows us to reconstruct histograms from a compressed IH over various ROI types: next we cover the case of rectangular regions, and later in Sec. 6 we extend the method to non-rectangular query ROI shapes.

5.2 Querying a TT-Compressed IH

Let $\mathcal{T}$ be a tensor data set of size $I_1 \times \dots \times I_N$, and ω a target region of interest (ROI), defined by the following indicator function,

$$\omega[x_1, \dots, x_N] = \begin{cases} 1 & \text{if } x_1 \in [i_1, j_1), \dots, x_N \in [i_N, j_N) \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathcal{S}$ be the summed area table of $\mathcal{T}$, and 0 and 1 denote either i or j for the 2^N vertices of ω. Thus, $\mathcal{S}[0, ..., 0]$ corresponds to $\mathcal{S}[i_1, ..., i_N]$, $\mathcal{S}[1, 0, ..., 0]$ to $\mathcal{S}[j_1, i_2, ..., i_N]$, and so on. The summation of $\mathcal{T}$ over ω can be obtained from $\mathcal{S}$ by adding and subtracting the following 2^N terms [16]:

$$\sum_{x_1=i_1}^{j_1-1} \cdots \sum_{x_N=i_N}^{j_N-1} \mathcal{T}[\boldsymbol{x}] = \sum_{p \in \{0,1\}^N} (-1)^{N-\|p\|_1} \cdot \mathcal{S}[p] \qquad (2)$$

The norm $\|\cdot\|_1$ counts the number of elements of value 1 in $[p]$ and determines the parity of each summand. For example, the SAT result in the 2D case is $\mathcal{S}[j_1, j_2] - \mathcal{S}[j_1, i_2] - \mathcal{S}[i_1, j_2] + \mathcal{S}[i_1, i_2]$. For N dimensions this requires 2^N look-ups in the multiarray $\mathcal{S}$, with 2^{N-1} positive terms and 2^{N-1} negative terms.

Thanks to multilinearity, tensor decompositions allow conversion of multidimensional operations into a sequence of equivalent 1D operations. In particular we can easily translate Eq. 2 into the TT compressed domain: it suffices to subtract the core slices that delimit the rectangle borders. Thus, we define

$$\mathbf{I}^{(n)} := \mathcal{I}^{(n)}[:, j_n, :] - \mathcal{I}^{(n)}[:, i_n, :]$$

for $n = 1, \ldots, N$; each is a matrix of shape $R_{n-1} \times R_n$. The last core $\mathcal{B}$ is unchanged: it is not a spatial axis, since it encodes the histogram bins. The desired histogram is then obtained as

$$\mathbf{I}^{(1)} \cdots \mathbf{I}^{(N)} \cdot \mathcal{B}[:, :, 0]$$

See the Appendix for an example illustration of these reconstruction formulas in the 2D case.

6 Non-rectangular ROIs

So far we have outlined how to integrate histograms only over rectangular regions. However, more general ROIs are supported as well, i.e. where the membership of every point to the query region is weighted by a real value in $[0, 1]$. Such cases are interesting e.g. when rotation-invariant descriptors are desired (i.e. where the region's membership function is radial) or when the membership is only estimated with a certain probability below 1 (e.g. a confidence heatmap arising from a segmentation procedure). Note that an IH in its original form is only optimal for rectangular regions and is not easily applicable in these more general cases.

In this section we make a distinction between reconstructing single histograms (as before) and the more general case in which blocks of histograms are desired.

6.1 Non-rectangular Reconstruction

In order to reconstruct one histogram from $\mathcal{I}$ over non-rectangular region we use the following identity:

$$\int_0^K f(x) \cdot g(x)\, dx = -\int_0^K \left(\int_0^x f(y)\, dy \right) \cdot g'(x)\, dx$$

which holds if either $f(x)$ or $g(x)$ are 0 at both $x = 0$ and $x = K$. In its discrete form, it means that to find the sum of a weighted array we can just compute the sum of the cumulative array (the IH in our case), weighted by the derivative of the weights. Each IH slice is represented by f, while g plays the role of our ROI's indicator function. In practice, we need the TT decomposition of our target ROI. Let $\mathcal{R} = [[\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(N)}]]$ be an N-dimensional TT tensor encoding the ROI with ranks $S_1, \ldots, S_{N-1}$, and $\Delta\mathcal{R}$ its derivative along all dimensions. Each bin b of the desired histogram follows from a dot product between $\Delta\mathcal{R}$ and a window on the corresponding slice of $\mathcal{I}$:

$$\langle \mathcal{I}[i_1{:}j_1, \ldots, i_N{:}j_N, b], \Delta\mathcal{R} \rangle \tag{3}$$

We compute the tensor dot product from Eq. 3 by successively fusing the cores of $\Delta\mathcal{R}$ and $\mathcal{I}^{(n)}$ via tensor contractions (for more details on dot products in the TT format, see [35]). The last core simply indexes the bins and does not vary; in particular we compute the dot product for all bins at once.

Algorithm 2 Given an IH $\mathcal{I}$ compressed with ranks $R_1, \ldots, R_N$, reconstruct a histogram over a non-rectangular region of bounding box $[i_1, j_1] \times \cdots \times [i_N, j_N]$ whose indicator function is approximated by a TT $\mathcal{R} = [[\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(N)}]]$ with ranks $S_1, \ldots, S_{N-1}$

procedure RECONSTRUCT($\mathcal{I}, \mathcal{R}$)
  // Compute $\Delta\mathcal{R}$ from $\mathcal{R}$ by deriving along all axes
  for $n = 1, \ldots, N$ **do**
    $\Delta\mathcal{R}^{(n)}[:, 0, :] = \mathcal{R}^{(n)}[:, 0, :]$
    for $i = 1, \ldots, I_n - 1$ **do**
      $\Delta\mathcal{R}^{(n)}[:, i, :] := \mathcal{R}^{(n)}[:, i, :] - \mathcal{R}^{(n)}[:, i-1, :]$
    end for
  end for
  $\mathcal{F} := \big(1\big)$ *// 1×1 tensor*
  for $n = 1, \ldots, N$ **do**
    $\mathcal{F} := \text{contraction}(\mathcal{F}, \mathcal{I}^{(n)})$
    // $\mathcal{F}$ has now size $I_n \times R_n \times S_{n-1}$
    $\mathcal{F} := \text{contraction}(\mathcal{F}, \Delta\mathcal{R}^{(n)})$
    // $\mathcal{F}$ has now size $R_n \times S_n$
  end for
  // $\mathcal{F}$ has now size $R_N \times S_N = R_N \times 1 = R_N$
  return $\mathcal{F} \cdot \mathcal{B}[:, :, 0]$ *// Vector-matrix product: the result is a histogram with B elements as expected*
end procedure

In Table 1 we show the asymptotic costs in terms of space, precomputing time and query time for several histogram reconstruction methods, including ours and the related method Wavelet-SAT [5] (which is lossless and limited to rectangular ROIs only).

6.2 Histogram Field Reconstruction

In this last contribution section we extend our framework to reconstruct many histograms at once, namely over a collection of sliding ROIs that produce a *histogram field*. This operation gives a mapping between each input pixel/voxel and the histogram of its neighborhood, and is equivalent to a bin-by-bin convolution with the ROI's indicator function. To this end we now use the identity

$$f * g = \left(\int f \right) * g'$$

instead of the one from Eq. 3. In discrete form, it allows us to compute the histogram of the input over a sliding ROI by simply convolving our IH $\mathcal{I}$ with the region's derivative $\Delta\mathcal{R}$. In other words, we need to convolve two tensors in the TT format. We implement this for separable regions $\omega = \omega^{(1)} \otimes \cdots \otimes \omega^{(N)}$, a case that is handled by simply convolving the TT cores along the spatial dimension [36]. More specifically, we build a new tensor $\mathcal{I}_* = [[\mathcal{I}_*^{(1)}, \ldots, \mathcal{I}_*^{(N)}, \mathcal{B}]]$ with each n-th core defined as

$$\mathcal{I}_*^{(n)} := \mathcal{I}^{(n)} * \Delta w^{(n)} \tag{4}$$

where $*$ denotes convolution of a 3D tensor (the TT core) with a vector along its second (spatial) axis and Δ is again the discrete differential operator, here applied to a vector.

7 Results

We implemented[1] and evaluated the proposed compression/decompression strategies on one image, three scalar volumes and one volume vector field.

1. Our code is available at *https://github.com/rballester/tthistograms*.

	Total space	Precomputing time	Query time (box ROI)	Query time (rank-S ROI)
Brute force	I^N	0	$O(K^N)$	$O(K^N)$
IH	$I^N B$	$O(I^N B)$	$O(2^N B)$	$O(2^N K^N B)$
IH (precomputed convolution)	$I^N B$	$O(N I^N \log_2(I) B)$	$O(2^N B)$	$O(2^N B)$ (assumes ROI shape never changes)
WaveletSAT [5]	No closed formula	$O(I^N \log_2(N)^N)$	No closed formula	Not supported
TT (proposed method)	$(N-1)IR^2 + IR + RB$	$O(NIR^3)$ [35]	$O(NKR^2) + RB$	$O(NIR^2S^2) + O(RB)$

TABLE 1: Asymptotic space and time costs for several methods. The whole uncompressed IH is used in the second and third algorithm. The third algorithm filters each bin slice by means of the fast Fourier transform (FFT), and the resulting convolutions are stored separately to achieve faster query times (but this assumes an invariant filter kernel).

7.1 Hardware and Software Used

Our code and tests were written in Python 3.5 and run on an Intel i7-4810MQ at 2.80GHz with 4 cores and 4GB of main memory. Operations for TT manipulation make use of the ttpy toolbox [37], a Python/FORTRAN library based on an earlier MATLAB package for the TT format. All compressed tensors as well as reconstructed histograms are handled in 64-bit floating point format. We compare our approximate results with respect to groundtruth histograms computed via brute-force counting, namely NumPy's `histogram()` function. As we are aware that brute-force histograms are highly-parallelizable, we also implemented a GPU-accelerated version of single histogram reconstruction that takes advantage of CuPy [38], a high-level NumPy-like interface for CUDA that can also seamlessly integrate handwritten kernels. Our kernel exploits CUDA's `atomicAdd()` operation and CuPy's `bincount_kernel`, and was run using an NVIDIA Quadro K2100M GPU with 2GB of memory.

7.2 Scalar Field Integral Histograms

We have evaluated our proposed compression with four scalar fields including a 4096^2 grayscale picture of a *Waterfall* [39] and three μCT scans of a *Bonsai* tree [40] (size 256^3), a *Lung* (size 512^3) and a *Flower* [41] (size 1024^3), all shown in Fig. 6 and originally having 8-bit depth.

As a preprocessing step, pixels or voxels are linearly mapped into the desired number of bins B: element-wise, $x \mapsto \lfloor x \cdot B/256 \rfloor$, with $B = 64$ and $B = 128$ in our experiments. Table 2 summarizes our compression algorithm's performance and results during the decomposition stage: time needed, compressed size, TT ranks, etc. Our compressed IH $\mathcal{I}$ takes at most a few hundred MB for the largest datasets, mainly depending on each data set's complexity; thus it is always one or two orders of magnitude smaller than the full uncompressed IH.

Fig. 7 demonstrates the results for two separable ROI query examples on the Waterfall image and their resulting histograms (exact and approximated). The query regions are a square and a 2D Gaussian, and are queried as described in Sec. 5 and Sec. 6 respectively.

Using the four scalar field data sets as a benchmark, we have gathered query performance results for a range of regions. These results are reported in Fig. 8. For each data set we consider ROIs of increasing size placed on the data set's center. For Gaussian regions we chose σ as 1/4 of the region's size in all cases. Each histogram was reconstructed 3 times; the final time is the average.

In relation to the brute-force approach, our new method performs significantly faster, even compared to a CUDA implementation, with more pronounced benefits for larger queries and datasets. As we expected, slicing operations (besides the bin counting itself) are a significant bottleneck in brute-force methods. This is much less of a burden for TT reconstruction, since it works with whole core slices and therefore needs fewer random accesses. The relative query error due to the compression is also shown to drastically diminish with increasing ROI query sizes. Note that rectangular queries demand a constant number of operations with our method, and this is reflected in the roughly constant TT reconstruction time. This is not the case for the Gaussian, whose costs increase moderately. Further discussions are given below in Sec. 8.

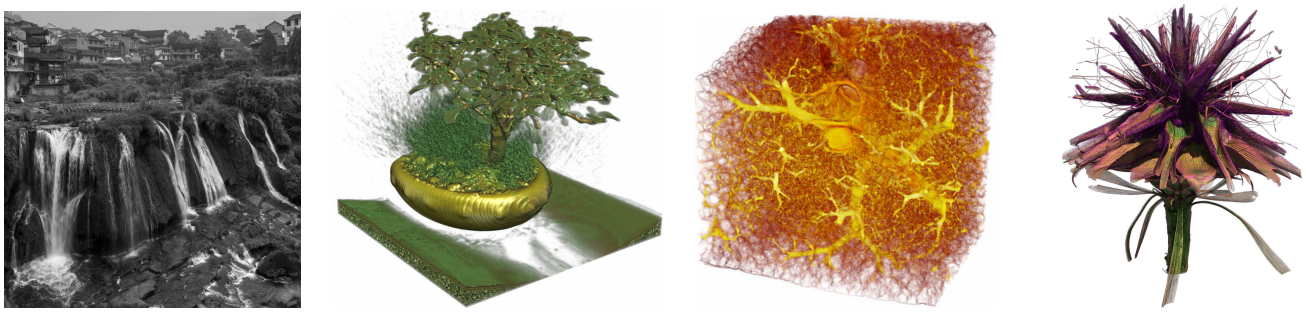
7.3 Vector Field Entropy

Our second experiment involves a 3D vector field, namely the wind speed coordinates from the 25th timestep of the Hurricane NCAR data set, available from [42]; see Fig. 9a. The original vector field contains 100 height slices, each of which has 500×500 data points. We removed the first 9 height slices, since they intersect with parts of Florida's terrain and thus contain missing values. The experiment consisted in computing 3D Shannon entropy fields based on the wind orientation histogram; such fields have applications in feature extraction, information-aware streamline placement and visualization, etc. [5], [43], [44]. To this end we quantize the wind direction at each voxel using 128 quasi-uniform regions on the spherical surface, see also Fig. 9b. The local histogram for a voxel is thus defined as the directional bin count for all wind information on a neighborhood of the voxel. The final field is the Shannon entropy of every local histogram, computed across the whole data set [43]. Once the window shape and size is provided, we compute the histogram field in one go as we detailed in Sec. 6.2.

Results are reported in Fig. 9 where we show the hurricane (a rendering of its vapor density), its global directional histogram, and several entropy fields for box and Gaussian local neighborhoods using both the proposed method and a brute-force approach. The latter is especially prohibitive for non-rectangular regions; it works by computing one FFT multidimensional convolution per histogram bin. Our method achieves a manifold speed-up factor, also for the rectangular case, while its results deviate very little from the exact groundtruth values given by the brute-force.

7.4 Cross-Correlation Queries

A simple, yet effective approach for interactive visualization of high-dimensional vector fields, for example of histogram features, is user-driven transfer function selection. Essentially, a 1D array $\boldsymbol{w}$ of B weights is defined and its dot product with each histogram is computed; the resulting scalar is displayed through a suitable colormap. If $\boldsymbol{w}$ and the histograms are all normalized, the result is the normalized cross-correlation (NCC) between a template and all possible window neighborhoods.



(a) Waterfall (4096^2) (b) Bonsai (256^3) (c) Lung (512^3) (d) Flower (1024^3)

Fig. 6: The four scalar fields used in our experiments: an image (a) and three μCT volumes (b), (c), (d).

	Waterfall	**Bonsai**	**Lung**	**Flower**	**Hurricane**
Size (MB)	16	16	128	1024	21.70
B	64	128	128	128	128
Decomposition time (s)	1282.26	1085.48	2070.83	28138.90	4958.32
Full IH (GB)	4	8	64	512	10.84
Compression target ϵ	0.0005	0.0001	0.0002	0.000015	0.0001
Compressed IH (MB)	181.32	38.34	205.15	85.61	281.28
IH compression ratio	22.59	213.69	319.45	765.50	39.49
TT ranks	100, 57	53, 137, 89	87, 271, 106	53, 121, 37	127, 535, 56

TABLE 2: Compression results for the 5 data sets we have tested.

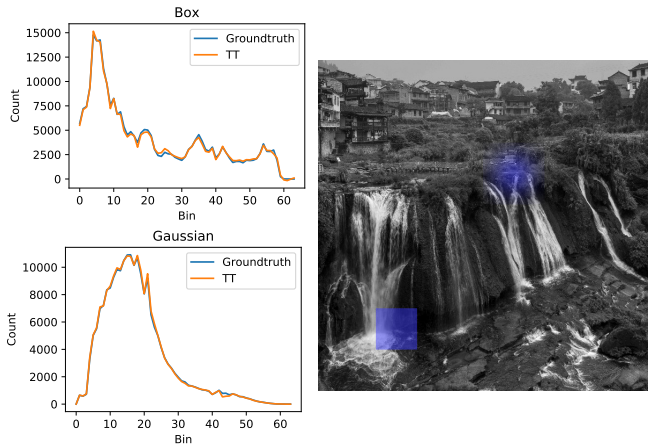


Fig. 7: Example: 64-bin histograms across two rank-1 rectangular 2D regions using TT compression to $(64, 51)$ ranks, with a 1:78.7 memory reduction from the original IH size. The TT box look-up over 512^2 pixels took 0.08ms while the brute-force took 4.2ms. The Gaussian query over 768^2 pixels took 4.4ms while the brute-force took 23.0ms.

As a last experiment we consider again the hurricane wind IH $\mathcal{I} = [[\mathcal{I}^{(1)}, \mathcal{I}^{(2)}, \mathcal{I}^{(3)}, \mathcal{B}]]$ (Sec. 7.3) and obtain its NCC w.r.t. arbitrary windows within our proposed TT representation. We first precompute offline a histogram field over all neighborhoods of a fixed size as in the previous section, and then store the norm of each histogram. During interactive exploration we allow the user to define a template window W whose histogram $\boldsymbol{w}$ we extract (Secs. 5 and 6) and normalize. We then weigh the last TT core of $\mathcal{I}$ with $\boldsymbol{w}$ along its second dimension, i.e. use $\mathcal{B} \times_2 \boldsymbol{w}$ instead of $\mathcal{B}$, and finally reconstruct (Eq. 4) and divide by the precomputed norms. The resulting multiarray summarizes each window's histogram into one single scalar, namely the correlation between the window's directional histogram and that of the template W. This way, the user can highlight and identify regions whose local wind behavior is similar. Note that the NCC lies between 0 and 1 as our feature vectors (histograms) are non-negative. Fig. 10 shows visualization results for a number of different windows W of size $8 \times 8 \times 91$, with response times under 2 seconds.

8 Discussion

Based on these numerical experiments we observe that our proposed compressed representation takes up much less space than the conventional IH approach as shown in Table 2. This is useful when the uncompressed IH strains or exceeds the available computational and memory resources. The compression is lossy and comes at the expense of a variable compression error, see also Fig. 8. We observe that, in general, bigger data sets can be compressed better than smaller ones. The same applies for sparse data sets: all tensor slices that are filled with zeros (e.g. Bonsai or Flower) are represented in a TT compressed format with zero-filled core slices, and in particular without increasing the TT ranks.

In terms of speed, TT reconstruction always becomes faster than naive brute-force traversal with ROIs of a certain size or larger. This overtaking point depends on the data set and prescribed ϵ, but usually is reached already by regions that are 100x or 1000x times smaller than the data set. As regards compression, our proposed incremental approach is robust and effective even for very large (512GB) integral histograms. Preprocessing times can take up to several hours, but such times are not uncommon for compression algorithms for large multidimensional data. If this time is taken into account, the break-even number of queries needed for our method to be overall faster than the naive traversal amounts to thousands (hundreds for histogram field reconstruction). However, we work under the asymmetry assumption, i.e. where interactive reconstruction is of paramount importance, especially when user interaction is involved.

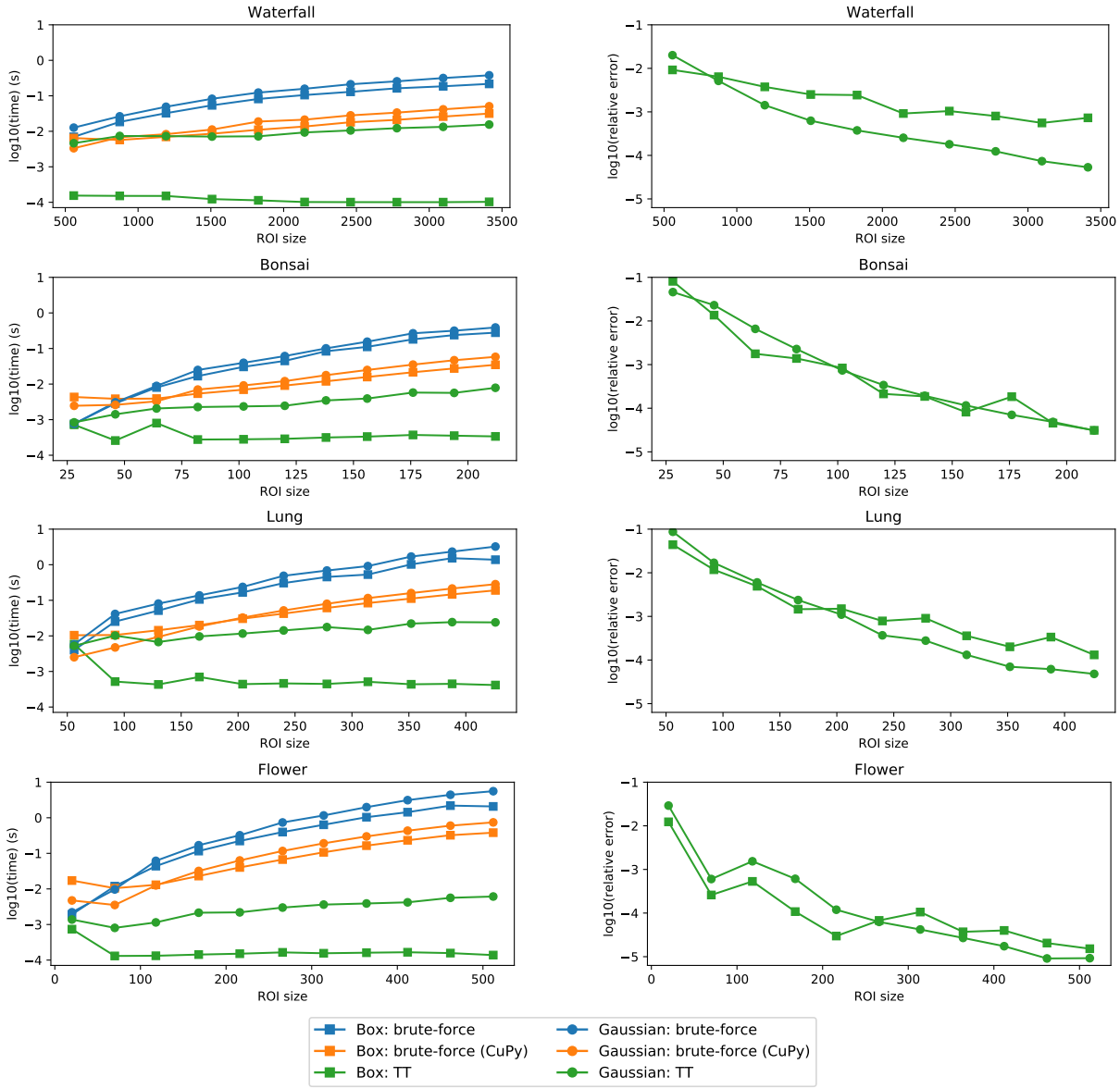


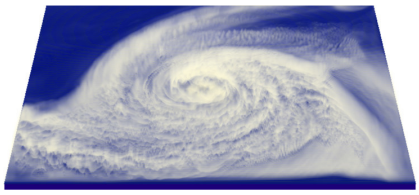
Fig. 8: Left: Average histogram reconstruction times for both box- and Gaussian-shaped regions. Right: Relative error of each reconstructed histogram. Our proposed TT representation generally outperforms brute-force approaches in terms of speed (even when they are CUDA-accelerated), and does so by an increasing margin as we increase the ROI size.

Note also that our algorithms scale well to higher dimensions, as tensor approximation is well defined and studied for $N \geq 2$, and the TT format is known to be an excellent representation for high dimensional data. Compared to wavelet-based alternative compression methods such as Wavelet-SAT [5], our method a) results in higher reduction rates (thanks to lossy compression and its use of adaptive transform bases); b) can reconstruct histograms over non-rectangular regions; and c) can apply transfer function weighing (and in particular, compute histogram field cross-correlations) at a small cost in the compressed domain. Its main drawback is of course the compression error introduced, but we found this relative error to be insignificant over medium to large ROIs, whereas the case of small ROIs is less important since brute-force traversal is actually the fastest method for such regions anyway.

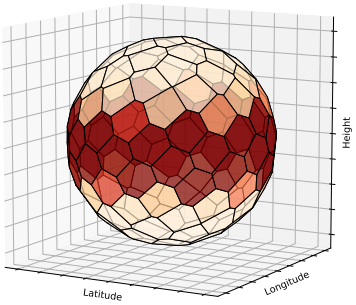
Our proposed reconstruction over Gaussian regions is significantly slower than its rectangular counterpart. However, it still outperforms the brute-force, usually even when exploiting GPU parallelism. We would like to highlight that, although general IHs are not well suited for non-rectangular regions, the tensor decomposition framework makes such queries possible thanks to its multilinearity and the range of compression-domain processing operations that it offers. The prescribed accuracy during compression is the main parameter of our system, and it results in a trade-off between smaller size and faster query time on one side versus lower error on the other side.

Limitations and Future Work

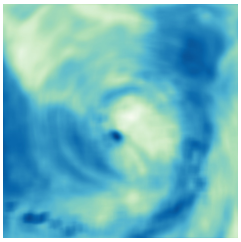
As mentioned before, the main limitation of tensor-compressed IH is related to small regions: in these cases a) the relative error of the method is higher, and b) the standard brute force traversal over a small area becomes reasonably fast, making alternative look-up methods less attractive. Regarding a), TT queries are fastest for rectangles; the break-even point against brute-force is



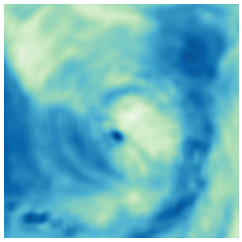
(a) Vapor density (volume rendering)



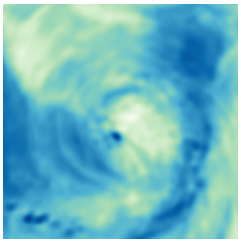
(b) Wind directional histogram, $B = 128$



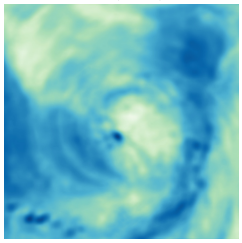
(c) Box regions Brute-force (59.8s)



(e) Box regions TT (1.9s)



(d) Gaussian regions Brute-force (1316.4s)



(f) Gaussian regions TT (23.8s)

Fig. 9: (a) A $500 \times 500 \times 91$ hurricane vector field at its 25th timestep; (b) directional histogram for its (latitude, longitude, height) wind coordinates, grouped into 128 bins on the spherical surface; (c-f) entropy fields for both box- and Gaussian-shaped neighborhoods of size $20 \times 20 \times 91$, computed via brute-force vs. our proposed method (Sec. 6.2).

more favorable, and there is more room for error reduction by reducing the error tolerance during decomposition. As a secondary limitation, our compression procedure is based on global L_2 norm and, in particular, there is no bound on the maximal absolute error. This is a usual situation in tensor-based methods, and is rarely a problem in practice.

Many operations, linear and otherwise, can be performed in the tensor compressed domain and have a potential impact in a range of applications for large and/or high-dimensional signals. As a future line of research we would like to exploit further ways of operating on compressed histograms and histogram fields. We would also like to port the current framework to other kinds of vector fields besides histograms.

9 Conclusions

We have contributed a novel tensor-based compression and query algorithm for integral histograms over multidimensional grid data. Our approach is based on an incremental tensor train decomposition. It demonstrates great memory reduction rates compared to raw integral histograms and allows for fast histogram computation over large regions at a high accuracy. We detailed first how to incrementally compress IHs that would take hundreds of GBs if represented explicitly. Once the compressed IH is available, we showed how to operate on its TT cores in order to properly access the encoded cumulative data and retrieve the desired result over a rectangular ROI at a high speed. We also showed how to map and query non-rectangular regions by exploiting their own tensor decomposition. Finally, we extended our reconstruction algorithms to support histogram field reconstructions, i.e. histograms over many regions at once.

To summarize, our framework can be helpful in applications that require fast approximate histograms, especially over medium to large query regions. We believe its potential can be exploited best in combination with the simple brute-force method, which can naturally compute histograms over small regions of arbitrary shapes without error and in reasonable times. Since most analysis and visualization applications are likely to require the original input data anyways, storing it along with our proposed compressed IH is a very feasible scenario.

Acknowledgments

This work was partially supported by the University of Zurich's Forschungskredit "Candoc" (grant number FK-16-012). We acknowledge the Computer-Assisted Paleoanthropology group and the Visualization and MultiMedia Lab (University of Zurich) for the acquisition of the Flower volume, and Prof. Dr. Schittny from the Institute of Anatomy (University of Bern) for the Lung.

References

[1] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 829–836.

[2] F. C. Crow, "Summed-area tables for texture mapping," in *Proceedings ACM SIGGRAPH*, 1984, pp. 207–212.

[3] E. Gobbetti, J. Iglesias Guitián, and F. Marton, "COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks," *Computer Graphics Forum*, vol. 31, no. 3, pp. 1315–1324, 2012.

[4] S. K. Suter, M. Makhinya, and R. Pajarola, "TAMRESH: Tensor approximation multiresolution hierarchy for interactive volume visualization," *Computer Graphics Forum*, vol. 32, no. 3, pp. 151–160, 2013.

[5] T.-Y. Lee and H.-W. Shen, "Efficient local statistical analysis via integral histograms with discrete wavelet transform." *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2693–2702, 2013.

[6] B. Weiss, "Fast median and bilateral filtering," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 519–526, 2006.

[7] Y. Wei and L. Tao, "Efficient histogram-based sliding window." in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3003–3010.

[8] R. Berger, S. Dubuisson, and C. Gonzales, "Fast multiple histogram computation using Kruskal's algorithm," in *Proceedings IEEE International Conference on Image Processing*, 2012, pp. 2373–2376.

[9] M. Kass and J. Solomon, "Smoothed local histogram filters," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 100:1–10, 2010.

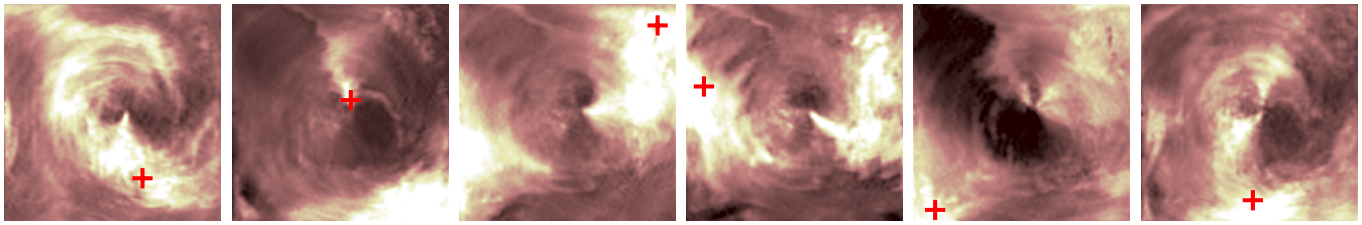


Fig. 10: Normalized cross-correlations between the hurricane wind vector field and six different template windows of size $8 \times 8 \times 91$. Brighter colors indicate higher histogram neighborhood correlations with a template whose center in each case is indicated by the red marker. This region-picking technique works in fact as a dynamic histogram transfer function selector that can e.g. highlight specific rain bands. Computation times ranged between 1.86s and 1.99s for these examples.

[10] M. Hadwiger, R. Sicat, J. Beyer, J. Krüger, and T. Möller, "Sparse PDF maps for non-linear multi-resolution image operations," *ACM Transactions on Graphics*, vol. 31, no. 6, pp. 133:1–12, 2012.

[11] R. Sicat, J. Krüger, T. Möller, and M. Hadwiger, "Sparse PDF volumes for consistent multi-resolution volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2417–2426, 2014.

[12] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2005, pp. 886–893.

[13] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, "Fast summed-area table generation and its applications," *Computer Graphics Forum*, vol. 24, pp. 547–555, 2005.

[14] D. Nehab, A. Maximo, R. S. Lima, and H. Hoppe, "GPU-efficient recursive filtering and summed-area tables," *ACM Transactions on Graphics*, vol. 30, no. 6, pp. 176:1–11, 2011.

[15] P. Schlegel, M. Makhinya, and R. Pajarola, "Extinction-based shading and illumination in GPU volume ray-casting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1795–1802, 2011.

[16] E. Tapia, "A note on the computation of high-dimensional integral images." *Pattern Recognition Letters*, vol. 32, no. 2, pp. 197–201, 2011.

[17] Y.-L. Wu, D. Agrawal, and A. El Abbadi, "Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes," in *Proceedings International Conference on Information and Knowledge Management*, 2000, pp. 414–421.

[18] P. S. Heckbert, "Filtering by repeated integration," in *Proceedings ACM SIGGRAPH*, 1986, pp. 315–321.

[19] C. Lundstrom, P. Ljung, and A. Ynnerman, "Local histograms for design of transfer functions in direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1570–1579, 2006.

[20] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[21] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi–modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[22] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[23] L. de Lathauwer, B. de Moor, and J. Vandewalle, "On the best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of higher-order tensors," *SIAM Journal of Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[24] R. Pajarola, S. K. Suter, and R. Ruiters, "Tensor approximation in visualization and graphics," in *Eurographics Tutorials*, 2013.

[25] R. Ballester-Ripoll and R. Pajarola, "Tensor decomposition methods in visual computing," in *IEEE Visualization Tutorials*, 2016.

[26] Q. Wu, T. Xia, and Y. Yu, "Hierarchical tensor approximation of multi-dimensional images," in *Proceedings IEEE International Conference in Image Processing*, vol. 4, 2007, pp. IV–49–IV–52.

[27] Q. Wu, T. Xia, C. Chen, H.-Y. S. Lin, H. Wang, and Y. Yu, "Hierarchical tensor approximation of multidimensional visual data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, pp. 186–199, 2008.

[28] S. K. Suter, J. A. Iglesias Guitián, F. Marton, M. Agus, A. Elsener, C. P. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola, "Interactive multiscale tensor reconstruction for multiresolution volume visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2135–2143, 2011.

[29] G. Wetzstein, D. Lanman, M. Hirsch, and R. Raskar, "Tensor displays: Compressive light field synthesis using multilayer displays with directional backlighting," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 80:1–11, 2012.

[30] R. Ruiters and R. Klein, "BTF compression via sparse tensor decomposition," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1181–1188, 2009.

[31] R. Ruiters, C. Schwartz, and R. Klein, "Data Driven Surface Reflectance from Sparse and Irregular Samples," *Computer Graphics Forum*, vol. 31, no. 2, pp. 315–324, 2012.

[32] R. Ballester-Ripoll and R. Pajarola, "Compressing bidirectional texture functions via tensor train decomposition," in *Proceedings Pacific Graphics Short Papers*, 2016.

[33] R. Ballester-Ripoll, D. Steiner, and R. Pajarola, "Multiresolution volume filtering in the tensor compressed domain," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, no. 99, pp. 1–14, 2017.

[34] R. Costantini, L. Sbaiz, and S. Süsstrunk, "Higher order SVD analysis for dynamic texture synthesis," *IEEE Transactions on Image Processing*, pp. 42–52, 2008.

[35] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[36] M. V. Rakhuba and I. V. Oseledets, "Fast multidimensional convolution in low-rank tensor formats via cross approximation," *SIAM J. Scientific Computing*, vol. 37, no. 2, 2015.

[37] "ttpy: Python implementation of the TT-toolbox," http://github.com/oseledets/ttpy.

[38] "CuPy: A numpy-compatible matrix library accelerated by CUDA," https://cupy.chainer.org/.

[39] "Furong Waterfalls," Wikimedia Commons, https://commons.wikimedia.org/wiki/File:1furongpanorama2012.jpg.

[40] "Real world medical datasets," http://volvis.org/.

[41] "Visualization and MultiMedia Lab's Research Data Sets," http://www.ifi.uzh.ch/en/vmml/research/datasets.html.

[42] "2004 SciVis Contest," http://sciviscontest-staging.ieeevis.org/2004/data.html.

[43] L. Xu, T.-Y. Lee, and H.-W. Shen, "An information-theoretic framework for flow visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1216–1224, 2010.

[44] M. Chen, M. Feixas, I. Viola, A. Bardera, H. Shen, and M. Sbert, *Information Theory Tools for Visualization*, ser. AK Peters Visualization Series. CRC Press, 2016.

Rafael Ballester-Ripoll received MSc degrees in Mathematics and Computer Science in 2012 from the Technical University of Catalonia (UPC), in the Center for Interdisciplinary Studies (CFIS). From 2012 to 2017 he was a doctoral candidate at the Visualization and MultiMedia Lab (VMML) of the University of Zurich. His research interests include tensor decompositions, scientific visualization, and signal processing and compression.

Prof. Dr. Renato Pajarola has been a Professor in computer science at the University of Zurich since 2005, leading the Visualization and Multi-Media Lab (VMML). He has previously been an Assistant Professor at the University of California Irvine and a Postdoc at Georgia Tech. He has received his Dipl. Inf-Ing. ETH and Dr. sc. techn. degrees in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 1994 and 1998 respectively. His research interests include real-time 3D graphics, scientific visualization and interactive 3D multimedia.