Data Reduction Techniques for Simulation, Visualization, and Data Analysis

Shaomeng Li[1,2], Nicole Marsaglia[2], Christoph Garth[3], Jonathan Woodring[4], John Clyne[1], and Hank Childs[2]

[1]National Center for Atmospheric Research [2]University of Oregon [3]Technische Universitat Kaiserslautern [4]Los Alamos National Laboratory

Abstract

Data reduction is increasingly being applied to scientific data for numerical simulations, scientific visualizations, and data analyses. It is most often used to lower I/O and storage costs, and sometimes to lower in-memory data size as well. With this paper, we consider five categories of data reduction techniques based on their information loss: 1) truly lossless, 2) near lossless, 3) lossy, 4) mesh reduction, and 5) derived representations. We then survey available techniques in each of these categories, summarize their properties from a practical point of view, and discuss relative merits within a category. We believe, in total, this work will enable simulation scientists and visualization/data analysis scientists to decide which data reduction techniques will be most helpful for their needs.

1. Introduction

Scientific visualization and data analysis most often use the post hoc paradigm, where simulation codes store data to a file system and where visualization programs read this simulation data at a later time. File I/O is increasingly a bottleneck for this paradigm, both for transfer rates and for storage capacity.

Slow I/O rates create problems for both simulation and visualization scientists. Simulation scientists, as data producers, can opt to write data less often. However, this approach can be problematic, as the resulting temporal sparsity may prevent proper analysis of the data. Visualization and data analysis scientists, as data consumers, need reasonable turnaround times for processing data with exploratory tasks. The majority of time to perform these tasks is typically spent both on reading data from disk and on computations that produce meaningful visualizations and statistics. As the ability to read data goes down, turnaround times go up.

Data reduction is a means to tackle I/O-related difficulties for both simulation and visualization scientists. For simulation scientists, reduced data sizes translated to less transfer time to disk and smaller storage space on disk. For visualization scientists, reading in fewer bytes also means a shorter turnaround time, improving interactivity. Each data reduction technique has varying tradeoffs between magnitude of reduction, processing time, and integrity. As a result, selecting a data reduction technique requires finding a middle ground between these tradeoffs, as well as considering the intended use case.

This paper mainly surveys data reduction techniques developed for scientific data arising from numerical simulations, and, to the best of our knowledge, this is the first all-in-one survey paper on this topic. In most cases, this type of data consists of floating-point values on certain mesh structures. The broader data reduction research community has developed a range of techniques for different types of data, primarily multimedia data such as audio and video. Many of these techniques can be applied to scientific data with little or no modification. We include these techniques in our survey provided there have been applications reported in the literature.

The rest of the introduction explains this survey's categorization of information loss associated with data reduction, which is arguably the most important property scientists care about in practice. Following that discussion is an explanation of other relevant properties of data reduction techniques, formalization of terminology, and some examples of of visualizations after data reduction. The introduction concludes with a roadmap for finding the most relevant sections of the paper.

1.1. Information Loss During Data Reduction

The classic view on information loss during data reduction is a binary choice: lossy or lossless. This view fails to capture differences between a wide range of data reduction techniques used by simulation and visualization scientists, whose application requirements are diverse from each other. Instead, we consider a five-category classification to qualify the information loss of a data reduction technique:

1. truly lossless compression;
2. near lossless compression;
3. lossy compression;
4. mesh reduction; and
5. derived representations.

While mesh reduction techniques and derived representations are both technically types of lossy compression, we classify them separately because they are outside the scope traditionally considered by the compression community — these two techniques tend to be even more lossy than what the compression community considers. Therefore, our classification could be viewed as a *spectrum*, with one end being strictly lossless (truly lossless) and the other being very lossy (derived representations), to the point that the original data often cannot be reconstructed. Looking from the perspective

of reduction factors, the two ends of the spectrum are modest reduction (truly lossless) and aggressive reduction (derived representations). Considering both information loss and reduction factors, scientists with extreme data problems may be willing to accept a certain level of information loss in exchange for the benefits of large reduction factors. In the rest of this subsection, we explain these five categories and survey their corresponding techniques in the rest of this paper.

Truly lossless compression: Truly lossless compression means data reconstructed after compression has exactly the same value on every bit as the original data, i.e., *bit-for-bit* identical. Scientists do not need to worry about this type of data reduction techniques with regard to data integrity, since any visualization and analysis outcome will not be altered. Many general-purpose compression techniques fall within this classification, and some have mature tools that realize their approaches, for example, *gzip*.

Near lossless compression: Near lossless compression means no information is discarded deliberately during data reduction, but floating-point rounding errors are introduced during arithmetic operations. Another way of understanding this property is that the techniques are lossless in mathematics but not in practice, with rounding errors being intrinsic for floating-point arithmetics. Though data is altered, scientists might accept these rounding errors, since they are theoretically at the same magnitude as the numerical models that generated the data. Many transformation-based techniques fall into this category.

Lossy compression: Lossy compression means that there is no requirement that reconstructed data points match the original data set. As a result, lossy techniques usually achieve a much more significant data reduction than lossless methods. The acceptable amount of information loss is highly application dependent, and scientists often need to examine the integrity of the resulting data and tune the reduction parameters on a case by case basis.

Mesh reduction: Mesh reduction techniques reduce the scale of a mesh (i.e., number of vertices and cells), thus reducing the amount of storage. Mesh reduction contrasts with lossy compression in that the underlying mesh is subject to reduction, while lossy compression does not touch the mesh itself. There are multiple ways to accomplish this goal. One way is to make meshes that resemble the original ones, but coarser. One benefit of a reduced mesh is that it also requires less memory to process, which benefits interactivity.

Derived representations: Derived representation techniques discard the original data entirely, enabling extreme data reduction. Instead, alternative representations are calculated and stored, so analysis routines can produce results similar to those when operating on the original data. The techniques in this class are often tailored toward specific visualization and analysis tasks. Scientists could be motivated to accept this class of techniques because of the potential for data reductions beyond what the other classes can offer. An example here is Cinema [AJO*14], which saves images of tens of thousands of visualizations across a wide range of parameters for further visual analytics; these images can still be orders of magnitude smaller than the simulation data.

1.2. Additional Properties of Data Reduction Techniques

Beyond information loss, additional properties are important to deciding which techniques to use. This section briefly explains these properties. Note that some properties apply to only some of the categories of data reduction technologies from Subsection 1.1.

Fixed error: With this property, an error tolerance can be specified as input to the reduction operator. The form of error tolerance can be an absolute deviation, a relative offset, or any pre-defined metric based on the application and/or technique. Fixed error allows scientists to carry out their analysis with more confidence, since they are assured of a limit in the amount of deviation. However, the downside is that the final size after reduction is often hard to predict, because not all data sets are equally amenable to compression. Truly and near lossless compression have fixed error, as the former has absolutely zero error, and the latter has floating-point rounding errors. That said, fixed error can be applicable to lossy compression and mesh reduction techniques as well.

Fixed size: This property is complementary to fixed error, i.e., the property where the maximum size of the data after reduction is guaranteed. That said, while the size after reduction can be set in advance, the resulting error is often unknown. We note that while some techniques could support fixed error and fixed size at the same time (e.g., uniform quantization), most advanced lossy techniques have to choose between them. Fixed size also applies to lossy compression and mesh reduction. Derived representations may have fixed size as well, since the final size of their representations is often user-settable.

Reduced memory footprint: With this property, the reconstructed form of the data set is smaller in size than the original version, meaning it takes less system memory for analyses. A reduced memory footprint speeds up both I/O time and execution time for algorithms, making this property very desirable for interactive applications. Lossy compression, mesh reduction, and derived representations can all have the reduced memory footprint property; with lossy compression, each data point is represented with fewer bits; with mesh reduction, a mesh is reduced to have fewer vertices; and with derived representations, the representations themselves usually take less memory.

Progressive data access: With this property, reduced data can be accessed in a progressive fashion, i.e., accessing part of the reduced data for a coarse reconstruction, and progressively accessing more data to obtain finer reconstructions. Progressive data access can be very helpful to alleviate I/O constraints during post hoc analysis, so it is also desirable for interactive applications. This property is most often available from lossy compression and mesh reduction techniques, with the former providing progressively increasing accuracy for individual data values, and the latter providing progressively increasing resolution for the underlying mesh or data domain.

1.3. Terminology

This section clarifies some terminology used in this paper. The scope of this survey is data reduction *techniques*. While many data reduction techniques are *compressors* (as in the lossless and lossy

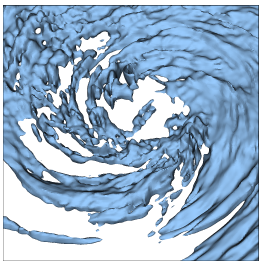


Figure 1: An isosurface visualization on uncompressed data (ground truth). This isosurface is on the temperature field from a simulation of Hurricane Isabel [KWB*].

compression categories), there are reduction techniques that are not compressors in the traditional sense, for example, mesh coarsening and topology representations. *Coders* are types of compressors that specifically apply an encoding algorithm to a data stream, and are often used as components of compressor products.

These techniques *reduce* the size of the data, also known as *compressing* the data in the case of compressors. The inverse process is then referred to as either *decompression* or *reconstruction*. When describing coders, these processes are also referred to as *encode* and *decode*. *Reduction factors* and *compression factors* are then the ratios between the original and the reduced forms of the data. For example, a reduction factor of four (4*X*, or 4:1) means that the reduced form is a quarter of the original form in size. Finally, the ability of a technique to perform data reduction is referred to as its *efficiency* or *effectiveness*. In lossless cases, the smaller the resulting data size, the more efficient a technique is. In lossy cases, a more efficient technique results in smaller size with the same level of error, or less error with the same target size.

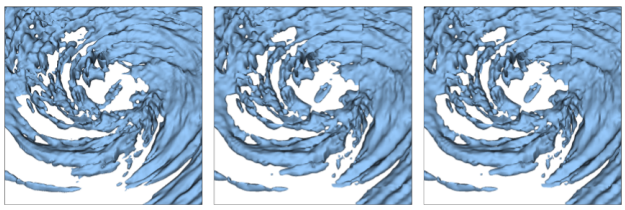
1.4. Visualizing Compression Results

This section illustrates the effects of data reduction for a representative visualization. Specifically, an uncompressed data set is compared with reconstructed data from different reduction strategies. The data reduction strategies come from three different categories and represent the potential reduction differences between each strategy.

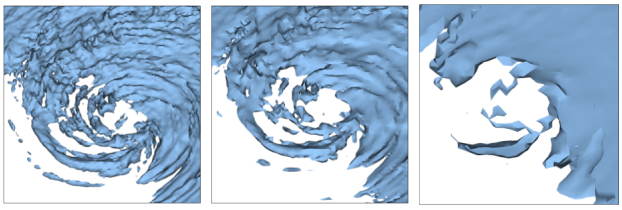
The data used for the visualizations is from a simulation of Hurricane Isabel, which was used for the inaugural IEEE Visualization contest in 2004 [KWB*]. The visualizations are of isosurfacing the temperature variable. All data are 32-bit floating-point values on a regular grid of $500 \times 500 \times 100$. Figure 1 shows this visualization on the uncompressed (original) data, plotted using the VisIt visualization package [CBW*12].

We picked three data reduction techniques from three categories: 1) lossy compression achieved by wavelet transform + coefficient prioritization (Subsection 4.4), 2) multi-resolution achieved by wavelet transforms (Subsection 5.2), and 3) mesh decimation (Subsection 5.1). VAPOR [NC12] and VisIt [CBW*12] were the tools we used to perform reduction.

Figure 2 shows visualizations of the same variable but on data that has gone through reduction. In Figure 2a, the grid remains at $500 \times 500 \times 100$ but the data is compressed at 8:1, 64:1, and 512:1 levels. For Figure 2b with multi-resolution, the grid resolution is lowered to be $250 \times 250 \times 50$, $125 \times 125 \times 25$, and $62 \times 62 \times 12$.



(a) Reduction: wavelet + coefficient prioritization (Subsection 4.4.1).



(b) Reduction: wavelet + multi-resolution (Subsection 5.2.3).

Figure 2: Contrasting two different reduction techniques. Both sets of images are isosurfaces applied to reduced data, with reduction levels 8:1, 64:1, and 512:1 from left to right.

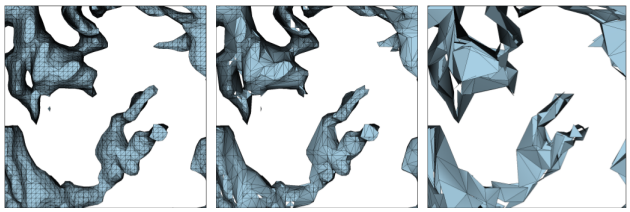


Figure 3: Close-up view of a portion of the triangle mesh of the isosurface. The ground truth is on the left. The mesh is reduced through mesh decimation (Subsection 5.1) with reduction levels 8:1 in the middle and 64:1 on the right.

With less grid points, the data size is reduced to 8:1, 64:1, and 512:1 levels respectively.

Figure 3 shows the effects of mesh decimation on the same isosurface. In this case, an isosurface was extracted on the original data, and then two separate mesh decimations were applied to the isosurface. The total number of triangles was reduced to $\frac{1}{8}^{th}$ and also to $\frac{1}{64}^{th}$ of the original contour mesh, corresponding to reduction levels of 8:1 and 64:1, respectively.

Overall, the lossy compression shows the best visualization quality: the contour only suffered subtle changes at 64:1 compression level, and still retained many details at 512:1. The multi-resolution and mesh decimation results suffered from more deterioration, with the 64:1 results already losing many details. On the other hand, the multi-resolution and mesh decimation results both reduce memory footprint, which is a desirable property for interactive applications. Of course, the best choice for reduction technique varies over use case and data set.

1.5. Guide to Reading This Paper

This paper surveys data reduction techniques based on the spectrum of information loss, spanning from zero information loss (truly lossless) to some information loss (lossy compression) to completely discarding the original data (derived representations). A total of five categories along this spectrum are described in their own sections, from Section 2 to Section 6. Some data reduction techniques span categories. In these cases, we describe the technique in depth in the most appropriate category section.

The paper was designed so that a reader could skip directly to the section(s) of most interest:

- For use cases where reconstructed data must be bit-for-bit accurate, the only applicable works in this paper can be found in Section 2. These techniques, on the whole, produce the least amount of data reduction.
- For use cases where accuracy within floating-point rounding error can be accepted, the reader should also consider Section 3.
- For lossy compression use cases, the reader should skip Sections 2 and 3 and proceed directly to Section 4, as the techniques in Section 4 provide significantly more data reduction than those in the preceding two sections.
- For use cases where the reconstructed data set needs to be much smaller in size than the original data, the reader should consult Sections 5 and 6. For the most part, Section 5 presents techniques that produces reduced forms of the original data (i.e., smaller memory footprint) that can be used for arbitrary visualizations and analyses, while Section 6 presents techniques that optimized only for specific visualizations and analyses.

2. Truly Lossless Compression

Truly lossless compressors are used in almost every area of computer science. These compressors work much better on text and integer data than on floating-point data. This is because a large portion of the bits in floating-point representations are effectively random, making lossless compression very difficult. The rest of this section surveys two families of classic coders in a relatively brief fashion, and then discusses a few truly lossless compressors that are specifically designed for floating-point scientific data.

2.1. Entropy-based Coders

We begin with a brief introduction to information theory, which gives a theoretical lower bound on the number of bits needed to encode each symbol from a source. According to Shannon [Sha01], for a source S made up of independent and identically distributed (*iid*) symbols $a_1, ..., a_n$, where each symbol has a respective probability of occurrence $p_1, ..., p_n$, the information content, or *entropy*, of S, is given by:

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i). \quad (1)$$

H provides an optimal lower bound on the expected number of bits needed to represent each a_i. Note that H is maximized when all symbols occur with equal probability ($p_i = 1/n$), and minimized when the probability of a single symbol is one and the rest are zeros. Thus, the entropy of a source determines a theoretical lower bound on the number of bits needed to represent the *iid* symbols without information loss.

The goal of entropy-based coders is to achieve the optimal bit representation as indicated by Equation 1. That is, entropy coding methods attempt to represent the symbols a_i by a set of *codewords*, c_i, in a way that minimizes the average number of bits per codeword with a lower bound given by H. These techniques are typified by the use of variable length codewords. That is, shorter codewords are assigned to represent symbols with higher probabilities of occurrence, and longer codewords are assigned to represent symbols with lower probabilities of occurrence. This strategy is used by *Huffman coding* [Huf52] and the more complex *arithmetic coding* [WNC87]. In practice, Huffman coders are best suited for cases when probabilities of occurrence are powers of $\frac{1}{2}$, while arithmetic coding encodes more efficiently in general. The textbook by MacKay [Mac03] provides good discussions of Shannon theory and entropy-based coding. Finally, the *bzip2* [Sew96] compression utility program uses Huffman coding internally, and in general it achieves better compression ratios than *gzip*.

2.2. Dictionary-based Coders

Dictionary-based coders work by matching symbols to a dictionary so only the matching information, but not the actual symbols, are stored. As an illustrative example, if we were to compress the current paragraph of text, a coder could store the page and index numbers of each word in a standard English dictionary. To achieve better efficiency, a dictionary-based coder usually constructs a dictionary for each compression task by using phrases occurring in the file to compress. This custom dictionary, in turn, must be stored as part of the compressed representation; smart ways to construct and store this dictionary can minimize this overhead.

Lempel and Ziv are pioneers in developing dictionary-based coders with their algorithms *LZ77* [ZL77] and *LZ78* [ZL78]. These two algorithms have inspired many variants that enjoy very wide usage today for general compression purposes. One of the derived algorithms, *DEFLATE* [Deu96], is used by both utility program *gzip* [gzi] and compression library *zlib* [zli].

Compared to entropy-based counterparts, dictionary-based coders tend to have a faster throughput, which is favorable for large scientific data compression. The fast throughput also leads to wide usage of dictionary-based coders in I/O middleware and filesystems to provide transparent compression. *HDF5* [FCY99] and *ADIOS* [LKS*08] are two I/O middleware solutions that support transparent compression with *zlib* among others as plug-ins [BLZ*14]. Two filesystems, *Btrfs* [RBM13] and *ZFS* [BM] on Linux and Solaris operating systems respectively, also natively support transparent compression through dictionary-based coders such as *zlib* and *gzip* [btr, OS12].

2.3. Lossless Coders for Scientific Data

Researchers have also developed lossless compressors specifically for scientific data. These compressors often exploit coherence (i.e., adjacent data points have similar values) between data points from scientific data, which is not common in a general binary file.

Predictive coding is the dominant approach to exploit data coherence in a lossless setting. That is, the use of *predictors* to predict future values based on previously encountered ones. Predictive coding involves two steps: prediction and compression. In the prediction step, a good predictor would produce small *residuals*, i.e., the difference between the predicted and actual values. In the compression step, these small residuals are then much more friendly to compression, because they exhibit significantly lower entropy [ILS05]. We note that predictive coding is used in lossy settings as well, as we will describe in more detail in Subsection 4.3. That said, this subsection focuses exclusively on the lossless options.

FPC [BR09] is a predictive lossless compressor for double-precision floating-point values. After linearizing multi-dimensional data, it uses two hash table-based predictors, FCM [SS97] and DFCM [GVDB01], to predict each incoming value. FPC then picks the more accurate predictor for each individual value. The residuals are produced by applying an XOR operation between the predictions and the true values, resulting in zeros at the common bit positions. With predictions close enough, these residuals will have all zeros at the most significant bits. A run-length compression is then used to effectively compress each residual. Compared to general purpose compressors like gzip and bzip2, FPC achieves higher compression ratios on most data sets, and consistently is at least one order of magnitude faster.

Both *MPC* [YMHB15] and *FPcrush* [BMYH16] chain multiple "algorithmic components" together in their compression pipeline. These algorithmic components include pre-conditioners to facilitate compression (e.g., mutators, shufflers, etc.), predictors, and compression modules. As a result, both MPC and FPcrush are capable of achieving higher compression ratios than gzip and bzip2. MPC and FPcrush differ in their target usage scenarios. MPC was designed for massive parallelism on GPUs, so it uses algorithmic components that require almost no internal state. The GPU implementation of MPC has one to two orders of magnitude higher throughput than parallel gzip and bzip2 on CPUs. FPcrush was designed for real-time usages, so it employs algorithmic components that are all of linear complexity. FPcrush also uses an asymmetric design, meaning that it is much faster to decompress than to compress. As a result, it achieves compression throughput on par with parallel gzip and bzip2, but its decompression is one order of magnitude faster.

Research for the topic of lossless scientific data compression has advanced in a few interesting directions. *Fpzip* [LI06] used a *Lorenzo* predictor [ILRS03] that is able to take advantage of data coherence along multiple dimensions (the Lorenzo predictor is described in more details in Subsection 4.3.1). Fout and Ma [FM12] proposed an adaptive compression framework that switches between five individual predictors — polynomial, Lorenzo, FCM, DFCM, and Mean — to achieve significantly better rates at a cost of slower performance. *ISOBAR* [SJS*12] pre-conditions data by picking only "compressible" bytes to feed into a compressor, so the achieved throughput is much higher. *BLOSC* [bloa] and *SPDP* [Bur] are two compressors that are also available as filters for HDF5 data format, with BLOSC featuring a very fast throughput.

2.4. Discussion on Truly Lossless Techniques

With respect to the reduction properties from Section 1.2, truly lossless compressors provide fixed error (i.e., no error at all), but not fixed size. They do not reduce memory footprint, nor provide any progressive data access. Since this topic is relatively mature, very good references exist with more information, namely textbooks by Salomon [Sal04] and Sayood [Say12].

Truly lossless compression is most often used to reduce the I/O and storage costs when saving data. It can also be used to archive data for future use without any fear of possibly losing science. The compression efficiency of different techniques is easy to compare: whichever one that gives the smallest file size is the best. When compressing scientific data, which is usually very big in size, the processing speed is also vital. BLOSC stands out in this regard, with a throughput sometimes surpassing plain memory copy [blob].

Despite the availability of general purpose compressors, the compressors with scientific data in mind are often better options. Among them, fpzip is widely used for its overall excellent performance in both compression efficiency and speed. When it was tested on the Rayleigh-Taylor simulation data from the Miranda code, fpzip achieved a 3.7X data size reduction. Further, in terms of overall throughput — which included time spent on both compression and actual I/O — it achieved a 2.7X improvement compared to a simple `fwrite()` of raw data [LI06].

3. Near Lossless Compression

Techniques in this family rely on some form of discrete mathematical transforms to map a signal from the spatio-temporal domain into a new domain, typically expressed as coefficients of a set of basis functions. With transforms that possess the properties of *information compaction* and *decorrelation*, the signal's information content can be concentrated into a relatively small number of coefficients, i.e., coefficients that are significantly more important than others. After the transformation, the number of resulting coefficients typically matches the number of samples from the original signal (thus no data reduction), but the resulting coefficients are more easily compressed than the original signal. In addition to decorrelation and energy compaction, it is essential that the transforms are invertible, allowing mathematically perfect reconstruction of the original input from the transformed values. Common transformations used in this step include the Fourier transform, wavelet transforms [SN96], and cosine transforms [ANR74]. More specifically, the high dimensional forms of these transforms (2D, 3D, etc.) are mostly used for scientific data, which are often in 2D and 3D structured grids.

For integer values, a selection of transformations exist that are specifically designed to be invertible and map integers to integers using only integer arithmetic, thus avoiding rounding errors [CDSY98]. With floating-point values, which is the focus of this survey, arithmetic operations generally introduce rounding errors. While not necessarily bit-for-bit lossless, these errors are of the same magnitude as rounding errors from numerical simulations. Hence we refer to techniques in this family as "near lossless." We survey a few near lossless compressors in the remainder of this sec-

tion, and discuss lossy encoding and compression of coefficients from transformations further in Subsection 4.4.

3.1. Mitigating Rounding Errors in Practice

To minimize floating-point rounding errors, a transformation could use higher precision floating-point representations to carry out calculations, provided the memory and computational resources allow for it. As an example, Trott et al. [TMM96] converted single precision floating-point values to double precision to perform wavelet transforms. The authors then used a data-specific statistics file to facilitate Huffman coding on the resulting coefficients, ultimately achieving moderate compression ratios on 3D curvilinear grids. A handful of researchers have explored more sophisticated ways to reduce or eliminate the introduction of rounding errors. Usevitch [Use07] explored mapping floating-point values into 278-bit integers to completely eliminate rounding errors, and applied JPEG2000 encoding to the wide-precision integers. However, the memory and computational overhead was significant. Lindstrom mapped small blocks of floating-point values to fixed-point representations in his *zfp* compressor [Lin14], and devised a custom transform to efficiently operate on the fixed-point values. Zfp then uses a custom encoder that better takes advantage of the properties of the transform, achieving compression ratios on par with some of the best lossless compressors.

3.2. Discussion on Near Lossless Compression Techniques

Near lossless compression has similar properties as truly lossless compression: fixed error (at floating-point rounding error magnitude) but not fixed size. It does not reduce memory footprint nor provide progressive data access.

Near lossless compression is rarely the sole objective of a compressor. Rather, it is often an optional mode with lossy compressors. For example, the above mentioned zfp was designed for lossy compression, and its near loss compression mode does not show a clear advantage in compression when compared to fpzip, which achieves bit-for-bit lossless compression [Lin14]. A wavelet-based lossy compressor from the *VAPOR* [NC12] package also supports near lossless data reconstruction, but it does not reduce data size in this mode. In short, for use cases involving near lossless compression, the lossless solutions from Section 2 should also be considered.

4. Lossy Compression

This section surveys lossy compression techniques. With these techniques, the reconstructed data values approximate, but do not, in general, exactly match the original. The rationale behind this approach is that some visualization and analysis tasks can be relatively unaffected by using lossy compression; this contrasts with numerical simulations which cannot tolerate such inaccuracies.

The techniques in this section do not reduce the size of the reconstructed data set. If the original data set contains N entries, then the reconstructed data set will have also have N entries. The only way that the memory footprint can be reduced is if the reconstructed data set has a reduced precision (e.g., from 64- to 32-bit representation). As a result, this section's techniques are mostly oriented around reducing storage size.

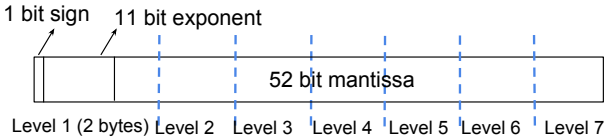


Figure 4: Illustration of the seven levels of detail for a double-precision floating-point value in MLOC [GRJ*12]. The first level consists of two bytes, and each successive level uses an additional byte. This illustration is inspired by [GRJ*12].

4.1. Truncation

Truncation is a widely used and simple technique for reducing data precision. For example, it is very common for numerical simulations to use double precision (i.e., 64-bit floating point values) in computation to maintain accuracy, while outputting single precision (i.e., 32-bit floating-point values) to save space.

MLOC [GRJ*12] truncates 64-bit floating-point values into multiple precision levels by dividing the eight bytes of a 64-bit value into seven groups, as Figure 4 illustrates. The first group of two bytes contain the exponent part and four most significant bits of the mantissa. These two bytes provide a very coarse approximation of the original value. Each additional byte increases the precision of the mantissa, thus improving the overall value precision. In fact, this organization provides a precision-based level of detail representation, which supports progressive data access.

Truncation has the advantage of being extremely simple and fast, and also provides fixed error and fixed size. It has the potential to reduce memory footprint, if the data type is changed. However, truncation is usually less effective at compressing data than other more complicated precision reduction techniques.

4.2. Quantization

Quantization reduces precision by mapping floating-point values to a finite set of approximations, which has a much smaller cardinality. We describe two flavors of quantization here: *scalar quantization* and the more complicated, yet powerful, *vector quantization*.

Scalar quantization: In its simplest form, scalar quantization provides an n-to-m ($n \gg m$) mapping for scalar values. An example is to represent all real numbers between 0.0 and 1.0 with a 256-step scale, which is often used for images: all values between $\frac{i}{256}$ and $\frac{i+1}{256}$ ($0 \leqslant i < 256$) are represented by the i^{th} step. Note that the binning scheme in this example uses the same width for every bin (step), which is effective for data with an uniform distribution. When the underlying data is non-uniform, the binning scheme can be adjusted to reflect the distribution, improving the distinguishing power between bins. In this regard, truncation can be seen as a form of non-uniform quantization, since floating-point representations have much more precision at small-magnitude values.

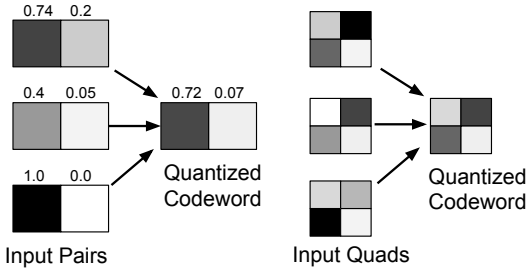


Figure 5: Vector quantization on vectors of size two (left) and four (right). The input vectors are pixels from a notional gray-scale image. The left illustration also includes gray-scale values of pixels.

Scalar quantization is often used to quantize coefficients coming out of transformations, such as the ones surveyed later in Subsection 4.4, since these coefficients tend to exhibit clear distribution patterns. Iverson et al. [IKK12] developed a compression technique for scientific data that has the same spirit of scalar quantization: sets (or regions) of the data that stay within an error bound are carefully chosen and represented by the mean values of those sets.

Vector quantization: Vector quantization extends scalar quantization to work on vectors. In this setting, a vector is not only from vector-valued data, such as velocity, tensor fields, or hyperspectral data, but also from groupings of adjacent values of the same field. For example, a 10×10 image could be organized into 25 vectors, each consisting of a 2×2 square of pixels.

Vector quantization operates by grouping vectors with similar values together, which means it takes advantage of similar patterns exhibited from vectors. All the vectors within the same grouping are then quantized to have the same value, which is the representative value of the group. The representatives are called *codewords*, and the mapping scheme is called a *codebook*. Figure 5 illustrates vector quantization of vectors of size two and four; they are pairs and quads of adjacent pixels from an artificial gray-scale image. In both cases, three similar vectors are mapped to a single vector, the codeword. That said, different vector sizes result in different compression ratios: a codeword encodes six pixels in the case of two-sized vectors, and twelve pixels for four-sized vectors.

In an n-dimensional vector space (for vectors of length n), input vectors are most likely distributed non-uniformly, thus a good binning scheme (codebook) non-uniformly allocates more codewords to dense regions. The process of finding a good codebook is then very much like finding a Voronoi-diagram-like partition for a space, and such algorithms including the LBG algorithm [LBG80] and Lloyd's algorithm [Llo82].

The computation for the encoding and decoding steps of quantization is usually asymmetric. Encoding is typically computationally heavy (finding a good codebook), and decoding is typically light (looking up a codebook). This property makes quantization suited for encode-once-and-decode-many-times use cases, for example, direct volume rendering [NH92]. Here contiguous blocks of size $I \times J \times K$ serve as individual vectors for quantization, and the decoding overhead was measured to be 5%. To achieve even faster rendering speed, the same authors pre-computed shading and ray tracing information for each of the $I \times J \times K$ blocks, and also applied vector quantization to compress them [NH93]. Overall, volume renderings from vector quantization have shown a significant advantage over scalar quantization results.

In general, quantization guarantees a fixed size after compression, but not a fixed error, although uniform scalar quantization guarantees both size and error. The decoded data is not reduced in memory footprint, nor is progressive data access supported. Though carefully designed codebooks improve its compression efficiency, quantization is most likely still not as good as some more complicated schemes. As a result, quantization is more often used as a component in other techniques. Finally, Gersho and Gray [GG91] have a good reference book on vector quantization, and Fowler has an open-source library, *QccPack* [Fow00] that supports many flavors of quantization.

4.3. Predictive Coding Schemes

The predictive coding schemes described in Subsection 2.3 can also be used for lossy compression. In principle, residuals from a predictor can be safely discarded if they are smaller than a pre-defined error tolerance, or get encoded otherwise. Since most predictive compressors also need to take into consideration cascading errors, predictions are made from previous approximations and residuals are calculated accordingly; this practice naturally results in a "fixed error" compression. To achieve efficient compression, the choice of predictors is often critical. The rest of this subsection surveys a few different classes of predictors.

4.3.1. Linear Prediction

A linear predictor is defined as a linear combination of previous values. In its simplest case, a linear predictor exhibits similarities to particle simulations where a particle remains at a fixed location, or travels along a straight line [EGM04].

Compvox [FY94] is a linear predictor that takes additional neighbor values into consideration. In the three-dimensional case, the following equation predicts a value at location (x,y,z):

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

The linear coefficients (a_1,a_2,a_3) are constantly updated as the scheme processes data to improve prediction accuracy. Further, the prediction errors in Compvox are efficiently coded with Huffman coding. In the lossless compression mode, Compvox achieves, on average, a 2:1 compression rate with several test data sets, improving on the widely used lossless compression tools gzip and zip.

The *Lorenzo predictor* [ILRS03] was proposed to compress scientific data of arbitrary dimensionality. This predictor works in units of n-dimensional cubes (squares in 2D, cubes in 3D, etc.). It uses known cube vertices to predict an unknown vertex:

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

Here, the vertex v is predicted to have value $E(v)$ using known values $F(u)$ from vertices of the same cube C. The degree between two vertices (i.e., the number of edges needed to traverse to reach

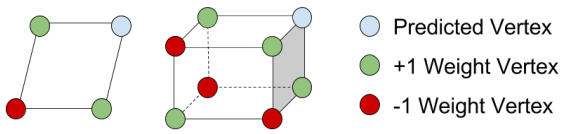


Figure 6: A Lorenzo predictor uses known values in a cube (green or red) to predict an unknown value (blue) for the 2D and 3D cases. Green vertices have $+1$ weight and red vertices have -1 weight. This figure is inspired by [ILRS03].

each other in a cube) is denoted as d_{uv}, and this degree determines the weight of each neighbor to be either $+1$ or -1. Figure 6 illustrates the weights for the 2D and 3D cases. The authors prove that, "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." *Fpzip* [LI06] uses the Lorenzo predictor and arithmetic coding for residuals. It is optimized to perform bit-for-bit lossless compression, as discussed in Subsection 2.3, though it supports lossy compression as well.

SZ-1.4 [TDCC17] is another compressor that utilizes a prediction model. It generalizes the Lorenzo predictor to larger cubes, i.e., cubes of size 3^n, 4^n, etc., where n is the dimensionality of the data. SZ-1.4 includes a quantization step when encoding prediction residuals, so it performs only lossy compression. The study by Tao et al. [TDCC17] showed that the compression efficiency and performance of SZ-1.4 are superior to fpzip.

4.3.2. Spline-fitting Prediction

Spline-fitting prediction tries to fit data points to a spline, and then uses this spline to predict future values. *ISABELA* [LSE*11] is a compressor that uses B-splines as predictors. ISABELA also includes a sorting step before prediction, with the reasoning being that a sorted array better fits a B-spline. This however requires indexing information for the sorted arrays to be decoded in their original ordering.

ISABELA naturally supports temporal compression by considering values over time at each location as its own data array to compress. It has then shown high performance in a storage framework to perform in situ compression [LSE*13]: each CPU core took less than one second to process ten million data points. Finally, ISABELA is also flexible enough to provide either fixed error or fixed size for compression.

4.3.3. Combination of Prediction Schemes

Multiple prediction schemes may be combined together to use in a single compressor product. Such compressors explore several prediction schemes for each prediction, and choose the best one for the corresponding data point. *SZ* [DC16] is a good example, using three prediction schemes: it predicts the next value from a linearized data stream to be 1) identical to the preceding one, 2) changing along a straight line, and 3) fitting a quadratic curve. A few compressors surveyed in Subsection 2.3 also adopt the strategy of combining multiple predictors, such as FPC [BR09] and the work by Fout and Ma [FM12]. To facilitate multiple predictors, these schemes often require reordering high-dimensional data into a linear fashion. Sophisticated data reordering can introduce storage overhead or otherwise lose information (e.g., the sorting used in ISABELA), but simple reorderings, such as using the intrinsic memory sequence adopted by SZ, does not have this concern.

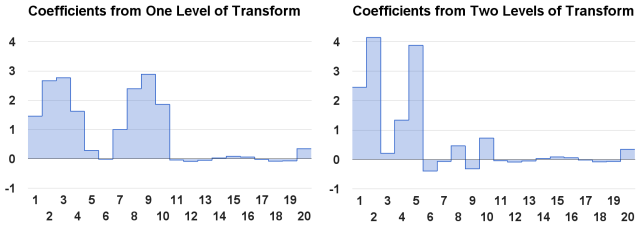


Figure 7: Wavelet coefficients from one (left) and two (right) levels of wavelet transform using the CDF 9/7 [CDF92] wavelet kernel. The X-axis is the index of each coefficient from 1 to 20, and the Y-axis is the actual value of each coefficient. The input array was a sine wave with length 20: $y = sin[x] + 1$ $(0 < x < 20)$.

4.4. Transform-based Compression Schemes

Transform-based lossy compression has the same spirit as near lossless compression (discussed in Section 3), where a transform is applied to compact information, followed by encoding the coefficients to achieve compression. In the transformation step, most techniques from this group require a structured grid as input, enabling 2D or 3D transforms to be applied. This subsection specifically considers lossy encoding. Most often these lossy coders take in a target size as input, then identify and keep the most important information until reaching the size budget. This results in a "fixed size" compression, with only a few exceptions.

After a transform, one of the most popular coding steps is to quantize floating-point coefficients to integers. Since the transformation concentrates coefficients, their integer representations will exhibit much less entropy. Classic lossless compressors can then be used to effectively compress these integers, for example, the entropy-based Huffman and arithmetic coding (see Subsection 2.1) and compression libraries gzip and zlib. Many applications, especially volume renderings, operate in a such fashion, using transforms such as cosine transform [YL95], wavelet transform [SSEM15, GS01, IP98a, KS99], and Fourier transform [TL93, WR00, CYH*97]. The rest of this subsection will discuss a few more modern compressor products in detail.

4.4.1. Wavelet Transform + Coefficient Prioritization

Wavelet transformations map input data (considered to be signals by wavelet theory) into the wavelet domain. With certain kinds of wavelets (orthogonal and near-orthogonal), the magnitudes of coefficients are proportional to their information content. The combination of these two properties, information concentration and magnitude proportional to content, means that only few wavelet coefficients will have large magnitudes, and the majority of the coefficients will be close to zero.

Figure 7 illustrates this information concentration property, by

plotting wavelet coefficients resulting from the transform of a sine wave ($y = sin[x] + 1$). The example used 20 data points as input, yielding 20 wavelet coefficients. The left subfigure plots wavelet coefficients produced by applying a wavelet transform directly to the sine wave samples, and the right subfigure plots coefficients produced by a second application of the transform, but this time applied to the coefficients resulting from the previous application. As is typical, each application exhibits more small-magnitude coefficients.

Coefficient prioritization is an approach for achieving data reduction; it prioritizes and saves the coefficients with the most information content, and discards the remaining ones (treated as as zeros). The coefficients are stored verbatim, thus any coefficient that is not discarded is preserved exactly. Coefficient prioritization essentially reorders the coefficients by sorting, which in turn incurs storage overhead to preserve the indices of the surviving coefficients. This can be done by a so-called "significance map," which faithfully records the index of each surviving coefficient.

VAPOR [NC12] is an open-source visualization package for the geoscience community that adopts the wavelet transform + coefficient prioritization strategy to achieve compression. This strategy has been shown to be effective in compressing turbulent flow data, as meaningful visual analytics can still be carried out even with a 256:1 compression ratio [LGP*15]. Performance-wise, VAPOR takes 14.2 seconds to compress a 4GB data set, and 3.8 seconds to decompress it on a 20-core machine [LGP*15]. Though simple to implement, this encoding scheme is somewhat expensive because of the need to sort all coefficients. Further, an approach to compute wavelet transform + coefficient prioritization using data parallel primitives achieved portable performance on multiple hardware architectures [LMC*17]. This approach demonstrated wavelets to be viable as an in situ compressor, with improved overall I/O (i.e., computational + actual I/O cost) on simulations taking more than 512 compute nodes [LLCC17]. Finally, the storage overhead introduced by significance maps is a factor that varies depending on the compression ratios, and can take up to 35% space of the reduced data format [LGP*15].

Wavelet compressors can also naturally accommodate time-varying sequences. This is because wavelet transforms are essentially one-dimensional operations and are easy to apply along every dimension, including the time dimension. Villasenor et al. [VED96] was among the first to apply this technique: they applied the CDF 9/7 one-dimensional wavelet filter bank over all three spatial dimensions and the temporal dimension to compress seismic reflection data. Li et al. [LSO*17] also applied 4D wavelet compression to scientific simulation data, and demonstrated an approximately 2X benefit over 3D wavelet compression with respect two visual analytics tasks. Finally, interested readers can learn more on wavelet transforms from the textbook by Strang and Nguyen [SN96].

4.4.2. Wavelet Transform + Advanced Coders

A handful of coders are specifically designed to encode wavelet coefficients. These coders represent the latest and most advanced development in this domain; they consistently achieve the highest compression ratios at a given accuracy (usually expressed as average errors) among all wavelet-based compression techniques. This section introduce how these coders work at a high level.

Wavelet transform + advance coders work in three steps: 1) wavelet transform, 2) quantization, and 3) encoding. In the first step, these techniques typically use a particular kind of wavelet kernel, namely the CDF 9/7 [CDF92] wavelet kernel. This is because the CDF 9/7 kernel possesses a number of properties that make it well suited for compression on a variety of data. In the second step, the real-valued wavelets coefficients are mapped to integers via quantization. The integer representations not only exhibit much less entropy, but also enable the concept of *bit planes*, whereby the most significant bits from all coefficients are represented in the most significant bit plane; all second most significant bits are represented in the second bit plane; and so on. In the third step, these coders encode bit plane by bit plane, rather than coefficient by coefficient. There is no storage overhead introduced by any of the three steps, thus the advanced coders achieve high compression efficiency. Also, this class of techniques are strictly lossy because of the intrinsically lossy nature of quantization in step 2.

Specific data structures are used to encode bit planes from the most to the least significant ones. These data structures exploit the spatial self-similarities of coefficients to achieve a high encoding efficiency. The encoded results are in the form of *bitstreams*, where the more significant bit planes are at the beginning, and the less significant bit planes are at the end. Note that, due to the compaction properties of wavelet transforms, only a few coefficients will have non-zero bits in the more significant bit planes. The address of each coefficient is implicit in the encoding scheme and does not require explicit recording. Starting from the beginning of the bitstream, any number of bits is able to reconstruct the data, enabling progressive data access. The reconstruction quality varies too: the more bits passed to the decoder, the more accurate the resulting reconstruction.

Common advanced encoding algorithms include ZeroTree [Sha93], Set Partitioning in Hierarchical Trees (SPIHT) [SP93], Set Partitioned Embedded bloCKs (SPECK) [IP98b, PINS04], Subband-Block Hierarchical Partitioning (SBHP) [CSD*00], and Embedded Block Coding with Optimized Truncation (EBCOT) [Tau00]. Among them 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], SBHP [LP06], and EBCOT [XXLZ01], and 4D versions of ZeroTree [ZJM*02], SBHP [LP07], and SPIHT [ZLMS04, LBMN05]. Wavelet coefficient coding is a big topic itself, and this survey includes only some representative and well-studied techniques. Other similar techniques are reported in [NS01, Rod99].

QccPack [Fow00] is an open-source package that implements the 2D and 3D versions of two of the advanced wavelet coders: SPIHT and SPECK. A study [BXH*17] on a climate data set with hundreds of variables confirmed that QccPack with SPECK has a superior efficiency to fpzip. The study found that, for most variables, SPECK achieves higher compression ratios than fpzip.

A few JPEG2000-compliant libraries are also available for floating-point compression tasks. This includes three reference implementations: *OpenJPEG* [ope], *JasPer* [jas], and *JJ2000* [jj2].

Another JPEG2000-compliant library, *Kakadu* [kak], was used by Woodring et al. [WMB*11] on POP (Parallel Ocean Program) data. Here, the standard two-dimensional JPEG2000 compression was applied on individual 2D layers of 3D variables, meaning coherence along the third axis was not exploited. This study demonstrated that JPEG2000 was effective in trading accuracy for smaller sizes to transmit data through the Internet.

4.4.3. Custom Transform: Zfp

Zfp [Lin14] is a newly emerging compressor, designed specifically for floating-point data. It uses a custom orthogonal block transform, which operates on blocks of 4^d in size (where d is the dimensionality), and makes use of a lifting scheme [DS98] in its implementation. Coefficients produced by this transform are then encoded from the most to the least significant bit planes, sharing the similar idea as coders surveyed in Subsection 4.4.2.

Zfp has a few unique characteristics. First, zfp has better random access capabilities because of its relatively small operational blocks: 4^3 in 3D cases. Many types of transform, including wavelets, would be ineffective on blocks this small. Second, calculations in zfp are relatively simple and thus inexpensive. It is reported to achieve 400MB/s in raw throughput [Lin14], and 3X to 6X I/O improvement when used as an in situ compressor, compared to writing/reading raw data [LCL16]. Finally, the newest release of zfp has the ability to operate in a "fixed error" mode in addition to the more traditional "fixed size" mode (as reported in [LCL16]).

4.4.4. Karhunen-Loéve Transform

The Karhunen-Loéve Transform (KLT) [Loè78] can be optimal in terms of both its energy compaction properties and its ability to decorrelate. Its basis functions are the eigenvectors of the covariance matrix Σ of its input signal. However, the basis functions are data dependent, and require recomputation for each new set of data. Moreover, the computational complexity of the eigenvalue problem of the covariance matrix is $O(N^3)$, while the complexity for other orthogonal transforms with a fixed set of basis functions matrix is at most $O(N^2)$. Though most likely impractical to use for general-purpose compression, it is used in specific settings such as hardware-accelerated volume rendering [FM07]. Also, it serves as a benchmark for comparison with other transforms. A detailed description of the KLT and its properties is available in the book by Wang [Wan12].

4.4.5. Tensor Decomposition

For the purpose of tensor decomposition, tensors are generalizations of higher order matrices (e.g., a 3D volume). Using matrix singular value decomposition (SVD), tensors can be decomposed into smaller approximations. One such tensor approximation is the Tucker decomposition, which is the focus of this subsection. The treatment by Kolda and Bader [KB09] has more detail on other forms of decomposition.

Tucker decomposition considers the input data, $\mathcal{X}$ of N-dimensions with $N \geqslant 3$, a tensor. The algorithm computes a least squares fitting of $\mathcal{X}$, resulting in a set of N basis factor matrices and an N-dimensional core tensor, $\mathcal{G}$. As an example, let $N = 3$ and

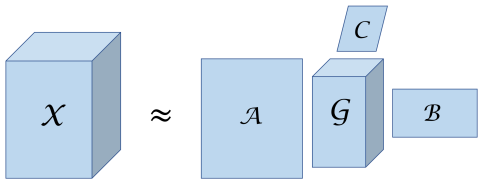


Figure 8: A 3D illustration of the Tucker decomposition. Given the input data/tensor $\mathcal{X}$, the least square fittings results in a core tensor $\mathcal{G}$, and three basis factor matrices $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$. $\mathcal{G}$ is considered an approximation of $\mathcal{X}$. Image inspired by [KB09].

let the input data $\mathcal{X}$ have the dimensions $\mathcal{R} \times \mathcal{S} \times \mathcal{T}$. Tucker decomposition produces three basis factor matrices, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, of sizes $\mathcal{R} \times \mathcal{L}$, $\mathcal{S} \times \mathcal{M}$, and $\mathcal{T} \times \mathcal{N}$, respectively, and a core tensor $\mathcal{G}$ of size $\mathcal{L} \times \mathcal{M} \times \mathcal{N}$. Figure 8 illustrates this decomposition. If $\mathcal{L} < \mathcal{R}$, $\mathcal{M} < \mathcal{S}$, and $\mathcal{N} < \mathcal{T}$ the core tensor $\mathcal{G}$ can be considered a compressed version of $\mathcal{X}$.

The basis factor matrices are the mapping to and from the original and reduced data. Unlike other tensor-based approaches with pre-defined basis factors, the Tucker decomposition results in basis factors that are data dependent.

The work by Ballester-Ripoll and Pajarola [BRP16] aimed to improve the the standard Tucker decomposition. They first presented core truncation, where they discarded the least significant ranks, or segments of the basis function matrices, and corresponding segments of the reduced core tensor. Alternatively, the authors also presented core thresholding, where only core coefficients with the smallest norm are removed, instead of the entire slices of the core tensor. This approach results in an improved compression ratio, but at the expense of longer reconstruction time. For parallel applications, Austin et al. [ABK16] were the first to utilize Tucker decomposition in a distributed-memory environment. Their method avoids data redistribution both locally and globally, and achieves performance improvements for both weak and strong scaling.

4.5. Discussion on Precision Reduction Techniques

Lossy compression is versatile enough to support many application scenarios, including both reducing I/O and storage cost for simulations and mitigating I/O constraints for visualizations and analysis (through progressive data access). There are many available lossy techniques to choose from. To help inform tradeoffs between different lossy compressors, we summarized their key properties in Table 1. Note this table only reports on properties found in public software; if a technique is reported in a research paper but not released for public availability, then we do not include it here.

Table 1 could help narrow down compressor options. For example, mesh type is an important factor to consider: while all these techniques support structured meshes, only five of them also support unstructured meshes (the first five). Requirements of either fixed error or fixed size, and whether progressive data access and lossless option are desired, could also help locate a compressor.

Recommending a method is difficult. A few rules of thumb are: 1) transform-based techniques (Subsection 4.4) are likely to

Table 1: Summary table of lossy compressors surveyed in Section 4.

Software Package	Lossless Option	Fixed Error	Fixed Size	Progressive Data Access	Supported Grid Type	Speed	Compression Efficiency	Section
MLOC	Yes[b]	Yes	Yes	Yes	(Un)structured	Fast	Low	4.1
Quantization	No	No	Yes	No	(Un)structured	Varies[d]	Low	4.2
Fpzip	Yes[b]	Yes	No	No	(Un)structured	Fast	High	4.3.1
ISABELA	No	Yes	No	No	(Un)structured	Medium	Medium	4.3.2
SZ	No	Yes	No	No	(Un)structured	Fast	High	4.3.3
SZ-1.4	No	Yes	No	No	Structured	Fast	High	4.3.1
VAPOR[a]	Yes[b]	No	Yes	Yes	Structured	Fast	High	4.4.1
QccPack[a]	No	No	Yes	Yes	Structured	Slow	High	4.4.2
JPEG2000	No	No	Yes	Yes	Structured	Varies	High	4.4.2
Zfp	Yes	Yes	Yes[c]	Yes	Structured	Fast	High	4.4.3

[a] This occurs when VAPOR uses the coefficient prioritization strategy, and when QccPack uses wavelet transform + advanced coders.
[b] MLOC and fpzip support truly lossless data reconstruction, while VAPOR and zfp supports near lossless data reconstruction.
[c] Zfp has the option to work on either fixed size or fixed error mode.
[d] Decompression is much faster than compression. Quantization is available from many libraries, including QccPack in this list.

yield the least average error (i.e., in terms of root-mean-square error (RMSE)) at a given compression level; 2) predictive coding schemes (Subsection 4.3) are good at providing a relative or absolute error guarantee; and 3) quantizations are useful primarily for very aggressive reduction to a small number of bins. Even with this knowledge, a user would almost certainly need to choose the technique on a case-by-case basis, depending on the specifics of the data and requirements of the intended analysis. This is because there are numerous factors to consider. For example, even if one transform-based technique yields the least RMSE, it may introduce large deviations to a small number of data points (i.e., large L^∞-norm). Moreover, factors such as the specific data and even a compressor's parameters can all have an effect on its performance. For example, high-dynamic range data may be treated better by one method than by another.

Besides the properties discussed above and those from Table 1, other considerations might include:

- will it work with "missing data" indicators that are present in many scientific data sets;
- how well does it support random data access;
- does it have restrictions on data sizes (e.g., powers of two);
- does it well support acceleration devices (e.g., GPUs); and
- how much parameter tuning is required for a "good" result?

Finally, for two specific uses: compressed GPU direct volume rendering and in situ systems, the survey papers by Rodríguez et al. [RGG*13] and Bauer et al. [BAA*16] provide more insights.

5. Mesh Reduction

This section surveys techniques that reduce data size by changing the underlying mesh to have fewer vertices and polygons. These techniques reduce the mesh in different fashions, but they all aim to remove redundancies or unimportant regions but keep the essence of the mesh or certain regions of interest. Most mesh reduction techniques benefit data consumers (e.g., visualization and analysis scientists), with the exception of temporal sampling which is mostly used by simulation scientists.

When operating on individual time slices, these techniques produce results that consume less memory when doing post hoc analysis, which is often desirable by visualization and data analysis scientists. This contrasts with the precision reduction techniques from the previous section, which mostly produced results that consumed the same amount of memory. In addition, many techniques in this section support progressive data access, so meshes are accessible with different reduction levels. Both properties are important for interactive data explorations. Mesh reduction can be viewed as providing fixed size, since the reduced size can be known in advance. With some techniques we also know what is lost during a reduction, so they can also be viewer as providing fixed error too. Finally, the reduced mesh typically has its data values at their full precision. That said, mesh reduction can also be used together with precision reduction on data values.

5.1. Mesh Decimation

Mesh decimation operates on unstructured meshes. Typically, it uses various criteria to eliminate vertices, edges, or cells, resulting in a new mesh with similar geometric and/or topological structures that is simplified has fewer vertices. After reduction, a reduced mesh can be evaluated using several different criteria, and some error metrics describing the differences after reduction have been captured in [CCM*00]. In the literature, triangle and tetrahedral meshes are considered most often. The rest of this subsection surveys a few techniques to achieve mesh decimation; among them edge collapse (Subsection 5.1.2) methods tend to provide the highest quality and finest granularity. A separate survey that covers this topic in more detail is available at [WDF11].

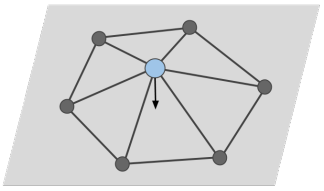


Figure 9: A candidate vertex (in blue) for removal is evaluated by its distance to the average 2D plane of its neighboring triangles. This illustration is inspired by [SZL92].

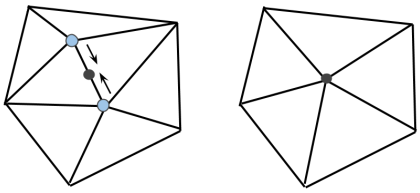


Figure 10: An illustration of edge collapse. In the original mesh (left), an edge candidate, indicated by two blue vertices, is selected to collapse down to the black vertex, and then the edges adjacent to the candidate are merged (right).

5.1.1. Vertex Removal and Tessellation

Vertex removal and tessellation is a classic mesh decimation algorithm that deletes vertices and then performs local tessellations to fill in the resulting holes. There are several criteria to select vertices to remove. One notable criteria is "distance-to-plane," as described by [SZL92]. The intuition behind this approach is to eliminate extraneous vertices that form flat planes, because they do not contribute much to the geometry of the mesh. For example, if a vertex is surrounded by a cycle of triangles, and it is "reasonably" close to the average plane of this group of triangles, this vertex can be prioritized for removal. Figure 9 illustrates this case. There are other vertex removal criteria that apply to higher dimensions as reported in [RO96].

5.1.2. Edge Collapse

Edge collapse works by repeatedly collapsing edges to vertices, and merging the resulting coincident edges. Figure 10 illustrates a basic edge collapse. Trotts et al. [THJW98, THJ99] picked candidate edges based on predicting the resulting deviations of local triangles and tetrahedra if they were collapsed. Garland and Heckbert [GH97] used quadric-based error metrics (QEM) to guide the edge collapse process that has provided low mean error mesh deviation. They then later extended this work to higher dimensions [GZ05]. Lindstrom and Turk [LT00] proposed image-driven simplification, which uses images of the original model against those of a simplified model to determine the cost of an edge collapse. This approach is particularly helpful for applications that require the simplified model to be visually similar to the original one.

Mesh decimation through edge collapse was explored as an out-of-core method by Wu and Kobbert [WK03] and by Vo et al. [VCL*07]. Both applications read the mesh from disk in a streaming fashion following a reasonably coherent ordering. Then the edges are examined against an error tolerance, and they are collapsed provided doing so will not introduce error beyond the tolerance. The latter approach [VCL*07] adds an additional property that the stream order of the mesh is preserved from input to output, which can be useful for additional processing afterwards. Finally, Pajarola and Rossignac [PR00] proposed a compressed progressive mesh (CPM) format, which encoded and transmitted refinement information in batches. The authors demonstrated that the encoding algorithm used in CPM format yielded higher compression ratios than many other formats.

View-dependent rendering in visualization is enabled by progressive meshes via edge collapse. The idea is to include the current view as one of the criteria to decide if an edge gets collapsed or not, for example, a "zoomed out" view would trigger edge collapse to speed up rendering. This technique is used in terrain rendering [Hop97, Hop98] as well polygonal model rendering [XV96, KLK04, CGG*04].

5.1.3. Vertex Clustering

Vertex clustering simplifies a mesh by merging a cluster of vertices into one. Rossignac and Borrel [RB93] used a single representative vertex to replace multiple mesh vertices within a cell, using a uniformly subdivided volume to partition a 3D mesh. Shaffer and Garland [SG01] proposed an improvement to this method by using adaptive space partitioning based on the analysis of the mesh.

Vertex clustering-based out-of-core mesh decimation was explored by Lindstrom and Silva [Lin00, LS01]. In their first 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 over the input mesh to generate the simplified version. While this approach initially required the system memory to hold the simplified version of the mesh, their second work removed this requirement. This new version of the algorithm compactly stored auxiliary information on disk instead, and managed to avoid expensive random accesses on disk.

5.2. Multi-resolution Techniques

Multi-resolution approaches build a hierarchy of data with different resolutions. Each version in the hierarchy is often referred to as a level. Typically, the finest level captures the native resolution of the data, and all other levels have successively lower (coarser) resolutions. Each of the lower resolution levels is an approximation of the original data, as they have fewer data points. We distinguish this category from the previous multi-resolution mesh decimation techniques because the methods in this subsection do not focus on preserving topology or connectivity, such as reducing point cloud data.

5.2.1. Sample Sets

Sampling is the easiest technique for providing a multi-resolution hierarchy. This approach treats the data set as a population and statistically samples points or cells from it. If the multi-resolution hierarchy has n levels, then the data set is sampled $n-1$ times, once for each lower-resolution level. Pre-calculating and storing sample sets (resolution levels) separately introduces significant storage overhead, which is referred to as "redundant" multi-resolution in some literature. As a result, some applications have opted to discard the native resolution level to reduce storage size [WAF*11].

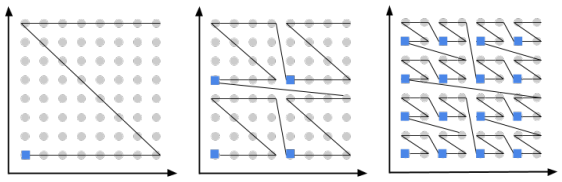


Figure 11: An example of several hierarchical Z-order curves traversing a 8×8 plane, providing three resolution levels. In each subfigure, data points in blue squares indicate the start of a Z-curve, and each turning point of that Z curve contributes to the current resolution level. From left to right, the three resolution levels are 2×2, 4×4, and 8×8. This illustration is inspired by [Cly12].

Smarter sampling can be more effective than simply uniform sampling. For example, Woodring et al. [WAF*11] used statistical sampling for interactive visualization and analysis. However, storage overhead is still a significant concern, which motivates the multi-resolution techniques discussed in the following sections.

5.2.2. Hierarchical Space-filling Curves

Hierarchical space-filling curves share the spirit of sampling for multi-resolutions but do not repeat any data point between resolution levels, resulting in "non-redundant" multi-resolution. Mathematically, space-filling curves are connected line segments that visit every data point in an n-dimensional space exactly once to create an index map that maps the n-dimensional space to a linear, one-dimensional curve. Given this linear mapping, data points along the curve can be hierarchically segmented into various multi-resolution levels. This allows data points from the same resolution level to be linearly organized together to achieve data locality, and also contribute to the reconstruction of higher resolution levels to avoid storage overhead. The most prominent space-filling curves include Z-curves (also known as Morton curves) [Mor66] and Hilbert curves [Hil91]. Data reordering using space-filling curves usually does not incur storage overhead, nor does it cause any information to be lost.

Figure 11 illustrates how Z-order curves traverse a 2D space to provide multiple resolution levels. These Z-shaped curves traverse a matrix to create lower resolution levels, and the curves are hierarchically organized such that data points from the coarser levels will automatically contribute to the finer multi-resolution levels, resulting in the same storage size as the original data. More details of how high-dimensional Z-order curves are segmented and mapped into hierarchical representations is discussed in [PF01].

Space filling curves are often used for progressive data access. MLOC [GRJ*12], which was described in Subsection 4.1 for its use of truncation, also uses Hilbert curves to organize data in a multi-resolution fashion to increase system interactivity. Pascucci and Frank [PF01] used Z-curves in an out-of-core scenario, where finer resolution levels were accessed as needed from disk. In this work, the authors demonstrated real-time interactions on a single server with a slicing operation on an $8,192^3$ grid. This work was later integrated into the *ViSUS* visualization framework [PSS*12].

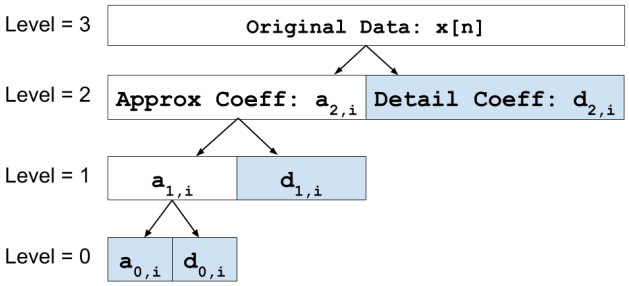


Figure 12: Hierarchical representation of wavelet coefficients after three passes of wavelet transform. Blue blocks are coefficients to be saved; in total they take the same amount of storage as the original data. Reconstruction goes from bottom to the top, and could stop at any intermediate level to get an approximation.

5.2.3. Wavelet-based Multi-resolution

Wavelet transforms are also frequently used to construct multi-resolution hierarchies. This is possible because wavelet transforms use band-pass filtering to transform input signals to the wavelet domain, and low frequency components can be used to reconstruct an approximation at a lower resolution. At the same time, low frequency components and high frequency components can be combined together to reconstruct a higher resolution level, resulting in "non-redundant" multi-resolution. In general, this strategy also achieves superior lower-resolution approximations compared to those from space-filling curves, since each data point is generated with consideration of its neighbors.

The wavelet transforms discussed here are the same as those in Subsection 4.4, although they are being used in a different fashion: to construct multi-resolution hierarchies. Multi-resolution hierarchies with wavelets work as follows. Wavelet transforms take a signal (data set) and represent it as a linear combination of a set of basis functions. The basis functions have two types: *scaling* functions, which capture the low frequency signal, and *wavelet* functions, which capture the high frequency signal. Coefficients with these two types of functions are *approximation* and *detail* coefficients, respectively. As their names suggest, they capture an approximation and the deviations of this approximation from the original data.

The wavelet transform is recursively applied to the approximation coefficients from the previous application of the transform, creating a hierarchy of coefficients, and resulting in a multi-resolution representation of the input signal. In the 1D case with a discrete signal $x[n]$, the following equation represents the linear combination of several basis functions with J lower-resolution levels:

$$x[n] = \sum_i a_{0,i} \cdot \phi_{0,i}[n] + \sum_{j=0}^{J-1} \sum_i d_{j,i} \cdot \psi_{j,i}[n]. \quad (4)$$

Here, $j = 0$ denotes the coarsest level, while $j = J$ denotes the finest level, which has the original resolution of $x[n]$. Scaling and wavelet basis functions are denoted as $\phi_{j,i}[n]$ and $\psi_{j,i}[n]$, respectively. The respective approximation and detail coefficients are $a_{j,i}$ and $d_{j,i}$. Figure 12 illustrates a multi-resolution hierarchy after three passes

of the wavelet transform ($J = 3$). The approximation coefficients — $a_{2,i}$, $a_{1,i}$, and $a_{0,i}$ — represent three different resolutions. Only $a_{0,i}$ needs to be kept in storage, while $a_{1,i}$ and $a_{2,i}$ are calculated on-demand from stored detail coefficients $d_{0,i}$ and $d_{1,i}$. This hierarchical fashion of storing coefficients and reconstructing the original signal enables progressive data access for wavelet-based multi-resolution.

Multi-resolution representations are especially useful in interactive visualization scenarios. Ihm and Park [IP99] evaluated the interactive use of their wavelet-based technique on the Visible Human data set [Ack98]. 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 interactive visualization uses finer resolutions to render objects closer to the viewpoint, while using coarser resolutions for objects further from the viewpoint. Guthe and Strasser [GS04] applied different resolution levels to different data blocks when rendering, aiming to use higher resolution levels on blocks that are more error sensitive. Further, Brix et al. [BMMB11] and Gao et al. [GWLS05] applied multi-resolution representations from wavelets on distributed systems. Both works use space-filling curves to achieve a better load balance among compute nodes. In another example, Weiss and Lindstrom [WL16] developed a method for level-of-detail surface generation, that preserves C0 (crack-free) continuity, using wavelet and octree data hierarchies. Finally, a wavelet-based multi-resolution technique is also adopted by VAPOR, an open-source package for scientific visualization and analysis, to provide better user interactivity [CMNR07, CR05].

5.2.4. Miscellaneous

There are several other techniques for providing multi-resolution data. Laplacian pyramids, which is a redundant multi-resolution representation, have been used for direct rendering of computed tomography data [GY95]. Curvelets [CDDY06] and surfacelets [LD07] are newer multi-resolution representations, similar to wavelets. They attempt to capture higher-dimensional features at lower resolutions, such as edges for curvelets and signal singularities for surfacelets, at the cost of introducing redundancies. The evaluation of these two techniques, as applied to turbulence data, can be found in [PLW*16]. Further, there are additional coding and processing schemes for multi-resolution processing with GPU-based direct volume rendering; these schemes are surveyed by Rodriguez et al. [RGG*13].

5.3. Subsetting

Subsetting is a simpler approach to change the mesh: it uses a portion rather than the entirety of the mesh. Subsetting is lossy when a subset of data is saved and the rest discarded. However, subsetting most often happens at the data retrieval phase, where subsets are read from disk for processing. We survey two types of subsetting: domain subsetting and query-based subsetting.

5.3.1. Domain Subsetting

Large simulation data is often divided into many domains (also known as blocks or pieces), with each domain separately accessible on disk. For example, a space with $4,096^3$ grid points could be divided into 64 domains of size $1,024^3$. Domain subsetting then allows algorithms to read and operate on only the subset of domains needed for a given analysis. Typically, the metadata for each domain can help decide if that domain is needed without actually reading the domain's content. Examples of domain subsettings include using per-domain scalar ranges to choose domains for isosurface calculations, and using per-domain spatial extents when slicing a volume.

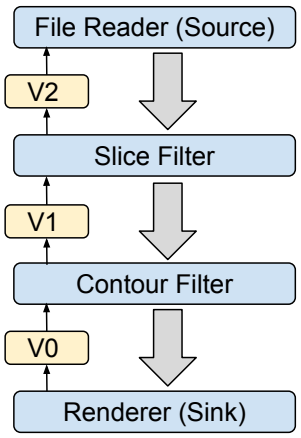


Figure 13: An example data flow network. Prior to execution, a contract that contains a list of domains to process is generated using all network constraints. The initial version of the contract, V0, comes from the renderer. V0 is then an input to the contour filter, which modifies the contract to make V1, and so on. V2 contains the intersection of the domain lists from the slice and contour filters. These domains are the only ones used during execution (thick gray arrows). This figure is inspired by [CBB*05].

VisIt [CBW*12], an open-source visualization system for scientific data, uses domain subsetting for its visualization pipeline. VisIt introduces a concept of a *contract*, which enables every filter in a pipeline of visualization operations to specify the list of domains that component requires [CBB*05]. When an update is issued at the sink of a pipeline, the contract travels upstream, from filter to filter, to inform each filter of the downstream filters' requirements, and allow the current filter to add its own requirements. Finally, the source at the beginning of the pipeline receives a contract containing all filters' requirements, and then reads in only the required domains. Figure 13 illustrates a visualization pipeline with four components. After each filter has updated the contract (upward arrows), the visualization pipeline starts execution (downward arrows) using only the subset of domains indicated by the contract. The benefits from reduced data size then propagate downstream to every filter.

5.3.2. Query-Based Subsetting

Query-based subsetting uses queries to find pieces of data that meet specific criteria. Queries are a form of subsetting, in that they enable loading portions of data rather than the entirety. Query-based subsetting borrows fast query techniques from database management systems to enable data access in a query style, and they are often employed in analysis scenarios where explorations are based on ad hoc queries.

An important note on query-based subsetting is that the query-enabled data set, which includes the original data and auxiliary information for querying, actually incurs storage overhead. Data is reduced during the data retrieval steps, with the reductions benefiting post hoc visualization and analysis (and not data storage).

	bitmap index			
A	b_0	b_1	b_2	b_3
0	1	0	0	0
2	0	0	1	0
3	0	0	0	1
3	0	0	0	1
1	0	1	0	0

Figure 14: A basic bitmap for a variable A that can take four distinct values: {0, 1, 2, 3}. Each row represents an occurrence of A; there are five occurrences in total. Queries can be answered by bitwise operations, for example, the query "$A < 2$" is answered with a bitwise OR on bits b_0 and b_1. This figure is inspired by [WAB*09].

Bitmap [O'N89, CI98] is an indexing technique that enables fast queries. With this technique, all of the values in a data set are organized by their corresponding bits. Queries are performed via bitwise logical operations, which are very fast on modern processors. Figure 14 illustrates a basic bitmap for a data set with four values: $\{0,1,2,3\}$. A limitation of bitmaps is that they are most effective on variables with low cardinality, i.e., a small number of distinct values.

Tree-based schemes from the database management community are also used for multi-dimensional data query. Such schemes include R-tree [Gut84], with its variant R*-tree [BKSS90] and R+-tree [SRF87], and B-tree [BM72], with its variant BBIO tree [CFSW01]. B-trees have been reported to have better performance for indexing unstructured tetrahedral meshes [PAL*06].

Bin-Hash [Gos08] is a data structure that aims to utilize massive parallelism for fast querying. Queries on this data structure are designed to favor GPU devices by consuming less memory at the cost of incurring extra computation. Bin-Hash has been used in query-driven visualizations (QDV) including an application on time-varying adaptive mesh refinement data [MGA*08] and another on extending QDVs with statistical analysis capabilities [GGA*11].

FastBit [WAB*09] is an open-source library for performing fast queries on scientific data. Its core querying technology is Bitmaps. In one application, FastBit was used to track the evolution of ignition kernels in a combustion simulation over three steps [WKCS03] — finding individual points that meet the conditions for ignition, grouping those points into connected components, and tracking the evolution of ignition over time. A separate application of FastBit, histogram-based parallel coordinates, proved to have good performance in exploring extremely large data ($\approx$ 1.5TB) on distributed-memory systems [RWC*08]. In this study, interactive query tasks and particle tracking tasks both saw 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 supported queries for certain data points using query criteria like "$Temperature > 32$" and "$CO_2 > 0.1$." Performance-wise, HDF5-FastQuery has demonstrated a consistent advantage over the R*-tree query, another query technology available from HDF5.

5.4. Temporal Sampling

Temporal sampling is used to reduce I/O cost for numerical simulations — all available time slices are sampled so only a subset of them get saved to disk. Sampling strategies are highly dependent on applications and analyses, with common ones including "save every i^{th} time slice," "save one time slice for every t_1 units of time simulated," and "save one time slice for every t_2 minutes of time running on a computer." Temporal sampling is used in almost all numerical modeling settings.

Recent research has also proposed techniques to choose time slices intelligently, rather than at evenly-spaced intervals. One idea is to save all time slices temporarily, and then choose the most representative ones from them [TLS12]. Though capable of choosing globally optimal time slices, this approach incurs significant I/O and computational burden. Another idea is to detect interesting events in situ as the simulation runs, and save the time slices when these events occur. Myers et al. [MLF*16], Ling et al. [LKA*17], and Salloum et al. [SBP*15] have used statistical methods, machine learning algorithms, and a sampling-based approach to trigger time slice savings, respectively. The specific event detection approach most likely needs to be tailored to each simulation, and the computational overhead that is imposed on the simulation needs to be carefully considered [SBP*15, LKA*17].

Temporal sampling is distinct from other mesh reduction techniques surveyed in this section in two ways. First, while previous techniques are mostly used by data consumers, i.e., visualization and data analysis scientists, temporal sampling is mostly used by data producers, i.e., simulation scientists. Second, while previous techniques mostly operate on spatial domains of single time slices, temporal sampling operates on the temporal domain with a series of time slices. In fact, time slices from temporal samplings can go through additional data reduction operators, maximizing the reduction factor. That said, temporal sampling fits as a mesh reduction technique, since it results in a coarser resolution, albeit in the time dimension.

5.5. Discussion on Mesh Reduction Techniques

In selecting a mesh reducing technique, the dimensions to consider are: time to reduce (latency on write), time to reconstruct (latency on read), total storage size (I/O bandwidth required on read and write), capability for progressive writing, capability for progressive reading, reconstruction error (lossy or lossless), and mesh type. In many cases, techniques are not suitable because they do not match user requirements. We organize our discussion for this topic around two considerations: reduction/reconstruction and resources.

How the data set will be reduced and how the data set will be reconstructed are important considerations for choosing a mesh reduction technique. In situ scenarios will likely place higher priority on the total time to reduce the data (the sum of compute plus I/O) to minimize the time taken away from the simulation. Alternatively if post hoc interactivity is a more important factor, then higher times can be tolerated in preparing the reduced data, such as reorganizing data to enable progressive reads (mesh decimation and wavelet-based multi-resolution) and generating auxiliary data structures for query-based subsetting (FastBit). That said, some uses cases put

Table 2: Summary table for mesh reduction techniques surveyed in Section 5.

Technique	Lossless Support	Fixed Error	Fixed Size	Progressive Data Access	Speed	Mesh Type	Section	Tools
Mesh Decimation	No	Yes	Yes	Yes	Varies	Unstructured	5.1	
Sample Sets	No	No	Yes	No	Fast	Structured	5.2.1	
Space-filling Curves	Yes[a]	No	Yes	Yes	Fast	Structured	5.2.2	ViSUS
Wavelet Multi-resolution	Yes[a]	No	Yes	Yes	Medium	Structured	5.2	VAPOR
Domain Subsetting	Yes[a]	Yes	Yes	No	Fast	(Un)structured	5.1.1	VisIt
Query-based Subsetting	Yes[a]	Yes	Yes	No	Medium	(Un)structured	5.3.2	FastBit
Temporal Sampling	No	Yes	Yes	No	Fast	(Un)structured	5.4	

[a] While wavelet multi-resolution achieves near lossless reconstruction, others achieve bit-for-bit lossless reconstruction.

a premium on minimizing the total storage size, regardless of reduction and reconstruction times. These use cases include remote transmission and long-term storage.

The minimum resources required on both reconstruction and reduction are also important considerations. For example, in situ scenarios will want to limit the amount of memory footprint required, where data sampling methods are viable. On the user end, if thin clients are desired, such as web browsers, this will bias the reduction methods to progressive and level-of-detail capable methods, such as progressive mesh decimation and multi-resolution. Alternatively, batch analysis scenarios have no need for progressive encoding and may not be concerned with memory footprint on read.

Finally, Table 2 summarizes mesh reduction techniques from this section, helping narrow down options to choose from.

6. Derived Representations

Techniques in this section serve the same overall goal of data reduction, but do not directly reduce the data set. Rather, they derive alternate representations from the original data, and then discard the original data completely. These derived representations are typically much smaller, effectively creating a data reduction. The derived representations can then be used for subsequent analyses.

Techniques in the derived representation category have some desirable properties. First, memory footprints for these techniques can be greatly reduced. Second, most techniques can fix the size of the derived representations, i.e., they can have the fixed size property. That said, since the original data is gone, the concepts of fixed error and progressive data access are often not applicable for these methods. However, it is sometimes possible to provide error bounds and/or progressive data access on the alternate representations themselves, or analyses that use these representations. The downside of this class of techniques is that they are less flexible than previously surveyed ones, because derived representations are usually highly tailored towards specific visualizations or analyses. We survey some popular derived representations that are either actively used or are actively being researched in the scientific visualization community.

6.1. Lagrangian Basis Flows

Traditionally, post hoc flow visualization techniques operate by having a simulation code store velocity vectors to disk, and having a separate visualization program load this data and advect particle trajectories via repeated velocity field evaluations. Temporal sampling on the available time slices reduces the amount of data to save. This is often referred to as the Eulerian approach in literature. In contrast, with the Lagrangian approach, advection is performed in situ during the simulation, and the resulting pathline data (so-called Lagrangian basis flows) are stored to disk. During post hoc processing, new pathlines can be generated from the basis flows. Spatial sampling on the seed points reduces the number of basis flows, thus reducing the amount of data to save. Research to date [ACG*14] indicates that this approach is more accurate than the traditional Eulerian approach while using less storage. To further improve the accuracy of Lagrangian schemes, C^1 cubic composite Bézier curves and cubic Hermite splines can be used to present pathlines [BJ15].

The fact that Lagrangian basis flows are generated in situ enables these basis flows to take advantage of the native temporal resolution of a simulation, which provides highly accurate pathlines. There are limited applications on Lagrangian basis flows right now. One noteworthy example includes a data structure that combines the Lagrangian representation with the traditional Eulerian representation [SXM16]. Using an indexing-based data structure, the information from both representations is combined for complex analyses, as demonstrated in studies of fusion, combustion, and cosmology data sets. Further, this combined representation enables out-of-core operations and efficient sampling of multi-resolution subsets.

6.2. Image Space Visualization

With image space visualization, the basic idea is to run visualization algorithms in situ, and save images and/or image-like results. A visualization scientist then uses the image results to explore the data as if they were generated in real-time by a traditional post hoc visualization program. Saying it another way, the results presented to end users is the same, but the post hoc visualization program is generating its results using pre-saved images rather than mesh-based simulation data. This approach reduces data size, among other benefits, at the cost of decreased visualization flexibility.

Proxy images [TCM10] are an images space visualization technique for volume rendering. Proxy images themselves are not viewable images, but they contain enough information to generate a variety of volume renderings, allowing for post hoc adjustments of

settings such as view, transfer functions, and lighting effects. To make the scheme work, three types of proxy images are needed: depth images to hold intensity intervals, multi-view perspective images to contain samples from neighboring viewpoints, and accumulated attenuation images to represent opacity mappings. In terms of data reduction, high resolution meshes benefit most from this technique. For example, a study focusing on a $2,048^3$ grid obtained two orders of magnitude reduction. Finally, this technique has been demonstrated in an in situ setting [TYC*11].

Volumetric depth images, or VDIs [FSE13] are another image space visualization technique for volume rendering, this time specifically focusing on raycasting. With this technique, samples along a ray are partitioned into segments based on similarity of values. All segments of a ray — including their color, opacity, and depth information — are stored in per-pixel lists. Lists for all rays form a VDI, which can be used to create images that are equivalent to their corresponding volume renderings. This technique was also been extended to deal with time series of volumes, as reported in [FFSE14]. Data reduction achieved through VDIs has been reported to be approximately one order of magnitude.

Cinema [AJO*14] is a framework for general-purpose image space visualization and exploration. The core of Cinema is a specification for an image database (i.e., a file format), meaning any program can generate a Cinema database (likely in situ). Post hoc processing then occurs via a viewer program that enables exploration via images in the database. On the image generation side, Cinema supports a rich set of features enabling domain scientists to make decisions on what imagery to save. Cinema exhibits impressive data reduction factors and performance. The authors note that for simulation data on the order of 10^{15} (petabytes) in size, and image sizes on the order of 10^6 (megabytes), many many images can be saved and still result in a storage reduction.

Additional image-based techniques have also been explored. Chen et al. [CYB08] implemented an Internet-based system for imagery exploration that supported six-dimensional navigation: azimuth, polar angle, viewpoint translation, time, and isocontour level. Ye et al. [YWM*15] used depth maps, which are multiple layers of isosurfaces in the image space, to perform feature extraction and tracking. They have also introduced a novel algorithm to calculate new isosurfaces from existing ones. Ye et al. [YMM13] also demonstrated a system that explores flow-field visualization imagery with navigation capabilities, such as changing the view angle, generating block cutaways, adjusting lighting, and changing transfer functions.

6.3. Topology-based Representations

Topology-based representations rely on concepts from mathematical topology to compute simplified or abstracted representations of scalar data, such as split trees, merge trees, and Morse-Smale complexes, etc. (see [HLH*16] for an overview). For example, a merge tree records join events between level set components as the threshold is increased, and has been shown to provide a good basis for feature segmentation [LPG*14]. Using this information, data can be stored in a reduced form, yielding substantial reduction in some applications [BKL*11, BWT*11, LPG*14]. Further, the reduced data still permits a flexible amount of post hoc exploration.

Topology-based data reduction has also been shown to be effective in combination with image-based storage. Based on a contour tree segmentation, the intersection of an image's viewing rays with the non-overlapping segments can be stored in a per-pixel list, yielding a *layered depth image* [BG15]. The resulting representation is compact and also allows post hoc simplification through merging of ray segments, based on contour tree information that is stored alongside the image. A particularly useful property of topology-based data representation is the ability to flexibly and correctly simplify the representation based on various measures such as volume or persistence [GNP*05]. This allows fine-grained control over the amount of reduction and preservation of important information.

Another approach of topology-based analysis is to determine features based on local extrema. For combustion analysis, Bremer et al. [BWT*11] reported an empirical data reduction factor of approximately $200X$ with additional compression. Later research showed the approach to be feasible in situ [BKL*11].

6.4. Histogram and Distribution Representations

Scientific data can be transformed into statistical distribution-based data sets using histogram and distribution representations. In this case, points or blocks of data are represented as probabilities of values in space and/or time, rather than individual point or cell values [LS15, TLB*11]. In these data representations, the overall data size may be reduced through aggregation of multiple time steps, spatial ranges, or ensembles into distributions.

These representations allow for statistical and theoretical calculations to occur quickly. That said, the resulting visualizations are less accurate, due to lack of spatial information, aggregation, and/or distribution precision. This statistical information can also be used for further data reduction, such as data selection and prioritization. For example, Biswas et al. [BDSW13] prioritized individual variables by their contribution to the information entropy in a multivariate data set. Finally, visualizations for high-dimensional data including usage of histograms and distribution representations are surveyed by Liu et al. [LMW*15].

6.5. Discussion on Derived Representations

For the most part, techniques in this section make assumptions about what sorts of visualizations or analyses end users will want to perform and then optimize specifically for those operations. For example, Lagrangian basis flows are only useful if the end user wants to study particle trajectories or visualization techniques based on particle trajectories. This contrasts with the previous sections where the results from reconstructing data could be used for arbitrary visualizations and analyses. As a result, choosing a technique from this section is straightforward. If the reduction technique allows for the desired visualizations and analyses, then it is likely a good option.

Assessing the computational cost for derived representation techniques involves considering both the generation of the derived representation and the usage of the derived representation for visualization and analysis. On the generation side, in situ settings place

particular emphasis on quick execution times, in order to not slow down the simulation. On the usage side, however, many of these techniques are much faster than their traditional counterparts. For example, both Lagrangian techniques and image space visualizations are able to provide interactivity that is difficult to achieve with traditional techniques.

7. Conclusion

Computing trends are likely to make data reduction be an increasingly important topic for simulation, visualization, and data analysis. The ability to generate and observe data is going up faster than the ability to store data, especially on supercomputers. Further, the desire to share and/or disseminate data worldwide creates another motivating use case, as data transmission speeds over the internet are often slower than disk, and again are not increasing as quickly as the size of scientific data. As a result, we believe that simulation scientists and visualization scientists will more and more frequently opt to add reduction techniques to their workflows. The five categories of reduction that we survey provide choices for the best fit for their workflows. The techniques have significant differences, with some providing perfect or near-perfect reconstruction and yet little data reduction and others providing great reductions but imperfect reconstructions. Other important differences include the nature of the data they store, the types of analyses they permit, whether they allow for progressive access, and whether the reconstructed version is smaller in size than the original (and by how much).

Reflecting on the techniques in the survey, it seems that the state of the techniques in each of our five categories vary. The truly lossless and near lossless categories are quite mature, including textbooks on their respective techniques and production software that is used ubiquitously (including non-scientific data use cases). For the remaining three categories, there appears to be significant opportunity. This state of affairs generally makes sense — the first two categories were well explored since they did not require the user to make compromises on data integrity, while the the latter three are now being explored since the state of computing appears to be requiring compromises. In particular, the category of derived representations seems to be a very active area of research in the last few years.

In terms of future work, we feel there is a great need for additional comparison between techniques. Such comparisons need to be carefully thought through, as factors such as input data, computing architecture, workflow (i.e., in situ processing, delivering data over the internet, etc.), parameters into the algorithms, and the analyses ultimately performed on reconstructed data all can play a significant role in assessing the benefit of a given technique.

References

[ABK16] AUSTIN W., BALLARD G., KOLDA T. G.: Parallel tensor compression for large-scale scientific data. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (May 2016), pp. 912–922. 10

[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 *Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV)* (Paris, France, Nov. 2014), pp. 67–75. 16

[Ack98] ACKERMAN M. J.: The Visible Human Project. *Proceedings of the IEEE 86*, 3 (Mar 1998), 504–511. 14

[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. 2, 17

[ANR74] AHMED N., NATARAJAN T., RAO K. R.: Discrete cosine transform. *IEEE Transactions on Computers 100*, 1 (1974), 90–93. 5

[BAA*16] BAUER A. C., ABBASI H., AHRENS J., CHILDS H., GEVECI B., KLASKY S., MORELAND K., O'LEARY P., VISHWANATH V., WHITLOCK B., BETHEL E. W.: In situ methods, infrastructures, and applications on high performance computing platforms. *Computer Graphics Forum 35*, 3 (2016), 577–597. 11

[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. 17

[BG15] BIEDERT T., GARTH C.: Contour tree depth images for large data visualization. In *Eurographics Symposium on Parallel Graphics and Visualization, Cagliari, Sardinia, Italy, May 25 - 26, 2015.* (2015), Dachsbacher C., Navrátil P. A., (Eds.), Eurographics Association, pp. 77–86. 17

[BJ15] BUJACK R., JOY K. I.: Lagrangian representations of flow fields with parameter curves. In *IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)* (2015), IEEE, pp. 41–48. 16

[BKL*11] BENNETT J., KRISHNAMOORTHY V., LIU S., GROUT R. W., HAWKES E. R., CHEN J. H., SHEPHERD J., PASCUCCI V., BREMER P.: Feature-based statistical analysis of combustion simulation data. *IEEE Trans. Vis. Comput. Graph. 17*, 12 (2011), 1822–1831. 17

[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. 15

[bloa] Blosc, an extremely fast, multi-threaded, meta-compressor library. Accessed: 2017-06-13. URL: http://www.blosc.org/. 5

[blob] Synthetic Benchmarks. Accessed: 2017-06-13. URL: http://www.blosc.org/synthetic-benchmarks.html. 5

[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 *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2014), IEEE, pp. 256–266. 4

[BM] BONWICK J., MOORE B.: ZFS: The last word in file systems. [Online, accessed July 02 2017]. URL: https://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf. 4

[BM72] BAYER R., MCCREIGHT E.: Organization and maintenance of large ordered indexes. *Acta Informatica 1* (1972), 173–189. 15

[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. 14

[BMYH16] BURTSCHER M., MUKKA H., YANG A., HESAARAKI F.: Real-time synthesis of compression algorithms for scientific data. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2016), pp. 264–275. 5

[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, 8

[BRP16] BALLESTER-RIPOLL R., PAJAROLA R.: Lossy volume compression using tucker truncation and thresholding. *Vis. Comput. 32*, 11 (Nov. 2016), 1433–1446. 10

[btr] Btrfs compression FAQ. [Online, accessed December 14 2016]. URL: https://btrfs.wiki.kernel.org/index.php/Compression. 4

[Bur] BURTSCHER M.: SPDP v1.0. [Online, accessed January 02 2018]. URL: http://cs.txstate.edu/~burtscher/research/SPDPcompressor/. 5

[BWT*11] BREMER P., WEBER G. H., TIERNY J., PASCUCCI V., DAY M. S., BELL J. B.: Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Trans. Vis. Comput. Graph. 17*, 9 (2011), 1307–1324. 17

[BXH*17] BAKER A., XU H., HAMMERLING D., LI S., CLYNE J.: Toward a multi-method approach: Lossy data compression for climate simulation data. In *Proceedings of the The 1st International Workshop on Data Reduction for Big Scientific Data (DRBSD-1)* (2017), Springer. 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 *Proceedings of IEEE Visualization (VIS'05)* (2005), IEEE, pp. 191–198. 14

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B. J., MEREDITH J. S., AHERN S., BONNELL K., MILLER M., WEBER G. H., HARRISON C., PUGMIRE D., FOGAL T., GARTH C., SANDERSON A., BETHEL E. W., DURANT M., CAMP D., FAVRE J. M., RUBEL O., NAVRATIL P., WHEELER M., SELBY P.: *VisIt: An End-User Tool for Visualization and Analyzing Very Large Data*, 1 ed., vol. 1 of *CRC Computational Science Series*. Taylor and Francis, Boca Raton, 2012, p. 520. 3, 14

[CCM*00] CIGNONI P., COSTANZA D., MONTANI C., ROCCHINI C., SCOPIGNO R.: Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)* (Oct 2000), pp. 85–92. 11

[CDDY06] CANDES E., DEMANET L., DONOHO D., YING L.: Fast discrete curvelet transforms. *Multiscale Modeling & Simulation 5*, 3 (2006), 861–899. 14

[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. 8, 9

[CDSY98] CALDERBANK A., DAUBECHIES I., SWELDENS W., YEO B.-L.: Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis 5*, 3 (1998), 332 – 369. 5

[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 *Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics* (2001), IEEE, pp. 59–151. 15

[CGG*04] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. In *ACM Transactions on Graphics (TOG)* (2004), vol. 23, ACM, pp. 796–803. 12

[CI98] CHAN C.-Y., IOANNIDIS Y. E.: Bitmap index design and evaluation. In *ACM SIGMOD Record* (1998), vol. 27, ACM, pp. 355–366. 15

[Cly12] CLYNE J.: Progressive data access for regular grids in high performance visualization. In *High Perfomance Visualization: Enabling Extreme-Scale Scientific Insight*, Bethel E., Childs H., Hanson C., (Eds.). Chapman & Hall/CRC, 2012, pp. 145–170. 13

[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. 14

[CP96] CHEN Y., PEARLMAN W. A.: Three-dimensional subband coding of video using the zero-tree method. In *Visual Communications and Image Processing* (1996), International Society for Optics and Photonics, pp. 1302–1312. 9

[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. 14

[CSD*00] CHRYSAFIS C., SAID A., DRUKAREV A., ISLAM A., PEARLMAN W. A.: SBHP-a low complexity wavelet coder. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'00)* (2000), vol. 4, IEEE, pp. 2035–2038. 9

[CYB08] CHEN J., YOON I., BETHEL W.: Interactive, internet delivery of visualization via structured prerendered multiresolution imagery. *IEEE Transactions on Visualization and Computer Graphics 14*, 2 (2008), 302–312. 17

[CYH*97] CHIUEH T.-C., YANG C.-K., HE T., PFISTER H., KAUFMAN A.: Integrated volume compression and visualization. In *Proceedings of Visualization'97* (1997), IEEE, pp. 329–336. 8

[DC16] DI S., CAPPELLO F.: Fast error-bounded lossy hpc data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (Chicago, IL, 06/2016 2016), IEEE, IEEE. 8

[Deu96] DEUTSCH P.: *DEFLATE compressed data format specification version 1.3*. Tech. rep., 1996. 4

[DS98] DAUBECHIES I., SWELDENS W.: Factoring wavelet transforms into lifting steps. *Journal of Fourier analysis and applications 4*, 3 (1998), 247–269. 10

[EGM04] ELLSWORTH D., GREEN B., MORAN P.: Interactive terascale particle visualization. In *Proceedings of the Conference on Visualization* (2004), IEEE Computer Society, pp. 353–360. 7

[FCY99] FOLK M., CHENG A., YATES K.: Hdf5: A file format and i/o library for high performance computing applications. In *Proceedings of Supercomputing* (1999), vol. 99, pp. 5–33. 4

[FFSE14] FERNANDES O., FREY S., SADLO F., ERTL T.: Space-time volumetric depth images for in-situ visualization. In *IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)* (2014), IEEE, pp. 59–65. 17

[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. 10

[FM12] FOUT N., MA K.-L.: An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2295–2304. 5, 8

[Fow00] FOWLER J. E.: QccPack: an open-source software library for quantization, compression, and coding. In *Proc. SPIE, Applications of Digital Image Processing XXIII* (August 2000), vol. 4115, SPIE, pp. 294–301. 7, 9

[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. 17

[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. 7

[GG91] GERSHO A., GRAY R. M.: *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991. 7

[GGA*11] GOSINK L. J., GARTH C., ANDERSON J. C., BETHEL E. W., JOY K. I.: An application of multivariate statistical analysis for query-driven visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 3 (2011), 264–275. 15

[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. 12

[GNP*05] GYULASSY A., NATARAJAN V., PASCUCCI V., BREMER P., HAMANN B.: Topology-based simplification for feature extraction from 3d scalar fields. In *16th IEEE Visualization Conference, VIS 2005, Minneapolis, MN, USA, October 23-28, 2005* (2005), pp. 535–542. 17

[Gos08] GOSINK L. J.: Bin-hash indexing: A parallel method for fast query processing. *Lawrence Berkeley National Laboratory* (2008). 15

[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 *Proceedings of the 41st International Conference on Parallel Processing (ICPP)* (2012), IEEE, pp. 239–248. 6, 13

[GS01] GUTHE S., STRASSER W.: Real-time decompression and visualization of animated volume data. In *Proceedings of IEEE Visualization (VIS'01)* (Oct 2001), pp. 349–572. 8

[GS04] GUTHE S., STRASSER W.: Advanced techniques for high-quality multi-resolution volume rendering. *Computers & Graphics 28*, 1 (2004), 51–58. 14

[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. 15

[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. 15

[GVDB01] GOEMAN B., VANDIERENDONCK H., DE BOSSCHERE K.: Differential fcm: Increasing value prediction accuracy by improving table usage efficiency. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA)* (2001), IEEE, pp. 207–216. 5

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *Proceedings of IEEE Visualization (VIS'02)* (2002), IEEE, pp. 53–60. 14

[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. 14

[GY95] GHAVAMNIA M. H., YANG X. D.: Direct rendering of laplacian pyramid compressed volume data. In *Proceedings of the 6th Conference on Visualization* (1995), IEEE Computer Society, p. 192. 14

[GZ05] GARLAND M., ZHOU Y.: Quadric-based simplification in any dimension. *ACM Transactions on Graphics (TOG) 24*, 2 (2005), 209–239. 12

[gzi] The Gzip Home Page. Accessed: 2017-06-13. URL: http://www.gzip.org/. 4

[Hil91] HILBERT D.: Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen 38*, 3 (1891), 459–460. 13

[HLH*16] HEINE C., LEITTE H., HLAWITSCHKA M., IURICICH F., FLORIANI L. D., SCHEUERMANN G., HAGEN H., GARTH C.: A survey of topology-based methods in visualization. *Comput. Graph. Forum 35*, 3 (2016), 643–667. 17

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 189–198. 12

[Hop98] HOPPE H.: Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Visualization'98 Proceedings* (1998), IEEE, pp. 35–42. 12

[Huf52] HUFFMAN D. A.: A method for the construction of minimum-redundancy codes. *Proceedings of the IRE 40*, 9 (1952), 1098–1101. 4

[IKK12] IVERSON J., KAMATH C., KARYPIS G.: *Fast and Effective Lossy Compression Algorithms for Scientific Datasets*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 843–856. 7

[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, 7, 8

[ILS05] ISENBURG M., LINDSTROM P., SNOEYINK J.: Lossless compression of predicted floating-point geometry. *Computer-Aided Design 37*, 8 (2005), 869–877. 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. 8

[IP98b] ISLAM A., PEARLMAN W. A.: Embedded and efficient low-complexity hierarchical image coder. In *Electronic Imaging'98* (1998), International Society for Optics and Photonics, pp. 294–305. 9

[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. 14

[jas] The JasPer Project Home Page. Accessed: 2017-06-13. URL: https://www.ece.uvic.ca/~frodo/jasper/. 9

[jj2] jj2000, A pure Java JPEG 2000 image codec. Accessed: 2017-06-13. URL: https://code.google.com/archive/p/jj2000/. 9

[kak] Kakadu Software. Accessed: 2017-06-13. URL: http://kakadusoftware.com/. 10

[KB09] KOLDA T. G., BADER B. W.: Tensor decompositions and applications. *SIAM Rev. 51*, 3 (Aug. 2009), 455–500. 10

[KLK04] KIM J., LEE S., KOBBELT L.: View-dependent streaming of progressive meshes. In *Proceedings of IEEE Conference on Shape Modeling Applications* (2004), IEEE, pp. 209–220. 12

[KP97] KIM B.-J., PEARLMAN W. A.: An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In *Proceedings of Data Compression Conference (DCC'97)* (1997), IEEE, pp. 251–260. 9

[KS99] KIM T.-Y., SHIN Y. G.: An efficient wavelet-based compression method for volume rendering. In *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications* (1999), IEEE, pp. 147–156. 8

[KWB*] KUO B., WANG W., BRUYERE C., SCHEITLIN T., MIDDLETON D.: IEEE Visualization 2004 Contest. [Online, accessed January 10 2018]. URL: http://vis.computer.org/vis2004contest/data.html. 3

[LBG80] LINDE Y., BUZO A., GRAY R.: An algorithm for vector quantizer design. *IEEE Transactions on Communications 28*, 1 (1980), 84–95. 7

[LBMN05] LALGUDI H. G., BILGIN A., MARCELLIN M. W., NADAR M. S.: Compression of fMRI and ultrasound images using 4D SPIHT. In *Proceeding of the International Conference on Image Processing (ICIP'05)* (2005), vol. 2, IEEE, pp. II–746. 9

[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. 10

[LD07] LU Y. M., DO M. N.: Multidimensional directional filter banks and surfacelets. *IEEE Transactions on Image Processing 16*, 4 (2007), 918–931. 14

[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. 9

[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. 14

[LI06] LINDSTROM P., ISENBURG M.: Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1245–1250. 5, 8

[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. 6, 10

[LKA*17] LING J., KEGELMEYER W. P., ADITYA K., KOLLA H., REED K. A., SHEAD T. M., DAVIS W. L.: Using feature importance metrics to detect events of interest in scientific computing applications. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)* (Oct 2017), pp. 55–63. 15

[LKS*08] LOFSTEAD J. F., KLASKY S., SCHWAN K., PODHORSZKI N., JIN C.: Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments* (2008), ACM, pp. 15–24. 4

[LLCC17] LI S., LARSEN M., CLYNE J., CHILDS H.: Performance impacts of in situ wavelet compression on scientific simulations. In *Proceedings of the In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization Workshop* (New York, NY, USA, 2017), ISAV2017, ACM. 9

[Llo82] LLOYD S.: Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. 7

[LMC*17] LI S., MARSAGLIA N., CHEN V., SEWELL C., CLYNE J., CHILDS H.: Achieving Portable Performance For Wavelet Compression Using Data Parallel Primitives. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)* (Barcelona, Spain, June 2017). 9

[LMW*15] LIU S., MALJOVEC D., WANG B., BREMER P.-T., PASCUCCI V.: Visualizing High-Dimensional Data: Advances in the Past Decade. In *Eurographics Conference on Visualization (EuroVis) - STARs* (2015), Borgo R., Ganovelli F., Viola I., (Eds.), The Eurographics Association. 17

[Loè78] LOÈVE M.: Probability theory, vol. ii. In *Graduate texts in mathematics*, vol. 46. Springer-Verlag, 1978. 10

[LP06] LIU Y., PEARLMAN W. A.: Resolution scalable coding and region of interest access with three-dimensional SBHP algorithm. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission* (2006), IEEE, pp. 647–654. 9

[LP07] LIU Y., PEARLMAN W. A.: Four-dimensional wavelet compression of 4-D medical images using scalable 4-D SBHP. In *Proceedings of Data Compression Conference (DCC'07)* (2007), IEEE, pp. 233–242. 9

[LPG*14] LANDGE A. G., PASCUCCI V., GYULASSY A., BENNETT J., KOLLA H., CHEN J., BREMER P.: In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014* (2014), Damkroger T., Dongarra J., (Eds.), IEEE Computer Society, pp. 1020–1031. 17

[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

[LS15] LU K., SHEN H.-W.: A compact multivariate histogram representation for query-driven visualization. In *IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)* (2015), IEEE, pp. 49–56. 17

[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. 8

[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. 8

[LSO*17] LI S., SANE S., ORF L., MININNI P., CLYNE J., CHILDS H.: Spatiotemporal wavelet compression for visualization of scientific simulation data. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)* (Sept 2017), pp. 216–227. 9

[LT00] LINDSTROM P., TURK G.: Image-driven simplification. *ACM Transactions on Graphics (ToG) 19*, 3 (2000), 204–241. 12

[Mac03] MACKAY D. J.: *Information theory, inference and learning algorithms*. Cambridge University Press, 2003. 4

[MGA*08] MEYER M., GOSINK L. J., ANDERSON J. C., BETHEL E. W., JOY K. I.: Query-driven visualization of time-varying adaptive mesh refinement data. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1715–1722. 15

[MLF*16] MYERS K., LAWRENCE E., FUGATE M., BOWEN C. M., TICKNOR L., WOODRING J., WENDELBERGER J., AHRENS J.: Partitioning a large simulation as it runs. *Technometrics 58*, 3 (2016), 329–340. 15

[Mor66] MORTON G. M.: *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966. 13

[NC12] NORTON A., CLYNE J.: The VAPOR visualization application. In *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, Bethel E., Childs H., Hanson C., (Eds.). Chapman & Hall/CRC, 2012, pp. 415–428. 3, 6, 9

[NH92] NING P., HESSELINK L.: Vector quantization for volume rendering. In *Proceedings of the 1992 Workshop on Volume Visualization* (1992), ACM, pp. 69–74. 7

[NH93] NING P., HESSELINK L.: Fast volume rendering of compressed data. In *Proceedings of IEEE Conference on Visualization (Visualization'93)* (1993), IEEE, pp. 11–18. 7

[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. 9

[O'N89] O'NEIL P. E.: Model 204 architecture and performance. In *High Performance Transaction Systems*. Springer, 1989, pp. 39–59. 15

[ope] OpenJPEG, An open-source JPEG2000 codec written in C. Accessed: 2017-06-13. URL: http://www.openjpeg.org/. 9

[OS12] ORACLE SOLARIS Z.: Administration guide, 2012. 4

[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. 15

[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. 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. 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. 14

[PR00] PAJAROLA R., ROSSIGNAC J.: Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 79–93. 12

[PSS*12] PASCUCCI V., SCORZELLI G., SUMMA B., BREMER P., GYULASSY A., CHRISTENSEN C., PHILIP S., KUMAR S.: The visus visualization framework. *EW Bethel, HC (LBNL), and CH (UofU), editors, High Performance Visualization: Enabling Extreme-Scale Scientific Insight, Chapman and Hall/CRC Computational Science* (2012). 13

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*. Springer, 1993, pp. 455–465. 12

[RBM13] RODEH O., BACIK J., MASON C.: Btrfs: The linux B-tree filesystem. *ACM Transactions on Storage (TOS) 9*, 3 (2013), 9. 4

[RGG*13] RODRÍGUEZ M. B., GOBBETTI E., GUITIÁN J. A. I., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S. K.: A Survey of Compressed GPU-Based Direct Volume Rendering. In *Eurographics 2013 - State of the Art Reports* (2013), Sbert M., Szirmay-Kalos L., (Eds.), The Eurographics Association. 11, 14

[RO96] RENZE K. J., OLIVER J. H.: Generalized unstructured decimation [computer graphics]. *IEEE Computer Graphics and Applications 16*, 6 (1996), 24–32. 12

[Rod99] RODLER F. F.: Wavelet based 3D compression with fast random access for very large volume data. In *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications* (1999), IEEE, pp. 108–117. 9

[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., , HAMANN B., BETHEL E. W.: 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. 15

[Sal04] SALOMON D.: *Data compression: the complete reference*. Springer Science & Business Media, 2004. 5

[Say12] SAYOOD K.: *Introduction to Data Compression*. Morgan Kaufmann Publishers Inc., 2012. 5

[SBP*15] SALLOUM M., BENNETT J. C., PINAR A., BHAGATWALA A., CHEN J. H.: Enabling adaptive scientific workflows via trigger detection. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2015), ISAV2015, ACM, pp. 41–45. 15

[Sew96] SEWARD J.: bzip2 and libbzip2. *avaliable at http://www. bzip. org* (1996). 4

[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. 12

[Sha93] SHAPIRO J. M.: Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing 41*, 12 (1993), 3445–3462. 9

[Sha01] SHANNON C. E.: A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review 5*, 1 (2001), 3–55. 4

[SJS*12] SCHENDEL E. R., JIN Y., SHAH N., CHEN J., CHANG C. S., KU S. H., ETHIER S., KLASKY S., LATHAM R., ROSS R., SAMATOVA N. F.: Isobar preconditioner for effective and high-throughput lossless data compression. In *2012 IEEE 28th International Conference on Data Engineering* (April 2012), pp. 138–149. 5

[SN96] STRANG G., NGUYEN T.: *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996. 5, 9

[SP93] SAID A., PEARLMAN W. A.: Image compression using the spatial-orientation tree. In *IEEE International Symposium on Circuits and Systems (ISCAS'93),* (1993), IEEE, pp. 279–282. 9

[SRF87] SELLIS T. K., ROUSSOPOULOS N., FALOUTSOS C.: The R+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases* (1987), Morgan Kaufmann Publishers Inc., pp. 507–518. 15

[SS97] SAZEIDES Y., SMITH J. E.: The predictability of data values. In *Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture* (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, pp. 914–922. 8

[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). 16

[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. 12

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

[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. 16

[TDCC17] TAO D., DI S., CHEN Z., CAPPELLO F.: Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. *Proceedings of the 31th IEEE International Parallel and Distributed Processing Symposium* (2017). 8

[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. 12

[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. 12

[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. 8

[TLB*11] THOMPSON D., LEVINE J. A., BENNETT J. C., BREMER P.-T., GYULASSY A., PASCUCCI V., PÉBAY P. P.: Analysis of large-scale scalar data using hixels. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (2011), IEEE, pp. 23–30. 17

[TLS12] TONG X., LEE T. Y., SHEN H. W.: Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Oct 2012), pp. 49–56. 15

[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. 9

[TYC*11] TIKHONOVA A., YU H., CORREA C. D., CHEN J. H., MA K.-L.: A preview and exploratory technique for large-scale scientific simulations. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)* (2011), pp. 111–120. 17

[Use07] USEVITCH B. E.: JPEG2000 compatible lossless coding of floating-point data. *Journal on Image and Video Processing 2007*, 1 (2007), 22–22. 6

[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 *Proceedings of Data Compression Conference (DCC'96)* (1996), IEEE, pp. 396–405. 9

[WAB*09] WU K., AHERN S., BETHEL E. W., CHEN J., CHILDS H., CORMIER-MICHEL E., GEDDES C., GU J., HAGEN H., HAMANN B., KOEGLER W., LAURET J., MEREDITH J., MESSMER P., OTOO E., PEREVOZTCHIKOV V., POSKANZER A., PRABHAT, RÜBEL O., SHOSHANI A., SIM A., STOCKINGER K., WEBER G., ZHANG W.-M.: FastBit: interactively searching massive data. In *Journal of Physics: Conference Series* (2009), vol. 180, IOP Publishing, p. 012053. 15

[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. 12, 13

[Wan12] WANG R.: *Introduction to orthogonal transforms: with applications in data processing and analysis*. Cambridge University Press, 2012. 10

[WDF11] WEISS K., DE FLORIANI L.: Simplex and diamond hierarchies: Models and applications. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 2127–2155. 11

[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 *15th International Conference on Scientific and Statistical Database Management* (2003), IEEE, pp. 65–74. 15

[WL16] WEISS K., LINDSTROM P.: Adaptive multilinear tensor product wavelets. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (Jan 2016), 985–994. 14

[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 *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Oct 2011), pp. 31–38. 10

[WNC87] WITTEN I. H., NEAL R. M., CLEARY J. G.: Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (1987), 520–540. 4

[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. 8

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMANN K., ERTL T.: Level-of-detail volume rendering via 3D textures. In *Proceedings of the IEEE Symposium on Volume Visualization* (2000), ACM, pp. 7–13. 14

[XV96] XIA J. C., VARSHNEY A.: Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th Conference on Visualization* (1996), IEEE Computer Society Press, pp. 327–ff. 12

[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. 9

[YL95] YEO B.-L., LIU B.: Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (1995), 29–43. 8

[YMHB15] YANG A., MUKKA H., HESAARAKI F., BURTSCHER M.: Mpc: A massively parallel compression algorithm for scientific data. In *2015 IEEE International Conference on Cluster Computing* (Sept 2015), pp. 381–389. 5

[YMM13] YE Y., MILLER R., MA K.-L.: In Situ Pathtube Visualization with Explorable Images. In *Eurographics Symposium on Parallel Graphics and Visualization* (2013), Marton F., Moreland K., (Eds.), The Eurographics Association. 17

[YWM*15] YE Y. C., WANG Y., MILLER R., MA K.-L., ONO K.: In situ depth maps based feature extraction and tracking. In *IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)* (2015), IEEE, pp. 1–8. 17

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

[ZL77] ZIV J., LEMPEL A.: A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory 23*, 3 (1977), 337–343. 4

[ZL78] ZIV J., LEMPEL A.: Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory 24*, 5 (1978), 530–536. 4

[zli] zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library. Accessed: 2017-06-13. URL: https://zlib.net/. 4

[ZLMS04] ZIEGLER G., LENSCH H. P., MAGNOR M., SEIDEL H.-P.: Multi-video compression in texture space using 4D SPIHT. In *IEEE the 6th Workshop on Multimedia Signal Processing* (2004), IEEE, pp. 39–42. 9