This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

Efficient and Flexible Hierarchical Data Layouts for a Unified Encoding of Scalar Field Precision and Resolution

Duong Hoang, Brian Summa, Harsh Bhatia, Peter Lindstrom
Pavol Klacansky, Will Usher, Peer-Timo Bremer, and Valerio Pascucci

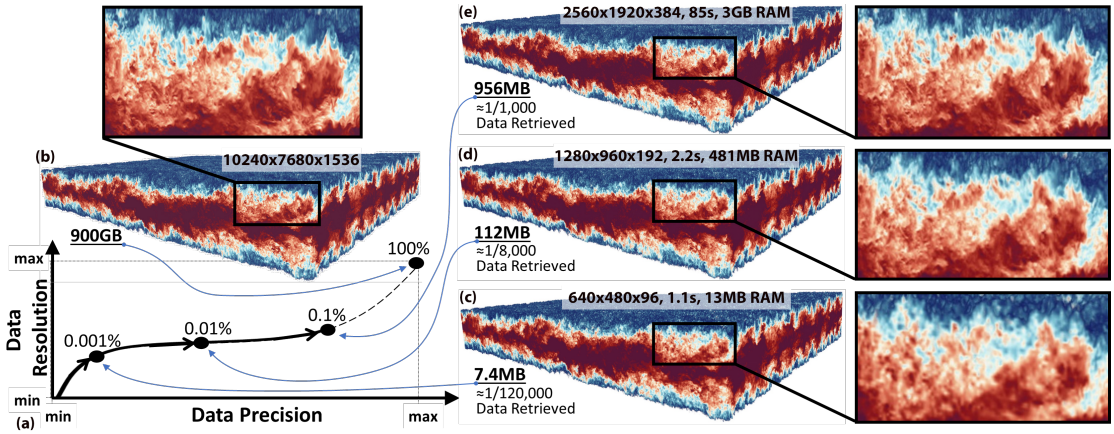


Fig. 1: We propose a hierarchical data layout that allows for various forms of progressive decoding that modulate improvements in both precision and resolution. Each progressive decoding traces a monotonic nondecreasing curve in the precision-resolution space from the origin, 0%, to the full data, 100% (shown in (a)). Using a 900 GB turbulent channel flow field [49] ($10240 \times 7680 \times 1536$, float64) (b), we demonstrate three approximations (c,d,e) of progressively increasing quality decoded along the curve in (a). The time to decode the data and RAM used are shown in the figure; data retrieved values are inclusive of the preceding points along the curve.

Abstract—To address the problem of ever-growing scientific data sizes making data movement a major hindrance to analysis, we introduce a novel encoding for scalar fields: a unified tree of resolution and precision, specifically constructed so that valid *cuts* correspond to sensible approximations of the original field in the precision-resolution space. Furthermore, we introduce a highly flexible encoding of such trees that forms a parameterized family of data hierarchies. We discuss how different parameter choices lead to different trade-offs in practice, and show how specific choices result in known data representation schemes such as ZFP [52], IDX [58], and JPEG2000 [76]. Finally, we provide system-level details and empirical evidence on how such hierarchies facilitate common approximate queries with minimal data movement and time, using real-world data sets ranging from a few gigabytes to nearly a terabyte in size. Experiments suggest that our new strategy of combining reductions in resolution and precision is competitive with state-of-the-art compression techniques with respect to data quality, while being significantly more flexible and orders of magnitude faster, and requiring significantly reduced resources.

Index Terms—scalar field, large-scale data, data compression, multiresolution, wavelet transform, coarse approximation

1 Introduction

With the advent of exascale computing and the increased availability of high-resolution experimental facilities, data can now be produced at sizes that overwhelm even the most efficient analytics or visualization algorithms. A common solution is to process large data sets into lower fidelity approximations for transfer, storage, and/or analysis. Reducing data fidelity primarily takes two forms: reducing resolution (i.e., number of data points) and reducing precision (i.e., the number of bits representing each data value). Generally, both types of techniques transform the original data into multiple "levels", such that meaningful information is disproportionately more concentrated in the first few levels, leaving subsequent levels with inessential information to be discarded. For resolution-based techniques, these levels often manifest in the form of a tree [67], a hierarchical space-filling curve [58], or a wavelet hierarchy [82]. With precision-based techniques, data samples are often decorrelated to form levels of precision, e.g., with energy concentration transforms [9, 28], followed by quantization to truncate away lower order precision levels (e.g., least significant bit planes). Both types of methods can achieve significant reduction. Therefore, both have seen widespread use.

Working individually with either approach, however, limits the achievable computational gains by maintaining either all bits for a few values or a few bits for all values. In this work, we show that both, previously separated, hierarchies can be combined into a single more general hierarchy, which we call a *precision-resolution tree*. Making both dimensions of possible data reduction available in a single, unified model lifts our approach out of the 1D reduction spaces (Fig. 2), where most existing techniques operate, thus ushering in new opportunities for different mixes of resolution and precision, which have been shown [39] to benefit different types of analysis tasks.

We therefore aim, foremost, not for pure compression performance, but for a data layout design that allows the flexibility of arbitrary incremental retrieval of *"chunks"* of data, progressively improving the resolution and/or precision of data. Furthermore, such data chunks

- *Hoang, Klacansky, Usher, and Pascucci are with SCI Institute, University of Utah. E-mail: {duong,klacansky,will,pascucci}@sci.utah.edu*
- *Summa is with Tulane Universiy. E-mail: {bsumma}@tulane.edu*
- *Bhatia, Bremer, and Lindstrom are with Center for Applied Scientific Computing, Lawrence Livermore National Laboratory. E-mail: {hbhatia,bremer5,lindstrom2}@llnl.gov*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

must be retrieved without excessive time overhead (e.g., due to complex decoding) or data overhead, such as re-reading of data (e.g., due to a lack of progressivity) and/or reading or decoding of unused data (e.g., due to a lack of random access).

With these goals in mind, we propose a new, highly flexible framework that can be leveraged to design novel, mixed-reduction strategies. We argue for choices of the framework's parameters that lean toward high degrees of progressivity and random access in all three domains — *precision, resolution, and space* — while achieving performance comparable to that of state-of-the-art pure compression techniques.

Contributions. In particular, we make the following contributions.

- We introduce a *precision-resolution tree* — a model that unifies the representation of spatial resolution hierarchies with precision-based data quantization approaches. In this model, adaptive approximations of data are modeled with the classical notion of a *valid cut* of the precision-resolution tree;
- We formalize a complete family of parameters for practical data layouts that encode such trees in the presence of compression and other practical considerations;
- We analyze the trade-offs associated with different choices in the degrees of freedom of this family of layouts and show how one can reproduce classical encoding schemes or design new ones with the advantage of being able to compare all within the same framework;
- We provide an empirical study, including system-level considerations, that allows designing a new encoding scheme for scientific data, which achieves competitive speed, memory usage, and compression rates. Our scheme allows decoding the data progressively while following different precision-resolution trade-off curves decided at read time, per the needs of the application.

Our proposed precision-resolution tree and family of data hierarchies are discussed in Section 3 and Section 4. The proposed system that implements these ideas is discussed in Section 5. We evaluate the system in Section 6 and show that even with increased complexity compared to traditional data layouts, appropriate design decisions and careful parameter choices enable the proposed layout implementation (available as opensource software [1]) to be both flexible and efficient.

2 Related work

Different types of data reduction approaches can be represented as fixed points or curves within the joint precision-resolution space (Fig. 2). Such techniques can be broadly classified as existing along the resolution axis, along the precision axis, or mixed.

Resolution-based reduction. Most resolution-based approaches subdivide the space to form a tree, where branches denote a spatial subdivision in one or more dimensions. How the dimensions are treated (simultaneously [47, 55] or independently [29, 81]) defines the shape of the tree. At coarse levels, data samples are obtained through some form of weighted averaging [13, 47, 61]. These trees often duplicate data for lower resolutions and therefore incur overheads in progressive data loading. Fine-level nodes can be discarded based on a threshold, thereby storing only sparse trees [10, 22, 30, 33, 37]. However, such approaches primarily support visualization and not numerical analyses.

Another group of techniques use data-dependent basis transforms [5–7, 31, 32, 73, 74], expressing the data as a linear combination of multi-dimensional basis functions forming levels akin to resolution. Common image and video compression methods also take this approach using variants of the discrete cosine transform [14, 71, 79], but with data-independent bases, thus trading quality for speed during encoding. These approaches are often limited in reconstructing coarse approximations, since often their "resolution levels" and the actual data samples in the original grid have no direct correspondence. Therefore, the (inverse) transform must be done at full resolution, and only subsequently can redundant samples be discarded, which is costly. Some approaches circumvent this limitation by constructing an octree before the transform [27, 32, 54]. Still, these approaches provide limited or no progression in precision.

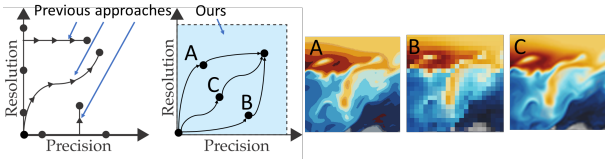


Fig. 2: Considering precision and resolution as two axes in the space of data reductions. (Left) Previous approaches operate either as fixed points or fixed progression curves in this space. (Middle) Our unified tree model supports multiple arbitrary progression curves simultaneously with one data layout, effectively covering the entire space. (Right) Reducing too much precision causes banding (A). Reducing too much resolution results in pixelization (B). This work explores how flexible combinations find better quality at comparable sizes (C).

Another notable approach, the IDX file format [43, 46], supports fast decoding by rearranging grid samples into a spatially coherent hierarchical space-filling curve [34, 58]. Because the rearrangement is efficient, IDX and techniques derived from it have been shown to scale to very large data sets [42, 59, 65, 72]. In this paper, we show that IDX's hierarchy [58] is a specific type of wavelet hierarchy, just without data filtering. Lacking data filtering means IDX does not have an interpolation basis and the coherency needed for effective compression.

Wavelets [20] are the most common transforms that support fast multiresolution decoding. Each transform step separates the input samples into equal halves: (1) *low-pass* coefficients representing a coarser grid, and (2) *high-pass* coefficients containing fine details absent in the first half. This transform can be recursively applied to produce a hierarchy of coefficients capturing details at multiple scales. Wavelet transforms are local and fast, and the coefficients are highly compressible [15, 16, 60, 66, 69, 76], and thus they are used in various data reduction systems [18, 51, 62, 82]. The multiresolution nature of the wavelet transform also makes it useful for, e.g., level-of-detail visualizations [35, 36, 40, 56, 64, 68, 77, 80]. Nevertheless, most of these systems do not take advantage of precision-based reduction, or do so only as a final lossy compression step with no progression. In contrast, our unified tree seamlessly consolidates resolution and precision.

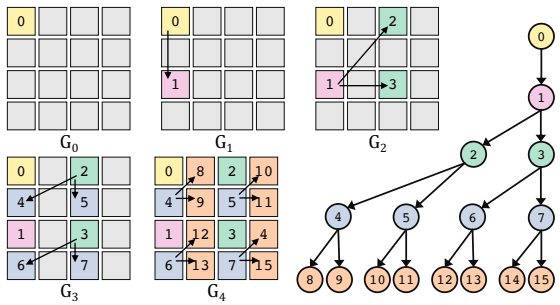
Precision-based reduction. The common theme among lossy compression approaches is to predict data values based on some model and encode the differences between the actual and the predicted values, which can be discarded/quantized. Here, better compression ratios can be achieved by predicting with more data points and using complex encoders, but often at the expense of speed. Although most resolution-based techniques stress progressivity, here the majority of techniques [3, 5, 26, 41, 53] adopt a single-error, write-once-read-once approach, where compression and decompression happen only at a predetermined quality. This approach requires the user to choose between reducing too much at write time or decoding too much at read time.

Other approaches [52, 60, 66] provide an additional degree of flexibility: the ability to specify a desired precision during decompression. Many of these approaches are used in combination with wavelets or other transforms, encoding the coefficients one bit plane at a time. Progression in precision is achieved by sorting the bit planes in decreasing order of significance, thus decoding the bit stream gives a progressive best effort approximation. For effective compression, however, a bit plane will often span multiple resolution levels, which complicates decoding when only a subset of the levels are desired.

Mixed (resolution-precision) reduction. JPEG2000 [76] allows for the selection of a small set of quality levels (computed at compression time) that are optimal combinations of resolution and precision in L_2 norm. This approach has disadvantages when applied to scientific data, since the preselected quality levels are quite limited, with little control over how those are achieved. We instead explore how the precision-resolution space can be navigated flexibly. JPEG2000 is also designed for imagery and not scientific data and as such does not support high-precision data. In addition, it is not concerned with large out-of-core data and therefore does not optimize for disk I/O. Overall, its optimization is tailored for visually appealing images and not necessarily the best for achieving scientific tasks, which, as shown

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics



(x,y)	$Z(x,y)$	P + 1 + S	S + 1 + P	g	f	h
(0,0)	0000: 0	+ + 0000	0000 + +	0	0	-
(1,0)	0001: 1	000 + 1 +	+ 1 + 000	4	8	4
(2,0)	0100: 4	0 + 1 + 00	00 + 1 + 0	2	2	1
(3,0)	0101: 5	010 + 1 +	+ 1 + 010	4	10	5
(0,1)	0010: 2	00 + 1 + 0	0 + 1 + 00	3	4	2
(1,1)	0011: 3	001 + 1 +	+ 1 + 001	4	9	4
(2,1)	0110: 6	01 + 1 + 0	0 + 1 + 01	3	5	2
(3,1)	0111: 7	011 + 1 +	+ 1 + 011	4	11	5
(0,2)	1000: 8	+ 1 + 000	000 + 1 +	1	1	0
(1,2)	1001: 9	100 + 1 +	+ 1 + 100	4	12	6
(2,2)	1100: 12	1 + 1 + 00	00 + 1 + 1	2	3	1
(3,2)	1101: 13	110 + 1 +	+ 1 + 110	4	14	7
(0,3)	1010: 10	10 + 1 + 0	0 + 1 + 10	3	6	3
(1,3)	1011: 11	101 + 1 +	+ 1 + 101	4	13	6
(2,3)	1110: 14	11 + 1 + 0	0 + 1 + 11	3	7	3
(3,3)	1111: 15	111 + 1 +	+ 1 + 111	4	15	7

Fig. 3: Illustration of a resolution tree, T_r, for a 4×4 grid and parameter $d = 1$, which leads to a branching factor of at most 2. The figure tabulates the Z-indices and the proposed bit manipulations thereof, leading to the functions f, g, and h, used for constructing the tree. The sequence of grids G_i shows the nested sub-grids arising from the top-down traversal of the tree. The nodes of the tree and the corresponding grid points are labeled by their indices (the output of f) and colored based on their levels (the output of g).

in Fig. 7, may require using very different kinds of error metrics.

The most relevant approach to the target of this work is VAPOR [19, 51], a data visualization toolkit that uses wavelets for compression and multiresolution access. VAPOR also exposes a set of predetermined quality levels at compression time. At read time, VAPOR can selectively fetch wavelet coefficients as the user increases the quality level and/or resolution level, reducing I/O and computation overheads. However, the quality levels correspond to the number of wavelet coefficients to decode and are not a direct control over data precision. This, together with the fact that only a few quality levels are supported (by default 4), means that the control over data quality is quite limited.

3 Unified Precision-Resolution Data Model

Given a scalar field defined on a regular grid, our goal is to build a hierarchy that captures both resolution and precision. For resolution, a traditional approach is that of kd-trees, quadtrees, or octrees. The hierarchy we propose differs from such trees primarily in that it has the same number of nodes as the number of grid points, and that approximations to the original field can be modeled using the well-established concept of a *cut* [17, 25] (Section 3.3). In particular, with a kd-tree, obtaining an approximation to the original field corresponds to cutting the tree and discarding all internal nodes, whereas with our hierarchy, all nodes above the cut define the approximation.

We first establish a set of rules that define such a *resolution tree*, T_r. Although T_r does not model precision yet, it is an important starting point for our unified *precision-resolution tree*, denoted as T_r^p. We will show that known hierarchies, such as the hierarchical-Z space-filling curve [58] and multiresolution wavelet hierarchies [63], are specific instances of T_r. T_r may be defined in many ways. Here, we propose a family of trees that can be characterized by a positive integer d, which controls the tree's branching factor. In particular, we require the root of T_r to have $2^d - 1$ children and every other internal node to have at most 2^d children. We first describe the construction of T_r with $d = 1$ (Section 3.1) before generalizing to $d > 1$ (Section 3.2). Hereafter, we refer to a data value stored at a specific grid point as *sample*, and as *grid point* when the value is not relevant.

3.1 Formulation of Resolution Tree for $d = 1$

To construct the resolution tree, we need to construct a map that takes a grid point to a node in the tree; this map consists of two functions. Without loss of generality, we describe these two functions — f and h (and an auxiliary one, g, to aid the derivation of f) — for a 2D grid, and illustrate them in Fig. 3.

The *index function*, $f(x,y) \to i$, maps the spatial coordinates (x,y) of a grid point to an integer $i \geq 0$ that represents the breadth-first traversal order of the corresponding node in T_r.

The *parent function*, $h(i) \to j$, maps the index, i, of a non-root node in T_r to that of its parent, j ($j < i$). With $d = 1$, each non-leaf node except the root has two children, so the parent function is simply $h(i) = \lfloor i/2 \rfloor$.

The *level function,* $g(x,y) \to l$, which maps the spatial coordinates of a grid point to an integer $l \geq 0$ — the level of the corresponding tree node, with the convention that the root is at level 0.

To understand g, consider traversing the tree in breadth-first order one level at a time (see Fig. 3). The set of nodes visited through level l constitutes a subgrid $G_l = \{f^{-1}(0), \ldots, f^{-1}(2^l - 1)\}$ (the set of points with indices $0, \ldots, 2^l - 1$). By design, $G_{l-1} \subset G_l$ and $G_{L-1} = G$, with L being the total number of levels, i.e., the complete grid is traversed by the end. Going from G_3 to G_4 in the shown example, 2^3 grid points at level 4 and with odd x coordinate are introduced, whereas going from G_2 to G_3, 2^2 grid points at level 3 and with odd y coordinate are introduced. This alternating pattern continues and can be formalized using the notion of the Z index formed by interleaving the bits of the two coordinates (known as the Morton code [78]). Assuming the interleaving pattern $yxyx\ldots yx$, Z indices that end with the bit 1 correspond to grid points with an odd x-coordinate, or $g(x_{odd}, y) = L - 1$ ($L = 5$ in this example). Similarly Z indices that end with bits 10 belong to level $L-2$, or $g(x_{even}, y_{odd}) = L - 2$. In general, Z indices that have a trailing bit pattern of 10...0 (one followed by m zeros) belong to level $g(x,y) = L - 1 - m$.

Formally, we partition the n bits of Z into a *prefix-1-suffix sequence*, $Z =$ P + 1 + S, where + is bit concatenation, the suffix S = 0...0 starts after the rightmost bit 1 in Z, and m is the number of bits in S. If Z has no ones, $Z =$ S, P is empty, and $m = n$.

Prop. 1. *A grid point (x,y) with $Z(x,y) = P + 1 + S$ belongs to level $l = g(x,y) = L - 1 - m$ of T_r. The nested grid sequence $G_0 \subset \cdots \subset G_{L-1}$ has $L = n + 1$ levels.*

Let (x_i, y_i) be the grid coordinates of a node i and (x_j, y_j) those of its parent j. Since $h(i) = \lfloor f(x_i, y_i)/2 \rfloor = f(x_j, y_j)$, in general the binary expansion of $f(x_j, y_j)$ has one more leading zero bit than that of $f(x_i, y_i)$. Further, since $g(x_i, y_i) = g(x_j, y_j) + 1$, the binary expansion of $Z(x_j, y_j)$ in general has one more trailing zero bit than that of $Z(x_i, y_i)$. Therefore, f can be obtained by swapping the prefix (P) and the trailing zeros (S) portions of $Z =$ P + 1 + S around the middle 1 bit. Effectively, the level-indicating bits (S) are brought to the front, so that lower resolution grid points are traversed first in T_r.

Prop. 2. *The index of a grid point (x,y), with $Z(x,y) = P + 1 + S$ is $f(x,y) = i = S + 1 + P$, and that of its parent is $h(i) = \lfloor i/2 \rfloor$.*

Exact calculations of f, g, and h for all the 16 grid points of a 4×4 grid, as well as the resulting T_r, are included in Fig. 3.

3.2 Generalized Formulation of Resolution Tree for $d > 1$

Bijective maps for tree construction. In general, a generic framework for a such resolution hierarchy, T_r, satisfies the following properties.

- There is a bijective map, $\mathscr{M}_d$, from the set of points of a grid G to the set of nodes in T_r. $\mathscr{M}_d$ consists of an index function $f_d(x,y) \to i$ and a parent function $h_d(i) \to j$. The root node of T_r has $2^d - 1$ children, and every other non-leaf node has at most 2^d children.
- Each node in T_r is associated with a *data value* that represents some approximation of the data. This data value may or may not be the same as the corresponding *sample*.

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

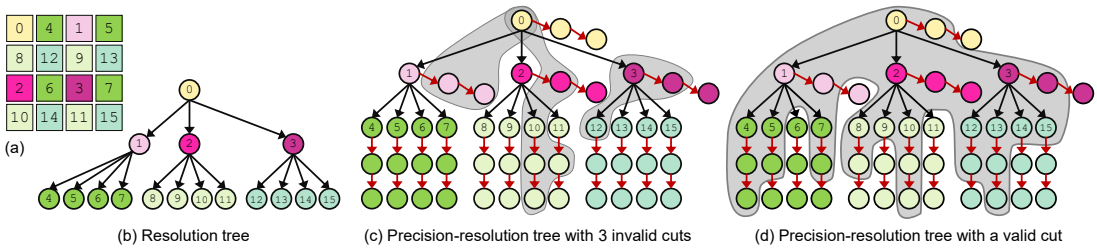


Fig. 4: The input is a 4×4 grid (a) from which we construct a T_r (b) using $d = 2$. Then, we extend T_r to form T_r^p (c) by adding nodes representing bit planes (the blank nodes) together with necessary edges (in red). The shaded regions correspond to three invalid cuts in (c) and one valid cut in (d). The three cuts in (c) are invalid since each shaded region intersects at least one path from the root to a leaf more than once.

- The root node of T_r (at level 0) represents an approximation of the entire field as a grid, G_0, corresponding to a single value, whereas a fully refined tree represents G exactly. Traversing T_r top-down from the root, the inverse map, $\mathscr{M}_d^{-1}$, can be invoked on the nodes visited at each subsequent level l, yielding a nested sequence $G_0 \subset G_1 \cdots \subset G_{L-2} \subset G_{L-1} = G$ of L grids, each providing an increasingly finer approximation of the field represented by G.

The functions f, g, and h defined in Section 3.1 are hereafter referred to as f_1, g_1, and h_1, and can be generalized to f_d, g_d, and h_d for any positive integer d. Although d can be used purely to control the branching factor of T_r regardless of the dimensionality (the same way a kd-tree, $d = 1$, branching factor 2 can be built on a 3D grid), letting d equal the dimensionality of the data can be more intuitive and natural since doing so leads to common types of wavelet subband decompositions.

Recall that g_1 was defined by counting the number of trailing zero bits in Z. To generalize, we instead count the number of groups of d trailing zero-bits (left-padding Z with `0`s if necessary) and denote that number as m_d. Then, $g_d(x,y) = L - 1 - m_d = \lceil n/d \rceil - m_d$ with $L = \lceil n/d \rceil + 1$ being the total number of levels in the tree and n the total number of bits in Z. Note that when d equals the dimensionality of the data, n is a multiple of d. To generalize the formulation of f_d, we partition Z into $Z = \mathtt{P} + \mathtt{F} + \mathtt{S}$, where `S` is the longest sequence of $d \times m_d$ trailing zero bits in Z, `F` is the next sequence of d bits, and `P` is the remaining prefix of Z. If Z has no ones, $Z =$`S` and both `P` and `F` are empty, and $m_d = \lceil n/d \rceil$.

Prop. 3. *A grid point (x,y) with $Z(x,y) = P + F + S$ belongs to level $l = g_d(x,y) = L - 1 - m_d$ of T_r. The nested grid sequence $G_0 \subset \cdots \subset G_{L-1}$ has $L = \lceil n/d \rceil + 1$ levels.*

Prop. 4. *The index of a grid point (x,y) with $Z(x,y) = P + F + S$ is $f_d(x,y) = i = S + F + P$, and that of its parent is $h_d(i) = \lfloor i/2^d \rfloor$.*

For example, consider the tree with $d = 2$ for a 4×4 grid in Fig. 4a. We have $Z(2,2) =$ `1100`. Here, `P` is empty, `F` = `11`, and `S` = `00`. Furthermore, $m_2 = 1$ and therefore $g_2(2,2) = 4/2 - 1 = 1$. Swapping `P` and `S` around `F`, we obtain $f_2(2,2) =$ `0011` $= 3$. The parent of this node has the index $h_2(3) = \lfloor 3/2^2 \rfloor = 0$.

If the input grid G is in 3D, with $\mathscr{M}_1$, each subgrid G_i grows twice as large on the next level (i.e., $|G_{i+1}| = 2|G_i|$), but only in one dimension at a time, similarly to a kd-tree. With $\mathscr{M}_2$ instead, each G_i grows in two dimensions at a time similarly to a quadtree and, therefore, $G_{i+1} = 4|G_i|$. With $\mathscr{M}_3$, G_i grows by 8 times like an octree, with expansion happening in all three dimensions with each increasing level. Note that $\mathscr{M}_1$ describes exactly the hierarchical-Z space-filling curve [58] (here, we provide an alternative formulation), whereas $\mathscr{M}_2$ and $\mathscr{M}_3$, respectively, describe the primal subdivision approach introduced by [50], as well as the most standard type of multiresolution 2D and 3D wavelet subband decompositions [70]. Thus, hierarchical-Z indexing can be considered a form of wavelet decomposition but without actual data filtering (also known as the *lazy wavelet transform*). Such concepts are unified in a formal framework in this study.

Subbands. In wavelet terms, a *subband* contains a subset of wavelet coefficients on the same level, with each subset spanning the entire spatial domain. In 1D, each wavelet transform step creates two subbands: one containing low-pass filtered samples (L) and the other high-pass filtered ones (H). In higher dimensions, more subbands may be created (four in 2D and eight in 3D). In our construction, the subband number is the rightmost group of d-bits that are not all zeros in the interleaved index Z. Since there are d bits in the group, there will be $2^d - 1$ subbands on the corresponding level (excluding bit pattern `0...0`, because this belongs to another level by definition).

Combinations of maps. Finally, we note that generalized resolution trees can be built by combining different maps. We may start with $\mathscr{M}_1$, but for the next level switch to $\mathscr{M}_2$, followed by $\mathscr{M}_3$, and so on. The bit interleaving pattern used for Z can also be arbitrary, with Morton code being a straightforward one, but by no means the only choice. One potential use case is when the input grid size is highly non-uniform, e.g., $32 \times 32 \times 512$, in which case, we may pick such interleaving patterns as *zzzzzyxzyxzyxzyxzyx* and apply four levels of $\mathscr{M}_1$ (on the *zzzz* part) followed by five levels of $\mathscr{M}_3$ (on the *zyxzyxzyxzyxzyx* part). Note that for non-power-of-two-dimension grids, we may insert "virtual" grid points so that the dimensions become powers of two, construct T_r, and then discard the virtual nodes. The previous example also serves to highlight the fact that in addition to d, the $\mathscr{M}_d$ maps are also parameterized by the bit interleaving pattern used to form Z. Furthermore, when data filtering such as the wavelet transform is used, a mismatch between d and the dimensionality of the data can lead to a T_r in which parent-child relationships between nodes do not reflect the correlations between corresponding coefficients. For example, if a 2D wavelet transform is performed such that the resulting subbands are those that are formed with f_1, a tree built from a map that combines f_1 and h_2 may work better than one built from $\mathscr{M}_1 = (f_1, h_1)$ because the former group wavelets with the same orientation at different scales.

3.3 Precision-Resolution Tree

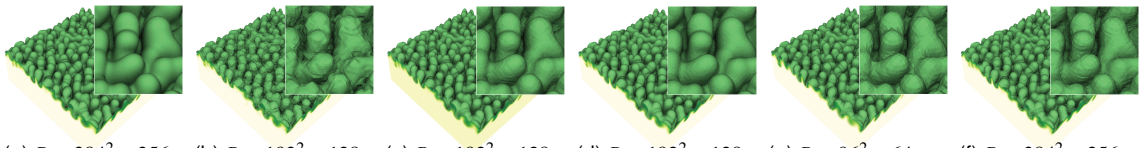
By definition, a T_r can be traversed only in the order of resolution (the vertical axis in Fig. 2), as the corresponding nodes store data values at full precision. We next focus on incorporating precision to T_r to form a *precision-resolution tree*, T_r^p. To achieve progression in precision, we first define the concept of a *bit plane*: assuming all sample values $\{V_k\}$ are P-bit integers, a *bit plane*, B_i $(0 \le i < P)$ is the set of bits (of the samples or transform coefficients) that share the same bit position, i. With this definition, we split each of T_r's nodes into a sequence nodes in T_r^p and connect the sequence through a chain, such that each node in the sequence now encodes a bit of the original node in T_r. Such a chain connects the bits of a value from the MSB (most significant bit) to the LSB (least significant bit). To finish the construction of T_r^p, we bring over all the edges of T_r, using the MSB node of each sequence as a proxy to the original node in T_r.

If the sample values are floating-point numbers, we express all values in the form $V_k = 2^E Q_k$ with a single exponent E and then consider the quantized integer values Q_k. We adopt the convention that B_0 is the least significant bit plane, and define a *precision level*, B_p, to be the set of all bit planes $\{B_i\}$, such that $i \ge P - p$ with P being the total number of bit planes. Likewise, a *resolution level* L_l is the set of all grid points whose corresponding nodes in T_r^p belong to levels that are at most l.

Approximations using valid cuts. Approximations to a scalar field in both precision and resolution can be defined using the classical notion of a *valid cut* [17, 25] of T_r^p (Fig. 4). Given a T_r^p and a subset C of its nodes that contains the root, a *cut* is defined as the set of edges leaving C. The cut is a *valid cut* if it does not contain two edges on the same path from the root of T_r^p to any leaf. If C corresponds to a valid cut, then C defines an approximation of the data in precision and resolution.

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics



(a) $R = 384^2 \times 256$, $\varepsilon = 0$, 288 MB (b) $R = 192^2 \times 128$, $\varepsilon = 1/4$, 195 KB (c) $R = 192^2 \times 128$, $\varepsilon = 1/32$, 357 KB (d) $R = 192^2 \times 128$, $\varepsilon = 1/128$, 563 KB (e) $R = 96^2 \times 64$, $\varepsilon = 1/128$, 199 KB (f) $R = 384^2 \times 256$, $\varepsilon = 1/16$, 1024 KB

Fig. 5: Isocontours extracted at different resolution (R) and precision levels (expressed as ε for absolute error) using our system. It is interesting to see that better surface quality can be obtained with less data at a lower resolution but higher precision level ((d) vs. (f)). The opposite can also be seen, where higher resolution but lower precision (c) results in better surface quality (than (e)). This example demonstrates both the versatility of our system (all approximations are obtained from the same data layout on disk), and the need for fine control in precision and resolution.

In our construction of T_r^p, obtaining a C corresponding to a valid cut is equivalent to obtaining an approximation by always retrieving coarse-resolution and higher order bits first. Doing so is desirable because coarse-resolution bits tend to contain more function energy and higher order bits have bigger impact on error. Finally, maintaining a valid cut in memory is simpler compared to maintaining an arbitrary cut, since the former requires only one "marker" on each path from the root to a leaf, whereas the latter requires specifying any subset of the edges.

4 Parameterized Family of Data Hierarchies

In this section, we introduce abstractions for data transformation and organization techniques often found in practice, to compute and store a T_r^p (with associated node values) on disk so that answering common data queries incurs minimal time, storage, and data transfer costs. We define four common types of queries: (1) Q_{prec} queries that request data at coarse resolution but high precision, (2) Q_{res} queries that request data at low precision but high resolution, (3) Q_{mixed} queries that request at some balanced combination of precision and resolution, and (4) Q_{roi} queries that request data at very high resolution and precision, but only for a subset of the whole grid (the region of interest, or ROI). In Fig. 5 we visually compare several approximations to the original field, produced by such a set of queries.

To design a data model that can serve such queries efficiently, we introduce the concepts of *bricks*, *blocks*, *chunks*, and *files*. Bricks and blocks provide logical grouping of nodes in T_r^p to facilitate Q_{roi} queries whereas chunks and files control the storage of tree nodes on disk to facilitate Q_{prec}, Q_{res} as well as fast I/O. In addition to the previously described bit interleaving pattern and map (Section 3), additional choices can be made for the sizes of bricks, blocks, chunks, and files, ultimately describing a family of physical hierarchies characterized by a set of parameters. Finally, we introduce two more parameters — the data transformation/filtering method (e.g., wavelet transform) and the encoding/compression scheme (e.g., ZFP), noting that the identity transform and verbatim/literal encoding are valid choices. The rest of the section discusses these parameters in detail.

4.1 Bricks as Units of Random Access

A frequent type of query in visualization and analysis is to retrieve a region of interest (ROI) in space, which corresponds to retrieving the corrresponding nodes in T_r^p. The most straightforward way to support this capability, in the presence of (potentially variable-rate) compression (which precludes implicit on-disk indexing), is to maintain pointers to the locations of all the nodes on disk. Although such a solution is very costly, this cost may be amortized by storing one pointer for every brick of nodes (or samples) instead. On each level, a *brick* is a set of $B_x \times B_y \times B_z$ contiguous samples in space. Partitioning the domain into bricks can be done either before or after the construction of T_r^p (and any data transformation). In both cases, such partitioning not only helps with Q_{roi} queries but also can speed up the construction of T_r^p, since smaller grids are better suited for caching and parallelization.

If partitioning is done after data transformation, there can be data dependencies among neighboring bricks, which complicates data reconstruction. For example, if wavelet transform is used, the support of each wavelet basis function may span across brick boundaries, requiring neighboring bricks to recover samples within a brick, thus negatively affecting I/O and decoding time. On the other hand, forming bricks before constructing local hierarchies avoids such issues while further facilitating parallelization during T_r^p construction [44], as bricks are now completely independent. The downsides of this brick-first approach are that a forest of (shallow) local hierarchies is created instead of a single global hierarchy, and the transform coefficients at the brick boundary may be artificially large, resulting in blocky artifacts during reconstruction (we present a solution for the latter in Section 5).

The problem of shallow local trees may be solved by merging the local trees fully (into a single tree) or partially (into a forest) to reach any desired number of hierarchy levels. We introduce parameters $L_{global} \geq 1$ and $L_{local} \geq 1$ to control the number of levels for the global and local hierarchies, respectively. Note that L_{local} can be smaller than the maximum number of levels possible, given a brick size. Merging of local trees is done by generating a tree that spans all local roots, while keeping the local edges intact (Fig. 6). The merged tree $\tilde{T}_r^p$ can be considered an approximation to the T_r^p that would have resulted from a non-bricking approach. Evidently, these are different trees, which may produce approximations corresponding to subtrees with different number of nodes for two identical queries. As such, one may expect $\tilde{T}_r^p$ to produce approximations with either significantly fewer or significantly more nodes compared to T_r^p, depending on the size of the query's ROI, However, having computed the exact number of nodes in several approximations for both trees, across a wide range of brick and ROI sizes, we observe that this number closely tracks the ROI size regardless of tree types, and the difference quickly becomes negligible as the ROI grows. This observation suggests that $\tilde{T}_r^p$ and T_r^p are functionally very similar.

4.2 Per-Brick Data Transformation or Filtering

Recall that each node in a T_r is associated with a data value, which is a transform coefficient computed from the original grid's sample values. In our model, the type of transform is a parameter orthogonal to the map used to construct the hierarchy. Typical types of transform include the many kinds of filters (wavelets, KLT, DCT, etc.) as well as the identity transform (in which case the resulting hierarchy is subsampling-based). A transform using the max operator is also useful in certain cases. Note that to inverse transform a max hierarchy, one needs to encode, for each parent node, which sibling is the largest among its children using d additional bits, assuming the $\mathscr{M}_d$ map. When the task is to detect the presence of isocontour components, as shown in Fig. 7 for example, averaging filters such as wavelets may not be ideal since they can lead to both false positives and negatives, forcing exploration of the full resolution data to guarantee no missing information. In such cases, a max hierarchy can yield coarse approximations with false positives but no false negatives, and therefore entire empty regions can be skipped since no new features can be created by refining those regions.

4.3 Blocks as Units of Compression

In practice, the transform coefficients in each brick are often partitioned into *blocks* and compressed as blocks. The ways blocks are formed can be broadly categorized into grouping grid samples versus grouping nodes in the corresponding tree (e.g., T_r). For example, ZFP [52] uses grid-based blocking, whereas JPEG2000 [76] and SPIHT [66] use tree-based blocking, in breadth-first and depth-first orders, respectively. Note that we can model this blocking behavior directly in T_r^p by letting each node represent a bit plane from not a single but a group of samples. A brick is then a collection of contiguous blocks. The blocking method may affect the size of the resulting subtrees corresponding to approximations produced for certain query types. As an example, for Q_{prec}

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

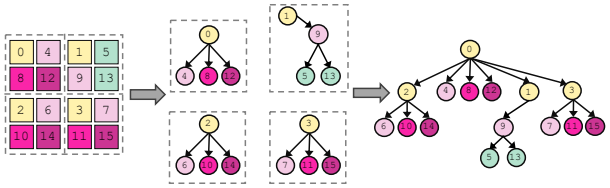


Fig. 6: A global tree can be constructed by merging local trees built independently from each brick. Local trees can be built using different maps (here $\mathscr{M}_1$ is used for one brick and $\mathscr{M}_2$ is used for the other three). The merging is done by generating a tree that spans all the local roots ($\{0,1,2,3\}$), while keeping the local edges intact.

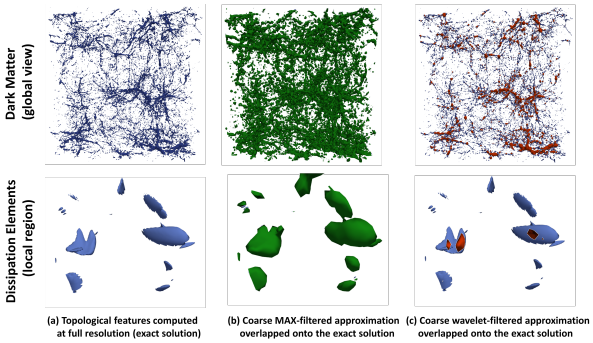


Fig. 7: A demonstration of the need for flexibility in the filtering operator. (b) the max-based approximation in green better preserves the features (blue) than (c) the wavelet-based approximation in red. The max operator tends to grow the components and potentially merge neighboring components whereas wavelets tend to average away components.

queries, depth-first (by resolution) and grid-domain (by space) blocking are not desirable since they tend to conflate samples across resolution levels. In our model, the blocking method, block size, and the compression or encoding method for each block are configurable parameters. Examples of encoding methods include ZFP, EBCOT, SPIHT, SPECK, etc. Verbatim encoding (i.e., no compression) is also a valid choice.

It is possible to use different encoding schemes for different blocks, which would typically require keeping a certain amount of metadata to indicate the scheme for each block. However, it is reasonable to expect that, in practice, the total number of schemes would be small. Thus, the cost of storing this metadata would be quite minimal and well amortized over the size of a brick (e.g, 64^3 coefficients) or a block (e.g., 4^3 coefficients). Nevertheless, our proposed implementation (Section 5) does not utilize this option, thus incurring no such metadata overhead.

4.4 Chunks as Units of Disk I/O

Next, we discuss how to store T_r^p on disk to facilitate efficient queries. In practice, disk I/O usually happens in big chunks of bytes, which we model using the concept of a *chunk*, our smallest unit of I/O. A chunk can be defined to have either a fixed size in bytes or a fixed extent in some space. Storage of chunks into files also has implications for performance. A large file can slow down chunk lookup (as there are too many chunks) and reduces parallelism during file I/O due to potential data races caused by multiple threads writing to the same file, especially in a distributed setting [45]. On the other hand, having too many small files puts pressure on the file system and increases the amount of metadata fetched at read time. In our framework, two parameters control these trade-offs: the number of chunks in a file and the ordering of chunks. An optimized chunk ordering can minimize disk seek latency and take advantage of the operating system's prefetching.

Finally, it is important to consider indexing for random access of levels and bit planes, as well as of blocks, bricks, chunks, and files. If few chunks are included in a file, a simple array of IDs and file offsets would suffice for random access of chunks. When the number of chunks per file is large, however, a B-tree [8] could perform better. Likewise, whether a file can be quickly accessed depends on the number of files per directory, which, if too large, is a bottleneck during file lookup. All such choices are parameters to our data model.

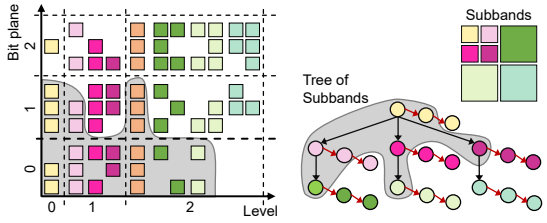


Fig. 8: Visualization of chunks in the precision-resolution space. The hierarchy shown here has three levels, seven subbands, and three bit planes. Each combination of subband and bit plane contains a number of chunks, shown as squares and colored by subband (light brown squares are extrapolated chunks). Each node of the tree represents an entire subband instead of individual coefficients/blocks. The shaded regions represent a cut in the tree and the corresponding subset of chunks.

4.5 Parameter Choices

In summary, we have proposed a family of hierarchical data models with various free parameters. Certain parameter choices lead to known data representations in the literature. For example, IDX [58] uses one brick the size of the entire grid and builds one tree using $\mathscr{M}_1$ with no data filtering. ZFP [52] builds no tree (0 levels), uses grid-based blocking, and encodes each block using the ZFP encoder. JPEG2000 [76] uses the term *tiles* to refer to our bricks, filters data with wavelets, uses depth-first blocking and refers to each block as a *code block*, and encodes each block using its own EBCOT encoder. Finally, the recently proposed system in [54] also defines blocks, bricks, and pages, with semantics that map very well to our blocks, bricks, and chunks. Certain combinations can be intriguing, such as encoding IDX's individual resolution levels using ZFP, or using the ZFP encoder as a lightweight replacement to EBCOT to encode wavelet coefficients.

5 System Description

We present a system that implements a member of our family of data layouts with concrete parameter choices and techniques to handle real-world data. As mentioned earlier, most reduction techniques support only one way to refine data, i.e., as a progression in either precision or resolution. Nevertheless, it is unclear whether such individual schemes can be combined sensibly into a 1D progression navigating the 2D precision-resolution space. In comparison, our approach does not impose any progression order on data chunks themselves, except for when valid cuts are concerned. Rather, we advocate for a database-inspired approach at system level, such that the data layout is designed to best serve chunks from the 3D precision-resolution-sample space in *any* order demanded by the analysis task at hand. Fig. 8 translates the understanding of *cuts* onto the 2D precision-resolution-sample space in terms of chunks of data.

System parameters. Hereafter, we describe the system for 3D data and use bricks of size $B_x \times B_y \times B_z$ samples, where $B_{x,y,z} = 2^{k_{x,y,z}}$ for $k_{x,y,z} \geq 0$. If any of $B_{x,y,z}$ equals 1, each brick belongs to a slice, allowing easy encoding of slice-based volumetric data (e.g., medical scans). We use the map $\mathscr{M}_3$ and let $L_{local} = 2$, while leaving L_{global} up to the user to control the hierarchy's depth. We use a breadth-first blocking approach limited to each wavelet subband to group coefficients into blocks of size 4^3 and encode each block with ZFP.

For chunking, we do not fix the chunk size in bytes, but let each chunk span the same number of samples in space (we settle on the value of 512^3 through the experiments discussed in Section 6). To form chunks, we partition the 3D space of precision, resolution, and spatial domain into tiles of size T_B (bit planes) $\times T_S$ (subbands) $\times T_G$ (grid points), and assign compressed bits from the same tile to one chunk (note that this is possible in our particular compression scheme, provided that T_G is a multiple of the block size). Thus, a *tile* can be considered the "extent" of a chunk in the aforementioned 3D space. A small T_G can result in chunks too small for optimal I/O. On the other hand, a large T_G creates spatial couplings that can penalize small Q_{roi} queries; likewise, large T_B and T_S can negatively affect Q_{res} and/or Q_{prec} queries. We let $T_B = T_S = 1$, and choose T_G such that $T_G = 2^k$ and $T_G \geq B_x, B_y, B_z$. For efficient I/O, we do not write data to disk as

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

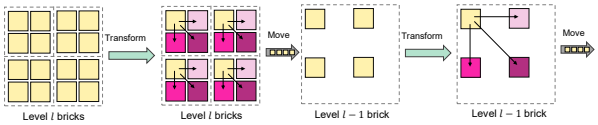


Fig. 9: At each level, we transform the samples in each brick to form local trees, and then move every 2^d root nodes (in yellow) of such trees into a parent brick, before repeating the process for the next coarser level.

soon as a brick is compressed, but buffer the compressed bits using one *bit stream* for each combination of subband and bit plane, and flush the bits only for each chunk. Equivalent bit planes across blocks (taking into account the block exponents) are written to the same stream.

Finally, we use the CDF 5/3 multilinear wavelets [20] for filtering, due to their simplicity and effectiveness in compression, and use the lifting approach [24] to compute the wavelet transform: $\hat{f}_{2i+1} = f_{2i+1} - \frac{1}{2}(f_{2i} + f_{2i+2})$ (w-lift) and $\hat{f}_{2i} = f_{2i} + \frac{1}{4}(\hat{f}_{2i-1} + \hat{f}_{2i+1})$ (s-lift) where f_i denotes the input sample value at index i and $\hat{f}$ the wavelet (odd-indexed) and scaling (even-indexed) coefficients. Below, we highlight important features of the proposed system and how we achieve them.

Encoding bricks independently. Encoding bricks independently of each other is key to handling out-of-core data sets, because the data can be processed one brick at a time during compression and reconstructed at brick level during decompression. Our hierarchy of L_{global} levels is built from the bottom up, with data transformation on each level performed in two steps. First, we perform the wavelet transform independently for each brick using $L_{local} = 2$, which corresponds to one transform pass in each dimension. Then, we move the coefficients in the coarsest subband into corresponding "parent" bricks of the next (coarser) level (typically each brick on level l is the parent to 2^3 bricks on level $l+1$ in 3D). When the coarser level bricks are fully formed, we again apply the same two transformation steps on those bricks. The recursion stops when the desired height for the global tree(s), L_{global}, is reached. This procedure is illustrated in Fig. 9.

Once the coefficients in the coarsest subband in a level-l brick are moved to its parent brick on level $l-1$, the rest of the coefficients in the level-l brick are encoded and then discarded. We want to move data as soon as possible (to avoid buffering too many bricks in memory), which can be achieved by a depth-first traversal of the set of bricks on all levels. We therefore visit bricks following a Z index created by interleaving the bits of each sample's spatial coordinates. Each Z index consists of a suffix (of length $\log_2(B_x B_y B_z)$ bits) that corresponds to a local coefficient index within the brick, and a prefix (the rest of the bits) to indicate the index of the brick itself. The interleaving pattern for Z is determined by first fixing a pattern for the suffix, and then repeating it as many time as possible for the rest of Z. Intuitively, bricks at level $l-1$ are formed by treating each level-l brick as a single sample. This strategy ensures a 2D brick stays on the same 2D slice in the next coarser level, for example.

Achieving smooth brick boundary. With our choice of wavelets (CDF 5/3), there is also a data dependency between neighboring bricks, which, if not handled, may lead to discontinuity artifacts at reconstruction time. To largely reduce this discontinuity, we use *lifting-based linear extrapolation* [12] to extrapolate each brick with one extra sample in each dimension. We could have used halo exchange to exchange the boundary values among neighboring bricks; however, we have chosen to extrapolate so that the bricks can be encoded (and decoded) independently of one another. In particular, the w-lift step extrapolates the even-length array $[\cdots a, b]$ to $[\cdots a, b, 2b-a]$ so that the last wavelet coefficient (in place of b) becomes zero. Unlike a simple linear extrapolation, which is done only once for the finest level, our approach intersperses the linear extrapolation steps with the lifting steps across hierarchy levels and across spatial dimensions. Such an extrapolation ensures zero-valued wavelet coefficients in all dimensions at the boundary during the forward transform, resulting in good compression. To support perfect reconstruction, however, we pay the extra cost of storing the (compressed) extrapolated samples, also in chunks (assuming a brick size of 64^3, the uncompressed extrapolated samples account for 5% of the data). Fig. 15 demonstrates that this extrapolation is

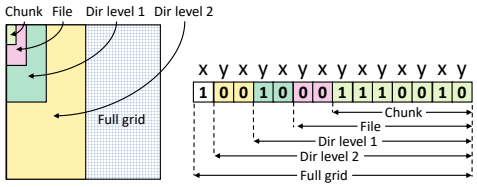


Fig. 10: Portions of the interleaved brick index are assigned to chunks, files, and directories. In this toy example, *spatially*, every 2^7 bricks form a chunk, every 2^2 chunks form a file, every 2^2 files form one directory, etc.

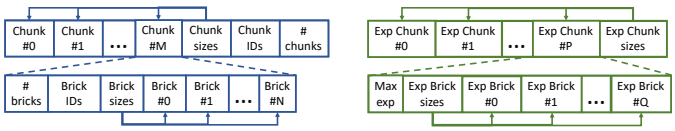


Fig. 11: File and chunk organization for compressed bit plane data (left), and for exponent data (right). Top is how chunks are stored in a file, and bottom is how a chunk is organized internally.

key to avoiding blocking artifacts. Note that the same technique can also extrapolate bricks with dimensions less than $B_x \times B_y \times B_z$ (those at the domain boundary) to $(B_x+1) \times (B_y+1) \times (B_z+1)$, provided that $B_{x,y,z}$ are powers of two, which they are in our system.

Achieving tight error bounds. Given an (absolute) error tolerance, we encode only enough bit planes to conservatively ensure the reconstruction error is within the tolerance. Beside ZFP's transform and quantization, the wavelet transform introduces additional range expansion that influences error and needs to be compensated for. We estimate range expansion at different levels using the infinity norms of the Kronecker products of 1D multiresolution wavelet synthesis matrices. As the first 10 norms (for the finest ten levels of resolution) are all less than 64, we conservatively encode/decode 6 additional bit planes on top of the number dictated by the input tolerance.

During block decoding, if the tolerance is τ and the block exponent is e, the smallest bit plane number to decode is $b-6$, with b being the largest integer such that $2^{b+e} \leq \tau$. In practice, the actual absolute error tends to be smaller than the input tolerance due to data smoothness, so more bit planes than needed are often decoded. Nevertheless, the required error tolerance is always respected, up to machine epsilon with respect to the range. Finally, we multiply wavelet coefficients with the norms of corresponding wavelet basis functions, ensuring equal energy contributions from the same bit plane on all levels.

Enabling fast lookup. Given a data query that contains an ROI, a resolution level, and an error tolerance, we need to quickly locate the relevant files, chunks, and bricks for decoding. At the API level, we do not make use of *cuts*, due to the complexity of providing such an API and because a cut can always be specified using a number of such queries. Since the topology of T_r^p is implicit (index-based), we can easily ensure all such queries produce valid cuts, as the only constraint to be satisfied is that whenever a node in T_r^p is retrieved, all of its ancestors (i.e., coarser level nodes and more significant bit planes) — identified by indexing — are also retrieved. In practice, given such a query, we locate the files containing the corresponding bits using a two-stage lookup. First, file lookup in the precision-resolution space is handled by defining a function that maps a point in that space to a file ID and a file path, which we define using bit packing (for file ID) and string concatenation (for file path). Given the requested nodes, we iterate over the relevant levels and bit planes, and apply this function each time to obtain the directories containing the relevant files.

Once inside such a directory, we perform spatial lookup to locate the exact files/chunks/bricks that intersect the requested ROI, relying on a single indexing scheme that works across brick, chunk, file, and directory levels. Starting from the least significant bit of a brick's interleaved index, portions of this index are assigned one-by-one to chunks, files, and directories (Fig. 10). This indexing scheme defines an implicit tree over the space of directories, files, and chunks, which enables a depth-first lookup algorithm that computes a list of relevant directories, files, and chunks in logarithmic time, skipping irrelevant

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

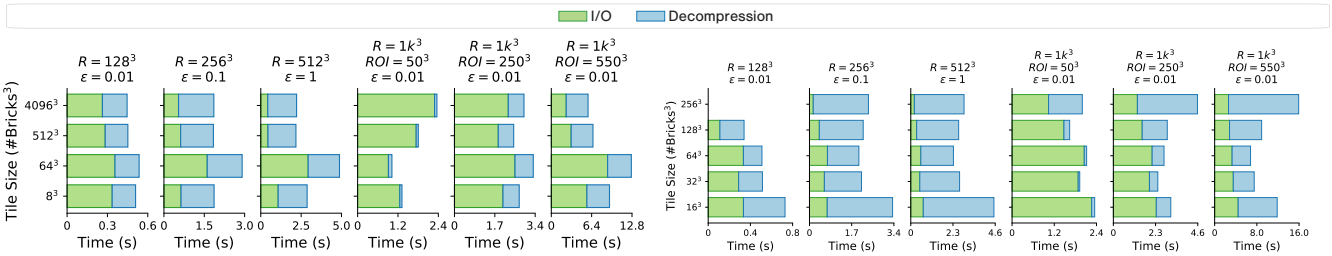


Fig. 12: I/O and decompression times for different tile sizes (left) and brick sizes (right) across a wide range of query types. The resolution and precision are controlled by the R and ε (absolute error) parameters, whereas ROI controls the size of the region of interest (which, when omitted, means the same as R). Tile size of 8^3 bricks and brick size of 64^3 samples appear to be sweet spots that work well across queries.

files entirely. Similar schemes have been used previously, e.g., in the context of sorting objects in a bounding volume hierarchy [48, 57]. Whenever a relevant file is found, we continue the traversal in the same way to find relevant chunks in the sub-tree under the file. We then fetch these chunks using the offsets stored in the file. Once a chunk is fetched, relevant bricks are located using the metadata stored in the chunk. Although the whole chunk is fetched, we decompress only the relevant bricks in the chunk to retrieve the requested bits.

Providing random access. Random access to compressed brick data is supported by storing in each chunk the number of bricks followed by the ID (interleaved brick index) and the size (in bytes) of every brick, encoded using base-128 compression [23]. Brick IDs are compressed by unary coding the differences between consecutive IDs, which tend to be 1. Likewise, to support random access of chunks, we store the (dictionary compressed) chunk IDs, chunk sizes, and the total number of chunks (Fig. 11, left) in each file. In practice, parameters are chosen such that a file can contain at most tens of thousand of chunks, so that such a flat indexing scheme still works well. The exponent data is also stored into chunks in a way that mirrors the bit plane chunks, but in a separate set of files (Fig. 11, right). A notable difference is that we also store the maximum exponent per chunk, which allows us to skip decoding an entire chunk if the maximum exponent is so small that the error tolerance is achieved even with no bit planes decoded.

Decoding progressively. Progressive decoding is the ability to resume decoding as new data arrives without redoing previous work. This capability is useful when a bit stream must be decoded as it is being transferred to show gradually improving results to the user. Supporting this feature often means keeping the state of the decoder in memory, which may not be practical depending on the size of this state. In our case, coefficients in a block always follow the same order and have the same precision, so the state for each block consists of the number of significant coefficients (an integer between 0 and 64), the current bit plane (an integer between 0 and 63), and the block exponent (11 bits for `float64`). Thus, the complete state needed for ZFP to resume decoding can be stored in 24 bits per block of 64 samples in 3D.

At decoding time, the steps performed for encoding are now done in reverse. In particular, bits are fetched from the streams, decoded, and "deposited" into a set of bricks, which start empty. Since not all bricks are present (depending on the query's ROI), we loosely maintain and update the current approximation in memory using a hashtable of bricks of wavelet coefficients, where the key consists of a brick's interleaved index and the brick's resolution level (including the subband). Each brick stores a grid of wavelet coefficients at a certain resolution level and subband. During decoding, bricks are updated with newly decoded coefficient bits, and the field is reconstructed through per-brick inverse wavelet transform, with missing coefficients assuming the value of 0. To keep a minimal state, we can also do away with the hashtable, and perform the inverse wavelet transform while having no access to previously decoded wavelet coefficients as they are presumably deallocated. Since the transform is linear, however, we can simply dequantize the (integral) increments obtained by decoding the current bit plane, inverse transform the floating-point increments, and update the field with the results.

Data set	Resolution × Precision (X × Y × Z) × Bits	Range [min, max]	Compression Ratio	Compression Speed (MB/s)
Pressure [49]	$(10240\times7680\times1536)\times64$	$[-0.23, 1.26]$	1.36×	6.5
Dissipation [38]	$(4096\times4096\times4096)\times32$	$[0.00, 82.67]$	1.41×	11.2
Dark matter [4]	$(2048\times2048\times2048)\times32$	$[0.00, 486.31]$	1.16×	14.2
Temperature [83]	$(2025\times1600\times400)\times64$	$[4.4819.24]$	1.95×	19.3
Mixed fraction [11]	$(920\times1400\times720)\times32$	$[0.00, 1.00]$	11.36×	26.0
Density [21]	$(1024\times1024\times1024)\times64$	$[1.00, 3.00]$	2.11×	35.9

Table 1: Tabulation of the data sets used for evaluation, with compression ratios and compression speeds when absolute tolerances are 5×10^{-8} for `float32` and 10^{-16} for `float64`.

6 Evaluation

We evaluate the efficacy of the proposed system using the data sets in Table 1, which features various types of scientific simulations. The test computer is a laptop with a 4-core CPU (2.8 GHz Intel Core i7-7700HQ), 32 GB of RAM and (unless otherwise specified) a 122 MB/s spinning hard drive. Note that we always use only one CPU core in all tests. Throughout this section, the term "tolerance" implies precision levels (which corresponds to the RMSE in the ideal case); a high tolerance corresponds to a low-precision level and vice versa.

In Table 1, we show the encoding times and near-lossless compression ratios for all the data sets used in this paper. Our encoding speed reduces linearly from 35 MB/s (for Density at 4 GB) to 6.5 MB/s (for Pressure at 900 GB), likely due to effects of the disk cache and overheads associated with updating bookkeeping data structures that grow linearly in the size of the data. For memory, we note that the largest data set, Pressure (900 GB), is encoded using only 2.5 GB of RAM. For near-lossless encoding (where the tolerance is set to about the machine epsilon relative to data range), we achieve varying degrees of reductions, but all are reduced to less than the original size. For all tested data sets, the overhead of metadata is on the order of $1/1000$ relative to the compressed data. Our system also supports also lossless compression of `float32` fields, but we note that in practice the last few bit planes are effectively noise and hence are expensive to compress while adding little to no value.

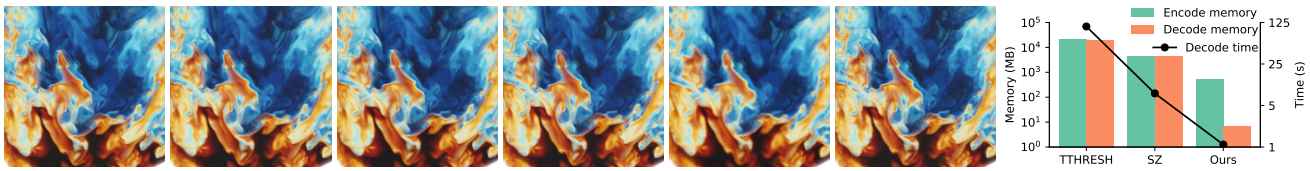
6.1 System Parameters

To find optimal values for the free parameters we execute concrete instances of the various query types on the different data sets. We have found that grouping chunks of different bit planes and intra-brick subbands in the same file reduces query time across the board (compared to separating them), likely due to disk prefetching and reduced seek time. Chunks belonging to different spatial file regions (Fig. 10) are still appropriately stored in separate files.

Tile size. To determine a tile size (i.e., T_G, the spatial extent of a chunk) that supports fast I/O across queries, we fix the brick size to 64^3 samples, vary the tile size from 2^3 to 16^3 bricks, execute different types of queries, and record the I/O as well as decompression time; the latter involves chunk/brick lookup, low-level decompression, inverse transforms and any in-memory data movement/transformation. The results can be seen in Fig. 12 (left). Larger tiles (hence larger chunks) tend to result in significantly reduced I/O time, especially at high-resolution levels; in our experiments the tile size of 16^3 bricks reports the lowest I/O time, except for the smaller ROIs of sizes 250^3 and 50^3. We choose the tile size of 8^3 bricks for the system as this size works almost as well and is better for small-ROI queries. With this choice, $T_G = 512^3$ since the brick size is 64^3. Note that since a chunk is only

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics



(a) Reference – 2 GB (b) SZ 6.5 MB, 0.976 SSIM (c) Ours 6.4 MB, 0.965 SSIM (d) TTHRESH 6.4 MB, 0.982 SSIM (e) Ours 800KB, 0.910 SSIM (f) TTHRESH 857 KB, 0.968 SSIM (g) Memory usage and decoding time, both in logarithmic scale.

Fig. 13: (Half of) Density — Comparison of data quality between (b) SZ, (c) ours and (d) TTHRESH at $300\times$ compression ratio. (e, f) Same comparison but at $2600\times$ ratio, (SZ did not produce a result here). (g) Plots of decode time and memory usage for the three methods. Our method uses orders of magnitudes less time and memory for decoding compared to SZ and TTHRESH.

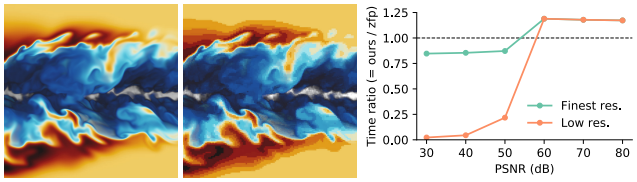


Fig. 14: Temperature — Our method (left, 2MB) supports very high compression ratios $> 1000\times$ where ZFP (middle, 12 MB) struggles, while maintaining the same decoding performance (right plot).

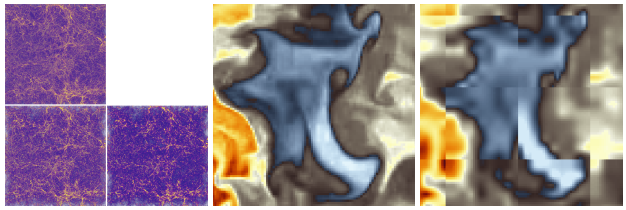


Fig. 15: (Left) Dark matter — Comparison of data quality between our system (bottom left, 109 MB, SSIM 0.78) and VAPOR (bottom right, 96 MB, SSIM 0.69) (Middle and right) Density — Ours (middle, 10 MB) is free from blocking artifacts visible with VAPOR (right, 13 MB).

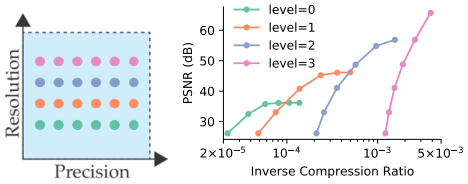


Fig. 16: Rate-distortion curves going through fixed-resolution points in precision-resolution space.

an I/O unit, the chunk size (controlled by T_G) as expected has almost no effects on the decompression time.

Brick size. We next vary the brick sizes from 16^3 to 128^3 samples, while keeping the tile size constant in number of samples (computed using the choices of 64^3-samples per brick and 8^3-bricks per tile). The results are shown in Fig. 12 (right), in which a few trends can be observed. First, I/O time is lower with larger brick sizes, likely due to less metadata for bookkeeping of bricks in each chunk, resulting in smaller chunks on average. Decompression time increases for bricks that are either too small or too large. Small bricks are faster to decompress themselves but also require reading and decoding a large amount of bookkeeping metadata. Using very small bricks helps only the tiny 50^3-ROI query. Based on these observations we choose our brick size to be 64^3 (i.e., $B_x = B_y = B_z = 64$), which works well across the queries for both I/O and decompression. Subsequent experiments assume the brick size of 64^3 samples and the tile size of 8^3 bricks.

6.2 Compression Comparisons

We compare our method against the state-of-the-art techniques, namely SZ [75], TTHRESH [5], JPEG2000 [76] (using OpenJPEG [2]), ZFP, and VAPOR [51]. Compared to SZ and TTHRESH, our decompression time and memory usage are orders of magnitude lower. Our data quality is competitive against both at $\approx 300\times$ compression ratio and is only slighty worse than TTHRESH at very high ratios (Fig. 13). Note that our results are decoded from a single data layout, whereas TTHRESH and SZ have to re-compress each time. Similarly, at comparable data sizes, JPEG2000 has slightly better data quality (39.0 dB versus 43.9 dB) at the expense of $2000\times$ higher memory usage and $15\times$ slower decompression time. Using a series of Q_{mixed} queries with increasingly lower tolerances, in Fig. 14 we show that our method achieves ZFP's decoding speed while enabling very high compression ratios. Furthermore, for mid- and low-quality levels (PSNR < 50 dB), our system can decode at lower resolutions, thus achieving significantly lower decoding time. Compared to VAPOR, our method achieves substantially better quality at $300\times$ compression ratio, while retrieving the data using one-fourth the memory and one-third the time, as well as avoiding blocking artifacts at very low bit rates (Fig. 15).

Reconstruction quality. Next, we study data quality for 16 approximations at the vertices of a 24-point grid in the precision-resolution space. We plot the PSNR at each point, and connects point that lie on the same resolution (Fig. 16). The rate-distortion curves suggest that the data quality at low-resolution levels quickly reaches a plateau (which is expected since the low number of data points, no matter how accurate, puts a hard limit on data quality measured in PSNR). The best rate-distortion curve can be thought of as an imaginary "envelope" that is the upper bound of all four individual curves.

7 Conclusion and Future Work

We modeled the full space of data reductions with a novel tree T_r^p that unifies precision and resolution, and accepts approximations as sub-trees produced by valid cuts of T_r^p. The parameters and trade-offs needed for practical data layouts encoding the tree are provided. Finally, we provided an empirical study on the system-level considerations leading to an efficient design, which we showed to be competitive with the state of the art. Our T_r^p captures the essential dependencies among nodes, but not all dependencies. For example, with filters such as wavelets, there exist data dependencies among neighboring nodes, required for inverse filtering (we handled this in our implementation). In addition, how cuts can be economically represented in memory is an open question. For future work, we would like to determine if and how optimal mixtures of reductions can be automatically found for scientific tasks with different notions of errors and costs. Extending the current framework to different grid types, unstructured data, and time-varying data is yet another direction. Finally, studying how applications such as interactive GPU rendering can benefit from a data layout such as the one proposed is important.

Acknowledgments

We thank the reviewers for their valuable feedback. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 17-SI-004 and Project No. 20-SI-001. This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. This research is supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work is supported in part by NSF DMS award 1664848, NSF IIS award 1657020, NSF OAC award 1842042, NSF OAC award 1941085, and NSF CMMI award 1629660. Release number LLNL-JRNL-813759.

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

REFERENCES

[1] IDX2: A compressed file format for scientific data. https://github.com/sci-visus/IDX2, last accessed on 09/07/20.

[2] OPENJPEG: An open-source JPEG 2000 codec written in C. http://www.openjpeg.org, last accessed on 08/15/20.

[3] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science*, 19:65–76, 2018.

[4] A. Almgren, J. Bell, M. Lijewski, Z. Lukić, and E. Van Andel. Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal*, 765:39, 2013.

[5] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. TTHRESH: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, 29:2891–2903, 2019.

[6] R. Ballester-Ripoll and R. Pajarola. Lossy volume compression using Tucker truncation and thresholding. *The Visual Computer*, 32(11):1433–1446, 2016.

[7] R. Ballester-Ripoll, S. K. Suter, and R. Pajarola. Analysis of tensor approximation for compression-domain volume visualization. *Computers & Graphics*, 47(C):34–47, 2015.

[8] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control (SIGFIDET '70)*, p. 107–141, 1970.

[9] S. Belkasim. Multi-resolution analysis using symmetrized odd and even DCT transforms. In *IEEE Data Compression Conference (DCC '11)*, pp. 447–447, 2011.

[10] J. Beyer, M. Hadwiger, and H. Pfister. State-of-the-art in GPU-based large-scale volume visualization. *Computer Graphics Forum*, 34(8):13–37, 2015.

[11] A. Bhagatwala, Z. Luo, H. Shen, J. A. Sutton, T. Lu, and J. H. Chen. Numerical and experimental investigation of turbulent DME jet flames. *Proceedings of the Combustion Institute*, 35(2):1157–1166, 2015.

[12] H. Bhatia, D. Hoang, G. Morrison, W. Usher, V. Pascucci, P.-T. Bremer, and P. Lindstrom. AMM: Adaptive Multilinear Meshes. *arXiv e-prints*, p. arXiv:2007.15219, July 2020.

[13] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

[14] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi, C. Chiang, Y. Wang, P. Wilkins, J. Bankoski, L. Trudeau, N. Egge, J. Valin, T. Davies, S. Midtskogen, A. Norkin, and P. de Rivaz. An overview of core coding tools in the AV1 video codec. In *Picture Coding Symposium (PCS '18)*, pp. 41–45, 2018.

[15] Y. Cho and W. A. Pearlman. Hierarchical dynamic range coding of wavelet subbands for fast and efficient image decompression. *IEEE Transactions on Image Processing*, 16(8):2005–2015, 2007.

[16] C. Chrysafis, A. Said, A. Drukarev, A. Islam, and W. A. Pearlman. SBHP-a low complexity wavelet coder. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 2035–2038, 2000.

[17] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Batched multi triangulation. In *IEEE Visualization Conference (VIS '05)*, pp. 207–214, 2005.

[18] J. Clyne. The Multiresolution Toolkit: Progressive access for regular gridded data. In *IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP '03)*, pp. 152–157, 2003.

[19] J. Clyne, P. Mininni, A. Norton, and M. Rast. Interactive desktop analysis of high resolution simulations: Application to turbulent plume dynamics and current sheet formation. *New Journal of Physics*, 9(8):301, 2007.

[20] A. Cohen, I. Daubechies, and J. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992.

[21] A. W. Cook, W. Cabot, and P. L. Miller. The mixing transition in Rayleigh-Taylor instability. *Journal of Fluid Mechanics*, 511:333–362, 2004.

[22] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '09)*, pp. 15–22, 2009.

[23] D. Cutting and J. Pedersen. Optimization for dynamic inverted index maintenance. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '90)*, p. 405–411, 1989.

[24] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, 4(3):245–267, 1998.

[25] L. De Fioriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *IEEE Visualization Conference (VIS '97)*, pp. 103–110, 1997.

[26] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS '16)*, pp. 730–739, 2016.

[27] J. Díaz, F. Marton, and E. Gobbetti. Interactive spatio-temporal exploration of massive time-varying rectilinear scalar volumes based on a variable bit-rate sparse representation over learned dictionaries. *Computers & Graphics*, 88:45–56, 2020.

[28] C. Egho and T. Vladimirova. Adaptive hyperspectral image compression using the KLT and integer KLT algorithms. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS '14)*, pp. 112–119, 2014.

[29] T. Fogal, H. Childs, S. Shankar, J. Krüger, R. D. Bergeron, and P. Hatcher. Large data visualization on distributed memory multi-GPU clusters. In *ACM Conference on High-Performance Graphics (HPG '10)*, pp. 57–66, 2010.

[30] T. Fogal, A. Schiewe, and J. Krüger. An analysis of scalable GPU-based ray-guided volume rendering. In *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV '13)*, pp. 43–51, 2013.

[31] N. Fout and K. L. Ma. Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1600–1607, 2007.

[32] E. Gobbetti, J. A. 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*, 31(34):1315–1324, 2012.

[33] E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7):797–806, 2008.

[34] Z. Gong, T. Rogers, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova. MLOC: Multi-level layout optimization framework for compressed scientific data exploration with heterogeneous access patterns. In *International Conference on Parallel Processing (ICPP '12)*, pp. 239–248, 2012.

[35] S. Guthe and W. Strasser. Advanced techniques for high-quality multi-resolution volume rendering. *Computers & Graphics*, 28(1):51–58, 2004.

[36] S. Guthe, M. Wand, J. Gonser, and W. Strasser. Interactive rendering of large volume data sets. In *IEEE Visualization Conference (VIS '02)*, pp. 53–60, 2002.

[37] M. Hadwiger, J. Beyer, W. K. Jeong, and H. Pfister. Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2285–2294, 2012.

[38] C. Harrison, H. Childs, and K. P. Gaither. Data-parallel mesh connected components labeling and analysis. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '11)*, p. 131–140, 2011.

[39] D. Hoang, P. Klacansky, H. Bhatia, P. Bremer, P. Lindstrom, and V. Pascucci. A study of the trade-off between reducing precision and reducing Resolution for data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1193–1203, 2019.

[40] I. Ihm and S. Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18(1):3–15, 2003.

[41] J. Iverson, C. Kamath, and G. Karypis. Fast and effective lossy compression algorithms for scientific datasets. In *International Conference on Parallel Processing (ICPP '12)*, pp. 843–856, 2012.

[42] S. Kumar, C. Christensen, J. A. Schmidt, P. Bremer, E. Brugger, V. Vishwanath, P. H. Carns, H. Kolla, R. W. Grout, J. Chen, M. Berzins, G. Scorzelli, and V. Pascucci. Fast multiresolution reads of massive simulation datasets. In *International Supercomputing Conference (ISC '14)*, pp. 314–330, 2014.

[43] S. Kumar, J. Edwards, P. Bremer, A. Knoll, C. Christensen, V. Vishwanath, P. Carns, J. A. Schmidt, and V. Pascucci. Efficient I/O and storage of adaptive-resolution data. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*, pp. 413–423, 2014.

[44] S. Kumar, D. Hoang, S. Petruzza, J. Edwards, and V. Pascucci. Reducing network congestion and synchronization overhead during aggregation of hierarchical data. In *IEEE International Conference on High Performance Computing (HiPC '17)*, pp. 223–232, 2017.

[45] S. Kumar, V. Vishwanath, P. Carns, J. A. Levine, R. Latham, G. Scorzelli, H. Kolla, R. Grout, R. Ross, M. E. Papka, J. Chen, and V. Pascucci. Efficient data restructuring and aggregation for I/O acceleration in PIDX.

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2020.3030381, IEEE Transactions on Visualization and Computer Graphics

In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*, pp. 1–11, 2012.

[46] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout. PIDX: Efficient parallel I/O for multi-resolution multi-dimensional scientific datasets. In *IEEE International Conference on Cluster Computing (CLUSTER '11)*, pp. 103–111, 2011.

[47] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization Conference (VIS '99)*, pp. 355–361, 1999.

[48] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. *Computer Graphics Forum*, 28(2):375–384, 2009.

[49] M. Lee and R. D. Moser. Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. *Journal of Fluid Mechanics*, 774:395–415, 2015.

[50] S. Lefebvre and H. Hoppe. Compressed random-access trees for spatially coherent data. In *Eurographics Conference on Rendering Techniques (EGSR'07)*, p. 339–349, 2007.

[51] S. Li, S. Jaroszynski, S. Pearse, L. Orf, and J. Clyne. VAPOR: A visualization package tailored to analyze simulation data in earth system science. *Atmosphere*, 10(9), 2019.

[52] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[53] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[54] F. Marton, M. Agus, and E. Gobbetti. A framework for gpu-accelerated exploration of massive time-varying rectilinear scalar volumes. *Computer Graphics Forum*, 38(3):53–66, 2019.

[55] K. Museth. VDB: High-resolution sparse volumes with dynamic topology. *ACM Transaction on Graphics*, 32(3):1–22, 2013.

[56] K. G. Nguyen and D. Saupe. Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20(3):49–57, 2002.

[57] J. Pantaleoni and D. Luebke. HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *ACM Conference on High Performance Graphics (HPG '10)*, p. 87–95, 2010.

[58] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC '01)*, pp. 45–45, 2001.

[59] V. Pascucci, D. E. Laney, R. J. Frank, G. Scorzelli, L. Linsen, B. Hamann, and F. Gygi. Real-time monitoring of large scientific simulations. In *ACM Symposium on Applied Computing (SAC '03)*, p. 194–198, 2003.

[60] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said. Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(11):1219–1235, 2004.

[61] S. Prohaska, A. Hutanu, R. Kahler, and H. C. Hege. Interactive exploration of large remote micro-CT scans. In *IEEE Visualization Conference (VIS '04)*, pp. 345–352, 2004.

[62] J. Pulido, D. Livescu, K. Kanov, R. Burns, C. Canada, J. Ahrens, and B. Hamann. Remote visual analysis of large turbulence databases at multiple scales. *Journal of Parallel and Distributed Computing*, 120:115 – 126, 2018.

[63] I. Ram, M. Elad, and I. Cohen. Generalized tree-based wavelet transform. *IEEE Transactions on Signal Processing*, 59(9):4199–4209, 2011.

[64] F. Reichl, M. Treib, and R. Westermann. Visualization of big SPH simulations via compressed octree grids. In *IEEE International Conference on Big Data (BigData '13)*, pp. 71–78, 2013.

[65] S. Rizzi, M. Hereld, J. Insley, M. E. Papka, T. Uram, and V. Vishwanath. Large-Scale Parallel Visualization of Particle-Based Simulations using Point Sprites and Level-Of-Detail. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '15)*, p. 1–10, 2015.

[66] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, 1996.

[67] H. Samet and A. Kochut. Octree approximation and compression methods. In *IEEE International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT '02)*, pp. 460–469, 2002.

[68] J. Schneider and R. Westermann. Compression domain volume rendering. In *IEEE Visualization Conference (VIS '03)*, pp. 293–300, 2003.

[69] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[70] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.

[71] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[72] B. Summa, G. Scorzelli, M. Jiang, P.-T. Bremer, and V. Pascucci. Interactive editing of massive imagery made simple: Turning Atlanta into Atlantis. *ACM Transactions on Graphics*, 30(2):1–13, 2011.

[73] S. K. Suter, J. A. I. Guitian, F. Marton, M. Agus, A. Elsener, C. P. E. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola. Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143, 2011.

[74] S. K. Suter, M. Makhynia, and R. Pajarola. TAMRESH – tensor approximation multiresolution hierarchy for interactive volume visualization. *Computer Graphics Forum*, 32(3pt2):151–160, 2013.

[75] D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS '17)*, pp. 1129–1139, 2017.

[76] D. S. Taubman and M. W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Springer, 2001.

[77] M. Treib, K. Bürger, F. Reichl, C. Meneveau, A. Szalay, and R. Westermann. Turbulence visualization at the terascale on desktop PCs. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2169–2177, 2012.

[78] P. van Oosterom and T. Vijlbrief. The spatial location code. In *International Symposium on Spatial Data Handling (SDH '96)*, pp. 1–17, 1996.

[79] G. K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):18–34, 1992.

[80] R. Westermann. A multiresolution framework for volume rendering. In *Symposium on Volume Visualization (VV '94)*, pp. 51–58, 1994.

[81] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. *Computer Graphics Forum*, 30(3):1151–1160, 2011.

[82] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens. Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV '11)*, pp. 31–38, 2011.

[83] C. S. Yoo, E. S. Richardson, R. Sankaran, and J. H. Chen. A DNS study on the stabilization mechanism of a turbulent lifted ethylene jet flame in highly-heated coflow. *Proceedings of the Combustion Institute*, 33(1):1619–1627, 2011.

1077-2626 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: The University of Utah. Downloaded on November 27,2020 at 19:18:31 UTC from IEEE Xplore. Restrictions apply.