Deep Hierarchical Super-Resolution for Scientific Data Reduction and Visualization

Paper Type: algorithm/technique

Abstract—We present an approach for hierarchical super resolution (SR) using neural networks on an octree data representation. We train a hierarchy of neural networks, each capable of $2\times$ upscaling in each spatial dimension between two levels of detail, and use these networks in tandem to facilitate large scale factor super resolution, scaling with the number of trained networks. We utilize these networks in a hierarchical super resolution algorithm that upscales multiresolution data to a uniform high resolution that reduces seam artifacts on octree node boundaries. We evaluate application of this algorithm in a data reduction framework by dynamically downscaling input data to an octree-based data structure to represent the multiresolution data before compressing for additional storage reduction. We demonstrate that our approach reduces seam artifacts common to multiresolution data formats, and show how neural network super resolution assisted data reduction can preserve global features better than compressors alone at the same compression ratios.

Index Terms—Octree, Super Resolution, Neural Networks.

1 Introduction

Data analysis and visualization are essential tasks for scientists to understand large-scale output from simulations [25, 53, 58, 60]. Typically, scientists run simulations which generate high resolution spatiotemporal data and write some portion of the generated data to storage for post-hoc analysis. However, with supercomputers reaching exascale computing capability (at least 1 exaFLOPS) this decade, scientists are able to generate more simulation data than on previous generation's petascale supercomputers. For example, Nyx [3] is a astrophysics simulation for bayronic and cold dark matter that can output up to 4096^3 size volumes per simulated timestep [15] which take up over 4 TB each, and the Johns Hopkins Turbulence Database (JHTDB) hosts computational fluid dynamics simulation results up to 8192^3 [82] which take up to 16 TB per timestep. However, current storage technology cannot save the data at the rate they are being generated [13], making it infeasible to store this much raw data for post-hoc analysis and visualization.

One approach for avoiding this I/O limitation is to use error-bounded lossy compressors like ZFP [44] and SZ [14] to reduce the data's footprint before saving it to persistent storage. Error-bounded lossy compressors ensure that a target error bound will be satisfied after (de)compression. While some compression methods aim to preserve features found in specific data types, such as the work by Liang et al. [42] to compress vector fields, error-bound compressors generally only consider the error between the raw floating point value and its (de)compressed value when compressing, which can distort spatial features present in the data when visualizing by not accounting for spatial coherence.

An alternate approach for avoiding the I/O bottleneck is to simply discard a portion of simulated data and recover it post-hoc via super resolution algorithms. Data is discarded by saving timesteps of the simulation infrequently, such as saving once every 100 timesteps, or by spatially subsampling the data, such as from N^3 to $(\frac{N}{2})^3$. Interpolation techniques like linear or bicubic interpolation are used to fill in the discarded data post-hoc, but can lead to overly smooth results and missing high-frequency features important for analysis. Due to the recent advancements of machine learning (ML), super resolution (SR) methods applied to scientific data have seen better feature reconstruction than interpolation methods [21–23, 84]. Using neural networks (NNs) for non-error-bounded lossy compression has been studied extensively for image compression with results exceeding JPEG and JPEG2000 [1, 5, 26, 66–68]. A major benefit of machine learning for data reconstruction is ML's ability to preserve global features well due to the learned statistics across large spatial and temporal domains. These global features are often learned implicitly, but can be learned explicitly via loss functions. Although these SR models for scientific data reconstruct global features well, they are limited to single scale factor SR. Scientific data is often modeled with hierarchical structures such as an octree, but current SR models cannot support this data format.

In this paper, we aim to give scientists a data reduction algorithm that uses neural network super resolution along with compressors in the data reduction-reconstruction pipeline to preserve global features better than compressors alone. Since a single network trained for $N\times$ SR would only offer two compression options - either the data is downscaled or it is not - we give more flexibility to data reduction by training a hierarchy of networks G to allow $2\times, 4\times, ..., 2^{|G|}\times$ super resolution in each spatial dimension. Then we reduce the data by dynamically downscaling the volume into an octree, a multiresolution data format, which is then compressed with a compressor. To accommodate this data format, we design a multiresolution super resolution algorithm for our hierarchy of NNs to upscale this octree data format to a uniform resolution with minimal seam artifacts, which are prevalent in multiresolution data [46, 72–74]. Our process is as follows:

1. **Pre-training** We collect full resolution simulation output and train a hierarchy of SR neural networks, one model at a time, with each model responsible for $2\times$ SR between downscaling levels (..., $8\times \rightarrow 4\times$, $4\times \rightarrow 2\times$, $2\times \rightarrow$full resolution).
2. **Error bounded data modelling and compression** Data is dynamically downscaled in a trial and error fashion into an octree-based data format conditional on a user set error bound. The multiresolution downscaled data is subsequently compressed using an error-bounded lossy compressor.
3. **Post-hoc data decompression and reconstruction** In post-hoc analysis, data is decompressed to the octree format and finally upscaled using our multiresolution super resolution algorithm with the pre-trained hierarchy of networks.

In our results, we evaluate the trained NN hierarchies and verify that they outperform baseline interpolation techniques. We demonstrate our data reduction framework's effectiveness at global feature preservation by qualitatively and quantitatively comparing reconstructed data with and without our method at the same compression ratios. We also demonstrate how our octree representation allows for further data reduction compared to using a uniform downscaling rate. Finally, we discuss the limitations and future work for our approach.

In summary, the main contributions of this paper are:

- A super resolution enhanced data reduction pipeline to retain global features present during visualization by using a pretrained NN hierarchy for super resolution.
- A hierarchy of NNs used for SR that give improved results over interpolation methods and allow SR at scale factors on a power of 2 up to $64\times$. Each network is trained individually, as opposed

to end-to-end as in many other SR algorithms.
- A multiresolution super resolution algorithm for an octree-based data representation that upscales multiresolution data to a uniform resolution with minimal seam artifacts.

2 Related Works

We review related works in three general data reduction techniques: hierarchical data representations, super resolution methods for scientific data, and compression techniques for scientific data. Our work is part of a larger category of research called AI4VIS or ML4VIS, which applies artificial intelligence algorithms to enhance parts of the visualization and analysis process used by scientists. We recommend the surveys by Wu et al. [80] and Wang et al. [76] for recent efforts in ML4VIS.

There is also related research concerning machine learning and both lossy image compression [5, 26, 52, 67, 68] and lossless image compression [12, 29, 70] within the computer vision community. One approach by Cao et al. [12] for lossless image compression uses NN super resolution and entropy coding. The networks in this approach learn a distribution of super resolution results and then compresses the ground truth's position within that distribution to losslessly compress, whereas ours is a deterministic mapping from low resolution to high resolution. For more information on machine learning assisted compression for image data, we refer to the survey by Patel et al. [55].

2.1 Hierarchical data representations for scientific data

Hierarchical data representations have been used in scientific applications to reduce computation and storage requirements by more efficiently allocating resources available. Rendering pipelines have utilized octrees [51] for efficient and scalable volume rendering by dynamically loading data from different octree depths depending on the current camera position and angle. Gobbetti et al. [19] organize large scalar fields into an octree structure and only feed the GPU information relevant to the current viewpoint and transfer function at each frame. Knoll et al. [37] create a technique for isosurface ray tracing for large octree volumes. Wilhelms and Van Gelder [79] use octrees for isosurface generation when the regions of interest are unknown. A key problem for rendering hierarchical data is the seam artifact that can occur between data blocks. Methods for reducing the effect of this issue are presented by Santos et al. [46], Wald et al. [72], and Wang et al. [74]. A survey of octree-based rendering methods is available by Knoll [36].

Aside from rendering, hierarchical data formats have also been used during simulation to improve performance. Losasso et al. [47] create a method for simulating water and smoke on an unrestricted octree to capture small scale visual detail and allow for efficient solving of the Poisson equation. Popinet [56] uses quadtrees and octrees for a flexible and efficient approach to solving time-dependent incompressible Euler equations. Adaptive mesh refinement (AMR) schemes, a similar hierarchical data representation, have been utilized in numerous scientific software packages to improve performance by using a coarse mesh where there is little detail and fine meshes where precision is more important. Berger and Colella first design the AMR scheme for use with hyperbolic partial differential equations [9] and two dimensional shock dynamics [8]. Different implementations of AMR are implemented in modern simulation packages such as Enzo [54], Chombo [49], LAVA [35], Nyx [3], and AMReX [83].

These hierarchical representations can also be used for data reduction. Knoll et al. [37] use an octree representation that only uses a fraction of the original data's memory footprint, similar to the approach of Velasco and Torres [71], to accelerate volume rendering. Bhatia et al. [10] introduce AMM, an adaptive multilinear mesh framework that reduces the memory footprint of raw data using tensor products of linear B-spline wavelets and allow a tradeoff between numerical precision and spatial resolution. Hoang et al. [28] use a precision-resolution tree, and an encoding of the tree, to perform data reduction on scalar field data. Ainsworth et al. [2] create a multilevel technique to compress univariate data that allows users to select a level that minimizes memory use while meeting some tolerance level. Another multilevel technique is presented in MGARD [43], which uses adaptive error based coefficient quantization to enable different tolerances at different levels of detail, and proposes to use the multilevel data decomposition as a preconditioner to terminate at an appropriate level.

Hierarchical data representations have also been used in ML works. Riegler et al. [57] create octree convolution, pooling, and upscaling layers for a NN to classify octree data representing bathtubs, beds, dressers, etc. Tatarchenko et al. [65] create a deep colvolutional decoder that learns to decode dense input to octree representations of 3D objects. Martel et al. [50] and Takikawa et al. [63] use hierarchical data representations to assist with implicit neural representation. Our hierarchical data representation is similar to AMR and typical octrees but with an optimization for our purposes of multiresolution super resolution.

2.2 Super resolution for scientific data

There are two categories of super resolution: spatial super resolution (SSR) and temporal super resolution (TSR). Spatial super resolution aims to increase the spatial resolution of input data. Zhou et al. [84] use a 3D convolutional neural network (CNN) to perform SSR on volumes with better feature reconstruction than trilinear interpolation or cubic-spline interpolation. Guo et al. [21] use three NNs in parallel to perform SSR on 3D vector fields. Xie et al. create tempoGAN [81], which upscales fluid flows for temporally consistent high resolution output. Fukami et al. compare two ML-based SSR methods for 2D fluid flow [16]. Höhlein et al. [30] use CNNs to perform wind field downscaling, which learns the mapping from coarse input data to fine predicted output data. SSR-TVD by Han and Wang [23] uses a generative adversarial network (GAN) for upscaling volumes such that the upscaled output frame is temporally coherent with adjacent timesteps. A similar line of research to SSR in the computer vision community is called image super resolution (ISR), and we refer readers to a survey by Wang et al. [78] for recent techniques. With the recent advances in super resolution, Jakob et al. have created a public 2D flow dataset with varying Reynold's number, citing that ML methods are a powerful method for interpolating flow maps and data reduction [32].

Temporal super resolution aims to increase the temporal resolution of input data. TSR-TVD by Han et al. [22] use a convolutional LSTM to learn the recurrence between timesteps efficiently to recover interpolated timesteps with higher accuracy than linear interpolation. Other methods perform both spatial and temporal super resolution together with spatiotemporal super resolution (STSR). Fukami et al. [17] performs STSR on fluid flow data using two networks in a 3D+1D approach, where the spatial dimension is upscaled first, and then the temporal dimension. Serifi et al. [59] experiment with CNN architectures for climate data downscaling in both the space and time dimensions. MeshfreeFlowNet [33], proposed by Jiang et al., trains a network to allow inference for arbitrarily sized spatiotemporal domains with PDEs impacting training through a loss function. Lu et al. [48] do not use super resolution, but rather an ML approach called implicit neural representation in which they overfit a coordinate network to a 3D scalar field and then query the network using coordinates instead of saving the volume to storage. Ours differs by reusing a network to upscale multiple timesteps instead of implicitly representing a single scalar field. Ours also differs from previous work as being the first to perform multiresolution super resolution using a hierarchy of neural networks with minimal seam artifacts introduced.

2.3 Compression for scientific data

Compressors for scientific data can be classified as lossless or lossy. Lossless compressors, like FPC [11], have perfect reconstruction of the original data. Lossy compressors do not perfectly reconstruct the original data, but usually deliver larger compression ratios for interactive visualization or data reduction. For example, COVRA [18] offers an interactive and efficient decompression system for visualizing volumentric data in realtime by decompressing hierarchical blocks on the GPU. CudaCompress [69] uses a wavelet-based compression system and can decode bricks on the GPU for efficient volume ray-casting. Lossy compressors can be further classified as non-error-bounded or error-bounded. Error bounded lossy compressors allow a user to specify a local error which the compressor will guarantee is not exceeded. Our

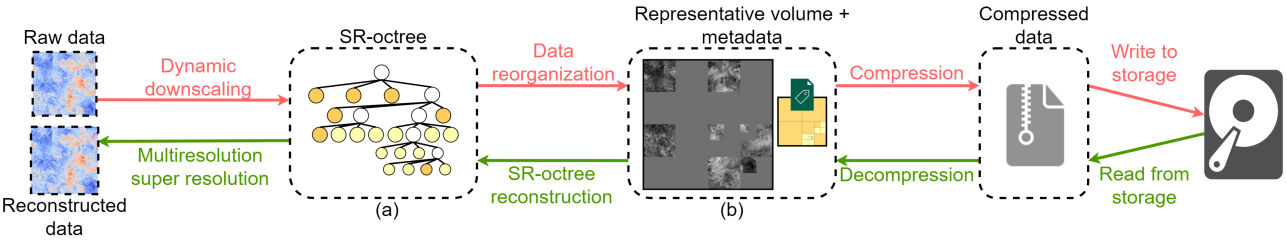

Fig. 1. An overview of our data reduction and reconstruction pipeline. During data reduction, raw data is reduced to a multiresolution format, represented with a data structure we design called an SR-octree (a). That multiresolution data is then reorganized into a representative volume (b), which is compressed along with metadata using a lossy/lossless compressor. During data reconstruction, the pipeline operates in reverse, except to recover the original data from the SR-octree, we use our multiresolution super resolution algorithm.

approach presented in this paper is considered an error-bound lossy data reduction algorithm that uses an error-bound compressor. For a comprehensive overview of state of the art scientific data compression, we refer to surveys by Li et al. [39], Balsa et al. [7], and Hoang et al. [27].

Error-bound lossy compressors can be classified as prediction- or transformation-based. FPZIP [45] and SZ [14, 41, 64] are prediction-based error-bounded lossy compressors. FPZIP uses a Lorenzo predictor [31] followed by integer mapping on both the predicted and actual data. Then the residuals between the predicted and actual data are sectioned into entropy codes for transmission. SZ creates an improved Lorenzo prediction and dynamically encodes blocks of data with either the Lorenzo prediction or an optimized linear regression method based on which method performs best. ZFP [44] is a transformation-based compressor that analyzes each 4^d sized block and encodes them with a specified number of bits per block through an orthogonal block transform and embedded coding. TTHRESH [6] is a lossy compressor based on the Tucker decomposition that compresses 3D and higher dimensional volumes. In our proposed pipeline, any of the mentioned compressors could be used for varying benefits and drawbacks depending on the compressor. Though some compressors may perform especially well under certain circumstances, we use SZ in our pipeline since it works for 2D as well as 3D and is considered state-of-the-art.

3 Overview

An overview for our data reduction and reconstruction pipeline is shown in Fig. 1. We use a hierarchy of NNs G where each network in the SR hierarchy is trained to perform $2\times$ super resolution between two downscaling levels. By using these networks in tandem, the SR hierarchy is capable of $2\times$, $4\times$, ..., $2^{|G|}\times$ SR, where $|G|$ is the number of networks in the hierarchy. For data reduction purposes, this gives more flexibility than a single model because data can be downscaled to any of those supported scale factors. We use this trained hierarchy during the multiresolution super resolution part of Fig. 1, and detail the SR hierarchy structure and architecture in Sect. 4.1.

We introduce a error-bounded dynamic downscaling algorithm, outlined in Sect. 4.2 and used in the dynamic downscaling part of Fig. 1, that will downscale spatial regions of a single volume to a multiresolution data format. To model the multiresolution data, we use an octree that is optimized for our purposes of dynamic downscaling, which we call an SR-octree (Fig. 1(a)). After the dynamic downscaling, data is stitched together into a representative volume and compressed using a compressor, shown in Fig. 1(b) and the subsequent compression.

Though the multiresolution format may reduce storage costs, standard NNs are not designed to perform multiresolution upscaling, and upscaling each octree node individually would introduce seam artifacts between nodes. We design a multiresolution super resolution algorithm for our hierarchy that reduces the impact of seam artifacts, detailed in Sect. 4.3, and used when reconstructing data in Fig. 1.

4 Our Method

Our NN hierarchy for super resolution is outlined in Sect. 4.1. Next, we discuss the multiresolution data format we use and the dynamic downscaling algorithm together in Sect. 4.2. Finally, we discuss our solution for upscaling the multiresolution data with minimal seam artifacts in Sect. 4.3.

4.1 Spatial super resolution hierarchy

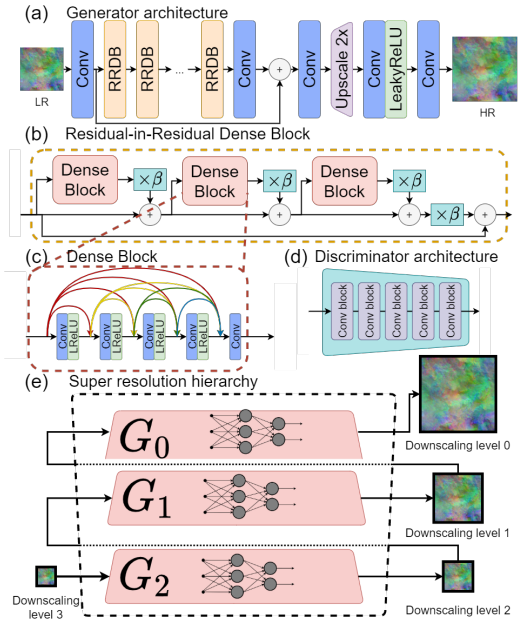

Fig. 2. (a) The spatial super resolution network architecture, utilizing Residual-in-Residual Dense Blocks (b) which contain dense blocks (c). (d) The discriminator network. (e) An example of a hierarchy of NNs trained to perform $2\times$ SR between different downscaling levels.

We use a hierarchy of SR networks to improve data reduction by allowing multiple SR factors. Most SR models support a static upscaling ratio of $N\times$ in each spatial dimension, which limits data reduction because the network only supports data downscaled by exactly $N\times$. Therefore, the compression ratio is limited to either $1\times$ (no data reduction) and $N^D\times$, where N is the upscaling ratio the model supports, and D is the number of dimensions.

In the computer vision field, researchers have created models that support multiple scale factor SR such as LapSRN [38] and SinGAN [61]. The PSNR degrades as the scale factor increases, but multiple SR factors gives the user a choice for the reconstruction quality/data reduction trade-off. In this paper, we use a hierarchy of identical generative adversarial networks (GANs), with each one trained to perform $2\times$ super resolution in each spatial dimension between downscaling levels. We define downscaling level 0 as full resolution and downscaling level i is data that is downscaled by a factor of 2^i in each dimension. Together, they form a hierarchy of networks G that supports up to $2^{|G|}\times$ SR. An example of a SR hierarchy is shown in Fig. 2(e).

4.1.1 Spatial super resolution generator

We use a super resolution generator architecture based on the enhanced super-resolution generative adversarial network (ESRGAN) [77], a

state-of-the-art GAN for upscaling single images from a low resolution (LR) input to a high resolution (HR) output. The full architecture is depicted in Fig. 2. We use the ESRGAN architecture due to three of the main features of the network: (1) Pre-upsample learning, (2) pixel/voxel shuffling, and (3) residual learning.

Pre-upsample learning allows us to reduce our memory requirement during training. *Pre-upsample* learning means that the input remains in the LR input space when convolution kernel weights are learned *before* being upscaled to the final output resolution. This method is the opposite of *post-upsampling*, which first upscales the data and then learns weights for convolution kernels in the HR space. Training and inferring in this high dimensional space with post-upsampling learning is much more computationally expensive than in the low dimensional space because there is twice the number of pixels/voxels in each dimension which leads to $4\times$ (in 2D) or $8\times$ (in 3D) as many convolution operations to happen per layer compared to the LR input. For example, pre-upsample learning would operate on a 32^2 input and then be upscaled to 64^2, whereas post-upsample learning would first upscale to 64^2, increasing memory and computation cost fourfold. Given the large volumes to train with and limited memory budget, pre-upsample learning is a better choice.

Recent work for volumetric super resolution has found success with a method called voxel shuffling, first presented in TSR-TVD [22], which is the 3D counterpart of pixel shuffling [62]. Voxel shuffling reshapes a volume of shape $[u^3C, L, W, H]$ to $[C, uL, uW, uH]$, where u is the upscaling ratio, C is the number of channels (feature maps) in the input, and L, W, and H are the 3 spatial dimension sizes. In our case, $u = 2$, and each spatial dimension is increased by a factor of 2, while the channel dimension reduces by a factor of 8. We use pixel shuffling in 2D, and voxel shuffling in 3D.

The ESRGAN architecture makes use of residual learning, which has been shown to improve results and stability of deep CNNs [24]. Residual learning means that output from layers in the network are added onto either input data (potentially transformed) or previous layer output. ESRGAN employs residual learning across many layers with d sequential residual-in-residual dense blocks (RRDB). RRDB have proven to be powerful for SR tasks due to the skip connections in the dense blocks between shallow and deeper features, which allow deeper layers to use feature maps from earlier in the network to influence weight learning. RRDBs also add two levels of residuals within them, giving them their name, shown in Fig. 2(b). First, each dense block [75], creates a residual which is added back to the input to the block. Second, the entire RRDB creates a residual which is added back to its input. Each residual is multiplied by a constant β between 0 and 1 (chosen to be 0.2 in Sect. 5.2) that controls the influence of each residual. This follows a technique called residual scaling which prevents instability in deep networks. After input has gone through the d RRDBs, that output is used as another residual and added back to a shallow feature map from before the first RRDB. The many residuals used offer pathways for gradients to be computed through, which helps work against the vanishing gradient problem.

4.1.2 Spatial super resolution discriminator

Each generator is coupled with a patch discriminator, shown in Fig. 2(d), which classifies overlapping patches as "real" (looks like its from the ground truth data) or "fake" (does not look like it is from the ground truth data). This network is designed to be relatively shallow with a small receptive field. Each Conv block (convolution, batch normalization, leaky rectified linear unit (LeakyReLU)) has a kernel size of 3 and a stride of 1. With only 5 layers there is a corresponding receptive field of 11 for each discriminator. This small receptive field is to prevent redundant learning across different scales. By keeping the receptive field small, the patch discriminator will only be judging the realness of *local* features in a small window, which were the responsibility of the paired generator to synthesize. Global features present in the data will either be ground truth LR data (which are already real), or are synthesized features from a previous generator. If the previous generators have learned to accurately reconstruct features at their scale, then there would be no reason to learn those features again in this discriminator.

We use a GAN for two reasons. First, they have shown to be powerful for reconstructing important high-frequency features for both images as well as scientific data. Second, we expect that the output from a GAN, specifically Wasserstein GANs (WGANs) [4], will more accurately match the distribution of ground truth data, which is important when using multiple super resolution generators from the SR hierarchy in tandem. When training, only ground truth data is used as both input and target output. Consequently, each network is not conditioned to super-resolve output from previous generators as in works like LapSRN [38] and SinGAN [61]. Therefore, the SR hierarchy is only learning to upscale data from within the ground truth distribution. By constraining the output of each generator to a specific distribution, that of the real data, we should see less compounding errors as networks are used sequentially. This distribution difference minimization is precisely the goal of WGANs.

4.1.3 Loss functions

We use two components in our loss function: a reconstruction component and a adversarial component, with a weight factor for each. The loss function is computed as:

$$\mathscr{L} = \lambda_{rec}\mathscr{L}_{rec} + \lambda_{adv}\mathscr{L}_{adv} \tag{1}$$

where λ_{rec} is the weight factor for $\mathscr{L}_{rec}$, our reconstruction loss, and λ_{adv} is the weight factor for $\mathscr{L}_{adv}$, our adversarial loss.

For the following loss functions, we denote the output from our network as $\hat{F}$ and the ground truth as F. Our generator G performs $G(F^{LR}) = \hat{F}$, where F^{LR} is F downscaled in each spatial dimension by a factor of 2, and we denote our discriminator as D.

Reconstruction loss. The reconstruction loss is defined as

$$\mathscr{L}_{rec} = \frac{1}{L \times W \times H} \sum_{i \in voxels} ||F_i - \hat{F}_i||_1 \tag{2}$$

where $||\cdot||_1$ is the l^1 norm. We use the l^1 norm instead of the l^2 norm because the l^2 norm tends to have overly-smoothed output [78]. $\mathscr{L}_{rec}$ is subject to minimization for the generator during optimization.

Adversarial loss. We use the WGAN-gradient penalty (WGAN-GP) loss [20] for two reasons. First, we expect that while super resolving multiple times through a hierarchy, error may compound at each level of the hierarchy. Since WGAN [4] aims to minimize the Earth-Mover's distance (Wasserstein-1 distance) between the output of the NN and the ground truth distributions, we can expect that NN output should have an output distribution that better matches the ground truth data than without the WGAN (i.e. a single CNN). This may help reduce compounding errors when performing high scale factor super resolution. Additionally, the gradient penalty increases training stability. The loss is defined as

$$\mathscr{L}_{adv} = \mathbb{E}_{F \in \mathbb{P}_r}[D(F)] - \mathbb{E}_{\hat{F} \in \mathbb{P}_g}[D(\hat{F})] \tag{3}$$

where $\mathbb{E}[\cdot]$ is the expectation operator, $\mathbb{P}_g$ is the model distribution of upscaled volumes, and $\mathbb{P}_r$ is the data distribution of the full resolution ground truth volumes. $\mathscr{L}_{adv}$ is subject to minimization for the generator network, to make the generator capture the data distribution of ground truth data well, and is subject to maximization for the discriminator, so that the discriminator can accurately tell real HR volumes apart from synthesized volumes from the generator.

4.2 Dynamic downscaling to a multiresolution data format

Now that we have a NN hierarchy G that can perform $2^i\times$ SR, for $1 \le i \le |G|$, we give more flexibility to the data reduction process by dynamically downscaling data according to the presence of features instead of uniformly downscaling. Scientific data do not always have complex features present throughout the entire volume, and large regions of a volume may be constant. Therefore, instead of uniformly downscaling a full volume by a single scale factor, we use an octree data representation and dynamic downscaling algorithm to reduce a volume dynamically while still meeting an error bound after SR.

4.2.1 Dynamic downscaling

Our dynamic downscaling algorithm is described with the following pseudocode, beginning with the full resolution volume F.

1. Downscale the input data D by $2\times$ in each dimension to $D\downarrow$
2. Reconstruct $\hat{F}$, the full resolution approximation of F, from the octree data including $D\downarrow$ by using our multiresolution SR algorithm (covered in Sect. 4.3).
3. Check to see if the error (RMSE) between F and $\hat{F}$ is acceptable.
   - 3 a) If the error is less than the error-bound, see if $D\downarrow$ can be downscaled further (return to step 1 with $D\downarrow$).
   - 3 b) Otherwise, reset $D\downarrow$ to D. Split D into its octants, and recurse on each octant (return to step 1 for each octant).

Note that step 2 reconstructs $\hat{F}$, the full resolution approximated volume of F, the ground truth. Though this is more expensive than checking octree nodes individually, we upscale all nodes together using our multiresolution super resolution algorithm to avoid seams. CNNs will create artifacts on output borders due to zero padding. In NNs, convolution layers use zero padding, which pads 0 values on the border of the input, to guarantee that the output feature maps have the same spatial dimensions as the input to the layer (i.e. 32^2 gets padded to 34^2 before a 3×3 convolution, so the output is 32^2). Guaranteed output size is desirable for specific operations like pixel/voxel shuffling. As a drawback, when convolutional kernels convolve with the edges and corners, the zero padding will influence the convolved feature map and increase error. This effect is magnified in deeper networks since multiple convolution layers with zero padding will feed in to each other. Zero padding already influences boundary voxels of output from an SR neural network, but performing SR on individual octants would increase the number of voxels affected by zero padding leading to an increased error along the boundaries between upscaled octants. Therefore, we develop the multiresolution super resolution algorithm to reduce zero padding influence which gives us an approximation $\hat{F}$ that has minimal seam artifacts and reports and accurate error for our error-bound.

We have three stopping criteria for any octant to not be subdivided further (3b in the above mentioned algorithm). First, if the size of an octant is only 2^D regardless of the downscaling level, where D is the number of dimensions, then it is not useful to split any more, since those octants of size 1^D cannot be downscaled further. Second, octants cannot be downscaled past $MAXDSL = |G|$ because the NN hierarchy cannot perform SR past that scale factor. Our final stopping condition is a *minChunk* hyperparameter that limits the algorithm from spatially subdividing if a data chunk only represents *minChunk* voxels in each dimension. Smaller *minChunk* will trade off compression ratio for time.

The worst case time complexity of our dynamic downscaling algorithm is $\mathcal{O}(2\times(\frac{N}{minChunk})^D \times \mathcal{O}(SR))$, where $\mathcal{O}(\cdot)$ is the time complexity, N is the spatial dimension with the fewest number of voxels in it, and $\mathcal{O}(SR)$ is the time complexity for the multiresolution super resolution algorithm. The worse case happens when no region can be downscaled at all. The best case time complexity is $\mathcal{O}(MAXDSL\times\mathcal{O}(SR))$, and happens when the entire volume can be downscaled to the lowest resolution while still meeting the error criteria. Since NN inference can be relatively slow compared to other upscaling algorithms (i.e. $\mathcal{O}(SR)$ is large, so step 2 in the pseudocode for dynamic downscaling at the beginning of Sect. 4.2 is computationally expensive), we use linear interpolation as an efficient conservative heuristic/proxy for the NN's reconstruction quality during dynamic downscaling. This reduces $\mathcal{O}(SR)$, but limits data reduction since the reconstructed data will likely have higher quality than the bound given since the NN is used to upscale instead of linear interpolation. We find this a necessary and acceptable trade-off since the neural network inference is up to 3 orders of magnitude slower than bilinear or trilinear interpolation (0.00007 sec. for bilinear interpolation vs. 0.05578 sec. for NN SR in an experiment). Since we use the interpolation heuristic, we perform a check after the dynamic downscaling finishes that uses the NN hierarchy instead of the interpolation proxy to ensure that the SR reconstructed data does meet the target error bound. If it is not met (which empirically has never happened), the dynamic downscaling process is repeated with an error bound that is progressively more strict until the error bound is met.

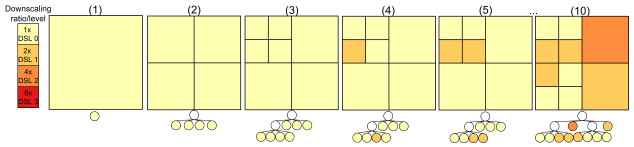

Fig. 3. An example of the progression of a volume (square) and SR-octree (tree structure beneath each square) during the dynamic downscaling process using a quadtree for a simple visualization. Data starts at full resolution with no downscaling in (1). A trial and error step failed, so the input was subdivided into its quadrants in (2). The same is repeated on the top left quadrant in (3). During (4, 5), the trial and error for the light orange ($2\times$) octants were successful, so that data is kept at downscaling level 1. The final example SR-octree is shown in (10), where DSL 0 regions could not be downscaled without failing the error-bound check or a stopping criteria was met.

4.2.2 SR-octree

To model the multiresolution data in our dynamic downscaling algorithm, we create a modified octree data structure called an SR-octree. Our SR-octree is designed to optimize data reduction time for our dynamic downscaling algorithm by minimizing the number of nodes in the tree and keeping data contiguous until it is spatially divided.

Like most octrees, children of a parent node represent the 8 octants that compose the parent. That is, the octree data structure defines the spatial partitioning, where the root node refers to the entire domain. Unique to the SR-octree is a *downscaling_level* attribute on each node that represents the factor that data has been downscaled in each spatial dimension within that node. This attribute allows us to represent contiguous large regions with a uniform downscaling ratio with one node, because our dynamic downscaling algorithm will only subdivide nodes when it is necessary. Specifically, each leaf node in the SR-octree holds a volume of data defined by the node's spatial partition and downscaling level. In Fig. 3, we show what the SR-octree looks like during the dynamic downscaling process. White nodes are internal nodes that hold no data and only point to their child octants. Nodes are colored according to their downscaling level shown by the key on the left.

4.2.3 Compression stage

After the SR-octree T is output from the dynamic downscaling algorithm, we use SZ [41], a state of the art error-bounded lossy compressor, to compress the multiresolution data. Though we use SZ, any compressor supporting grid-structured floating point data may be used.

Instead of compressing each node's data separately, which may have a large overhead for each node depending on the compressor, we combine all data into a single *representative volume*, shown in Fig. 1(b)-left, to compress. To fill the representative volume with the SR-octree data, we stitch together all nodes into an empty full resolution volume. Since some data is downscaled at different scale factors, we have empty space in the volume, which we choose to fill the the average data value so that run-length encoding can efficiently compress these areas.

Starting with a full resolution empty volume might be excessive if all data has been downscaled by a minimum scale factor. For example, if all nodes in the SR-octree have a downscaling level of at least 1, then we can instead stitch together the SR-octree nodes into an empty volume that is half of the original volume size in each spatial dimension. This extends for any minimum downscaling level.

After creating the representative volume, we compress with SZ. We want to compress without distorting the data such that the reconstructed volume's error after SR is still acceptable. Since CNNs learn spatial statistics, we use a pointwise error bound of $\varepsilon = \sqrt{\frac{F.max - F.min}{10^{P/10}}}$ for SZ's pointwise error-bound compression, where P is the reconstructed PSNR in decibels and F is the original volume. This ε guarantees that the PSNR of the (de)compressed representative volume will be at least P dB, and serves as a starting point for our target error bound that scales with the reconstructed PSNR so that compression ratio scales with the error-bound. Since compressing the data may cause unforeseen error after decompression and SR, we perform a sanity check that performs decompression and multiresolution super resolution with the

NN hierarchy to ensure that error-bound is still met, which empirically has always been the case. When the error-bound is not met, we use a binary search approach to find the optimal ε.

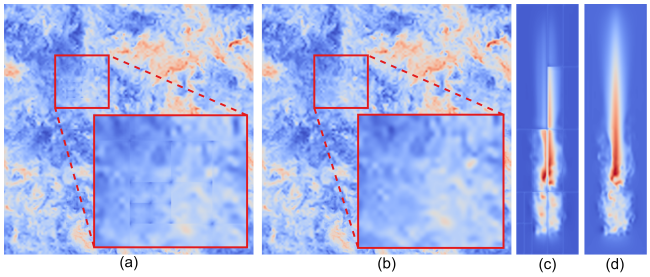

Fig. 4. An example of upscaling multiresolution data with NNs by (a,c) upscaling nodes individually or (b,d) our proposed multiresolution super resolution algorithm on Iso2D data (a,b) and Plume data (c,d).

4.3 Multiresolution super resolution for SR-octrees

Lastly, we present our algorithm for upscaling our SR-octree multiresolution data format while minimizing the impact of seam artifacts. A naive solution to upscale the multiresolution data is to use the pretrained NNs to perform super resolution on each data chunk individually based on the downscaling factor each node is saved with. The upscaled individual chunks would finally be stitched together into a single full resolution volume of data. However, this approach introduces *seam* artifacts, or boundary artifacts, between data chunks, illustrated in Fig. 4. Seams are a problem common to hierarchical rendering techniques [46, 72, 74], but is even more pronounced when using NNs due to zero padding influence outlined in Sect. 4.2.

Therefore, we require an approach that upscales all data together instead of separately to reduce the impact of seam artifacts. Though there is data distributed across many different downscaling levels, the only level that can uniformly represent the data without any separate upscaling is at the largest downscaling level (coarsest level), since smaller downscaling levels can be reduced further to the largest downscaling level. This gives us a uniform LR version of the raw data that we can begin the upscaling process at that will not create seam artifacts when upscaled.

4.3.1 Upscaling process

Our upscaling process begins at the largest downscaling level $MAXDSL = |G|$, where G is our SR hierarchy. Since all data is saved with at most $MAXDSL$, this means that all data nodes can be individually downscaled to the maximum downscaling level and stitched together into a single volume. From here, we upscale this volume to $MAXDSL-1$ which will not introduce seam artifacts since the full volume was upscaled at once. The upscaled volume is now an approximation of the ground truth raw data at $MAXDSL-1$. However, we do not need to use approximated voxels when downscaled data exists in the SR-octree. If a node in the octree has $downscaling\_level \leq MAXDSL-1$, then we opt to replace the approximated voxel values in our volume with the more accurate voxel values from the node representing them. We call these approximated values that should be overwritten *stale voxels*. We can repeat our upscaling $\rightarrow$ stale voxel overwriting until we reach $downscaling\_level = 0$. Since we are only ever upscaling the full volume instead of many small volumes, we are not introducing seam artifacts on node boundaries. It is worth noting that when stale voxels are overwritten, there may be discontinuity between the overwritten voxels and predicted voxels surrounding them, creating a seam. However, in all experiments and visualizations, we do not find any observable seams.

4.3.2 Downscaling process

In our upscaling process (Sect. 4.3.1), we claim that all data can be downscaled uniformly to $MAXDSL = |G|$, and begin upscaling one level at a time starting there. However, downscaling all nodes to $MAXDSL$ and putting them together into a volume is not always possible because very small nodes (limited by *minChunk* in Sect. 4.2) cannot always be downscaled to $MAXDSL$. As an example, a single voxel node cannot be downscaled by a factor of 64.

We resolve this issue with a downscaling process which mirrors our upscaling process. The primary issue is that some small nodes at low downscaling levels cannot be downscaled to $MAXDSL$ in one large downscaling step. Instead, we can take many smaller downscaling steps which mirrors how we take many small upscaling steps. Our SR-octree construction guarantees that single voxel nodes at $MINDSL$, where $MINDSL$ is the minimum downscaling level present in any node of an SR-octree T, must have siblings with the exact same size (a single voxel) and downscaling level. Therefore, these siblings can be joined to make a larger node that is 2^D that can be downscaled by $2\times$.

This motivates the following downscaling process:

1. Set $MINDSL = 0$.
2. Combine all single voxel data nodes that have $downscaling\_level = MINDSL$ to a single node of size 2^D.
3. Downscale all nodes with $downscaling\_level = MINDSL$ (including the combined nodes from step 2) by $2\times$ in each spatial dimension.
4. Set $MINSDL = MINDSL + 1$.
5. If $MINDSL = MAXDSL$, finish. Else, return to the step 2.

After finishing, all data is downscaled to a single volume at $MAXDSL$.

Since we downscale regions of data by a factor of 2 multiple times, we require the following property in the downscaling algorithm used. Given a full resolution volume F, scale factor S, and downscaling algorithm $down(F,S) : \mathbb{R}^{|F|} \rightarrow \mathbb{R}^{|F|/2}$, it must hold that $down(F, 2 \times S) = down(down(F,S),S)$. When this property holds, there is no effect from downscaling a region multiple times compared to downscaling it once at a large downscaling factor. For instance, downscaling a region by a factor of $2\times$ three times would be the same as downscaling it by $8\times$ once. We see this property in all pooling algorithms (min, max, mean pooling) as well as subsampling.

4.3.3 Multiresolution super resolution end-to-end, optimization, and time complexity

To summarize the multiresolution super resolution algorithm, the downscaling process will first combine and downscale nodes, starting from the smallest downscaling level (fine) to the largest downscaling level (coarse), by a factor of $2\times$ at a time. Then, the uniform LR data is upscaled by the NN one downscaling level at a time, with stale voxels overwritten with non-approximated data when available. Since all data is upscaled uniformly step by step, the impact of seam artifacts is minimized, and the NN does not suffer from boundary artifacts at each node's edges because zero padding is minimized.

The time complexity of this SR-octree super resolution algorithm with a caching optimization is $\mathcal{O}(|N| + MAXDSL(\mathcal{O}(DS) + \mathcal{O}(SR)))$, where $|N|$ is the number of nodes, $MAXDSL$ is the maximum downscaling level, $\mathcal{O}(DS)$ is the time complexity for the downscaling algorithm, and $\mathcal{O}(SR)$ is the time complexity for the super resolution algorithm. Our implemented caching optimization gives a memory complexity of $\mathcal{O}(\mathcal{O}(F)(5+\frac{3}{b}))$, where $\mathcal{O}(F)$ is the memory complexity of the data to be compressed and b is the number of bytes used to represent each element (4 for single precision floating point numbers, which we use for all data). Though our SR-octree upscaling algorithm is designed with NNs in mind, it can be used with any upscaling algorithm, such as other interpolation techniques (bilinear, bicubic, trilinear, etc).

5 Evaluation

We evaluate our framework through a series of experiments that compare our approach to baseline approaches across five datasets. Information about each dataset is listed in Sect. 5.1. In Sect. 5.2, we describe hyperparameters and training procedure used for training the NN hierarchies that perform SR on LR data as described in Sect. 4.1 and illustrated with Fig. 2(e). In Sect. 5.3, we evaluate the neural network SR results against bilinear interpolation and bicubic interpolation baselines for 2D datasets, and trilinear interpolation baseline for 3D datasets. Sect. 5.4 compares our method with and without dynamic downscaling. In Sect. 5.5, we evaluate our data reduction/reconstruction pipeline by

comparing PSNR/SSIM metrics over varying compression ratios and renderings for our method against our baseline state of the art lossy compressor SZ [41]. All PSNR/SSIM metrics listed are calculated in the data space as opposed to image space.

5.1 Datasets

We experiment with three scalar field datasets from Johns Hopkins Turbulence Databases (JHTDB) [40] that are time-varying results from a direct numerical simulation (DNS) for turbulent fluid flow and two scalar fields not from JHTDB. Information about each dataset and corresponding trained hierarchies are shown in Table 1. **Iso3D magnitude** is a 3D velocity magnitude dataset from a DNS of turbulent fluid flow at Taylor-scale Reynolds number $R_\lambda \sim 433$, hosted in the "isotropic1024coarse" dataset in the JHTDB. We take the middle z-axis slice (z=512) of the Iso3D magnitude dataset to create an **Iso2D magnitude** dataset for a testing our approach with 2D data. We use the pressure field from the "mixing" dataset in JHTDB (**Mixing3D pressure**). We also experiment with a solar plume dataset (**plume**), which is a velocity magnitude field from a solar plume simulation that looks into the effect the solar plume plays on the heat, momentum, and magnetic field of the sun. Plume is originally $512 \times 126 \times 126$, but we use trilinear interpolation to resample to $512 \times 128 \times 128$ for reasons discussed in Sect. 6. Lastly, **vorts** a 128^3 vorticity magnitude scalar field dataset from a pseudo-spectral simulation of vortex structures.

5.2 Network training and hyperparameters

One NN hierarchy is trained per dataset, with settings for each shown in Table 1. We perform a training/test set split by training on the first k timesteps and testing on the final $N-k$ timesteps, where N is the total number of timesteps for the dataset (# train + # test in Table 1), and k is the number of training timesteps. We train this way to mimic a use case where a network is trained on data already available and then applied on newly generated simulation results.

During training, networks are optimized with the Adam optimizer [34] with default decay rates $\beta_1 = 0.9, \beta_2 = 0.999$ for the first and second moment estimation. We use $\lambda_{rec} = 1.0$ and $\lambda_{adv} = 0.1$ for our loss function defined in Sect. 4.1.3. All networks use 96 kernels in every convolution layer (see Fig. 2) and 3 RRDBs to achieve a good balance of computational efficiency/memory use and reconstruction quality. Following Wang et al. [77], the RRDBs use $\beta = 0.2$ for residual scaling, explained in Sect. 4.1.1. Table 1 shows how many models are in each trained hierarchy. We use Pytorch's `DistributedDataParallel` to train in parallel across 8 NVidia A100 40GB Tensor Core graphics cards on a single NVidia DGX A100 system. The full resolution 3D data cannot fit in the GPU's memory during training, so we crop training volumes to 96^3 and use a random starting position for the crop to increase training example diversity. We augment training with random flipping in any subset of the spatial dimensions to further increase training diversity. Each model is trained for 50 epochs, at which point each had converged. Training time takes between 1–5 hours, but is only necessary once per dataset. Storage cost for each model is listed in Table 1.

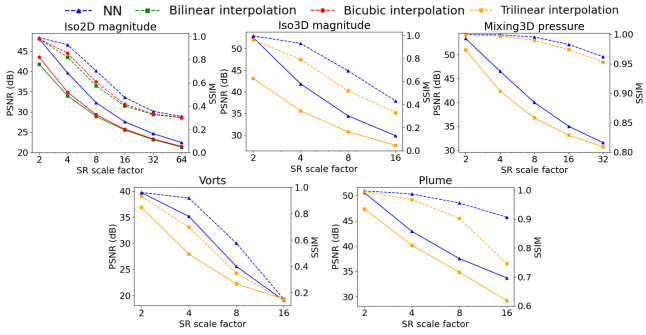

Fig. 5. Comparison between our SR NNs and linear/cubic interpolation baselines across scale factors. Solid lines show PSNR values, dashed lines are SSIM values.

5.3 Evaluating neural network hierarchy SR

We provide PSNR/SSIM charts for each scale factor trained in each network in Fig. 5 plotted against interpolation baselines. Our results show that the NNs are properly trained and outperform the baseline SR methods for each scale factor for both PSNR and SSIM. These results also verify that using multiple trained NNs in tandem for SR factors of $4\times$ or higher does not cause unforeseen reconstruction artifacts which is important because the networks were not trained conditional on the output of previous networks.

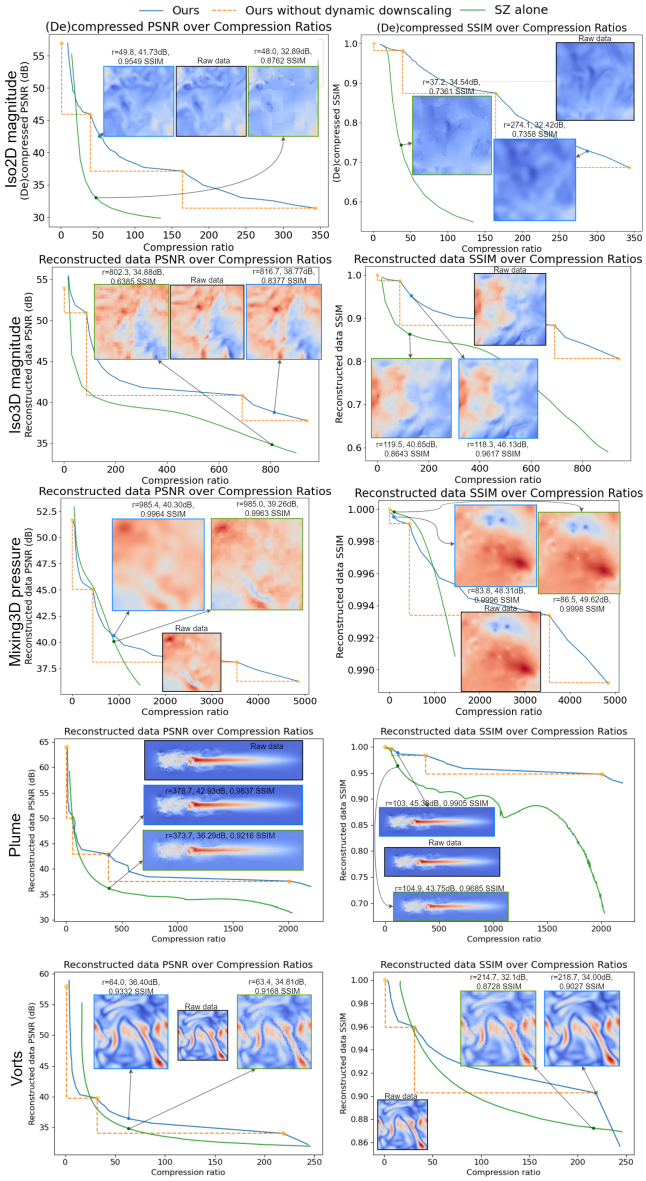

Fig. 6. The data reduction results for SZ alone compared to our method w/ and w/o the SR-octree for dynamic downscaling for each dataset. Downscaling ratio r and metrics are shown near examples.

5.4 Dynamic downscaling and SR-octree evaluation

In Fig. 6, we can see that using our approach (in blue) that dynamically downscales data to the SR-octree allows for more flexibility in terms of target PSNR/SSIM error-bounds than our approach with dynamic downscaling disabled (in orange). To disable dynamic downscaling in our approach, we use the same algorithm presented in Sect. 4.2, but do not split data into its octants when uniform downscaling cannot occur (step 3b). The result of this augmented process is a uniformly downscaled volume instead of a multiresolution volume. Since data cannot be downscaled at arbitrary scale factors, only 2^i for $1 \leq i \leq |G|$,

Dataset name	$\|G\|$	# train	# test	data size	G size	train time	Red. time	Rec. time	*minChunk*
Iso3D magnitude	4	100	10	1024^3	184MB	3.8 hrs.	685 sec.	204 sec.	32
Iso2D magnitude	6	400	100	1024^2	71MB	2.8 hrs.	5.6 sec.	10^{-5} sec.	16
Mixing3D pressure	5	100	10	512^3	230MB	4.6 hrs.	120 sec.	28 sec.	16
Plume	4	20	9	$512 \times 128 \times 128$	123MB	1.5 hrs.	5.16 sec.	4.5 sec.	4
Vorts	4	20	9	128^3	123MB	2.65 hrs.	46.1 sec.	4.2 sec.	4

Table 1. Information about each dataset we evaluate with. $|G|$ is the total number of networks in the trained NN hierarchy G. Data reduction time is shortened to Red. time, and data reconstruction time to Rec. time. *minChunk* is a hyperparamter for dynamic downscaling set in Sect. 4.2. Reconstruction is performed on the same machine as training (listed in Sect. 5.2) but only 1 GPU is used. SZ's compression/decompression time is negligible, using less than 1% of the total time listed above during compression and decompression.

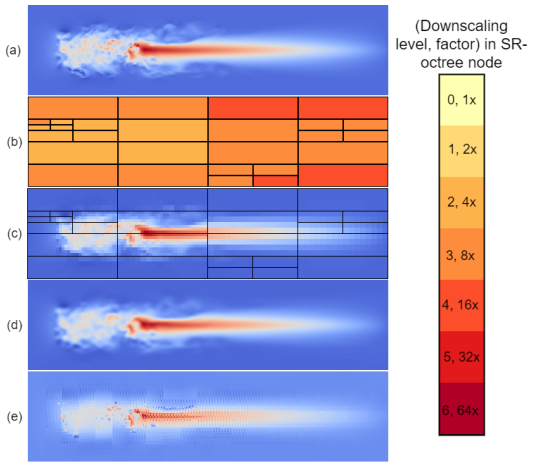

Fig. 7. (a) A single timestep of the Plume dataset used for testing. (b) Downscaling levels in a saved SR-octree (c) The resulting multiresolution data (SR-octree data). (d) The result of using our multiresolution super resolution algorithm with a pretrained neural network hierarchy on the SR-octree data from (c). (e) The (de)compressed result from using SZ alone at the same compression ratio as our method that produced (d).

where G is the NN hierarchy, the compression ratio will increase as i increases, but the only data points will be on those few samples between $i = 1$ and $i = |G|$. Since error-bounds between those points will default to the lower downscaling level to have an acceptable error, our approach without dynamic downscaling and an SR-octree follows a step function. Introducing our SR-octree and dynamic downscaling allows for PSNR/SSIM targets to bridge the gap between downscaling levels more smoothly, and enables higher compression ratios for given error-bounds as a result. In Fig. 7, we show a color mapping for the resulting downscaling levels in the SR-octree our dynamic donwscaling algorithm creates for a timestep of the Plume dataset with a target error-bound of 38 dB PSNR, which has a final compression ratio of 550× (same volume visualized in Fig. 8). Visualizations are shown in 2D (a single slice of the z-axis), but the experiment and metrics are given for the full 3D volume. The empty, or rather more simple regions, are downscaled further while the more complicated areas are allocated more voxels. The upscaled result (d) achieves 40.65 dB PSNR and 0.9707 SSIM, while SZ alone at a similar compression ratio (e) achieves 35.25 dB PSNR and 0.9136 SSIM and has visual artifacts.

5.5 Data reduction and reconstruction evaluation

Qualitative results with volume and isosurface rendering We show renderings of the reconstructed (a) Iso2D magnitude data, (b) Iso3D magnitude data, (c) Mixing3D data, (d) Plume data, and (e) Vorts data in Fig. 8, with each dataset at a fixed compression ratio. We compare the ground truth visualization to (1) the SR-octree, the multiresolution representation of data saved to storage in our data reduction pipeline, (2) our end result from using our multiresolution super resolution algorithm to upscale the SR-octree to the original raw data's resolution, and (3) SZ's decompressed data. In other words, (2) is visualizing the SR-octree just before multiresolution super resolution with the NNs during data reconstruction from Fig. 1. Across the visualizations for each dataset, we see that SZ inserts block–shaped artifacts that impact visualization, whereas our method more accurately captures visual features. Even though our PSNR/SSIM are not as high as SZ's at the compression ratio for (c), the isosurface visualization, we can see that our method is free from artifacts and can show the isosurfaces better. We can also see that the multiresolution super resolution algorithm with our NN hierarchy reduces seams present from the multiresolution data.

6 Limitations

This section comments on the limitations of our approach. We address the computational efficiency of our pipeline, the limitation of application to single scalar fields, the use of 2× scale factor between the hierarchy levels, and our network's weakness at pointwise error bounds.

Data reduction/reconstruction computational efficiency. One limitation of our approach is the time it takes for data reduction and reconstruction. Though we use linear interpolation as a conservative approximation for the quality the NN will produce, which greatly reduces data reduction time, our data reduction process can still take up to 4 minutes depending on volume size, shown in Table 1, and scales with the size of the volume to reduce. To improve data reduction speed, nodes could be processed in parallel individually, but then seams would affect estimated reconstruction quality which would reduce the final compression ratio. During data reconstruction, model inference (i.e. $\mathcal{O}(SR)$) would be quicker if a more lightweight model was used compared to ESRGAN.

Reducing single scalar fields. Since SR networks perform best on input with similar features to what the network learned on, our approach favors an uncompressed curated dataset to train with from the same simulation the user plans to create more data from. For single scalar field data reduction (no time-varying data to train with), our approach would work best if a NN hierarchy could train using other similar data, or a subvolume of the uncompressed data, that has features present throughout the entire volume. We also expect the storage cost of the NN hierarchy (up to 230MB, Table 1) to be amortized across an arbitrary number of scalar fields to reduce, which is not as effective for non-time-varying data. Otherwise, storage overhead from the NN would reduce the effective compression ratio.

Handling of non-power-of-two sized volumes. Our use of a 2× scale factor upscaling within each NN limits data shape, since we can only perform SR with a factor of 2^i, for $1 \leq i \leq |G|$. Therefore, data can only be downscaled (and upscaled) by up to the largest factor of two that a spatial dimension is divisible by, which can limit data reduction. Resampling the data to a size more compatible is a short term solution (as we did for Plume), but for support of arbitrary spatial dimension sizes, another kind of NN SR architecture is needed.

Preservation of strict pointwise error bounds. Our approach struggles to be competitive with SZ alone when the error-bound metric requested by the user is maximum relative pointwise error bounds, but works much better when PSNR/RMSE/SSIM is the target error-bound. NNs learn from loss values that average error across the entire spatial domain which can give a good result on average, but does not punish the network for having high relative pointwise error outliers. With our approach's focus on visualization and large compression ratios we expect this side effect, but other networks or training strategies, such as implicit neural representations, might bound maximum relative pointwise error better than our approach.

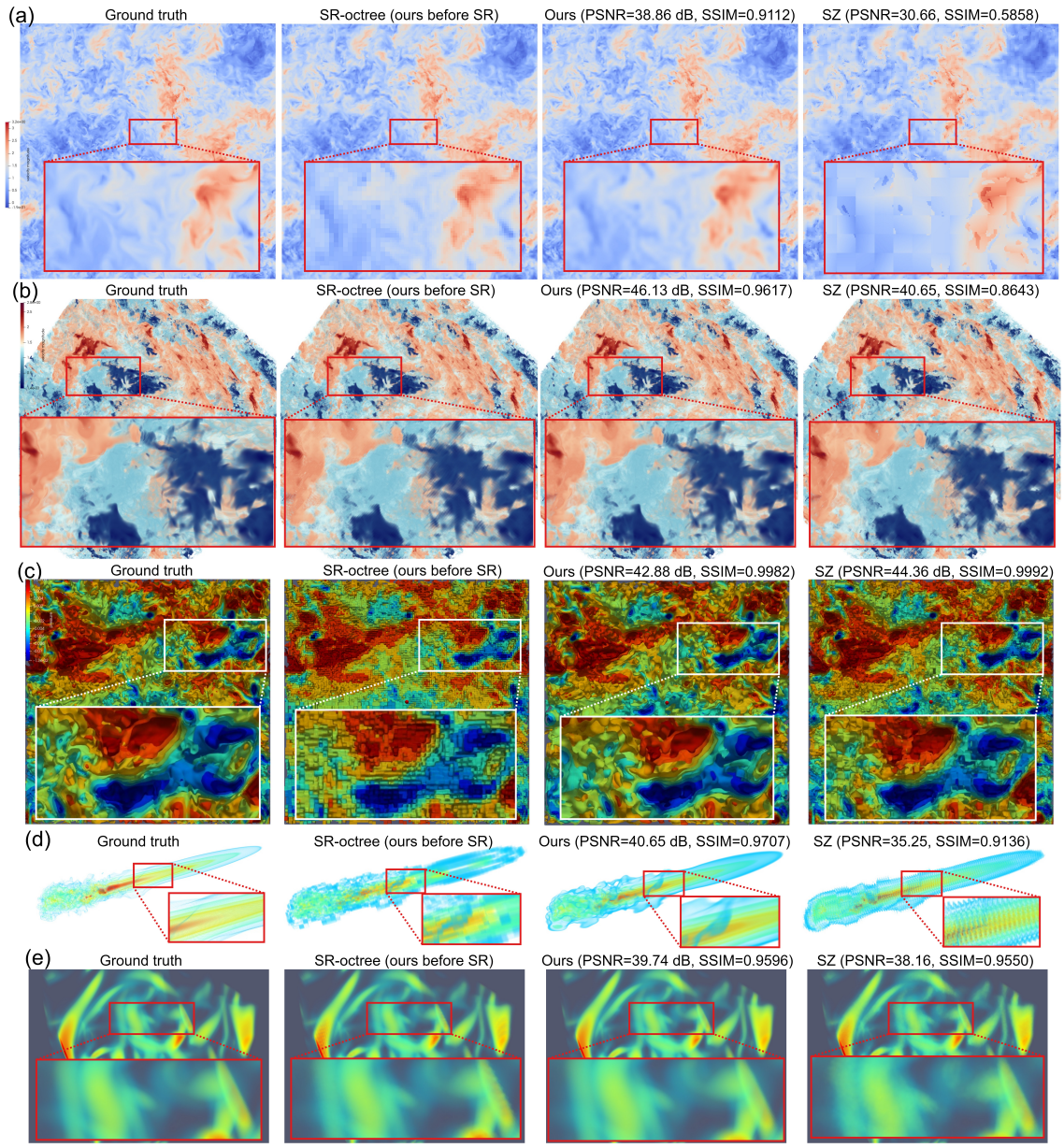

Fig. 8. (A) A colormapped image for the reconstructed Iso2D magnitude data. Our compression ratio: 87.8×, SZ's compression ratio: 88.5×. (B) A volume rendering of the reconstructed Isomag3D data. Our compression ratio: 118.3×, SZ's compression ratio: 119.5×. (C) Isosurface volume rendering for 8 isovalues linearly spaced in the reconstructed Mixing3D data. Both compression ratios: 544×. (d) Volume rendering of the solar plume dataset. Our compression ratio: 555×, SZ's compression ratio: 548×. (e) Volume rendering of Vorts dataset. Our compression ratio: 63×, SZ's compression ratio: 61.7×.

7 Conclusion and Future Work

In this paper, we present a multiresolution super resolution algorithm for upscaling hierarchical data using NNs while minimizing seam artifacts on node boundaries. The multiresolution super resolution algorithm is comprised of a downscaling process followed by an upscaling process to reduce the effect of seam artifacts between nodes. We demonstrate one use of this algorithm within a data reduction and reconstruction pipeline for scientific data. Our quantitative results show that our method can enhance SZ's compression capability with better PSNR/SSIM at similar compression ratios, especially as compression ratios grow. Our visualizations of reconstructed data show that our method contains no seam artifacts between node boundaries while also retaining visual features better than SZ alone. In future work, the compressor and NN architecture can be swapped out for other compatible compressors or architectures. For instance, architectures that are more lightweight will have less storage overhead, and some improved training routine for a model might reduce the amount of training data necessary for accurate super resolution, or improve training speed. We experiment with SZ and and ESRGAN, but any compressor capable of compressing grid–structured or multiresolution data could be used instead, and any NN architecture capable of 2× SR can be used as the generator in the NN hierarchy. Specialized network architectures, such as a network created for vector field SR, may perform better in their specialized domain. Outside of data reduction, the multiresolution super resolution algorithm we propose may be useful in other scientific visualization algorithms, such as realtime multiresolution volume rendering.

REFERENCES

[1] SO/IEC 15444-1. Information technology–JPEG 2000 image coding system, 2000. Standard, International Organization for Standardization.

[2] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data—The univariate case. *Computing and Visualization in Science*, 19(5):65–76, 2018.

[3] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. V. Andel. Nyx: A Massively Parallel AMR Code for Computational Cosmology. *The Astrophysical Journal*, 765(1):39–53, 2013.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein Generative Adversarial Networks. In *Proc. 2017 International Conference on Machine Learning*, pp. 214–223, 2017.

[5] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *Proc. 2017 International Conference on Learning Representations*, 2017.

[6] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. TTHRESH: Tensor Compression for Multidimensional Visual Data. *IEEE Transactions on Visualization and Computer Graphics*, 26(9):2891–2903, 2020.

[7] M. Balsa Rodríguez, E. Gobbetti, J. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. Suter. State-of-the-Art in Compressed GPU-Based Direct Volume Rendering. *Comput. Graph. Forum*, 33(6):77—100, 2014.

[8] M. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

[9] M. J. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

[10] H. Bhatia, D. Hoang, G. Morrison, W. Usher, V. Pascucci, P.-T. Bremer, and P. Lindstrom. AMM: Adaptive Multilinear Meshes. arXiv:2007.15219 [cs.GR], 2020.

[11] M. Burtscher and P. Ratanaworabhan. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Transactions on Computers*, 58(1):18–31, 2009.

[12] S. Cao, C.-Y. Wu, and P. Krähenbühl. Lossless Image Compression through Super-Resolution. arXiv:2004.02872 [eess.IV], 2020.

[13] F. Cappello, S. Di, S. Li, X. Liang, A. M. Gok, D. Tao, C. H. Yoon, X.-C. Wu, Y. Alexeev, and F. T. Chong. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications*, 33(6):1201–1220, 2019.

[14] S. Di and F. Cappello. Fast Error-Bounded Lossy HPC Data Compression with SZ. In *Proc. 2016 IEEE International Parallel and Distributed Processing Symposium*, pp. 730–739, 2016.

[15] B. Friesen, A. Almgren, Z. Lukić, G. Weber, D. Morozov, V. Beckner, and M. Day. In situ and in-transit analysis of cosmological simulations. *Computational Astrophysics and Cosmology*, 3(4):1–18, 2016.

[16] K. Fukami, K. Fukagata, and K. Taira. Super-Resolution Reconstruction of Turbulent Flows With Machine Learning. *Journal of Fluid Mechanics*, 870:106–120, 2019.

[17] K. Fukami, K. Fukagata, and K. Taira. Machine-learning-based spatio-temporal super resolution reconstruction of turbulent flows. *Journal of Fluid Mechanics*, 909:A9, 2021.

[18] E. Gobbetti, J. Iglesias Guitián, and F. Marton. COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. *Computer Graphics Forum*, 31:1315–1324, 06 2012.

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

[20] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved Training of Wasserstein GANs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., *Proc. 2017 Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[21] L. Guo, S. Ye, J. Han, H. Zheng, H. Gao, D. Z. Chen, J. Wang, and C. Wang. SSR-VFD: Spatial Super-Resolution for Vector Field Data Analysis and Visualization. In *Proc. 2020 IEEE Pacific Visualization Symposium*, pp. 71–80, 2020.

[22] J. Han and C. Wang. TSR–TVD: Temporal Super-Resolution for Time-Varying Data Analysis and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):205–215, 2020.

[23] J. Han and C. Wang. SSR-TVD: Spatial Super-Resolution for Time-Varying Data Analysis and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2020, early access.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[25] K. Heitmann, S. Habib, H. Finkel, N. Frontiere, A. Pope, V. Morozov, S. Rangel, E. Kovacs, J. Kwan, N. Li, S. Rizzi, J. Insley, V. Vishwanath, T. Peterka, D. Daniel, P. Fasel, and G. Zagaris. Large-Scale Simulations of Sky Surveys. *Computing in Science Engineering*, 16(5):14–23, 2014.

[26] L. Helminger, A. Djelouah, M. Gross, and C. Schroers. Lossy image compression with normalizing flows. In *Proc. 2021 Neural Compression Workshop at International Conference on Learning Representations*, pp. 1–10, 2021.

[27] D. Hoang, P. Klacansky, H. Bhatia, P. Bremer, P. Lindstrom, and V. Pascucci. A Study of the Trade-off Between Reducing Precision and Reducing Resolution for Data Analysis and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1193–1203, 2019.

[28] D. Hoang, B. Summa, H. Bhatia, P. Lindstrom, P. Klacansky, W. Usher, P. Bremer, and V. Pascucci. Efficient and Flexible Hierarchical Data Layouts for a Unified Encoding of Scalar Field Precision and Resolution. *IEEE Transactions on Visualization and Computer Graphics*, 27(02):603–613, 2020.

[29] E. Hoogeboom, J. Peters, R. van den Berg, and M. Welling. Integer discrete flows and lossless compression. In *Proc. 2019 Advances in Neural Information Processing Systems*, pp. 1–11, 2019.

[30] K. Höhlein, M. Kern, T. Hewson, and R. Westermann. A comparative study of convolutional neural network models for wind field downscaling. *Meteorological Applications*, 27(6):e1961, 2020.

[31] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak. Out-of-core Compression and Decompression of Large n-dimensional Scalar Fields. *Comput. Graph. Forum*, 22(3):343–348, 2003.

[32] J. Jakob, M. Gross, and T. Günther. A Fluid Flow Data Set for Machine Learning and its Application to Neural Flow Map Interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1279–1289, 2021.

[33] C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, K. Kashinath, M. Mustafa, H. Tchelepi, P. Marcus, M. Prabhat, and A. Anandkumar. MeshFreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework. In *Proc. 2020 International Conference for High Performance Computing, Networking, Storage and Analysis*, number 9, pp. 1–15. IEEE Computer Society, 2020.

[34] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. 2015 International Conference on Learning Representations*, 2015.

[35] C. C. Kiris, M. F. Barad, J. A. Housman, E. Sozer, C. Brehm, and S. Moini-Yekta. The LAVA Computational Fluid Dynamics Solver. In *52nd Aerospace Sciences Meeting*. 2014.

[36] A. Knoll. A Survey of Octree Volume Rendering Methods, 2006. Scientific Computing and Imaging Institute, University of Utah.

[37] A. Knoll, I. Wald, S. Parker, and C. Hansen. Interactive Isosurface Ray Tracing of Large Octree Volumes. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pp. 115–124, 2006.

[38] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. Fast and Accurate Image Super-Resolution with Deep Laplacian Pyramid Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(11):2599–2613, 2019.

[39] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs. Data Reduction Techniques for Simulation, Visualization and Data Analysis. *Computer Graphics Forum*, 37(6):422–447, 2018.

[40] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9, 2008.

[41] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In *Proc. 2018 IEEE International Conference on Big Data*, pp. 438–447, 2018.

[42] X. Liang, H. Guo, S. Di, F. Cappello, M. Raj, C. Liu, K. Ono, Z. Chen, and T. Peterka. Toward Feature-Preserving 2D and 3D Vector Field Compression. In *In Proc. 2020 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 81–90, 2020.

[43] X. Liang, B. Whitney, J. Chen, L. Wan, Q. Liu, D. Tao, J. Kress, D. Pugmire, M. Wolf, N. Podhorszki, and S. Klasky. MGARD+: Optimizing Multilevel Methods for Error-bounded Scientific Data Reduction, 2020.

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

[45] P. Lindstrom and M. Isenburg. Fast and Efficient Compression of Floating-Point Data. *IEEE transactions on visualization and computer graphics*, 12:1245–1250, 2006.

[46] P. Ljung, C. Lundström, and A. Ynnerman. Multiresolution interblock interpolation in direct volume rendering. In *Proc. 2006 Eurographics/IEEE VGTC Symposium on Visualization*, pp. 259–266, 2006.

[47] F. Losasso, F. Gibou, and R. Fedkiw. Simulating Water and Smoke with an Octree Data Structure. In *Proc. 2004 ACM SIGGRAPH*, pp. 457—462, 2004.

[48] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. Compressive Neural Representations of Volumetric Scalar Fields. arXiv:2104.04523 [cs.LG], 2021.

[49] M. Adams, P. Colella, D. T. Graves, J.N. Johnson, N.D. Keen, T. J. Ligocki. D. F. Martin. P.W. McCorquodale, D. Modiano. P.O. Schwartz, T.D. Sternberg and B. Van Straalen. Chombo Software Package for AMR Applications – Design Document. Lawrence Berkeley National Laboratory Technical Report LBNL-6616E.

[50] J. N. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein. ACORN: Adaptive Coordinate Networks for Neural Representation. *ACM Trans. Graph. (SIGGRAPH)*, 2021, early access.

[51] D. Meagher. Geometric Modeling Using Octree Encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[52] F. Mentzer, G. D. Toderici, M. Tschannen, and E. Agustsson. High-Fidelity Generative Image Compression. In *Proc. 2020 Advances in Neural Information Processing Systems*, pp. 11913–11924, 2020.

[53] B. Nouanesengsy, J. Woodring, J. Patchett, K. Myers, and J. Ahrens. ADR visualization: A generalized framework for ranking large-scale scientific data using Analysis-Driven Refinement. In *Proc. 2014 IEEE Symposium on Large Data Analysis and Visualization*, pp. 43–50, 2014.

[54] B. W. O'shea, G. Bryan, J. Bordner, M. L. Norman, T. Abel, R. Harkness, and A. Kritsuk. Introducing Enzo, an AMR cosmology application. In T. Plewa, T. Linde, and V. G. Weirs, eds., *Adaptive Mesh Refinement—Theory and Applications*, pp. 341–349. Springer, 2005.

[55] M. I. Patel, S. Suthar, and J. Thakar. Survey on Image Compression using Machine Learning and Deep Learning. In *Proc. 2019 International Conference on Intelligent Computing and Control Systems*, pp. 1103–1105, 2019.

[56] S. Popinet. Gerris: A Tree-Based Adaptive Solver for the Incompressible Euler Equations in Complex Geometries. *Journal of Computational Physics*, 190(2):572–600, 2003.

[57] G. Riegler, A. O. Ulusoy, and A. Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proc. 2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6620–6629, 2017.

[58] R. B. Ross, T. Peterka, H.-W. Shen, Y. Hong, K.-L. Ma, H. Yu, and K. Moreland. Visualization and parallel I/O at extreme scale. *Journal of Physics: Conference Series*, 125(012099):1–10, 2008.

[59] A. Serifi, T. Günther, and N. Ban. Spatio-Temporal Downscaling of Climate Data Using Convolutional and Error-Predicting Neural Networks. *Frontiers in Climate*, 3:26–41, 2021.

[60] C. Sewell, K. Heitmann, H. Finkel, G. Zagaris, S. T. Parete-Koon, P. K. Fasel, A. Pope, N. Frontiere, L.-T. Lo, B. Messer, S. Habib, and J. Ahrens. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In *Proc. 2015: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 50:1–50:11, 2015.

[61] T. R. Shaham, T. Dekel, and T. Michaeli. SinGAN: Learning a Generative Model From a Single Natural Image. *2019 IEEE/CVF International Conference on Computer Vision*, pp. 4569–4579, 2019.

[62] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. In *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1874–1883, 2016.

[63] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. 2021.

[64] D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. In *Proc. 2017 IEEE International Parallel and Distributed Processing Symposium*, pp. 1129–1139, 2017.

[65] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs. In *Proc. 2017 IEEE International Conference on Computer Vision*, pp. 2107–2115, 2017.

[66] L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. In *Proc. 2017 International Conference on Learning Representations*, 2017.

[67] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable Rate Image Compression with Recurrent Neural Networks. In *Proc. 2016 International Conference on Learning Representations*, 2016.

[68] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full Resolution Image Compression with Recurrent Neural Networks. In *Proc. 2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5435–5443, 2017.

[69] M. Treib, K. Burger, F. Reichl, C. Meneveau, A. Szalay, and R. Westermann. Turbulence Visualization at the Terascale on Desktop PCs. *IEEE transactions on visualization and computer graphics*, 18(12):2169–2177, 2012.

[70] R. van den Berg, A. A. Gritsenko, M. Dehghani, C. K. Sønderby, and T. Salimans. IDF++: Analyzing and Improving Integer Discrete Flows for Lossless Compression. In *Proc. 2021 International Conference on Learning Representations*, pp. 1–19, 2021.

[71] F. Velasco and J. Torres. Cell Octrees: A New Data Structure for Volume Modeling and Visualization. In *Proc. 2001 Vision Modeling and Visualization Conference*, pp. 151–158, 2001.

[72] I. Wald, C. Brownlee, W. Usher, and A. Knoll. CPU volume rendering of adaptive mesh refinement data. In *Proc. SIGGRAPH Asia 2017 Symposium on Visualization*, pp. 1–8, 2017.

[73] I. Wald, S. Zellmann, W. Usher, N. Morrical, U. Lang, and V. Pascucci. Ray Tracing Structured AMR Data Using ExaBricks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):625–634, 2021.

[74] F. Wang, I. Wald, Q. Wu, W. Usher, and C. R. Johnson. CPU Isosurface Ray Tracing of Adaptive Mesh Refinement Data. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1142–1151, 2019.

[75] H. Wang, D. Su, C. Liu, L. Jin, X. Sun, and X. Peng. Deformable non-local network for video super-resolution. *IEEE Access*, 7:177734–177744, 2019.

[76] Q. Wang, Z. Chen, Y. Wang, and H. Qu. Applying Machine Learning Advances to Data Visualization: A Survey on ML4VIS. arXiv:2012.00467 [cs.HC], 2020.

[77] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. C. Loy. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. In L. Leal-Taixé and S. Roth, eds., *Computer Vision – ECCV 2018 Workshops*, pp. 63–79. Springer, 2019.

[78] Z. Wang, J. Chen, and S. C. H. Hoi. Deep learning for image super-resolution: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020, to appear.

[79] J. Wilhelms and A. Van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[80] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu. Survey on Artificial Intelligence Approaches for Visualization Data. arXiv:2102.01330 [cs.HC], 2021.

[81] Y. Xie, E. Franz, M. Chu, and N. Thuerey. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow. *ACM Transactions on Graphics*, 37(4):95:1–95:15, 2018.

[82] P. Yeung, X. Zhai, , and K. Sreenivasan. Extreme events in computational turbulence. In *Proc. 2015 National Academy of Sciences*, pp. 12633–12638, 2015.

[83] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, and M. Zingale. AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, 4(37):1370–1374, 2019.

[84] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. Volume upscaling with convolutional neural networks. In *Proc. 2017 Computer Graphics International Conference*, pp. 1–6, 2017.