Data Reduction Techniques for Scientific Visualization and Data Analysis

Samuel Li[1]

[1]Department of Computer and Information Science, University of Oregon

1. Introduction

The classic paradigm for scientific visualization and data analysis is *post-hoc*, where simulation codes write results on the file system and visualization routines read them back to operate. This paradigm sees file I/O as an increasing bottleneck, in terms of both transfer rates and storage capacity. As a result, simulation scientists, as data producers, often sample available time slices to save on disk. This sampling process is not risk-free, since the discarded time slices might contain interesting information.

Visualization and data analysis scientists, as data consumers, see reasonable turnaround times of processing data a precondition to successfully perform tasks with exploratory natures. The turnaround time is often spent on both reading data from disk, and computations that produce meaningful visualizations and statistics.

Data reduction is a means to tackle the above-mentioned difficulties both simulation and visualization scientists face. For simulation scientists, data in reduced form requires less transfer time and storage space on disk, giving more freedom to utilizing available hardware resources. For visualization scientists, reading in less bytes also means a shorter turnaround time, giving a better interactivity.

Besides relieving burdens of I/O, a few data reduction techniques also reduce the memory footprints at the same time for visualization computations. This is achieved by either lowering the grid resolution, or simplifying a given mesh. The benefit of reduced memory footprint is that computations proceed even faster, greatly improving the turnaround time for data explorations.

Data reduction comes at a cost as well. The most obvious cost is a computational overhead that consumes CPU cycles. If computational overhead becomes too high, it can potentially diminish the benefit of a faster I/O. Some data reduction techniques, namely the lossy ones, even hurt the data integrity. These techniques aims to recover the data into a close approximation, and it is up to the scientists to decide if the lost integrity is acceptable. Overall, adopting data reduction into a work flow requires finding a middle ground between data size, processing time, and integrity, and where this middle ground lies is very task-dependent.

This paper surveys techniques to achieve data reduction for simulation, visualization, and data analysis. We keep a few different usage scenarios in mind when doing this survey. These scenarios are distinguished by three important properties of available data reduction techniques; they serve as three dimensions to guide a scientist to dive into a collection of techniques that best meet her requirements. Note that a single technique does not necessarily keep staying in one scenario given that many techniques can operate in different modes. We identify these three dimensions as follows, and elaborate the ideas behind them in Subsection 1.1.

1. *lossy* or *lossless* data reduction;
2. *reduce on write* or *reduce on read*; and
3. resulting in *original memory footprint* or *reduced memory footprint*.

1.1. Use Case Dimensions

Lossy or lossless: This dimension is concerned with potential information loss after applying data reduction. With lossy reduction, the reduced form can never recover data identical to the original one. Lossy techniques aim to find a good balance between information loss and data reduction. With lossless reduction, the exact information from the original data is preserved. Lossless techniques aim to eliminate any redundancies in the data.

The choice between lossy and lossless reduction depends on the requirements of the analysis routines. On the one hand, analyses that require the best available precision to operate on have to choose lossless compression. On the other hand, many analyses, especially scientific visualization routines, can tolerate some degree of information loss. This tolerance can result in great rewards, for example, certain visual analytics tasks can still carry on with 256 : 1 lossy wavelet compression [LGP*15], while one of the state-of-the-art lossless coders can achieve up to 3.7 : 1 compression on another scientific data set [LI06]. That is one order of magnitude difference. Moreover, data reduction techniques that are tailored toward specific visualization or analysis tasks can often smartly discard data that is deemed not as useful, thus achieving even higher reduction rates. For example, Cinema [AJO*14] achieves orders of magnitude reduction for visual exploration tasks.

Reduce on write or reduce on read: This dimension is concerned with the stage where data reduction is performed: when the simulation code is writing results, or when the visualization is reading data from storage. Reduce on write refers to operators that create a representation of data with reduced size, and write this new representation to the permanent storage. Reduce on read is less intuitive: it refers to the case where a portion (as opposed to the entirety) of the data on disk is read out for analysis. Simple techniques such as subsetting (Subsection 2.7) achieves reduction on read by only retrieving domains of interest, and more complicated techniques (e.g. multi-resolution in Subsection 2.2) are able to recover the entire domain with a portion of the data, often at a penalty of inferior quality. Reducing on write and reducing on read are not exclusive in theory: if a technique supports reduce on read, one can also just write that portion of the data to disks. In this survey, we focus on how a technique is mostly used in practice regarding this dimension.

The notion of reduce on read is important to interactive tasks such as data explorations, since reading less data means a shorter turnaround time. The level of reduce on read is usually adjustable, with the option to use the entire saved data to obtain the most accurate available recovery. As a result, reduce on read enables workflows like *progressive data access*.

Original memory footprint or reduced memory footprint: This dimension is concerned with the capability of a technique to reduce memory footprint in addition to reducing the size in storage. This dimension looks at data recovered from its reduced form, rather than what is stored. Using an example to explain this idea, a compressed image would normally *not* reduce memory footprint after it was decoded into RGB values to display. However, it could reduce memory footprint if it was shrunk to have smaller dimensions — smaller dimensions mean less RGB values to display. For scientific visualizations, reduced memory footprint can be achieved by simple techniques such as subsetting (Subsection 2.7) where only a subset of domains is loaded, or more complicated techniques such as mesh simplification (Subsection 2.6), where a similar but dramatically simplified mesh is used for further operations.

The benefit of reduced memory footprint is two-fold. Firstly, for data sets larger than the memory capacity, one can still fit them into memory for processing with reduced memory footprint. Secondly, even if a data set fits within memory in its original form, the reduced form takes less time to process. Often times, the amount of reduction in memory footprint is adjustable and the option to go to the most accurate recovery is available. As a result, reduction on memory footprint is favorable for interactive tasks as well.

1.2. Paper Organization

This survey paper will separate the descriptions of data reduction techniques and their use cases. Techniques descriptions are presented in Section 2, where all surveyed techniques are categorized into seven groups. This section describes how each technique achieves data reduction, and readers are expected to obtain a high-level understanding of their reduction mechanism. We also group use cases of data reduction, but based on the combinations of properties discussed in Subsection 1.1. This categorization scheme results in eight use cases in total. We chose this categorization scheme for accessibility, i.e., a scientist can focus on a use case of interest based on her requirements. Section 3 through Section 10 survey these eight use cases. Of course one technique can be used in multiple use cases, and thus appear in multiple sections. Finally, we conclude this survey in Section 11.

2. Techniques Overview

There are seven categories of data reduction techniques in this paper, and each subsection overviews one category. Some data reduction techniques work by taking inputs and producing outputs, so we refer to them as *compressors*. We also use the term *coders*, which are very similar to compressors, except that coders can also act as a component of another compressor. For example, an entropy coder can be used by itself, or as a component in a wavelet compressor.

2.1. General-Purpose Compressors

General-purpose compressors are used in almost every field of computer science and on almost all types of data. Also, because of the general-purposed nature of these compressors, they are often used together with other data reduction techniques to help further improve the reduction efficiency. We survey three families of such coders: entropy-based coders, dictionary-based coders, and vector quantization.

Entropy-based compressors work by encoding strings based on their information content, or *entropy*, to achieve the best coding efficiency. For example, Huffman coding [H*52] identifies the frequency of each symbol's occurrence, so it is able to use fewer bits to encode high-frequency symbols, and more bits for low-frequency symbols. The number of bits Huffman coding uses approaches the theoretical entropy of input when each symbol has a probability close to powers of $\frac{1}{2}$. Other prominent members in this family include arithmetic encoding [WNC87] and the very simple run-length coding. Arithmetic encoding could be seen as a generalization of Huffman encoding and is more efficient with arbitrary probability distributions. Run-length coding replaces consecutive occurrences of a symbol with a pair of symbols and its number of occurrences. Entropy-based compressors are often used together with other techniques since data going through certain treatments (transforms, predictions, for example) can become close to constant with very small fluctuations.

In terms of our three dimensions, entropy-based compressors are considered lossless, reduce on write, and resulting in the original memory footprint.

Dictionary-based compressors work by matching strings to a dictionary so only the matching information, and not the actual strings, is stored. As an example, to compress the current paragraph of text, a coder keeps the page and index of each word in a standard English dictionary. For general purpose compression, the dictionary is usually constructed for each compression task, and thus becomes a storage overhead. Smart ways to construct and store this dictionary can minimize this overhead. LZ77 [ZL77] and LZ78 [ZL78] are among the first dictionary-based coders. They have many variants that enjoy very wide uses, including DEFLATE [Deu96], LZW [Wel84], LZO [Obe05], LZ4 [Col11], and LZMA/LZMA2 [Pav99], to name a few. Among them DEFLATE and LZMA2 are the underlying algorithms of popular compression format `.gz` and `.xz`, respectively. Dictionary-based coders tend to have a faster throughput than entropy-based coders, and they also can be configured towards either a higher compression ratio or a faster throughput.

Dictionary-based compressors are considered lossless, reduce on write, and resulting in the original memory footprint, in terms of our three dimensions.

Quantization is a popular lossy compression approach that is

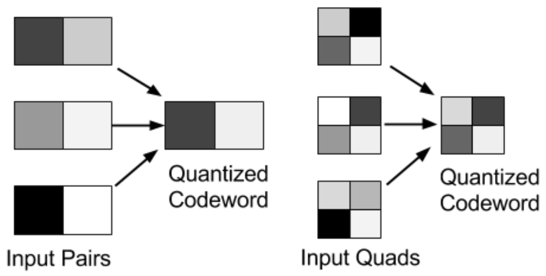


Figure 1: Vector quantization on vectors of size two (left) and size four (right).

applicable to any quantities, and also sees heavy use in the signal processing community. We describe two flavors of quantization here: scalar quantization and the more complicated, yet powerful, vector quantization.

Scalar quantization provides an n-to-m ($n \gg m$) mapping for scalar values, so all possible input values are represented by a finite set of discrete values. An example is to represent all floating-point values between 0.0 and 1.0 with a 256-step gray scale for image output. Using scalar quantization, all values between $\frac{i}{256}$ and $\frac{i+1}{256}$ are represented by the ith gray step.

Vector quantization generalizes scalar quantization by providing such n-to-m ($n \gg m$) mapping for vectors. It takes input vectors instead of single values, and maps them to a much smaller set of vectors as output. Figure 1 illustrates vector quantization of vectors in size two and four. These vectors could be a pair of adjacent pixels or a quad of pixels from an image. The resulting vectors are called *codewords*, and the n-to-m mapping scheme is called a *codebook*. In practice, both the size of the codebook and the length of the vector are important parameters to adjust.

Vector quantization becomes advantageous when the underlying data exhibits coherence, i.e., similarities between neighbor data points. Given such an input data set, the input vectors then distribute in the vector space following certain patterns. A good codebook then allocates more codewords to dense areas in the vector space, providing better ability to distinguish them apart. The process of finding a good codebook is very much like finding a voronoi-diagram-like partition for a space. Such successful algorithms include the LBG algorithm [LBG80] and Lloyd's algorithm [Llo82].

Quantization is considered lossy, reduce on write, and resulting in the original memory footprint in terms of our three dimensions. Also, [GG12] provides a good reference on vector quantization for signal compression.

2.2. Multi-resolution Approach

The multi-resolution approach builds a hierarchy of the data volume with different resolutions. The finest level is the native resolution of the original data, and every other level has a lower resolution in this hierarchy. Data size is reduced at the lower resolution levels, but usually not at the native resolution level.

Multi-resolution provides a means to achieve progressive data access by loading a small amount of data to reconstruct an approximation with lower resolutions first, and then keeps loading more data to reconstruct better approximations until reaching the full resolution. This property makes it suitable for data exploration tasks. This subsection surveys three families of multi-resolution approaches.

Sampling for multi-resolution is the simplest technique to achieve multi-resolution. It recursively samples the data volume, and each sample results in a new separate level with lower resolution. The fact that each resolution level is separate, however, introduces significant storage overhead going towards the native resolution. As a result, some applications opt to discard the native resolution level to actually reduce data size [WAF*11].

Sampling for multi-resolution is considered lossy, capable of both reduce on write and reduce on read, and resulting in reduced memory footprint.

Space-filling curves share the spirit of sampling for multi-resolution, but intelligently organize data such that multi-resolution is achieved without storage overhead: data points from the lower resolution levels always contribute to the reconstruction of higher resolution levels. Mathematically, space-filling curves are continuous that visit every data point in an n-dimensional space exactly once; they map an n-dimensional space to a one dimensional curve. This mapping has a unique property that a space-filling curve would finish visiting all data points in one resolution level before visiting the next level. Data locality is achieved by keeping the data points belonging to the same resolution level together. That said, commonly used approaches to store high dimensional arrays in memory, such as row-major and column-major storage, also map an nD space to a 1D representation, but they do not provide data locality. The most prominent space-filling curves are Z-curves [Mor66] and Hilbert curves [Hil91]. We use the former as an example to explain how space-filling curves work.

Z-curves traverse a data volume from the coarsest resolution level to finer levels in a Z-shaped pattern. It traverses a resolution level all at once and then go to a finer level. Storage wise, data points from the coarser levels also contribute to the finer levels, so no overhead is introduced even at the native resolution. Figure 2 illustrates how a Z-order curve traverse data points in a 16×16 matrix from the coarsest 1×1 resolution to an 8×8 resolution. Higher dimensional Z-order curves are also discussed in [PF01].

Space-filling curves are considered lossy, capable of both reduce on write and reduce on read, and resulting in reduced memory footprint.

Transform-based multi-resolution approaches transform data into frequency domain, and then use low frequency components to construct approximations with lower resolutions. This approach is technically sound because if we look at the input data as signals, the low frequency components represent a general trend while the high frequency components represent detailed deviations on that trend. There are three classic transforms in this family: cosine transform, Fourier transform, and wavelet transform. Compared to space-filling curves, transform-based techniques take into account similarities between neighbor data points, resulting in better approxima-

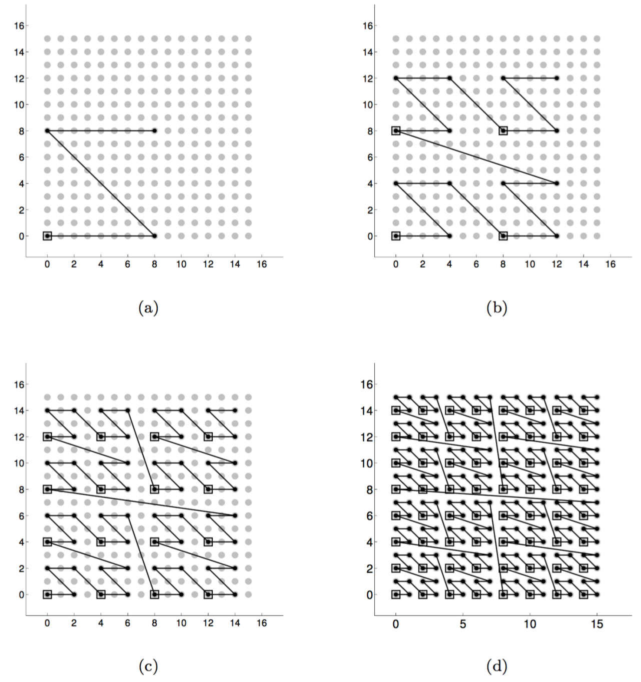


Figure 2: A Z-order curve traverses a 16×16 plane with four resolution levels. (a) shows the coarsest level in 2×2 dimension, and (b), (c), and (d) each shows a finer level, until all data points are visited in (d).

tions at lower resolution levels. We will use wavelet transforms as an example to explain transform-based multi-resolutions. Theories behind the cosine transform, Fourier transform, and wavelet transform are covered in detail by [RY14, Sne95, SN96] respectively.

Mathematically, wavelet transforms take in a signal and represent it using a linear combination of a set of basis functions. The basis functions have two flavors: scaling functions to capture the low frequency components, and wavelet functions to capture the high frequency components. As a result, wavelet coefficients from the scaling functions contain approximations and coefficients from the wavelet functions contain the detailed deviations. Wavelet transforms can be recursively applied on the approximation coefficients, resulting in a hierarchical representation of the input signal. In a 1D case with a discrete signal $x[n]$, this linear combination is thus represented as the following equation with J lower-resolution levels:

$$x[n] = \sum_i a_{0,i} \cdot \phi_{0,i}[n] + \sum_{j=0}^{J-1} \sum_i d_{j,i} \cdot \psi_{j,i}[n]. \tag{1}$$

Here $j = 0$ denotes the coarsest level and $j = J$ denotes the finest level with original resolution where $x[n]$ is at. Scaling and wavelet basis functions are denoted as $\phi_{j,i}[n]$ and $\psi_{j,i}[n]$, respectively. Their coefficients, $a_{j,i}$ and $d_{j,i}$, are approximation and detail coefficients accordingly.

Figure 3 illustrates a hierarchy after three passes of wavelet transform ($J = 3$). The approximation coefficients — $a_{2,i}$, $a_{1,i}$, and $a_{0,i}$ — are three different resolution levels. Note only $a_{0,i}$ is kept in

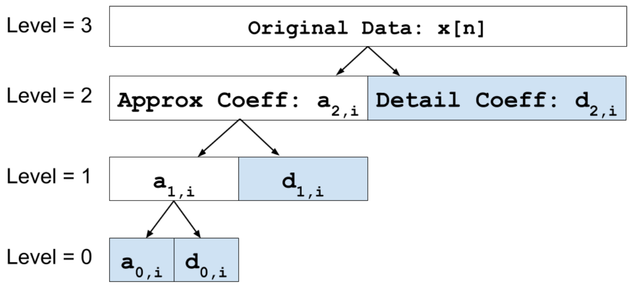


Figure 3: Hierarchical representation of wavelet coefficients after three passes of wavelet transform. Blue blocks are coefficients to be saved; in total they take the same amount of storage as the original data. Reconstruction goes from bottom to the top, and could stop at any intermediate level to get an approximation.

storage, and $a_{1,i}$ and $a_{2,i}$ are calculated on-the-fly with detail coefficients $d_{0,i}$ and $d_{1,i}$. Overall, the blue blocks in Figure 3 are saved to disk using the same amount of storage as the original data $x[n]$. In the case of higher dimensions, the same transform is applied in all dimensions, resulting in lower-resolution squares (2D) and cubes (3D) for example.

Transform-based multi-resolution is considered lossy, capable of both reduce on write and reduce on read, and resulting in reduced memory footprint.

Miscellaneous: We note that there are other techniques to achieve multi-resolution. Laplacian pyramid was used for direct rendering of computed tomography data, and the authors found it significantly faster than wavelet transforms [GY95]. Curvelets [CDDY06] and surfacelets [LD07] are newer multi-resolution technologies that attempt to capture specific features from the data set (edges for curvelets and signal singularities for surfacelets). However, both introduce storage overheads. Evaluation of these two techniques on turbulence data can be found at [PLW*16].

2.3. Predictive Coding Schemes

Predictive coding schemes work by using a predictor to predict the next values based on known ones. A good predictor results in small residuals, which is the difference between the predicted and real values. These residuals can be safely discarded if they are smaller than an error tolerance. Otherwise, they need to be coded efficiently. Entropy-based coders are popular with these residuals, since they usually contain significantly lower entropy than the original data. Predictive coding schemes surveyed here are mainly from two communities: video processing and scientific data compression. The former introduced motion compensated predictions, and the latter introduced a wide range of predictors and compressors. In this section, we will briefly describe motion compensated predictions, and survey those scientific data oriented predictors in more detail.

Motion compensated prediction, or MCP [Ana89], is specifically designed for video coding. MCP assumes picture changes

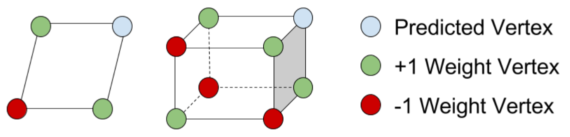


Figure 4: A Lorenzo predictor uses known values in a cube (green or red) to predict an unknown value (blue) in 2D and 3D cases. Green vertices have $+1$ weight and red vertices have -1 weight.

from frame to frame are mostly due to translations of objects in the scene or panning of the camera. Said another way, picture changes are due to the movement of several large groups of pixels. MCP first identifies pixels moving together, and then uses a single vector — a motion vector — to record the motion of this group. Therefore, motion vectors serve as the predictors and are encoded. The more pixels moving in groups, the more compression MCP achieves. Because of the nature of MCP, it is mostly used for time-varying data sets.

MCP is mostly used as a lossy, reduce on write operator with the original memory footprint.

Linear predictors predict the data points with constant values, or change in a linear fashion. This intuition has synonyms in particle simulations that particles will stay at the same locations, or travel along a straight line [EGM04]. To add some heuristic into linear predictors, one can pick out the smoothest changing dimension to make predictions in a multi-dimensional data set, as proposed in [ZTGB02].

Compvox [FY94] improves linear predictors by taking into consideration more neighbor values by using the following equation to predict a value at (x,y,z) in a 3D volume:

$$\tilde{v}(x,y,z) = a_1 v(x-1,y,z) + a_2 v(x,y-1,z) + a_3 v(x,y,z-1). \quad (2)$$

The linear coefficients (a_1, a_2, a_3) are constantly updated as the coder keeps processing data to improve prediction accuracy. The overall compression is lossless, Compvox achieves a 2:1 compression ratio on average with a few test data sets, which was better than both popular compression tools gzip and zip.

These linear predictors were proposed as lossless, reduce on write operators that result in the original memory footprint.

Lorenzo predictor [ILRS03] was proposed for large scientific data sets in arbitrary dimensions. This predictor works in units of n-dimensional cubes (squares in 2D, cubes in 3D, hypercubes in even higher dimensions): it uses the known values of the cube vertices to predict the only unknown value at a corner of the cube:

$$E(v) = \sum_{u \in C} (-1)^{d_{uv}+1} \cdot F(u) \quad (3)$$

Here the vertex v is predicted using values from other vertices u that belong to the same cube C. The degree between two vertices (number of edges connecting them) is denoted as d_{uv}, and this degree determines the weight of each neighbor to be either $+1$ or -1. Figure 4 illustrates the weights in 2D and 3D cases. According to the authors with proof, "the Lorenzo predictor is of highest possible order among all predictors that estimate the value of a scalar field at one corner of a cube from the values at the other corners."

Lorenzo predictor can be both lossy or lossless, achieves reduce on write, and results in the original memory footprint.

FPZIP [LI06] is a compressor for productions that employs the Lorenzo predictor. FPZIP optimizes coding for residuals (the difference between the predicted and actual values) by mapping the numbers to unsigned integer representations and applying arithmetic coding. Least significant bits can be discarded when mapping to integers to achieve lossy compression. The major advantage of FPZIP over the original Lorenzo predictor compressor in [ILRS03] was its compression speed: it is two times faster.

FPZIP can be both lossy or lossless, achieves reduce on write, and results in the original memory footprint.

ISABELA [LSE*11] uses a B-splines curve fitting approach. ISABELA is unique in that it sorts data points before going through the curve fitting step, which greatly improves prediction accuracy. Sorting requires the coder to keep the indexing information for decoding. Given that each single index requires $log_2 N$ bits to encode in an N sized array, the total storage requirement for indices is $N \times log_2 N$ bits. As a result, ISABELA is advantageous with smaller arrays. Thus, the authors proposed splitting the data into windows for efficient storage, more specifically, 1024 sized windows for this study.

ISABELA is mainly used as a lossy, reduce on write operator that results in the original memory footprint.

Combinations of predictors are also proposed. These compressors try a few predictors for each prediction, and choose the best one at that data point. **FPC** [BR09] uses two hash table based predictors, FCM [SS97] and DFCM [GVDB01]. **SZ** [DC15] uses three curve-filling predictors that predict the next value to be 1) the same as the preceding value, 2) fitting a linear curve, and 3) fitting a quadratic curve, as Figure 5 illustrates. FPC and SZ both linearize the data before predicting, meaning that they treat multi-dimensional data as if they were one-dimensional. FPC and SZ have different handles on prediction errors. FPC uses leading-zero encoding to compress residuals, so it strictly performs lossless compression. SZ uses an error tolerance to decide if a data point is "unpredictable," and if it is, SZ records the unpredictable values separately. As a result, SZ works best as a lossy compressor when a large portion of data points could be predicted within the error tolerance.

Based on evaluations in [DC15], SZ provides better efficiency (larger compression rates with fixed error, or less error with fixed compression rate) than a number of coders including the above-mentioned FPC, ISABELA, and FPZIP. SZ is also among the fastest coders, which makes it appealing for real-world uses.

2.4. Transform-based Compression

Techniques in this family all have a certain type of transform operation in their hearts, and often involve additional coders to encode the coefficients from the transforms. Transform-based techniques achieve compression by smartly discarding either coefficients with the least information content, or least significant bit

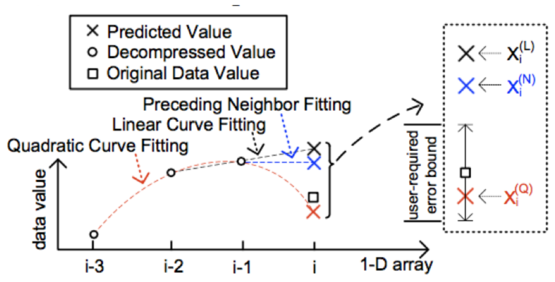


Figure 5: Three predictors used in SZ.

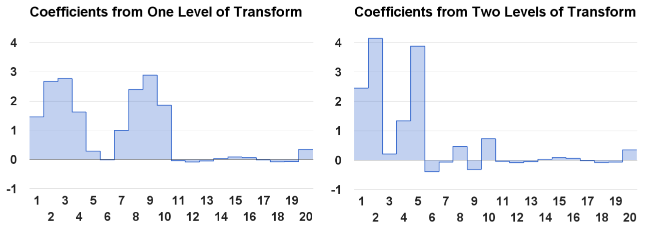


Figure 6: Wavelet coefficients from one and two levels of wavelet transform using the CDF 9/7 kernel. The input array was a sine wave with length twenty.

planes. As a result, they provide lossy compression. Note this approach differs from transform-based multi-resolution, where coefficients were discarded if they represented the details of the original data. From a practical point of view, transform-based compression in this subsection recovers data with the full resolution, whereas transform-based multi-resolution recovers data with lower resolution.

The most successful transform in compression applications is wavelet transform, which is adopted in the newest JPEG2000 [SCE01] still image compression standard. Cosine transform was used in the predecessor of JPEG2000, the JPEG [Wal92] still image compression standard, which still enjoys wide use today. The rest of this subsection surveys a few different transforms with optional coder combinations. We also provide an example showing how a transform "compacts" information to facilitate compression in the wavelet transform + significance map subsection.

Wavelet transform + significance map is the most straightforward approach to achieve compression with wavelets: it saves the wavelet coefficients with the most information content, and treats the rest coefficients as zeros. A significance map is no more than a mechanism to bookkeep the locations of those non-zero coefficients. Data compressed using this approach thus has two parts: 1) significant coefficients which contain most information, and 2) a significance map which keeps where they are.

Wavelet transforms are able to compact information into a small number of coefficients. Moreover, with orthogonal and near-orthogonal wavelet kernels, the magnitudes of resulting wavelet coefficients are proportional to their information content. As a result, saving the coefficients with the largest magnitudes provides efficient compression. We use an example to illustrate this "information compaction": a sine wave with twenty data points goes through a wavelet transform with the CDF 9/7 [CDF92] wavelet kernel. This transform was recursively performed for two levels, as described by Equation 1. Figure 6 plots the resulting wavelet coefficients, and it shows fewer coefficients have large magnitudes after each level of transform. Longer input arrays usually allow more levels of transform, which further compacts information.

Wavelet transform + significance map is a lossy, capable of both reduce on write operator that results in the original memory footprint.

Wavelet transform + general purpose compressors involves two operations: first it performs wavelet transform, and then uses general purpose compressors to compress the resulting wavelet coefficients. The rational is that the coefficients contain much less entropy after wavelet transforms, which makes them friendly to compression. Quantization, which itself is a general purpose compressor, is often combined together with other compressors, since quantization further reduces entropy substantially. The choice of other general purpose compressors includes gzip in [SSEM15], run-length encoding in [IP98a, KS99], Huffman coding in [TMM96], and a combination of run-length and Huffman coding in [GWGS02].

Wavelet transform + general purpose compressors is mostly used as a lossy, reduce on write operator that results in the original memory footprint.

Wavelet transform + state-of-the-art coders uses coders that are specifically designed to encode wavelet coefficients, which is contrary to the general purpose compressors in the previous subsection. These coders represent the latest and the most advanced development in this domain, and they consistently achieve the highest compression accuracies among all wavelet-based compression techniques. We briefly introduce how these coders work, and leave the details to the actual papers that introduced the coders.

These state-of-the-art coders usually work in three steps: 1) wavelet transform, 2) quantization, and 3) encoding. In the wavelet transform step, the CDF family of wavelet kernels [CDF92], especially the member with nine and seven filter sizes which is referred as CDF 9/7, produces coefficients most friendly to further encoding. In the quantization step, all coefficients are represented as integers. Integer representations enable the concept of *bit plane*, where the most significant bits from all coefficients fit in the most significant bit plane; all second most significant bits fit in the second significant bit plane; and so on. Wavelet transform compacts information into a small number of coefficients, so the first few significant planes are expected to have only a few non-zero bits. In the encoding step, specific data structures are used to encode the bit planes from most to least significant ones. These data structures exploit the spatial self-similarities of coefficients to achieve high encoding efficiency. The encoding results are in the form of bitstreams, and starting from the beginning any length of the bitstream is able to reconstruct the data.

Data structures in this family include ZeroTree [Sha93], Set Partitioning in Hierarchical Trees (SPIHT) [SP93], Set Partitioned Embedded bloCKs (SPECK) [IP98b, PINS04], and Embedded Block

Coding with Optimized Truncation (EBCOT) [Tau00]. We note that EBCOT is the coder that was adopted by the JPEG200 standard. Though originally proposed to code 2D images, most of these coders are extended to higher dimensions, including 3D versions of ZeroTree [CP96], SPIHT [KP97], SPECK [TPM03], and EBCOT [XXLZ01], and 4D versions of ZeroTree [ZJM*02], and SPIHT [ZLMS04,LBMN05]. We note that wavelet coefficient coding is a big topic itself, and what we surveyed here are some representative and well-studied techniques. There are many other techniques, for example reported in [CSD*00,LP07,NS01,Rod99].

Wavelet transform + state-of-the-art coders is lossy, capable of both reduce on write and reduce on read, and result in the original memory footprint.

ZFP is a newly emerged transform-based compressor specifically designed to encode scientific data in high throughput [Lin14]. It uses a custom block transform to encode data in blocks of 4^3 size, and also an embedded coding scheme to encode coefficients one "bit plane" by another, which is a similar workflow to the "wavelet transform + state-of-the-art coder." The transform of ZFP, however, was carefully designed to make use of a lifted implementation, which significantly reduces the operations needed per transform. As a result, ZFP was reported to achieve a throughput of 400MB/s.

ZFP is recommended by its authors to be used as a lossy and reduce on write compressor, and it results in the original memory footprint.

Other transforms are also used for data reduction, for example, the Fourier transform, cosine transform (DCT) [ANR74], and Karhunen-Loeve transforms (KLT) [Loe78]. They work in the similar fashion as the "wavelet transform + general purpose compressors" family techniques, where general purpose compressors are used to encode the coefficients from transforms. The choice of general purpose compressors include quantization and entropy-based coders, for example, in techniques introduced in [CYH*97, CL97, FM07] respectively. One way to use these transforms is to perform "compression domain volume rendering," which skips the inverse transform step to perform volume rendering directly with the compressed data. For example, frequency domain volume rendering has been reported in [TL93, WR00].

2.5. Save Derived Information

Techniques in this section do not attempt to reduce the size of simulation data itself. Instead, they apply certain operations on the data, derive additional information, and save the derived information. One might confuse this class of techniques with *in-situ* visualization, where visualization is performed simultaneously with numerical simulations. Saving derived information differs from *in-situ* visualization in that 1) the derived information is not necessarily the target visualization and analysis; 2) the derived information still allows explorations in *post-hoc* style as other data reduction techniques do. The rest of this section surveys such techniques in the scientific visualization community.

Lagrangian representations are derived representations of pathlines for flow analysis [ACG*14]. Traditionally the velocity field of simulation data is saved on disk, and advections are performed to obtain pathlines during analysis routines. Using Lagrangian representations, the advection operation is performed *in-situ* along with the simulation, so the resulting pathlines are saved on disk. The amount of data reduction is controlled by how many pathlines are saved, e.g., less pathlines result in more aggressive data reduction. An arbitrary pathline can be interpolated from nearby pathlines on the fly during analysis routines. These saved pathlines are also expected to be very accurate because they are calculated from every time step of a simulation, as opposed to a temporal subset, which is often the case. The high temporal accuracy is also why this technique is named "Lagrangian representation." To further improve the smoothness of Lagrangian pathlines, the C^1 cubic composite Bézier curves and cubic Hermite splines were proven to be viable representations [BJ15].

Lagrangian representation is lossy, reduce on write, and resulting in reduced memory footprint.

Volumetric depth images, or VDIs [FSE13], are derived from raycastings with auxiliary information. More specifically, this technique partitions samples along a ray into *segments* based on their similarities, and keeps all segments of a ray in a list. These segments contain color, opacity, and depth information. A complete set of VDIs further contains the model view and projection matrices used during the generation of segments, which enables arbitrary camera settings for volume renderings in the data exploration stage. The final volume rendering can be performed at interactive rates due to the saved segments. Data reduction achieved through VDIs is at one order of magnitude. A VDI application on a time series of volumes is also reported in [FFSE14].

VDI is lossy, reduce on write, and resulting in reduced memory footprint.

Proxy images are specially designed images from renderings to perform image-space rendering for analysis. For example, three types of proxy images are used in the framework proposed by [TCM10]: *depth image*, *multi-view perspective image*, and *accumulated attenuation*. Depth images are generated for multiple intensity intervals to enable semi-transparent rendering and complex lighting. Multi-view perspective images contain samples that would only be visible from neighboring viewpoints to enable view changes for explorations. Accumulated attenuations are used to represent opacity mappings to enable transfer function explorations. In terms of data reduction, the resulting proxy images are orders magnitude smaller than the simulation data. They are also less computational expensive than real renderings in most cases.

Proxy image is lossy, reduce on write, and resulting in reduced memory footprint.

2.6. Mesh Simplification

Mesh simplification operates on mesh data, with triangle and tetrahedral meshes most commonly seen. It achieves data reduction by calculating a new mesh with less vertices, yet remains a similar structure of the original mesh. Usually, a mesh simplification routine uses certain criteria to decide which vertices to eliminate, as surveyed in this subsection

Vertex removal and tessellation is a classic mesh simplification

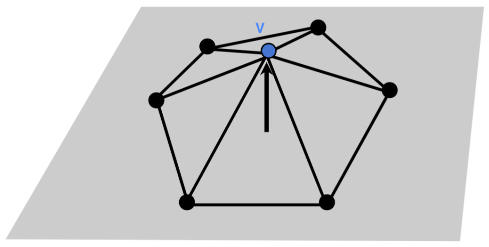


Figure 7: A candidate vertex is evaluated by its distance to the average plane.

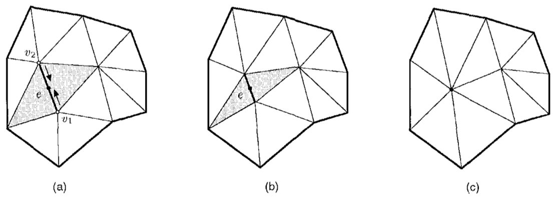


Figure 8: An edge collapses to a single vertex.

algorithm. It deletes identified vertices and perform local tessellations (triangulation and tetrahedralization in 2D and 3D) to fill the resulting hole. One of the vertex selection criteria, for example, was "distance-to-plane" from [SZL92]. The intuition is to eliminate vertices that are almost on a flat plane. Figure 7 illustrates this idea. If a vertex V is surrounded by a complete cycle of triangles, an average plane of these triangles is calculated. This cycle of triangles is reasonably flat if the distance from V to the average plane is small enough (e.g. below a threshold). V is thus prioritized to be removed. Similar simplification technique is also generalized to higher dimensions in [RO96], where the authors use n-dimensional Delaunay tessellations to fill the resulting hole from the removal of vertices.

Vertex removal is lossy, reduce on write, and resulting in reduced memory footprint.

Edge collapsing Edge collapsing works by recursively collapsing an edge to a vertex, removing one edge and one vertex at a time. Figure 8 illustrates a basic edge collapse: edge e, defined by two vertices v_1 and v_2, collapses to a third vertex e on this edge, resulting stretch on triangles sharing v_1 or v_2. Trotts et al. [THJW98, THJ99] pick the next edge to collapse based on prediction of deviations of local triangles and tetrahedra if it were collapsed. Garland et al. [GH97] use quadric-based error metrics (QEM) to guide the edge collapsing process, which have proven to give better results in the mean error sense. Their approach is later extended to higher dimensions in [GZ05].

Edge collapsing is lossy, reduce on write, and resulting in reduced memory footprint.

Vertex clustering Vertex clustering simplifies a mesh by using one vertex to represent a cluster of them. Rossignac et al. [RB93] used uniform grids to divide a 3D volume into small cubes and use a single vertex as the representative of that cube. This representative vertex could be either the center of mass of all vertices, or the most important vertex. Shaffer et al. [SG01] proposed an improvement by using adaptive partitioning of the space. Their algorithm requires a two-pass processing of the mesh: the first pass analyzes the mesh, and the second pass is able to allocate more cells to regions with more details.

Vertex clustering is lossy, reduce on write, and resulting in reduced memory footprint.

2.7. Subsetting

Subsetting achieves data reduction by keeping or retrieving only a portion of the entire data set. Subsetting can be achieved on spatial domains, on variables, and on a querying basis, as the rest of this subsection shows.

Domain subsetting uses meta data of domains to select a subset of domains for further analysis. Domain subsetting allows algorithms to only read in, and further operate on, a subset of data that meets the data range requirement. The granularity of subsetting is often at the spatial domain level, because the meta data of each domain can help decide if the current domain is needed without actually reading data from the whole domain. Examples of domain subsettings include using scalar ranges to choose domains for isosurface calculation, and use spatial extents for slicing a volume.

We consider this technique lossless because the domains read out keep their original form. Domain subsetting also achieves reduce on read, and results in reduce memory footprint.

Variable subsetting is often used in multivariate visualization scenarios, where the large number of variables become an obstacle for analysis. Variable subsetting techniques help identify variables that are more relevant to specific tasks, and subset these variables for further analysis. Variable subsetting reduces memory footprint and also unloads the perception burden of human on multivariate visualizations.

Variable subsetting could be achieved by dimension reduction techniques, such as Multidimensional Scaling (MDS), Principal Component Analysis (PCA) and Independent Component Analysis (ICA) in [GXY12,SKK06,TLMM02] respectively. Variable subsetting could also build on information theory. For example, Biswas et al. [BDSW13] reported an information-aware framework in which the contribution of individual variables to the total entropy is calculated, and variables are grouped or selected based on their individual and mutual information.

We consider variable subsetting lossy because the variables are selected from dimension reduction techniques, rather than selection criteria set by the analyst. There might be information loss during dimension reduction. Variable subsetting also achieves reduce on read, and results in reduced memory footprint.

Query-Based Subsetting Queries find pieces of data that meet specific criteria. We treat them as a form of subsetting in that they enable loading portions rather than the entirety of data. Query-based subsetting is often used in analysis scenarios where *ad-hoc* query dominates the data read process. Query-based subsetting borrows fast query techniques from database management systems to enables data access in query style. As a result, it enables reduction on read applications.

RID	A	bitmap index =0	=1	=2	=3
1	0	1	0	0	0
2	1	0	1	0	0
3	2	0	0	1	0
4	2	0	0	1	0
5	3	0	0	0	1
6	3	0	0	0	1
7	1	0	1	0	0
8	3	0	0	0	1
		b_1	b_2	b_3	b_4

Figure 9: A basic bitmap for a variable A that can take four distinc values: {0, 1, 2, 3}. Each Row Identifier (RID) represents an occurance of A Queries can be answered by bitwise operations, for example, query "$A < 2$" is answered by bitwise OR on bits b_1 and b_2.

Bitmap [O'N89, CI98] is an indexing technique that allows fast queries. Using Bitmaps, each value in a set of N elements is represented by a Row IDentifier (RID). Each RID consists of N bits with a single bit valued one and $N-1$ valued zero. Queries are performed by bitwise logical operations, which are very fast on modern processors. Figure 9 illustrates a basic bitmap for a set with four elements: $\{0,1,2,3\}$. The limitation of bitmaps is that they are most effective on variables with low cardinality. For scientific data with floating point values, bitmaps are used with binning, as explained in [WAB*09].

There are also tree-based schemes for multi-dimensional data query borrowed from the database management community. Such schemes include R-tree [Gut84] with its variants R*-tree [BKSS90] and R+-tree [SRF87], and B-tree [BM72] with its variants BBIO tree [CFSW01]. B-tree is reported to have better performance in indexing unstructured tetrahedral meshes [PAL*06], but overall these schemes are not popular among scientific visualization community.

Query-based subsetting is lossless, reduce on read, and resulting in reduced memory footprint.

3. Use Case 1: Lossless, Reduce on Write, Original Memory Footprint

This section surveys applications that utilize lossless operators to achieve reduce on write, and no reduction in memory footprint. Such use cases are the simplest in the sense that they only tradeoff between two variables: computational time and data size. Technique wise, general purpose compressors (Subsection 2.1) and predictive coders (Subsection 2.3) are most often seen for this use case. Besides direct compression on scientific data sets, there are also applications to use these compressors for I/O middlewares and file systems with transparent compression.

We note here that by lossless we mean a "bit-to-bit" recovery. That is, every bit going through an encoding and decoding sequence stays the same. For floating-point scientific data, this definition excludes transform-based compression techniques (Subsection 2.4). That is because though transforms themselves are mathematically invertible, their calculations on floating-point values introduce arithmetic errors. These tiny errors may accumulate, eventually resulting in recoveries that are not "bit-to-bit" identical. That said, these transforms are lossless if operated on fixed-point values, for example, gray scale images. This section will not cover transform-based techniques because we focus on applications on floating-point scientific data.

3.1. Data Set Compression

General-purpose compressors (Subsection 2.1) and predictive coders (Subsection 2.3) are used for direct compression on data sets in this setting. For example, tetrahedral and triangular meshes are compressed using arithmetic coding respectively in [GGS99, PK05]. The former managed to use just slightly more bits to encode the mesh connectivity than its binary entropy. The combination of predictors and general-purpose compressors are also used to compress volume data [ZTGB02, FY94] and particle data [EGM04]. This combination performs better than general-purpose compressors (e.g., gzip, zip) alone, as discovered in [FY94].

The best available coders for scientific data in this use case are those designed for floating-point values, namely FPZIP [LI06] and FPC [BR09]. These coders achieve not only better compression ratios than general-purpose compressors (e.g., gzip, bzip2), but also higher throughputs. A high throughput is essential to reduce overall I/O time (compression + actual I/O) in a simulation run. A reduced overall I/O is proven to be true for FPZIP [LI06], and is likely to hold up for FPC as well given its reported numbers [BR09].

3.2. Transparent Compression on I/O

Specific I/O middleware and file systems have transparent compression built in, meaning that data is stored in compressed forms without notifying any applications accessing the data. HDF5 [The] and ADIOS [BLZ*14] are two I/O middlewares providing such functionality, and Btrfs [RBM13] and ZFS [BM07] are two file systems with transparent compression. The actual throughput of this kind of I/O middlewares or file systems is again greatly impacted by the underlying compression algorithms. As a result, there are usually multiple compressors to choose from for different use cases. For example, Btrfs recommends using LZO for high throughput, and zlib for high compression ratio [btr].

4. Use Case 2: Lossless, Reduce on Read, Reduced Memory Footprint

This section surveys applications that use a lossless operator to achieve reduce on read, and also reduce memory footprint. These applications use such operators to speed up I/O when reading data in for analysis tasks (reduce on read), and also speed up the analysis routine or even load data that does not fit in memory (reduce memory footprint). Subsetting (Subsection 2.7) is a viable operator to achieve these goals. We survey applications using domain subsetting and query-based subsetting in this section.

4.1. Domain Subsetting for Visualization Pipeline

VisIt [CBW*12], an open-source visualization system for scientific data, uses domain subsetting for its visualization pipeline. VisIt introduces a concept of "contract," which provides an opportunity for every component of the pipeline to modify it with the list of domains that component requires [CBB*05]. When an update is issued at the bottom of the pipeline, the contract travels upstream to inform each component of the downstream components' requirements, and allow the current component to determine its requirements. Finally, the source at the beginning of the pipeline receives

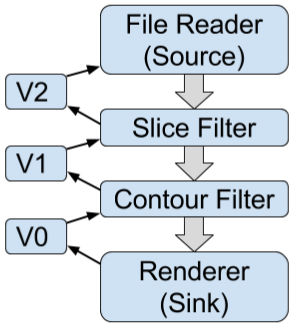


Figure 10: An example pipeline. During the update phase (denoted by thin arrows), contract version 0 (V0), comes from the sink. V0 is then an input to the contour filter, which modifies the contract to make contract version 1 (V1). The details of the execution phase are then passed to the source, which begins the execute phase (denoted by thick arrows).

a contract containing all components' requirements, and reads in only the required domains.

The following example illustrates this contract-based visualization pipeline using domain subsetting. This pipeline has four components as Figure 10 shows. Each component has a chance to identify a list of domains that it needs based on its parameters and the metadata of each domain, i.e., the slice filter picks out the domains covering the spatial extents of the slice, and the contour filter picks out the domains containing the desired isovalue. The resulting contract would contain the intersection of these two domain lists. As the visualization pipeline starts execution, the file reader only reads in the subset of domains indicated by the contract, and the benefits from a reduced data size propagates downstream to every filter.

4.2. Query-based Subsetting for Fast Analysis

FastBit [WAB*09] is an open-source toolkit to perform fast query on scientific data. It uses Bitmaps in its heart for queries. In an application to track the evolution of ignition kernels in a combustion simulation, FastBit is used to speed up this analysis [WKCS03]. During step 1 — finding points satisfying conditions for ignition kernels — FastBit naturally performed queries on the data. During step 2 — grouping the points into connected components — each connected component was represented as a bitmap. During step 3 — tracking the evolution of ignition kernels by computing the overlap among them — was completed quickly with bitwise AND operation. Another application of FastBit, histogram-based parallel coordinates, proved to have satisfactory performance in exploring extremely large data ($\approx$ 1.5TB) on distribute-memory systems [RWC*08]. Both tasks of interactive query and particle tracking enjoyed significant speedups from FastBit. Finally, FastBit was also used to provide a query interface to the popular scientific data format HDF5, resulting in HDF5-FastQuery [GSS*06]. Data stored with HDF5-FastQuery then supports queries for certain data points using query criteria like "$Temperature > 32$" and "$CO_2 > 0.1$."

5. Use Case 3: Lossy, Reduce on Write, Original Memory Footprint

This section surveys applications that use a lossy operator to achieve reduce on write, but not reduce memory footprint after recovery. This use case is similar to what we surveyed in Section 3, but uses lossy compressors. On the one hand, compared to lossless approaches, the loss of accuracy enables much more aggressive compression. On the other hand, the inaccurate recovery needs to be carefully studied to make sure 1) analyses can still carry on (i.e. limited false negatives) and 2) analyses are not misled by compression artifacts (i.e. limited false positives). Overall, this lossy, reduce on write, no reduction on memory approach is easily applied to many applications, and it is a major use case of data reduction.

One commonly adopted approach for simulation runs falls into this use case: temporal sampling. Though sampling strategies differ, temporal sampling loses some time steps in exchange of relieved write burden on disks. The saved time steps are still in their original forms without changing the memory footprint. The rest of this section describes applications that use more complicated operators to reduce sizes of single three-dimensional volumes and a series of time-varying data set.

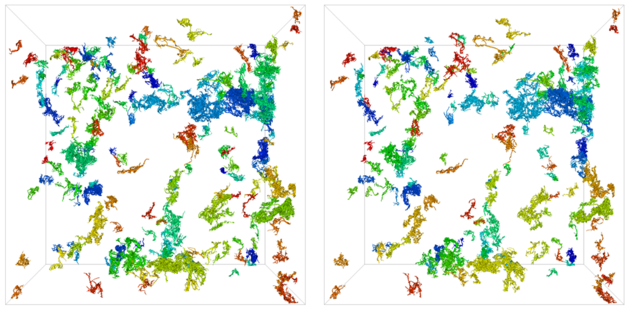
5.1. Three Dimensional Data Volume Compression

Grid data volumes are a very common output of numerical simulations. A lossy operator significantly reduces the size of the data volumes into a compressed form, which is suitable to save to disks. The input data volume is recoverable from the compressed form, but errors may be introduced at individual grid points. A few techniques from different families are used to compress grid data volumes.

Vector quantization (Section 2.1) was used in direct volume rendering [NH92]. Here contiguous blocks of size $I \times J \times K$ serve as individual vectors for quantization. This application has taken advantage of the fast decoding speed of vector quantization, which only involves looking up the codebook. The decompression overhead was measured to be 5%. However, the compression step was time consuming since finding a good codebook was expensive. To achieve even faster rendering speed, the same authors precomputed shading and ray tracing information for each $I \times J \times K$ block, and still applied vector quantization to compress them. The resulting volume rendering system was very fast because it retrieves most information from the codebook.

Predictive coders SZ and FPZIP (Section 2.3) fit into this use case as well [DC15, LI06]. Both coders feature fast coding and decoding speed, making them suitable for potential in-situ use. One advantage of predictive coders for lossy compression is that they can easily bound the absolute errors, i.e., work in an *error bound* mode. This is because the coders can easily obtain the absolute error by comparing the prediction and real values, and make adjustments accordingly. Error bound is important for certain applications where the scientists need a guarantee on the accuracy of their data. Meanwhile, error bound compression leads to an uncertainty of the final compressed file size, i.e., no *size bound*. As a result, a scientist who would like to adopt these predictive coders might need to run experiments to have a sense of the compression ratios on her data. Test results reported in the SZ paper suggest that SZ actually achieves the highest compression in most data sets, making it an appealing choice for real world uses.

Wavelet transform with VAPOR: VAPOR is an open-source visualization package for the climate community [NC12]. It uses wavelets to provide both multi-resolution and compression to the data, and we focus on its compression capability in this section.



(a) Baseline results from the uncompressed raw data. (b) Results from lossy wavelet compression with VAPOR at 256:1 ratio.

Figure 11: Visual analytics of high enstrophy areas from a turbulent flow simulation. Each one of qualified areas is identified as a connected component and has a unique color. The two renderings are from the raw and compressed versions of the same data set.

VAPOR adopts the wavelet transforms + significance map strategy to achieve compression (Section 2.4). This strategy proves to be effective in compressing turbulent flow data, as meaningful visual analytics can still carry out even with a 256 : 1 compression rate [LGP*15]. Figure 11 illustrates the comparison between a baseline rendering and a rendering with lossy compressed data. Using transform-based strategy, VAPOR naturally performs size bound compression by saving a target portion of all coefficients. However, it also introduces storage overheads to keep the significance map. Therefore, VAPOR takes more space than $\frac{1}{X}$ of the raw data in practice in a $X : 1$ compression setting.

Wavelet transforms with the state-of-the-art coders QccPack [Fow00] is an open-source package that includes two of the state-of-the-art wavelet coders: SPIHT and SPECK (Section 2.4). These coders output coefficients in the form of bit streams. A bit stream has the property that any portion of it can be used to recover the input data as long as this portion starts from the beginning of the stream. Of course, the longer the bit stream used to recover, the more accurate the recovery is, with a caveat that there are diminishing returns. This property provides great flexibility to applications: one can cut the output of bitstream at any point, and also read in any portion of the bit stream that is already on disk. In another word, QccPack with SPIHT and SPECK coders is able to achieve reduce on read in addition to reduce on write. Size bound compression is naturally achieved here.

There is currently limited scientific visualization applications of these state-of-the-art wavelet coders, one example being JPEG2000 applied on large scale climate data [WMB*11]. The authors demonstrated that JPEG2000 was effective in trading accuracy for smaller sizes to transmit data through internet. They argued that L^{∞} norm is a better measurement to communicate accuracy with domain scientists. We point out that in this application, the standard two dimensional JPEG2000 compression was applied on individual 2D layers though the original data was in 3D. This approach did not take advantage of data coherence in the vertical direction. Using a state-of-the-art wavelet coder that naturally supports 3D volumes, such as SPIHT and SPECK in QccPack, is likely to yield better accuracies.

Transform-based coder ZFP is a transform-based compressor that was designed for floating-point data (Section 2.4). ZFP has the advantage of supporting finer random access compared to wavelet transform techniques, since it performs transforms in relatively smaller blocks. In the case of 3D data volumes, they are 4^3 blocks. ZFP supports both size bound and error bound compression mode. The original ZFP reported in [Lin14] achieves size bound compression by setting a target number of bits for every 4^3 block. Later with the release of 0.41 version of ZFP, it gained support for error bound compression [LCL16].

An advantage of ZFP is its fast coding and decoding speed. This could be seen from a successful application of ZFP to relieve the I/O burden on wave-propagation simulations [LCL16]. The workflow with ZFP was to compress the strain fields before writing them to the disk, read the compressed strain fields out, and then decompress them before using them in kernel calculations. The overall I/O time of the new workflow would include both calculation time for compression and decompression, and the actual I/O time on the compressed data. Experiments show that compared to I/O time with the raw data, the overall I/O time of this new workflow is $3X$ to $4X$ shorter for write and $5X$ to $6X$ shorter for read depending on the preset error tolerance during compression.

5.2. Time-varying Data Volume Compression

Time-varying data is from time slices saved from numerical simulations. Though policies of saving time slices differ, it is likely that coherence exists from slice to slice. Compression techniques making use of this coherence are referred as "spatiotemporal" compressors in some literatures, and we survey their representative applications in this subsection.

Wavelet transforms are used in time-varying data volume compression. This is partially because of the fewer technical difficulties encountered during the compression: wavelet transforms are essentially one dimensional operations and easy to apply along the time dimension. For example, Villasenor et al. [VED96] applied the CDF 9/7 1D wavelet filter bank over all four dimensions to compress seismic reflection data. The authors argued that cosine transform based techniques, such as JPEG and MPEG, are not suitable for such data, and proved the effectiveness of wavelet transforms. There are also applications of wavelet-based spatiotemporal compression applied to time-varying 3D medical images. Zeng et al. [ZJUH01, ZJM*02] established the feasibility of spatiotemporal wavelet compression of time-varying echocardiography images. In the former of their work, the authors pointed out that the degree of coherence present may differ between dimensions, and thus warranted different handling. Lalgudi et al. [LBM*05, LBMN05] evaluated 4D wavelet spatiotemporal compression on functional MRI (fMRI) data obtained as a time series of 3D images of the brain. They confirmed the coherence difference between dimensions: more benefit gain is observed from the time dimension than the z dimension. These evaluations also show that on fMRI data sets, among the three tested state-of-the-art coders — ZeroTrees, SPIHT, and JPEG2000 (see Section 2.4) — JPEG2000 and

SPIHT are considerably more efficient than ZeroTrees, and SPIHT is slightly better than JPEG2000. We note that some medical data sets are in integer format, thus lossless compression is possible even going through transforms. When using the state-of-the-art coders on floating point values, a quantization step is needed to represent the numbers in fixed point format, prohibiting lossless recovery.

Motion compensated predictions (MCP, see Section 2.3) was proposed for video compression along the time dimension, so it can naturally be applied to time-varying data from simulations. BM3D [DLVL02] is a motion estimation scheme for time-varying 3D data volumes. It features two strategies to search for matches in the next time step of a group of data points: an exhaustive and a heuristic "hill climbing" strategy. The latter strategy uses a two level steepest ascent method to guide the search. "Hill climbing" also proves to be a more practical strategy since it is only slightly less accurate but two orders of magnitude less expensive to compute. Sanchez et al. [SNA06] used an MCP-based video format, H.264, to compress 4D medical imagery, and later developed their custom compressor [SNA08]. In the latter of their work, they used one pass of motion compensation to reduce redundancies between 2D slices in both spatial and temporal dimensions, followed by another pass of motion compensation on the residuals from the first pass. The final residuals and motion vectors were coded by arithmetic coding. In lossless compression mode, this proposed approach achieves significantly higher compression rates than both 3D-JPEG2000 and H.264 coders. MCP could also be used together with wavelet transforms based on the idea to use wavelets to de-correlate coherence in spatial domains, and use MCP to de-correlate coherence in the temporal domain, as explored in [GS01] and [KYLS05]. The latter shows that MCP reduces bit rate by 25% compared to the wavelet-only approach when performing lossless compression on medical imagery. Again, some of these applications report results as lossless because they worked on integer values. The same techniques applied to floating point values would require a quantization step and introduce non-reversible errors.

ISABELA (Section 2.3) is designed to perform lossy spatiotemporal compression on scientific data. ISABELA features high scalability, which makes it really appealing for in-situ use with large scale simulations. A storage framework was built on top of ISABELA to demonstrate the low overhead, fast random access, compression effectiveness, and high scalability in in-situ uses [LSE*13] ISABELA is also flexible enough to provide error bound or size bound modes.

FPZIP and ZFP in 4D mode (Section 2.3) are also available options for time-varying data. The designs of both coders are easy to extend to higher dimensions. We refer back to the discussion in Subsection 5.1.

6. Use Case 4: Lossy, Reduce on Write, Reduced Memory Footprint

Applications in this use case reduce the data size before writing to disk. The reduced form would also lower memory footprints in addition to file sizes. Two families of techniques fit into this use case: saving derived information (Subsection 2.5) and mesh simplification (Subsection 2.6). We will cover their applications in this section. Also, we note that multi-resolution approaches fit into this use case as well if the user decides to save a lower-resolution version of his data for further use, as indicated in [WAF*11]. However, multi-resolution is more widely used as a "reduce on read" technique to improve interactivity of analyses and data explorations. Therefore, we decide to put multi-resolution application in Section 8 instead for interested readers to refer to.

6.1. Mesh Simplification on Unstructured Grids

Mesh simplification (Section 2.6) is for unstructured meshes, for example, tetrahedron meshes in 3D and triangle meshes in 2D. Based on quality criteria, a simplified mesh could have one or two orders magnitude less number of triangles, which makes it much easier for interactive visualizations. A few applications successfully performed mesh simplification on extremely large meshes in an out-of-core fashion.

Edge collapsing based out-of-core mesh simplification was proposed in [WK03, VCL*07]. Both applications read the mesh from disk in a streaming fashion with a certain ordering, i.e., the geometries are roughly sorted by one coordinate. Then the edges are examined against certain error tolerance — they are collapsed if the collapse does not bring more error than the error tolerance. The approach in [VCL*07] is even able to preserve the stream order of the mesh between input and output.

Vertex clustering based out-of-core mesh simplification were also proposed in [Lin00, LS01]. In the former work the authors applied the quadric error metric to guide vertex clustering. The benefit was both better positioning of vertices, and the requirement of only a single pass of the input model to generate the simplified version. This application requires the system memory to hold the simplified version of the mesh. This requirement was removed in their later work ([LS01]). This new version of algorithm compactly stores auxiliary information on disks instead, and managed to avoid expensive disk random accesses.

6.2. Image Space Visualization and Exploration

Image space visualization techniques save explorable images rather than the simulation data, thus dramatically reduce both the file size on disk and footprint in memory. One way to think about image space visualization is that it sacrifices the visualization flexibility to achieve reduced sizes, as opposed to simulation data which provides great flexibility but requires more storage. Cinema [AJO*14] is a such image space visualization framework.

Cinema couples simulations to generate the image database in-situ. Cinema supports a rich set of settings for a domain scientist to make decisions between the final imagery size and its flexibility. For example, the scientist is able to specify visualization objects to create, the parameter space of the selected objects, the camera space of the objects, and the quality of generated images. Based on the specifications, Cinema is able to provide an estimate cost in terms of storage computational overhead. After creating the image database, Cinema supports three types of visual analytics tasks: interactive exploration, querying, and image compositing. While the capability of interactive exploration is limited by the specifications defined during image database creation, the querying and image compositing capabilities are great enhanced by the fact that is this an image-based technique.

Cinema exhibits impressive data reduction factors and performance. The authors argued that the imagery is on the order of 10^6 whereas the simulation data could go onto the order of 10^{15}. The authors also showed that both the in-situ creation of the image database and performing various visualization routines take constant time.

6.3. Lagrangian Representations for Flow Analysis

Lagrangian representations keep the pathline information and are designed for flow analysis. They are usually generated *in-situ* with the numerical simulations, and can take advantage of high temporal fidelity to provide potentially highly accurate pathlines. Sauer et al. [SXM16] further proposed a data structure that combines Lagrangian representations with traditional Eulerian representations: Lagrangian particles are indicated by vectors originating from the Eulerian grid points. With an indexing-based data structure, the combined information from both representations is queried efficiently for complex analyses, as demonstrated in studies of real world fusion, combustion, and cosmology data sets. The additional benefits of this combined representation include out-of-core operations and efficient sample of multi-resolution subsets.

7. Use Case 5: Lossy, Reduce on Read, Reduced Memory Footprint

Multi-resolution approaches (Subsection 2.2) are widely applied in this use case. A lower resolution version of the data is preferred sometimes because it saves I/O and consumes less memory, both helping to provide better interactivity for data explorations.

We note that mesh simplification using either vertex removal or edge collapsing is a progress process — it iterates to remove one vertex or edge at a time. Techniques with this nature has the potential to create a multi-resolution hierarchy, as the authors indicated in [SZL92, RO96]. Unfortunately, we are not aware of any applications using it in this fashion.

7.1. Space Filling Curves for Progressive Data Access

Z-order curves have been used for out-of-core data access, where portions of data must be kept in the next levels of memory given the limit of available memory. Pascucci et al. [PF01] used this technology to create a global index of very large data sets to speed up data request. A visualization algorithm — slicing — was used to demonstrate this benefit by working on the available data in a progressive fashion; it reduces the overall computation time. Hilbert curves are used in MLOC [GRJ*12] — a framework for compressed scientific data exploration with heterogeneous data access patterns — to organize data in a multi-resolution fashion. Here spatial-temporal data sets are linearized by Hilber curves with data points of the same resolution level stored together. This improved data locality alleviates I/O constraints.

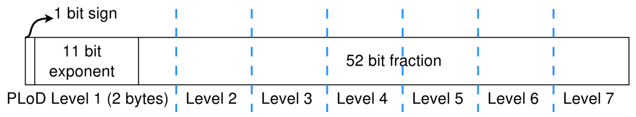


Figure 12: All 64 bits in a double-precision value are divided into seven groups. These seven groups represent seven precision-based levels of details to achieve reduction in read.

7.2. Wavelet Transforms for Progressive Data Access

Volume rendering on Magnetic Resonance (MR) data (e.g. the Visible Human Project [Ack98]) is among the first applications to use wavelet-based multiresolution representations. Muraki [Mur92, Mur93] proved the viability of wavelet-based multi-resolution representations in volume rendering 3D MR data back in early 1990's, and he pointed out a potential use case in data transmission through computer networks: a rough approximation can be sent first, followed by enhanced details at selected regions. Nguyen et al. [NS01] divided 3D data volumes into 16^3 cubes and applied wavelet transform on each of the cubes, achieving both a multi-resolution representation and random access at the individual cube granularity. Similar applications are also reported in [IP98a, Rod99] with discussions on random access strategies.

Multi-resolution representations are especially useful in interactive visualization scenarios, because users can start from a coarse approximation of the volume, and keep refining the entire or a portion of the volume as more data is available. Ihm et al. [IP99] evaluated the interactive use of their wavelet-based technique on the Visible Human data sets. Guthe et al. [GWGS02] then achieved frame rates adequate for practical interactive use on this data set with multiple optimizations. Both LaMar et al. [WWH*00] and Weiler et al. [LHJ00] explored the idea that the interactive visualization uses finer resolutions to render objects closer to the view point, while to use coarser resolutions for objects further from the view point. Guthe et al. [GS04] further applied different resolution levels to different data blocks when rendering, aiming to set a screen-space error bound: more error sensitive blocks are rendered using higher resolution levels. The screen-space error is calculated by comparing the rendered image using the current set against the image rendered using the blocks that map one voxel to a single pixel. Further, Brix et al. [BMMB11] and Gao et al. [GWLS05] applied multi-resolution representations from wavelet on distributed systems. Interestingly, both works use space-filling curves to achieve a better load balance among compute nodes. Finally, wavelet-based multi-resolution approach is also adopted by VAPoR, an open-source package for scientific visualization and analysis, to provide a better user interactivity [CMNR07, CR05].

8. Use Case 6: Lossy, Reduce on Read, Original Memory Footprint

This use case is similar to Case 5, since they are both for progressive data access. This use case is different from Case 5 in that recovered data will not reduce memory footprint. For grids and meshes this means they will have the same number of vertices after recovery from the compressed form. A few techniques that are able to use partial saved data to recover the original data fit into this use case.

MLOC [GRJ*12] provides reduction in read by using less number of bits to reconstruct each double-precision value that originally has 64 bits. More specifically, MLOC divides the eight bytes of a double-precision value into seven groups: the first group has two bytes containing the sign, exponent, and the first four bits of the fraction, and each group next has one byte containing eight more bits of the fraction. The first group thus has a coarse approximation of the data, and each extra group adds more precision. This strategy is referred as Precision-based Level of Details, and illustrated in Figure 12. Using the coarse levels of details in MLOC (e.g., level 1 through level 6) reduces I/O cost in read.

Two wavelet-based techniques — wavelet transform + significance map and wavelet transform + state-of-the-art coders (Subsection 2.4) — are capable of supporting this use case as well. The former sorts all wavelet coefficients based on their information content, and use a subset of coefficients containing most information for reconstruction. Actually, this subset can be cut at any point to start reconstruction, achieving reduction on read. The latter encodes wavelet coefficients one bit-plane by another, from the most significant ones to less significant ones. Again, less number or partial bit-planes can be used to start reconstruction, achieving reduction on read. The QccPack [Fow00] implementation of SPIHT and SPECK coders supports this use: a user is able to specify a smaller bit rate for data reconstruction despite the available bit rate on file, thus achieving reduction on read. Though capable, these wavelet-based techniques have not been used in this lossy, reduction on read, and no reduction in memory fashion in real-world applications to our knowledge.

9. Use Case 7: Lossless, Reduce on Write, Reduced Memory Footprint

This use case is similar to Case 2 (Section 4), with the difference that we focus on reduction on write here, but reduction on read in Case 2. In theory, domain subsetting adopted in Case 2 could be used here, where the scientist deems a subset of domains contains all data needed, and the discarded domains does not lose anything useful. This might be applicable for some applications, for example, in a wind farm simulation, we probably only want to save domains near the wind turbines rather than the entire wind farm. After all, this use case is relatively rare, and we are not aware of any formal discussions of this kind.

10. Use Case 8: Lossless, Reduce on Read, Original Memory Footprint

This use case is similar to Case 1 (Section 3) with the difference that we focus on reduction on read here, but reduction on write in Case 1. Based on our definition of reduction on read, it becomes tricky to require both lossless and reduction on read at the same time. By reduction on read we mean reconstruction using part of the saved data is possible. If this reconstruction were lossless then the saved data has redundancy, and the redundancy is easily eliminated during writing. Since every lossless data reduction technique aims to reduce redundancy even before writing to disks, we decide this use case not practical.

11. Conclusion

This paper surveys common techniques used for visualization and analysis of scientific data sets. They are distinguished by three important characteristics: lossy or lossless, reduce on write or reduce on read, and results in original memory footprint or reduced memory footprint. A scientist may use these three characteristics to guide himself to choose a data reduction technique for her use case. Existing applications with data reduction integrated into the work flow are also surveyed in this paper, and they are organized around the three characteristics of available techniques. We believe this survey serves as a starting point for a visualization scientist as well as a simulation scientist to explore data reduction options to tackle his I/O constraints.

References

[ACG*14] Agranovsky A., Camp D., Garth C., Bethel E. W., Joy K. I., Childs H.: Improved post hoc flow analysis via lagrangian representations. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on* (2014), IEEE, pp. 67–75. 7

[Ack98] Ackerman M. J.: The visible human project. *Proceedings of the IEEE 86*, 3 (1998), 504–511. 13

[AJO*14] Ahrens J., Jourdain S., O'Leary P., Patchett J., Rogers D. H., Petersen M.: An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2014), IEEE Press, pp. 424–434. 1, 12

[Ana89] Anandan P.: A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision 2*, 3 (1989), 283–310. 4

[ANR74] Ahmed N., Natarajan T., Rao K. R.: Discrete cosine transform. *IEEE transactions on Computers 100*, 1 (1974), 90–93. 7

[BDSW13] Biswas A., Dutta S., Shen H.-W., Woodring J.: An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 2683–2692. 8

[BJ15] Bujack R., Joy K. I.: Lagrangian representations of flow fields with parameter curves. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on* (2015), IEEE, pp. 41–48. 7

[BKSS90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: The r*-tree: an efficient and robust access method for points and rectangles. In *ACM SIGMOD Record* (1990), vol. 19, Acm, pp. 322–331. 9

[BLZ*14] Boyuka D. A., Lakshminarasimham S., Zou X., Gong Z., Jenkins J., Schendel E. R., Podhorszki N., Liu Q., Klasky S., Samatova N. F.: Transparent in situ data transformations in adios. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on* (2014), IEEE, pp. 256–266. 9

[BM72] BAYER R., MCCREIGHT E.: Organization and maintenance of large ordered indexes. *Acta Informatica 1* (1972), 173–189. 9

[BM07] Bonwick J., Moore B.: ZFS: The last word in file systems. 9

[BMMB11] Brix K., Melian S., Müller S., Bachmann M.: Adaptive multiresolution methods: Practical issues on data structures, implementation and parallelization. In *ESAIM: Proceedings* (2011), vol. 34, EDP Sciences, pp. 151–183. 13

[BR09] Burtscher M., Ratanaworabhan P.: FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers 58*, 1 (2009), 18–31. 5, 9

[btr] Btrfs compression FAQ. [Online, accessed December 14 2016]. URL: https://btrfs.wiki.kernel.org/index.php/Compression. 9

[CBB*05] Childs H., Brugger E., Bonnell K., Meredith J., Miller M., Whitlock B., Max N.: A contract based system for large data visualization. In *Visualization, 2005. VIS 05. IEEE* (2005), IEEE, pp. 191–198. 9

[CBW*12] Childs H., Brugger E., Whitlock B., Meredith J., Ahern S., Pugmire D., Biagas K., Miller M., Harrison C., Weber G. H., Krishnan H., Fogal T., Sanderson A., Garth C., Bethel E. W., Camp D., Rübel O., Durant M., Favre J. M., Navrátil P.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. Oct 2012, pp. 357–372. 9

[CDDY06] Candes E., Demanet L., Donoho D., Ying L.: Fast discrete curvelet transforms. *Multiscale Modeling & Simulation 5*, 3 (2006), 861–899. 4

[CDF92] Cohen A., Daubechies I., Feauveau J.-C.: Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics 45*, 5 (1992), 485–560. 6

[CFSW01] Chiang Y.-J., Farias R., Silva C. T., Wei B.: A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. In *Parallel and Large-Data Visualization and Graphics, 2001. Proceedings. IEEE 2001 Symposium on* (2001), IEEE, pp. 59–151. 9

[CI98] Chan C.-Y., Ioannidis Y. E.: Bitmap index design and evaluation. In *ACM SIGMOD Record* (1998), vol. 27, ACM, pp. 355–366. 9

[CL97] Chan R. K., Lee M.: 3d-dct quantization as a compression technique for video sequences. In *Virtual Systems and MultiMedia, 1997. VSMM'97. Proceedings., International Conference on* (1997), IEEE, pp. 188–196. 7

[CMNR07] Clyne J., Mininni P., Norton A., Rast M.: Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. *New Journal of Physics 9*, 8 (2007), 301. 13

[Col11] Collet Y.: Lz4–extremely fast compression. 2

[CP96] Chen Y., Pearlman W. A.: Three-dimensional subband coding of video using the zero-tree method. In *Visual Communications and Image Processing'96* (1996), International Society for Optics and Photonics, pp. 1302–1312. 7

[CR05] Clyne J., Rast M.: A prototype discovery environment for analyzing and visualizing terascale turbulent fluid flow simulations. In *Visualization and Data Analysis 2005* (San Jose, CA, USA, Mar. 2005), Erbacher R. F., Roberts J. C., Grohn M. T., Borner K., (Eds.), vol. 5669, SPIE, pp. 284–294. 13

[CSD*00] Chrysafis C., Said A., Drukarev A., Islam A., Pearlman W. A.: Sbhp-a low complexity wavelet coder. In *ICASSP* (2000). 7

[CYH*97] Chiueh T.-c., Yang C.-k., He T., Pfister H., Kaufman A.: Integrated volume compression and visualization. In *Visualization'97., Proceedings* (1997), IEEE, pp. 329–336. 7

[DC15] Di S., Cappello F.: Fast error-bounded lossy hpc data compression with sz. In *IPDPS 2016* (Chicago, IL, 06/2016 2015), IEEE, IEEE. 5, 10

[Deu96] Deutsch P.: *DEFLATE compressed data format specification version 1.3*. Tech. rep., 1996. 2

[DLVL02] De Leeuw W., Van Liere R.: BM3D: motion estimation in time dependent volume data. In *Proceedings of the conference on Visualization'02* (2002), IEEE Computer Society, pp. 427–434. 12

[EGM04] Ellsworth D., Green B., Moran P.: Interactive terascale particle visualization. In *Proceedings of the conference on Visualization'04* (2004), IEEE Computer Society, pp. 353–360. 5, 9

[FFSE14] Fernandes O., Frey S., Sadlo F., Ertl T.: Space-time volumetric depth images for in-situ visualization. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on* (2014), IEEE, pp. 59–65. 7

[FM07] Fout N., Ma K.-L.: Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1600–1607. 7

[Fow00] Fowler J. E.: QccPack: An open-source software library for quantization, compression, and coding. In *International Symposium on Optical Science and Technology* (2000), International Society for Optics and Photonics, pp. 294–301. 11, 14

[FSE13] Frey S., Sadlo F., Ertl T.: Explorable volumetric depth images from raycasting. In *2013 XXVI Conference on Graphics, Patterns and Images* (2013), IEEE, pp. 123–130. 7

[FY94] Fowler J. E., Yagel R.: Lossless compression of volume data. In *Proceedings of the 1994 Symposium on Volume Visualization* (New York, NY, USA, 1994), VVS '94, ACM, pp. 43–50. URL: http://doi.acm.org/10.1145/197938.197961, doi:10.1145/197938.197961. 5, 9

[GG12] Gersho A., Gray R. M.: *Vector quantization and signal compression*, vol. 159. Springer Science & Business Media, 2012. 3

[GGS99] Gumhold S., Guthe S., Strasser W.: Tetrahedral mesh compression with the cut-border machine. In *Proceedings of the conference on Visualization'99: celebrating ten years* (1999), IEEE Computer Society Press, pp. 51–58. 9

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 8

[GRJ*12] Gong Z., Rogers T., Jenkins J., Kolla H., Ethier S., Chen J., Ross R., Klasky S., Samatova N. F.: MLOC: Multi-level layout optimization framework for compressed scientific data exploration with heterogeneous access patterns. In *Parallel Processing (ICPP), 2012 41st International Conference on* (2012), IEEE, pp. 239–248. 13, 14

[GS01] Guthe S., Strasser W.: Real-time decompression and visualization of animated volume data. In *Visualization, 2001. VIS '01. Proceedings* (Oct 2001), pp. 349–572. doi:10.1109/VISUAL.2001.964531. 12

[GS04] Guthe S., Strasser W.: Advanced techniques for high-quality multi-resolution volume rendering. *Computers & Graphics 28*, 1 (2004), 51–58. 13

[GSS*06] Gosink L., Shalf J., Stockinger K., Wu K., Bethel W.: Hdf5-fastquery: Accelerating complex queries on hdf datasets using fast bitmap indices. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)* (2006), IEEE, pp. 149–158. 10

[Gut84] Guttman A.: R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1984), SIGMOD '84, ACM, pp. 47–57. URL: http://doi.acm.org/10.1145/602259.602266, doi:10.1145/602259.602266. 9

[GVDB01] Goeman B., Vandierendonck H., De Bosschere K.: Differential fcm: Increasing value prediction accuracy by improving table usage efficiency. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on* (2001), IEEE, pp. 207–216. 5

[GWGS02] Guthe S., Wand M., Gonser J., Strasser W.: Interactive rendering of large volume data sets. In *Visualization, 2002. VIS 2002. IEEE* (2002), IEEE, pp. 53–60. 6, 13

[GWLS05] Gao J., Wang C., Li L., Shen H.-W.: A parallel multiresolution volume rendering algorithm for large data visualization. *Parallel Computing 31*, 2 (2005), 185–204. 13

[GXY12] Guo H., Xiao H., Yuan X.: Scalable multivariate volume visualization and analysis based on dimension projection and parallel coordinates. *IEEE transactions on visualization and computer graphics 18*, 9 (2012), 1397–1410. 8

[GY95] Ghavamnia M. H., Yang X. D.: Direct rendering of laplacian pyramid compressed volume data. In *Proceedings of the 6th conference on Visualization'95* (1995), IEEE Computer Society, p. 192. 4

[GZ05] Garland M., Zhou Y.: Quadric-based simplification in any dimension. *ACM Transactions on Graphics (TOG) 24*, 2 (2005), 209–239. 8

[H*52] Huffman D. A., et al.: A method for the construction of minimum-redundancy codes. *Proceedings of the IRE 40*, 9 (1952), 1098–1101. 2

[Hil91] Hilbert D.: Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen 38*, 3 (1891), 459–460. 3

[ILRS03] Ibarria L., Lindstrom P., Rossignac J., Szymczak A.: Out-of-core compression and decompression of large n-dimensional scalar fields. In *Computer Graphics Forum* (2003), vol. 22, Wiley Online Library, pp. 343–348. 5

[IP98a] Ihm I., Park S.: Wavelet-based 3d compression scheme for very large volume data. In *Graphics Interface* (1998), vol. 98, Citeseer, pp. 107–116. 6, 13

[IP98b] Islam A., Pearlman W. A.: Embedded and efficient low-complexity hierarchical image coder. In *Electronic Imaging'99* (1998), International Society for Optics and Photonics, pp. 294–305. 6

[IP99] Ihm I., Park S.: Wavelet-based 3d compression scheme for interactive visualization of very large volume data. In *Computer Graphics Forum* (1999), vol. 18, Wiley Online Library, pp. 3–15. 13

[KP97] Kim B.-J., Pearlman W. A.: An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (spiht). In *Data Compression Conference, 1997. DCC'97. Proceedings* (1997), IEEE, pp. 251–260. 7

[KS99] Kim T.-Y., Shin Y. G.: An efficient wavelet-based compression method for volume rendering. In *Computer Graphics and Applications, 1999. Proceedings. Seventh Pacific Conference on* (1999), IEEE, pp. 147–156. 6

[KYLS05] Kassim A. A., Yan P., Lee W. S., Sengupta K.: Motion compensated lossy-to-lossless compression of 4-d medical images using integer wavelet transforms. *IEEE transactions on information technology in biomedicine 9*, 1 (2005), 132–138. 12

[LBG80] Linde Y., Buzo A., Gray R.: An algorithm for vector quantizer design. *IEEE Transactions on communications 28*, 1 (1980), 84–95. 3

[LBM*05] Lalgudi H. G., Bilgin A., Marcellin M. W., Tabesh A., Nadar M. S., Trouard T. P.: Four-dimensional compression of fMRI using JPEG2000. In *Medical Imaging* (2005), International Society for Optics and Photonics, pp. 1028–1037. 11

[LBMN05] Lalgudi H. G., Bilgin A., Marcellin M. W., Nadar M. S.: Compression of fMRI and ultrasound images using 4d SPIHT. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on* (2005), vol. 2, IEEE, pp. II–746. 7, 11

[LCL16] Lindstrom P., Chen P., Lee E.-J.: Reducing disk storage of full-3d seismic waveform tomography (f3dt) through lossy online compression. *Computers & Geosciences 93* (2016), 45–54. 11

[LD07] Lu Y. M., Do M. N.: Multidimensional directional filter banks and surfacelets. *IEEE Transactions on Image Processing 16*, 4 (2007), 918–931. 4

[LGP*15] Li S., Gruchalla K., Potter K., Clyne J., Childs H.: Evaluating the efficacy of wavelet configurations on turbulent-flow data. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (2015), pp. 81–89. 1, 11

[LHJ00] LaMar E., Hamann B., Joy K. I.: Multiresolution techniques for interactive texture-based volume visualization. In *Electronic Imaging* (2000), International Society for Optics and Photonics, pp. 365–374. 13

[LI06] Lindstrom P., Isenburg M.: Fast and efficient compression of floating-point data. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5 (2006), 1245–1250. 1, 5, 9, 10

[Lin00] Lindstrom P.: Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 259–262. 12

[Lin14] Lindstrom P.: Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics 20*, 12 (2014), 2674–2683. 7, 11

[Llo82] Lloyd S.: Least squares quantization in pcm. *IEEE transactions on information theory 28*, 2 (1982), 129–137. 3

[Loe78] Loeve M.: Probability theory, vol. ii. *Graduate texts in mathematics 46* (1978), 0–387. 7

[LP07] Liu Y., Pearlman W. A.: Four-dimensional wavelet compression of 4-d medical images using scalable 4-d sbhp. In *Data Compression Conference, 2007. DCC'07* (2007), IEEE, pp. 233–242. 7

[LS01] Lindstrom P., Silva C. T.: A memory insensitive technique for large model simplification. In *Proceedings of the conference on Visualization'01* (2001), IEEE Computer Society, pp. 121–126. 12

[LSE*11] Lakshminarasimhan S., Shah N., Ethier S., Klasky S., Latham R., Ross R., Samatova N. F.: Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In *Euro-Par 2011 Parallel Processing*. Springer, 2011, pp. 366–379. 5

[LSE*13] Lakshminarasimhan S., Shah N., Ethier S., Ku S.-H., Chang C.-S., Klasky S., Latham R., Ross R., Samatova N. F.: Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience 25*, 4 (2013), 524–540. 12

[Mor66] Morton G. M.: *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966. 3

[Mur92] Muraki S.: Approximation and rendering of volume data using wavelet transforms. In *Proceedings of the 3rd conference on Visualization'92* (1992), IEEE Computer Society Press, pp. 21–28. 13

[Mur93] Muraki S.: Volume data and wavelet transforms. *IEEE Computer Graphics and applications*, 4 (1993), 50–56. 13

[NC12] Norton A., Clyne J.: The VAPOR visualization application. In *High Performance Visualization*, Bethel E., Childs H., Hanson C., (Eds.). 2012, pp. 415–428. 10

[NH92] Ning P., Hesselink L.: Vector quantization for volume rendering. In *Proceedings of the 1992 workshop on Volume visualization* (1992), ACM, pp. 69–74. 10

[NS01] Nguyen K. G., Saupe D.: Rapid high quality compression of volume data for visualization. In *Computer Graphics Forum* (2001), vol. 20, Wiley Online Library, pp. 49–57. 7, 13

[Obe05] Oberhumer M.: Lzo real-time data compression library. *User manual for LZO version 0.28, URL: http://www. infosys. tuwien. ac. at/Staff/lux/marco/lzo. html (February 1997)* (2005). 2

[O'N89] O'Neil P. E.: Model 204 architecture and performance. In *High Performance Transaction Systems*. Springer, 1989, pp. 39–59. 9

[PAL*06] Papadomanolakis S., Ailamaki A., Lopez J. C., Tu T., O'Hallaron D. R., Heber G.: Efficient query processing on unstructured tetrahedral meshes. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (2006), ACM, pp. 551–562. 9

[Pav99] Pavlov I.: 7z format, 1999. [Online, accessed October 30 2016]. URL: `http://www.7-zip.org/7z.html`. 2

[PF01] Pascucci V., Frank R. J.: Global static indexing for real-time exploration of very large regular grids. In *Supercomputing, ACM/IEEE 2001 Conference* (2001), IEEE, pp. 45–45. 3, 13

[PINS04] Pearlman W. A., Islam A., Nagaraj N., Said A.: Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE transactions on circuits and systems for video technology 14*, 11 (2004), 1219–1235. 6

[PK05] Peng J., Kuo C.-C. J.: Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 609–616. 9

[PLW*16] Pulido J., Livescu D., Woodring J., Ahrens J., Hamann B.: Survey and analysis of multiresolution methods for turbulence data. *Computers & Fluids 125* (2016), 39–58. 4

[RB93] Rossignac J., Borrel P.: Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in computer graphics*. Springer, 1993, pp. 455–465. 8

[RBM13] Rodeh O., Bacik J., Mason C.: Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS) 9*, 3 (2013), 9. 9

[RO96] Renze K. J., Oliver J. H.: Generalized unstructured decimation [computer graphics]. *IEEE Computer Graphics and Applications 16*, 6 (1996), 24–32. 8, 13

[Rod99] Rodler F. F.: Wavelet based 3d compression with fast random access for very large volume data. In *Computer Graphics and Applications, 1999. Proceedings. Seventh Pacific Conference on* (1999), IEEE, pp. 108–117. 7, 13

[RWC*08] Rübel O., Wu K., Childs H., Meredith J., Geddes C. G., Cormier-Michel E., Ahern S., Weber G. H., Messmer P., Hagen H., et al.: High performance multivariate visual data exploration for extremely large data. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), IEEE Press, p. 51. 10

[RY14] Rao K. R., Yip P.: *Discrete cosine transform: algorithms, advantages, applications*. Academic press, 2014. 4

[SCE01] Skodras A., Christopoulos C., Ebrahimi T.: The jpeg 2000 still image compression standard. *IEEE Signal processing magazine 18*, 5 (2001), 36–58. 6

[SG01] Shaffer E., Garland M.: Efficient adaptive simplification of massive meshes. In *Proceedings of the conference on Visualization'01* (2001), IEEE Computer Society, pp. 127–134. 8

[Sha93] Shapiro J. M.: Embedded image coding using zerotrees of wavelet coefficients. *Signal Processing, IEEE Transactions on 41*, 12 (1993), 3445–3462. 6

[SKK06] Salama C. R., Keller M., Kohlmann P.: High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1021–1028. 8

[SN96] Strang G., Nguyen T.: *Wavelets and filter banks*. SIAM, 1996. 4

[SNA06] Sanchez V., Nasiopoulos P., Abugharbieh R.: Lossless compression of 4d medical images using h. 264/avc. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings* (2006), vol. 2, IEEE, pp. II–II. 12

[SNA08] Sanchez V., Nasiopoulos P., Abugharbieh R.: Efficient 4d motion compensated lossless compression of dynamic volumetric medical image data. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on* (2008), IEEE, pp. 549–552. 12

[Sne95] Sneddon I. N.: *Fourier transforms*. Courier Corporation, 1995. 4

[SP93] Said A., Pearlman W. A.: Image compression using the spatial-orientation tree. In *ISCAS* (1993), vol. 93, pp. 279–282. 6

[SRF87] Sellis T., Roussopoulos N., Faloutsos C.: The r+-tree: A dynamic index for multi-dimensional objects. 9

[SS97] Sazeides Y., Smith J. E.: The predictability of data values. In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on* (1997), IEEE, pp. 248–258. 5

[SSEM15] Sasaki N., Sato K., Endo T., Matsuoka S.: Exploration of lossy compression for application-level checkpoint/restart. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International* (2015), IEEE, pp. 914–922. 6

[SXM16] Sauer F., Xie J., Ma K.-L.: A combined eulerian-lagrangian data representation for large-scale applications. *IEEE Transactions on Visualization and Computer Graphics* (2016). 13

[SZL92] Schroeder W. J., Zarge J. A., Lorensen W. E.: Decimation of triangle meshes. In *ACM Siggraph Computer Graphics* (1992), vol. 26, ACM, pp. 65–70. 8, 13

[Tau00] Taubman D.: High performance scalable image compression with ebcot. *Image Processing, IEEE transactions on 9*, 7 (2000), 1158–1170. 7

[TCM10] Tikhonova A., Correa C. D., Ma K.-L.: Visualization by proxy: A novel framework for deferred interaction with volume data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1551–1559. 7

[The] The HDF Group: Using compression in HDF5. [Online, accessed December 14 2016]. URL: `https://support.hdfgroup.org/HDF5/faq/compression.html`. 9

[THJ99] Trotts I. J., Hamann B., Joy K. I.: Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 224–237. 8

[THJW98] Trotts I. J., Hamann B., Joy K. I., Wiley D. F.: Simplification of tetrahedral meshes. In *Visualization'98. Proceedings* (1998), IEEE, pp. 287–295. 8

[TL93] Totsuka T., Levoy M.: Frequency domain volume rendering. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 271–278. 7

[TLMM02] Takanashi I., Lum E. B., Ma K.-L., Muraki S.: Ispace: Interactive volume data classification techniques using independent component analysis. In *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on* (2002), IEEE, pp. 366–374. 8

[TMM96] Trott A., Moorhead R., McGinley J.: Wavelets applied to lossless compression and progressive transmission of floating point data in 3-d curvilinear grids. In *Visualization'96. Proceedings.* (1996), IEEE, pp. 385–388. 6

[TPM03] Tang X., Pearlman W. A., Modestino J. W.: Hyperspectral image compression using three-dimensional wavelet coding. In *Electronic Imaging 2003* (2003), International Society for Optics and Photonics, pp. 1037–1047. 7

[VCL*07] Vo H. T., Callahan S. P., Lindstrom P., Pascucci V., Silva C. T.: Streaming simplification of tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (2007), 145–155. 12

[VED96] Villasenor J. D., Ergas R., Donoho P.: Seismic data compression using high-dimensional wavelet transforms. In *Data Compression Conference, 1996. DCC'96. Proceedings* (1996), IEEE, pp. 396–405. 11

[WAB*09] Wu K., Ahern S., Bethel E. W., Chen J., Childs H., Cormier-Michel E., Geddes C., Gu J., Hagen H., Hamann B., et al.: Fastbit: interactively searching massive data. In *Journal of Physics: Conference Series* (2009), vol. 180, IOP Publishing, p. 012053. 9, 10

[WAF*11] Woodring J., Ahrens J., Figg J., Wendelberger J., Habib S., Heitmann K.: In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 1151–1160. 3, 12

[Wal92] Wallace G. K.: The jpeg still picture compression standard. *IEEE transactions on consumer electronics 38*, 1 (1992), xviii–xxxiv. 6

[Wel84] Welch T. A.: A technique for high-performance data compression. *Computer 17*, 6 (1984), 8–19. 2

[WK03] WU J., KOBBELT L.: A stream algorithm for the decimation of massive meshes. In *Graphics interface* (2003), vol. 3, pp. 185–192. 12

[WKCS03] WU K., KOEGLER W., CHEN J., SHOSHANI A.: Using bitmap index for interactive exploration of large datasets. In *Scientific and Statistical Database Management, 2003. 15th International Conference on* (2003), IEEE, pp. 65–74. 10

[WMB*11] WOODRING J., MNISZEWSKI S., BRISLAWN C., DEMARLE D., AHRENS J.: Revisiting wavelet compression for large-scale climate data using jpeg 2000 and ensuring data precision. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on* (Oct 2011), pp. 31–38. doi:10.1109/LDAV.2011.6092314. 11

[WNC87] WITTEN I. H., NEAL R. M., CLEARY J. G.: Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (1987), 520–540. 2

[WR00] WESTENBERG M. A., ROERDINK J. B.: Frequency domain volume rendering by the wavelet x-ray transform. *IEEE Transactions On Image Processing 9*, 7 (2000), 1249–1261. 7

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMANN K., ERTL T.: Level-of-detail volume rendering via 3d textures. In *Proceedings of the 2000 IEEE symposium on Volume visualization* (2000), ACM, pp. 7–13. 13

[XXLZ01] XU J., XIONG Z., LI S., ZHANG Y.-Q.: Three-dimensional embedded subband coding with optimized truncation (3-d escot). *Applied and Computational Harmonic Analysis 10*, 3 (2001), 290–315. 7

[ZJM*02] ZENG L., JANSEN C. P., MARSCH S., UNSER M., HUNZIKER P. R.: Four-dimensional wavelet compression of arbitrarily sized echocardiographic data. *Medical Imaging, IEEE Transactions on 21*, 9 (2002), 1179–1187. 7, 11

[ZJUH01] ZENG L., JANSEN C., UNSER M. A., HUNZIKER P.: Extension of wavelet compression algorithms to 3d and 4d image data: Exploitation of data coherence in higher dimensions allows very high compression ratios. In *International Symposium on Optical Science and Technology* (2001), International Society for Optics and Photonics, pp. 427–433. 11

[ZL77] ZIV J., LEMPEL A.: A universal algorithm for sequential data compression. *IEEE Transactions on information theory 23*, 3 (1977), 337–343. 2

[ZL78] ZIV J., LEMPEL A.: Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory 24*, 5 (1978), 530–536. 2

[ZLMS04] ZIEGLER G., LENSCH H. P., MAGNOR M., SEIDEL H.-P.: Multi-video compression in texture space using 4d spiht. In *Multimedia Signal Processing, 2004 IEEE 6th Workshop on* (2004), IEEE, pp. 39–42. 7

[ZTGB02] ZHAO R., TAO T., GABRIEL M., BELFORD G. G.: Lossless compression of very large volume data with fast dynamic access. In *Proc. SPIE* (2002), vol. 4925, p. 180. 5, 9