In-situ multi-resolution and temporal data compression for visual exploration of large-scale scientific simulations

Henry Lehmann[*] Bernhard Jung[†]

Virtual Reality and Multimedia Group,
Technical University Bergakademie Freiberg,
Germany

Abstract

Today's large-scale scientific simulations generate massive data sets that pose challenges both for data storage in HPC environments during the simulation phase and the subsequent data analysis phase. A promising approach for reducing the amount of data written out during simulation run is in-situ compression. However, even the compressed data sets are typically still too large for interactive visual data exploration which calls for multi-resolution data layouts. The recently proposed ISABELA method for lossy in-situ compression was shown to outperform other compression methods for scientific data sets. In this paper, we propose two main extensions to the ISABELA method: (1) an interlaced data layout that supports decompression of multi-resolution views of the data without overhead in the compressed format; (2) a new temporal compression scheme for improving the compression rate by exploiting temporal coherence in the data set. The compressed multi-resolution data can easily be transformed to the VTK AMR (adaptive multi-resolution) data format to support interactive exploration in ParaView and other visualization tools based on VTK. During the simulation phase, there is no significant increase of computational demands for the generation of complete multi-resolution compressed data sets as compared to flat ISABELA compression. During the analysis phase, due to the AMR data layout, our method supports selective loading of regions of interests as well as progressive loading of data sets, thus enabling interactive visualizations of large-scale scientific simulations.

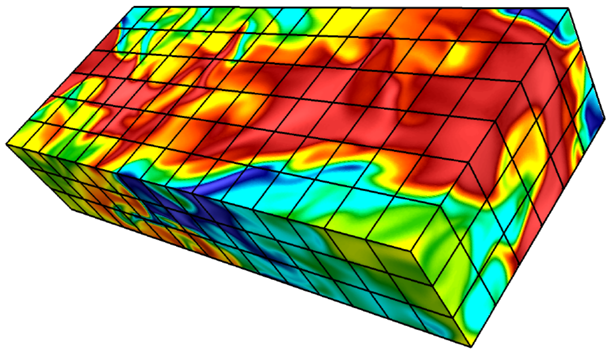
Keywords: in-situ compression, multi-resolution visualization, high-performance computing, time-dependent scientific data

Index Terms: E.4 [Coding and Information Theory]: Data compaction and compression; G.1.2 [Mathematics of Computing]: Approximation—Spline and piecewise polynomial approximation

1 Introduction

Today's HPC environments make use of massive parallelism in order to maintain a steady increase in computing performance. However, this increase in compute power is not matched by improvements in I/O capabilities and the handling of massive data sets produced in HPC environments has become a major problem both in the simulation phase and the subsequent analysis phase [9][7][12][1].

On the simulation side, data movement from main memory to mass storage constitutes a bottleneck which forces scientists to subsample their data or write out data infrequently, defeating the purpose of high-resolution simulations. A promising solution are in-situ approaches for compression, visualization, feature extraction and data indexing [10][1] where data are processed while still re-



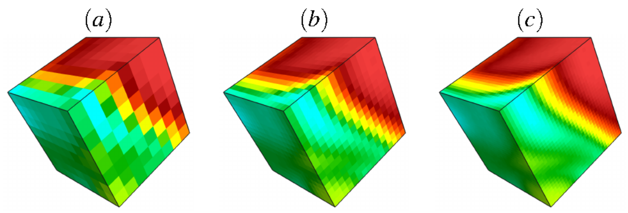


Figure 1: *Top:* The full voxel grid is subdivided into compressed AMR (adaptive multi-resolution) blocks. *Bottom:* By interlacing, each AMR block is represented at multiple levels of resolution. E.g., for compression block size 8^3 and three levels of resolution: (a) coarsest resolution 8^3 voxels, (b) 16^3 voxels, (c) full resolution 32^3 voxels.

siding in main memory of the HPC hardware, thus bypassing storage on slow media. In-situ visualization, e.g. explorable images [11], can achieve a significant decrease in data size for interactive visualization or previewing results. In contrast, in-situ compression methods, e.g. [5] and [7], are able to reconstruct the original data set, supporting not only visual but also non-visual data analysis in a post-processing step. In-situ methods including in-situ compression are widely considered central for coping with the large data challenges of future extreme-scale simulations [10][6][3][4].

For the post-processing phase, these developments suggest that future visualization systems must be enabled to directly load data from in-situ compressed formats. However, given the massive size of today's and future scientific data sets, even compressed versions of the data sets are much too large for interactive visual data exploration. This calls for extensions of in-situ compression algorithms that directly support multi-resolution output.

In this paper we extend the ISABELA algorithm [7] for in-situ compression of scientific data. We apply a data interlacing technique yielding low-resolution views from subsets of the voxels of the original data (Figure 1) without introducing overhead into the compressed format. Furthermore we propose a new temporal compression method to enhance the compression rate in high-

[*]e-mail: henry.lehmann@informatik.tu-freiberg.de
[†]e-mail: jung@informatik.tu-freiberg.de

IEEE Symposium on Large Data Analysis and Visualization 2014
October 9–10, Paris, France
978-1-4799-5215-1/14/$31.00 ©2014 IEEE

resolution, time-dependent simulations. The intended benefits of this work concern both simulation and visual analysis. On the simulation side, scientists may save disk space, increase the spatial or temporal resolution of the simulation, or run longer simulations. During analysis, the multi-resolution data representation allows for the quick generation of coarse, but interactive visualizations of the complete data set that can be progressively refined if desired.

This paper is structured as follows: In Section 2 we briefly describe the ISABELA compression algorithm. Our first main contribution, an interlacing scheme for in-situ generation of compressed multi-resolution representations of scientific data sets based on ISABELA is introduced in Section 3. In Section 4 we explain the progressive loading of data sets based on the multi-resolution layout. As our second main contribution, we present a new approach for temporal compression based on ISABELA in section 5. An experimental evaluation of our approach is given in Section 6. We discuss our approach in Section 7 before we finally conclude in Section 8.

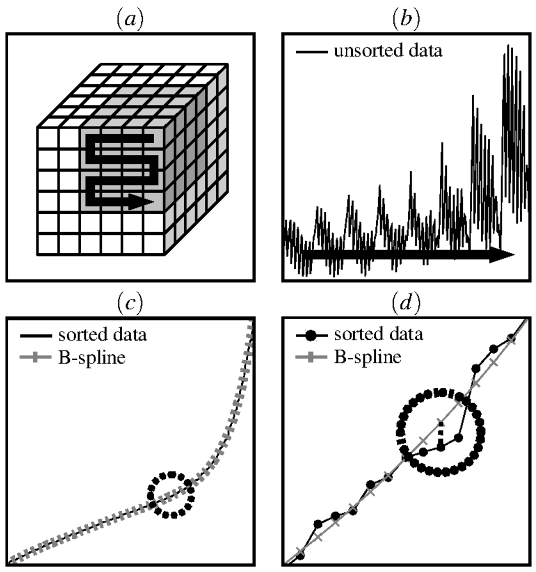


Figure 2: Illustration of ISABELA-like compression: *(a)* subdivision, *(b)* linearization *(c)* sorting/B-spline fit, *(d)* error correction.

2 ISABELA Compression Algorithm

ISABELA (*In-situ Sort-And-B-spline Error-bounded Lossy Abatement*) is a communication-free, block-based, lossy, in-situ compression algorithm for scientific data which negligibly effects the simulation run-time [7]. As argued in [7] lossless compression techniques are generally not suitable for scientific data while other lossy compression techniques, such as Wavelet compression, are significantly outperformed by ISABELA. In a typical configuration, ISABELA can reduce data size up to $\sim$ 20% while guaranteeing a relative error of at most 1% [7].

The ISABELA algorithm is based on linearization of subvolumes, sorting sequences, B-spline regression and error correction as depicted in Figure 2. Compressed blocks consist of a permutation (sort-order), B-spline coefficients and quantized errors. The permutation describes positions of data values in the original sequence and therefore in the subvolume of the data set. The block-based approach exploits the continuous nature of data from scientific simulations and groups together values with local coherence. Since sorting establishes an optimal signal-to-noise ratio in a monotone sequence [7], the B-spline fit only has a very low error w.r.t. the original data. Instead of the original data, only the permutation, B-spline coefficients and error correction bits are stored which yields the excellent compression rates.

block size N		512	1024	2048	4096
permutation		14.06%	15.63%	17.19%	18.75%
B-spline co-	16	3.13%	1.56%	0.78%	0.39%
efficients M	32	6.25%	3.13%	1.56%	0.78%

Table 1: Distribution of memory inside ISABELA compression block for `double` precision data as percentage of uncompressed data size (memory requirements for error correction not shown).

Table 1 shows the memory distribution (without error correction) inside an ISABELA compression block for different block sizes N and number of B-spline coefficients M. Since permutation and B-spline coefficients are fixed size, the compression rate (*compressed/raw* data size) of ISABELA cannot exceed a lower bound which is given by $\mathrm{cr}(N,M) = (\lceil \mathrm{ld}(N) \rceil N + 64M)/(64N)$, where the permutation consists of $\lceil \mathrm{ld}(N) \rceil N$ and the B-spline coefficients of $64M$ bits. As Table 1 shows, fixed start-up costs for storing the permutation and B-spline coefficients amount to about 17% in typical configurations. This bound can be overcome by temporal compression schemes, where start-up costs may be reduced or, as in the approach introduced in Section 5, fully dispensed with non-reference time steps. However, even the compressed data sets are typically still too large for interactive visual data exploration which can be addressed with multi-resolution data layouts.

3 Multi-Resolution Support

In plain ISABELA, decompressing compression blocks one by one yields the full resolution of the data set, which may not fit in main memory and thus cannot be visualized in one pass. In order to support the (de)compression at several levels of resolution, we designed an interlaced data layout. The refinement of one level to the next level of resolution (LOR) doubles the number of voxels in all three dimensions and thus yields eight times more voxels. The procedure is thus similar to spanning an octree with D levels.

In ISABELA, a compression block of $N = n^3$ voxels constitutes the smallest loadable data unit. With D levels of resolution, our interlacing scheme covers a grid of $(n \cdot 2^{D-1})^3$ voxels, called an *AMR (Adaptive Multi Resolution) block*. E.g., in a typical configuration $n = 8$ and $D = 3$, each AMR block covers $(8 \cdot 2^2)^3 = 32^3$ voxels (cf. Figure 1). For in-situ compression, these 32^3 voxels are assumed to reside in the same compute process which is a reasonable assumption in our experience.

The interlacing scheme is designed such that the lowest resolution of one AMR block has the same number of voxels as one ISABELA compression block (and thus can be stored as exactly one compression block). Following the selection scheme shown in Figure 3 *(a)* all levels of resolution can be stored in distinct compression blocks, so that voxels of one level can be (de)compressed independently from voxels at other levels of resolution. In Figure 3, voxels of the same resolution are shown in the same gray value with dark to light going from low to high resolution. Voxels belonging to the same compression block are found by following several Z-curves in parallel as shown in Figure 3 *(b)*. Figure 3 *(c)* depicts three compression blocks at different levels of resolution. In Figure 3 *(d)*, voxels are reordered such that voxels of one compression block are next to each other in order to be fed into the ISABELA compression algorithm easily. The only overhead on top of plain in-situ ISABELA compression is the division of the data set into AMR blocks and the extraction of voxels into compression blocks according to the levels of resolution. It is clearly visible that, following the interlacing scheme, lower resolutions can be reconstructed by only

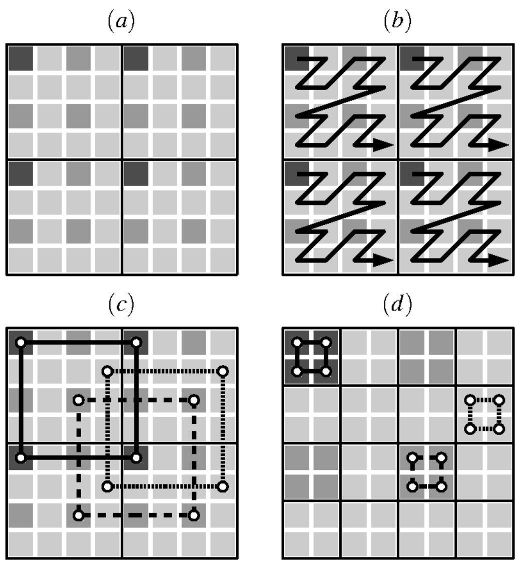


Figure 3: Multiple levels of resolution extracted as subsets of voxels of one AMR block in 2D. *(a)* Selection scheme, voxels of same color correspond to same LOR, higher resolutions (lighter) contain lower resolutions (darker). *(b)* Compression blocks are extracted by following several Z-curves in parallel. *(c)* Three compression blocks, each at a different level of resolution. *(d)* Voxels are reordered to be fed into ISABELA compression algorithm.

loading and decompressing a subset of all compression blocks.

4 Refinement and Progressive Loading

The compressed format shown in section 3 can directly be used to generate lower resolutions of the data set during decompression by loading required compression blocks and reordering voxels. In the highest resolution one AMR block has $(n \cdot 2^{D-1})^3$ voxels. Decreasing the resolution halves the number of voxels in each dimension but the grid occupies the same spatial region as shown in Figure 4 *(a)-(c)*.

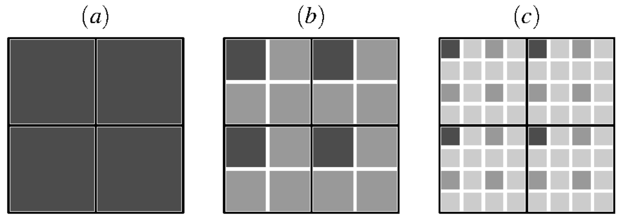


Figure 4: Refining one AMR block with $D = 3$ and $n = 2$ in 2D. *(a)* level one, lowest resolution, 2×2 voxel, same size as one compression block. *(b)* level two, 4×4 voxel, composed of level one and three compression blocks. *(c)* level three, 8×8 voxel, composed of level two and twelve compression blocks.

By storing the data according to our interlacing scheme, the data can be loaded partially for visualization. This provides the user an interactive tool for data exploration which can save time for e.g. adjusting visualization parameters and setting viewpoints. Alternatively data can be loaded depending on an error metric or progressively until the full resolution is reached. Figures 5 and 6 show the views to the data for one level 2 and resp., level 3 AMR block being refined partially.

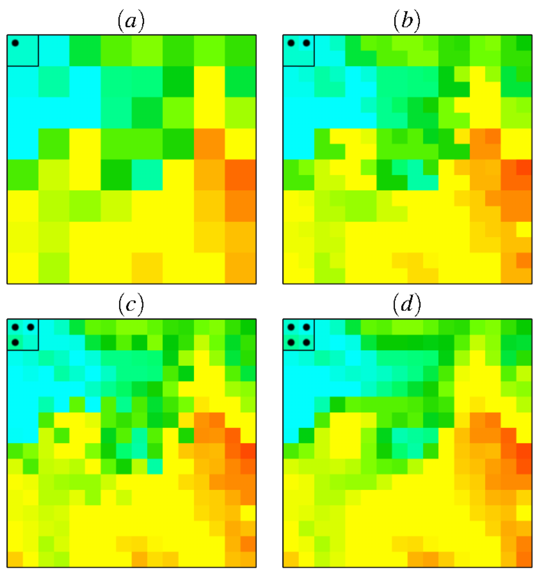


Figure 5: Partially refining one AMR block with $D = 3$ and $n = 8$ at level two in 2D. *(a)* one compression block (from level one) but no further blocks from level two loaded, all voxels are duplicated from level one. *(b),(c)+(d)* one, two, three compression blocks of level two loaded respectively.

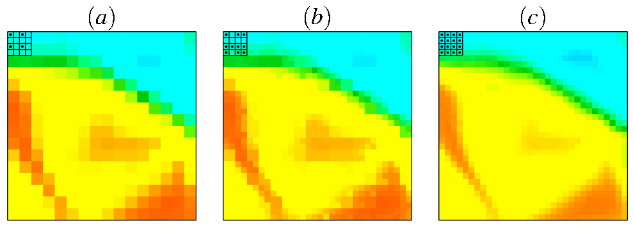


Figure 6: Partially refined AMR block with $D = 3$ and $n = 8$ at level three in 2D. *(a)-(c)* 4 (i.e. all level 2, but no level 3 compression blocks), 10, and all 16 compression blocks loaded.

Each LOR inside one AMR block is represented by one uniform grid and so e.g. $D = 3$ levels of resolution in one AMR block correspond to three uniform grids of size $(n \cdot 2^0)^3$, $(n \cdot 2^1)^3$, $(n \cdot 2^2)^3$ for level one to three respectively as already depicted in Figure 4 *(a)-(c)*. In order to refine the AMR block partially, only a subset of compression blocks of the next higher resolution are loaded. All remaining voxels in the new LOR are filled by duplicating the values of the parent voxel of the next lower resolution. This means even if the AMR block only is partially refined and only a fraction of the compressed data was used to generate it, all voxels are filled with data values (although values are approximate). As compression blocks comprise voxels of one LOR and due the interlacing scheme based on Z-curves, decompression of one compression block always results in voxels which are evenly distributed along the AMR block (cf. Figure 3 *(b)+(c)*).

5 A New Method for Exploiting Temporal Coherence

Compression in ISABELA has the lower bound on memory stated in section 2 as storage requirements for the permutation and number of B-spline coefficients induce start-up costs for the compression of a data block that are independent from the data values being compressed. To overcome this bound, in [7] temporal compression is applied to benefit from the data's continuous nature in the time dimension. Concretely, for each non-reference time step, the change of positions of values inside the sorted sequence of a compression block is encoded using differences w.r.t. the permutation of the last reference time step. Due to the large degree of temporal coherence in typical scientific data sets, differences are typical small. This yields a lot of repetitive bit patterns which are amenable to `zlib` compression, improving the overall compression rate by $\sim 2-5\%$ [7].

Our approach to temporal coherence aims at totally getting rid of the memory needed to store the permutation and B-spline inside compression blocks for non-reference time steps. In contrast to difference encoding between permutations, our approach reuses the same permutation and B-spline for several time steps. Only the first time step of this sequence, the reference time step, contains the permutation and the B-spline. Thus, the minimum bound of the compression rate (without error correction) is lowered to $\mathrm{cr}_T(N,M) = \mathrm{cr}(N,M)/T$ where T is the length of the sequence until the insertion of a new reference time step. E.g., for a block size of $8^3 = 512$, 16 B-spline coefficients, and $T = 3$ the minimal compression rate (for the unlikely case that no error correction is necessary) decreases from 17.19% by 1/3 to 5.73% .

In a reference time step, the value sequence $\bar{X}_0$ of length N is compressed using ISABELA as shown in Figure 2. This yields the permutation P_0, the sorted sequence X_0 and the B-spline Y_0 sampled at N locations approximating the values in X_0. Moving on to the next time steps, the permutation P_0 is reused to reorder the new data values $\bar{X}_i$ obtaining X_i. For smooth data, the reordered data values X_i retain the approximate shape of the B-spline Y_0, although the reordered sequences become increasingly noisy over time. For data from high resolution simulations, the noise increases gradually as shown in Figure 7.

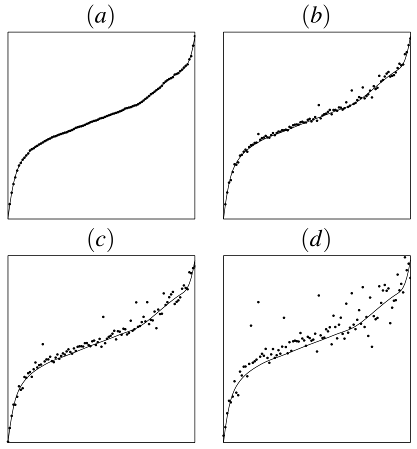


Figure 7: *(a)* in the reference time step, data values $\bar{X}_0$ are sorted, with their sort-order being represented by permutation P_0. *(b),(c)+(d)* in the subsequent time steps, permutation P_0 is reused for reordering the new data values X_i which gradually introduces noise in the sequences.

Due to the increasing noise in the sort order, applying B-spline regression in non-reference time steps would lead to inaccurate fits of the data. Instead, we simply reuse the B-spline Y_0 from the last reference time step. In this way, no B-spline coefficients must be stored for non-reference time steps. However, error correction can be expected to require the more bits, the more noise is present in the sequences.

For error correction in non-reference time steps, the noisy data values are first snapped towards the B-spline Y_0 by moving them horizontally along the abscissa. Thus, the values get closer to the B-spline Y_0 and the remaining error quantization can work efficiently. The horizontal snapping is similar to the difference encoding between permutations. However, in our approach, the values are not entirely sorted after they have been snapped to the B-spline. The snapping process moves the values only as far as needed, until the user-defined error bound is reached, which is illustrated in Figure 8.

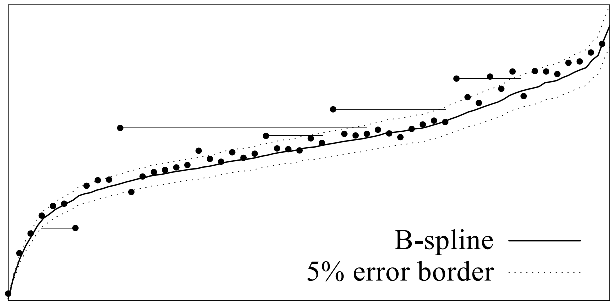


Figure 8: As first step of error correction, noisy data values are snapped towards the B-spline Y_0 of the reference time step.

For purposes of the following explanation, without loss of generality, the reordered value sequence X_i is assumed to only consist of positive data values x. If the data value $x = X_i[j]$ is greater than the corresponding value of the spline $y = Y_0[j]$, the data value is moved to the right, otherwise to the left. Concretely, if $x > y$, the snapping procedure evaluates the spline with an increasing index offset $1, 2, \ldots$. Snapping stops when $y' = Y_0[j + offset]$ meets the user-defined error bound or when y' is the last spline value smaller than x. Similarly, if $x < y$, the snapping procedure finds the spline index, whose value $y' = Y_0[j - offset]$ is the first spline value that either meets the error bound or is smaller than x. In some cases no spline value y' smaller than x exists. To handle this situation, the smallest data value is stored inside the compression block and used as value $y' = x_{min}$ for error correction. The number of positions $offset$ the value x is moved together with its sign is called Δ.

At the end of the snapping procedure there are snapping tuples (x, Δ, y, y'), where $y' \leq x$, which are used to further correct the error if necessary. The idea of the snapping mechanism is to minimize the distance between the data value and the spline, thereby reducing the amount of error correction bits necessary. However, in regions where the B-spline has a very low gradient the snapping produces very large steps and the snapping mechanism might not be beneficial.

Therefore, the remaining error correction is tried without and with snapping. Without snapping, quantization is accomplished by $k_1 = \lfloor (1 - y/x)/\rho) + 1/2 \rfloor$ and the reconstruction by $\hat{x}_1 = y \cdot (1 + k_1 \cdot \rho)$ which follows from the definition of the relative error $|x - y|/|x| \leq \rho$.[1] If $|k_1| \leq |\Delta|$ and $\mathrm{re}(x, \hat{x}_1) \leq \rho$, snapping is discarded, Δ is set to zero and the reconstructed value is based on y. Otherwise, quantization is accomplished with snapping by $k_2 = \lfloor (1 - y'/x)/\rho) + 1/2 \rfloor$ and the reconstruction by $\hat{x}_2 = y_2 \cdot (1 + k_2 \cdot \rho)$. The reconstructed value $\hat{x}_2$ always meets the maximum bound on the relative error, since $y' \leq x$ holds. At the

[1]For other error definitions, e.g. absolute error, the quantization formulae are easily adapted. In our implementation, we use a modified definition of the relative error $|x - y|/max(|x|, \hat{\varepsilon}) \leq \rho$. The constant $\hat{\varepsilon}$ restricts the relative error from going towards infinity when x is close to zero.

end of the error correction step the *correction tuple* is either given by $(\Delta = 0, k_1)$ or by $(\Delta \neq 0, k_2)$.

At the end of the procedure, correction tuples are compressed using `zlib`, as high-resolution simulations often exhibit high temporal coherence, yielding small numbers (Δ, k) with many repetitive bit patterns. In case the size of the compressed correction tuples exceeds the memory needed for a new permutation and B-spline coefficients, the algorithm falls back to plain ISABELA, picking the approach that yields better compression.

6 Results

For evaluation of the AMR and temporal compression approaches, we applied them to the *forced isotropic turbulence* data set from the *Johns Hopkins Turbulence Databases* [8] and the *turbulent combustion simulation S3D Direct Numerical Solver* data set [13]. As the latter data set contains large border regions with zero-valued cells for one of the attributes and time-dependent effects are less pronounced during the initial time steps, we cropped the data set to the interesting, non-trivially compressible center region and used time steps from the second half of the simulation in our experiments. Concretely, we cropped the *combustion* data set with scalar attributes *mixfrac* and *vort* to a $480 \times 192 \times 96$ center region with time steps $60-122$ of the original data set. The *isotropic* data set with vector *velocity* was cropped to a $64 \times 64 \times 64$ center region, using all 1024 time steps of the original data set.

Figure 9 provides a visual impression of low-resolution views generated from compressed three-level AMR representations of the data sets. From left to right, views generated from $1/64$ (level 1), $1/8$ (level 2), and the complete number (level 3) of voxels are shown. We conclude that already low-resolution visualizations provide approximate views of the data sets that might be sufficient for quick browsing during interactive analysis.

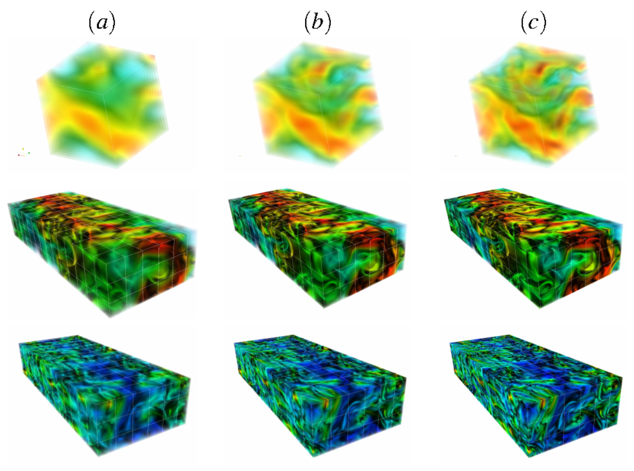


Figure 9: Decompressing data sets from different AMR levels: *(a)* $1/64$, *(b)* $1/8$ and *(c)* all voxels from original data set. *Top row:* isotropic, *velocity*. *Center row:* combustion, *mixfrac* *Bottom row:* combustion, *vort*.

The interlacing scheme for generation of AMR representations samples voxels from high-resolution data sets evenly for coarse levels of resolution. This, however, means that voxels used for low-resolution views are more distant from each other and therefore enjoy less spatial coherence. Due to the reduced spatial coherence between voxels in compression blocks, a negative impact on compression rates can be expected. Figure 10 compares the compression rates of data sets being compressed with only one level of resolution (i.e. no AMR), two LOR, and three LOR. The Figure

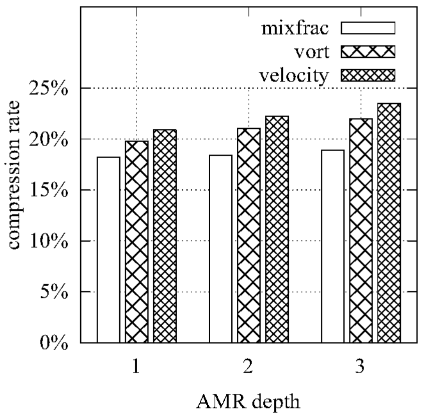


Figure 10: Costs of AMR in terms of compression rate. Compression rate increases by 1-3% for three levels of resolution as compared to data compression with only one level of resolution.

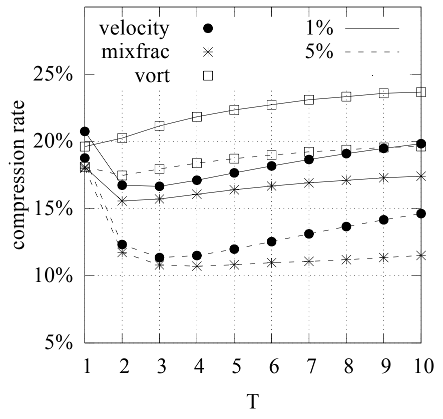
shows that compression rates increase moderately by 1-3% when a three level AMR is used as compared to a flat data layout.

Figure 11 evaluates the performance of our temporal compression approach without simultaneous AMR generation. The algorithm was run without and with fallback to plain ISABELA enabled, which allows the algorithm in non-reference time steps to choose between error correction w.r.t. the last reference time step and the generation of a new permutation and B-spline (see Section 5). The data sets were compressed with different values of T, i.e. the insertion rate of reference time steps. Figure 11 *(a)* shows the compression rate without fallback to plain ISABELA. It can be seen that for all variables but *vort* the compression improves for $T = 2 \ldots 4$. For larger T the gain in compression rate decays, since the noise of reordered sequences gets too strong and the compression worsens. The variable *vort* doesn't improve at all for $\rho = 1\%$ and only improves slightly for 5%. Figure 11 *(b)* shows the compression rate with fallback to plain ISABELA enabled. In comparison to the no-fallback case, the increase in the the compression rate for larger values T is clearly reduced. Best compression results are again obtained for smaller values of T ($T = 2-4$). For *velocity* and *mixfrac*, improvements of the compression rate of $\sim 2-4\%$ for $\rho = 1\%$ and $\sim 5-6\%$ for $\rho = 5\%$ can be observed. Compression of the attribute *vort* again benefits to a lesser extent from temporal compression, which can be attributed to a strong fluctuation in data values.

Figures 12 and 13 show the progress of the compression rate for one compression block in a randomly chosen sequence of 10 time steps for the isotropic and combustion data set respectively. *Top* and *bottom* rows show the compression rate without and with AMR for *(a)* $\rho = 1\%$ and $T = 2$ and *(b)* $\rho = 5\%$ and $T = 3$. It is clearly visible that a new permutation and the B-spline are only stored in reference time steps. Non-reference time steps usually contain snapping and quantization information. Sometimes, e.g. visible in Figure 13 bottom left, the amount of error correction bits in non-reference time steps becomes too large, resulting in a fallback to plain ISABELA.

7 Discussion

Support for compression into an AMR data format allows for the extraction of low resolution views and progressive loading of scientific data sets. This facilitates interactive data exploration e.g. when changing visualization parameters or viewpoints. Lower resolutions views to the data set are constructed by interlacing vox-



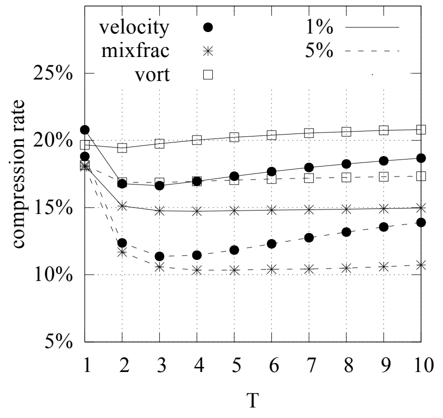


Figure 11: Performance of temporal compression. Every T-th time step ($T = 1 \dots 10$) a reference time step is inserted, $T = 1$ corresponds to no temporal compression. *Top:* without fallback and *Bottom:* with fallback to plain ISABELA.

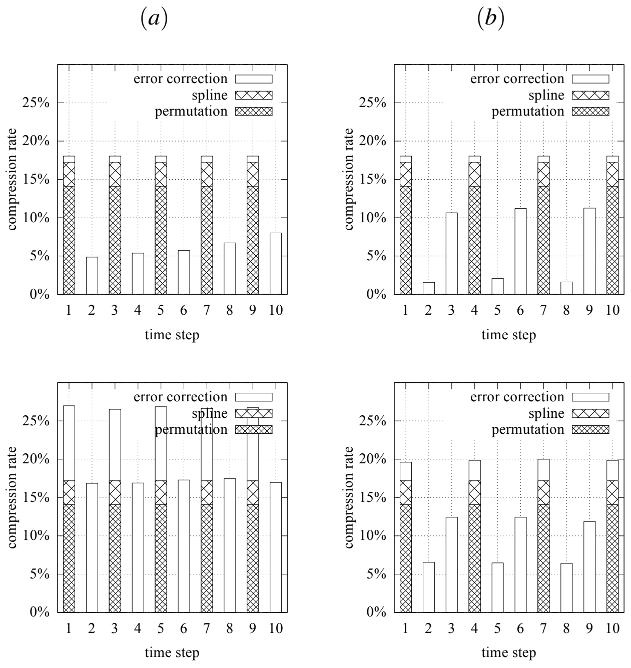


Figure 12: AMR + temporal coherence with isotropic, *velocity*. Compression rate for a sequence of time steps with *(a)* $\rho = 1\%$, $T = 2$ and *(b)* $\rho = 5\%$, $T = 3$ and *Top:* without AMR and *Bottom:* with AMR depth $D = 3$.

Variable	AMR	$T = 1$	$T = 2$	$T = 3$	$T = 4$
velocity	no	20.9%	17.1%	17.0%	17.4%
velocity	yes	23.5%	19.0%	18.6%	18.8%
mixfrac	no	18.2%	15.4%	15.1%	15.0%
mixfrac	yes	19.0%	17.3%	17.5%	17.8%
vort	no	19.6%	19.4%	19.7%	20.0%
vort	yes	21.8%	22.0%	22.5%	23.0%

Table 2: Compression rates for variables *velocity*, *mixfrac* and *vort*, relative error $\rho = 1\%$, showing impact of AMR ($D = 3$) and temporal compression ($T = 1$: every time step is a reference time step, i.e. no temporal compression, $T = 2$: every 2nd time step is a reference time step, etc.).

els of the full resolution data set, similar to the Adam7 algorithm used in progressive PNG. Compared to plain ISABELA, only minimal run-time overhead during compression is introduced. In order to generate data at lower resolutions, only a fraction of the total number of compression blocks have to be loaded, which decreases bandwidth requirements during interactive exploration.

In order to exploit temporal coherence in time-dependent data sets, we introduced a new scheme for temporal compression. Whereas plain ISABELA stores sort-order information and B-spline coefficients for each time step, our approach, instead, reuses sort-orders and B-splines from previous reference compression blocks. Although this introduces some noise in the sort orders at non-reference time steps, the increased costs for error correction are clearly outweighed by the costs saved for not storing a sort-order and a B-spline at all for non-reference time steps. While the difference encoding method for temporal compression from [7] effectively reconstructs the correct sort-order and B-spline at every time step that may be very close approximations of the real data values, our error correction scheme reduces the error only as much as needed to guarantee the user-defined relative error. This too indicates that our method requires less memory than difference encoding for temporal compression.

While the interlacing scheme for construction of the AMR data representation reduces the spatial coherence within the voxels of a compression block and thus has a negative effect on the compression rate, temporal coherence improves the compression rate. Table 2 compares the costs incurred by AMR and saved by temporal compression. It shows (again), that AMR introduces a penalty $1 - 3\%$ in compression rate, which can however be compensated through temporal compression. We conclude that the introduction of an AMR scheme has only minor negative effects on the compression rate which are far outweighed by its positive effects on interactive data exploration.

For sake of completeness, we further applied our compression method to the complete, uncropped combustion data set (Table 3). As expected, due to the large border regions with zero-valued *mixfrac*-attributes, the compression rate significantly improves for that attribute to 11.6% without and 14.1% with AMR. For the *vort* attribute, which has fluctuating values also in the border regions, the compression rate is similar to the compression rate in the center region.

Both introduced extensions to ISABELA, i.e. the multi-resolution support and the temporal compression scheme were designed with the goal of keeping ISABELA's *in-situ* compression

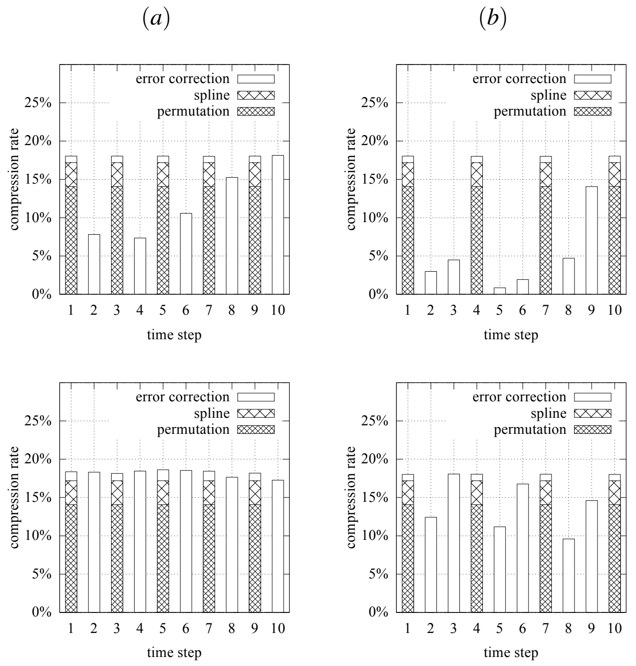


Figure 13: AMR + temporal compression with combustion, $mixfrac$. Compression rate for a sequence of time steps with *(a)* $\rho = 1\%$, $T = 2$ and *(b)* $\rho = 5\%$, $T = 3$ and *Top:* without AMR and *Bottom:* with AMR depth $D = 3$.

Variable	AMR	$T = 1$	$T = 2$	$T = 3$	$T = 4$
$mixfrac$	no	18.9%	13.6%	12.2%	11.6%
$mixfrac$	yes	21.4%	16.0%	14.6%	14.1%
$vort$	no	19.6%	19.5%	19.8%	19.9%
$vort$	yes	22.8%	22.7%	23.0%	23.2%

Table 3: Compression rates for the complete combustion data set, $480 \times 720 \times 120$ voxel, 122 time steps, with relative error bound $\rho = 1\%$, AMR depth $D = 3$, and every T-th time step as reference time step.

capabilities intact. Our AMR blocks have a typical size of 32^3 voxels, which can be expected to fit in the same compute process in HPC environments. Similarly, our temporal compression method requires only data from one reference time step in addition to the current time step in main memory. Moreover, data from the reference time steps is limited to permutation and B-spline information, which amount to $\sim 17\%$ of the full time step in typical configurations. These properties ensure the locality of the compression method.

As noted in [7], ISABELA introduces only a negligible overhead on simulations in terms of runtime. Furthermore, as compression blocks can be processed independently from each other, ISABELA is highly amenable to parallelization both during compression and decompression. In our own experiments, we measured the time for loading (read + decompress) the compressed subset of *isotropic* data set, consisting of 64^3 voxels and 1024 time steps. The dataset was compressed with AMR depth $D = 3$ and temporal compression setting $T = 3$, yielding a compressed data set of 1.42 GB. Experiments were conducted on a standard desktop workstation[2]. Figure 14 shows the loading times for all 1024 time steps of the full resolution data set where $1 \ldots 6$ threads were used. For example, with a

[2]Intel Core i7 980 @ 3.33 GHz CPU with six physical cores (twelve virtual threads through Intel Hyper-Threading), SATA 1TB hard disk

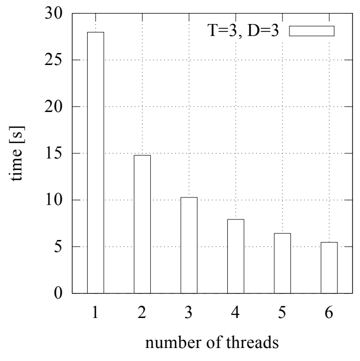


Figure 14: Total loading times for 1024 compressed time steps of the 64^3 isotropic data set, for different numbers of threads used for decompression.

serial implementation (1 thread), loading took 29.9 seconds, while the use of 6 threads reduced the loading time to 5.4 seconds, i.e. about $1/6$. These results confirm the excellent scalability of ISABELA decompression in the number of threads/compute cores. Average loading times for a single time step are thus only ~ 5.3 ms when six threads are used. Similarly, loading all time steps at the lowest resolution, i.e. $1/64$ of the data, takes about 85 ms with 6 threads.

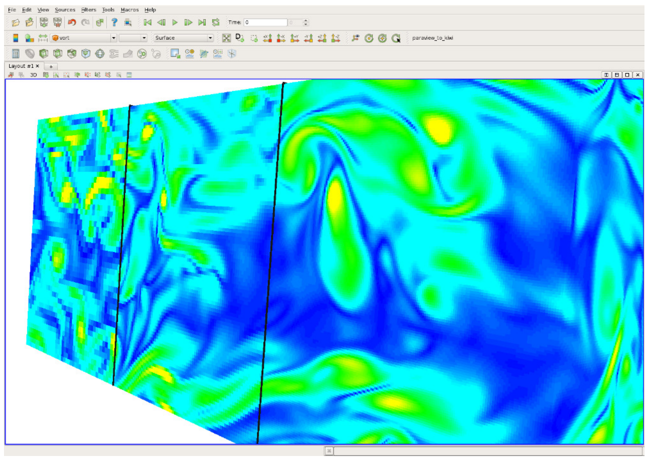


Figure 15: Compressed AMR data set loaded into Paraview via a custom reader: high resolution in front, mid resolution in center, low resolution in back.

Finally, our multi-resolution compression is interoperable with the Adaptive Multi Resolution (AMR) format of the ParaView/VTK toolkit [14] which follows the AMR scheme described in [2]. The VTK AMR format can be built up either in main memory during decompression or by explicit file conversion. This enables data import into ParaView (Figure 15) and other analysis tools that support the AMR format of VTK. Figure 16 shows the isotropic data set at different levels of resolution in the XSITE-CAVE, a 50 megapixel Virtual Reality display. Fast loading times, as possible with ISABELA compression, are particularly important in immersive environments. Future work will address issues of controlling AMR resolution levels in immersive settings, both interactively and automatically.

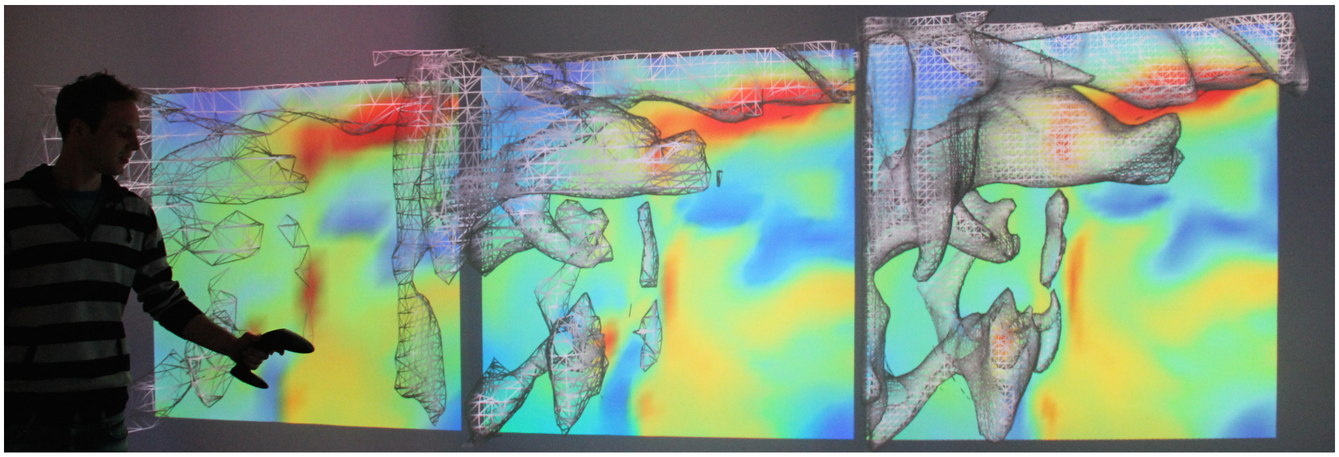


Figure 16: The isotropic data set at different LOR in a CAVE. From left to right, the resolutions of the plane plot and iso-surface are increasing.

8 Conclusion

We extended the ISABELA method for in-situ compression of scientific data sets with an interlacing data layout supporting direct adaptive multi-resolution (AMR) output and with a new approach for temporal compression. While the latter aims at enhancing the compression rate during the simulation phase, the former aims to support interactive data exploration during the analysis phase.

The AMR data output is based on subdividing the full data set into AMR blocks, which can be (de)compressed independently at multiple levels of resolution by following an interlacing scheme. The interlacing approach also supports progressive loading as lower resolution data are part of higher resolution representations. The interlacing scheme is in principle independent of ISABELA and can be applied to other in-situ compression algorithms, which is a topic for future work.

The main idea behind our method for temporal compression is to defer sorting and B-spline regression by only applying it every T-th time step in order to save memory in non-reference time steps. To handle the increasing noise in sort-orders for non-reference time steps, a new error correction scheme was introduced.

Both ISABELA extensions, AMR and the temporal compression scheme, support in-situ compression of scientific data in HPC environments. The in-situ compressed multi-resolution data can be converted straightforwardly into the Adaptive Multi Resolution (AMR) format of the ParaView/VTK toolkit.

Acknowledgements

This research was supported by the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center 920. The S3D Turbulent Combustion Simulation data set is made available by Dr. Jackqueline Chen at Sandia Laboratories through US Department of Energy's SciDAC Institute for Ultrascale Visualization.

References

[1] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 49:1–49:9, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[2] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82(1):64–84, May 1989.

[3] C. Brislawn, J. Woodring, S. Mniszewski, D. DeMarle, and J. Ahrens. Subband coding for large-scale scientific simulation data using JPEG 2000. In *Image Analysis and Interpretation (SSIAI), 2012 IEEE Southwest Symposium on*, pages 201–204, April 2012.

[4] J. Dongarra, J. Hittinger, J. Bell, L. Chacon, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster, and S. Wild. *Applied Mathematics Research for Exascale Computing*. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Feb 2014.

[5] J. Iverson, C. Kamath, and G. Karypis. Fast and effective lossy compression algorithms for scientific datasets. In *Proceedings of the 18th international conference on Parallel Processing*, Euro-Par'12, pages 843–856, Berlin, Heidelberg, 2012. Springer-Verlag.

[6] S. Klasky, H. Abbasi, J. Logan, M. Parashar, K. Schwan, A. Shoshani, M. Wolf, S. Ahern, I. Altintas, W. Bethel, L. Chacon, C. Chang, J. Chen, H. Childs, J. Cummings, S. Ethier, R. Grout, Z. Lin, Q. Liu, X. Ma, K. Moreland, V. Pascucci, N. Podhorszki, N. Samatova, W. Schroeder, R. Tchoua, K. Wu, and W. Yu. In situ data processing for extreme scale computing. In *Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC)*, Denver, CO, 06/2011 2011.

[7] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In E. Jeannot, R. Namyst, and J. Roman, editors, *Euro-Par (1)*, volume 6852 of *Lecture Notes in Computer Science*, pages 366–379. Springer, 2011.

[8] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, A. Burns, S. Chen, E. Szalay, and G. Eyink. A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence," arxiv.org, 2008.

[9] K.-L. Ma. In situ visualization at extreme scale: Challenges and opportunities. *Computer Graphics and Applications, IEEE*, 29(6):14 – 19, nov.-dec. 2009.

[10] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova. In-situ processing and visualization for ultrascale simulations. *Journal of Physics: Conference Series*, 78(1):012043, July 2007.

[11] A. Tikhonova, H. Yu, C. D. Correa, J. H. Chen, and K.-L. Ma. A preview and exploratory technique for large-scale scientific simulations. In T. Kuhlen, R. Pajarola, and K. Zhou, editors, *EGPGV*, pages 111–120. Eurographics Association, 2011.

[12] P. C. Wong, H.-W. Shen, C. R. Johnson, C. Chen, and R. B. Ross. The top 10 challenges in extreme-scale visual analytics. *IEEE Computer Graphics and Applications*, 32(4):63–67, 2012.

[13] C. Yoo, J. Chen, and R. Sankaran. Direct numerical simulation of turbulent lifted hydrogen/air jet flames in heated coflow. In *2nd ECCOMAS Thematic Conference on Computational Combustion in Delft, the Netherlands*, 2007.

[14] G. Zagaris, C. Law, and B. Geveci. Visualization and analysis of AMR datasets. *Kitware Source*, 18:2–6, 07 2011.