Reconstructing Unsteady Flow Data from Representative Streamlines via Diffusion and Deep Learning Based Denoising

Pengfei Gu
University of Notre Dame

Jun Han
University of Notre Dame

Danny Z. Chen
University of Notre Dame

Chaoli Wang
University of Notre Dame

***Abstract*—We propose VFR-UFD, a new deep learning framework that performs vector field reconstruction (VFR) for unsteady flow data (UFD). Given integral flow lines (i.e., streamlines), we first generate low-quality UFD via diffusion. VFR-UFD then leverages a convolutional neural network to reconstruct spatiotemporally coherent, high-quality UFD. The core of VFR-UFD lies in recurrent residual blocks that iteratively refine and denoise the input vector fields at different scales, both locally and globally. We take consecutive time steps as input to capture temporal coherence and apply streamline-based optimization to preserve spatial coherence. To show the effectiveness of VFR-UFD, we experiment with several vector field data sets to report quantitative and qualitative results and compare VFR-UFD with two VFR methods and one compression algorithm.**

Index Terms: Flow visualization, vector field reconstruction, unsteady flow data, convolutional neural network, data reduction.

■ **SCIENTISTS** generate unsteady flow data (UFD) from their scientific simulations to better understand different physical and natural phenomena (e.g., hurricanes and tornados). In these cases, they often produce the corresponding integral lines (i.e., streamlines and pathlines) from the Eulerian description of UFD for further exploration. Due to the limited disk storage and I/O bandwidth, we consider a scenario where scientists do not store raw UFD but only these line representations during simulation time for subsequent post hoc analysis. In this paper, we aim to achieve vector field reconstruction (VFR) using a small set of stored flow lines. That is, given these flow lines obtained at simulation time, we recover the corresponding UFD in high quality via a deep learning approach.

The key question of using flow lines to recover UFD is which representation should be chosen: streamlines or pathlines. Pathlines seem to be an obvious choice as we deal with UFD. However, streamlines have also been used for unsteady flow analysis and visualization [1], [2] even though they do not capture the temporal change. We opt to use streamlines in this work due to their simplicity, available representative selection solutions, and easy utilization in training deep neural networks. First, unlike pathlines, streamlines can be *independently* traced from each time step, making it straightforward to handle during simulation time. Moreover, for pathline tracing, particles could go out of bound after a certain number of time steps. Additional particles should be placed to ensure proper domain coverage and maintain a similar number of pathlines at each time step. This is not a severe problem for streamline tracing as we can discard short streamlines and keep tracing new streamlines until the target number of streamlines is met. Second, for efficient data reduction, we want to strike a good balance between the size of flow line data saved and the quality of reconstructed vector fields. Assuming *representative* flow lines are good at capturing flow features and revealing the underlying vector fields, using streamlines can save more storage as we can leverage existing representative streamline selection solutions [3], [4], [5]. To our best knowledge, selecting representative pathlines, however, is still an open problem. Third, streamlines can easily provide *dense* domain coverage at each time step, and we can use every streamline point for VFR. For example, for a UFD of 100 time steps, if we trace 300 streamlines per time step, with an average of 200 points per line, all $60,000$ vectors at each time step can be used for loss computation during neural network training. However, only a fraction of pathline points covers the domain at the *exact* time steps (imagining slicing pathlines along the time dimension). For example, assuming the same UFD, even though we trace $6,000$ pathlines, with an average of $1,000$ points per line (so both streamline and pathline representations store the same number of points), we only have at most $6,000$ vectors at each exact time step for loss computation. The rest of the pathline points that are at non-exact time steps cannot be easily utilized. This is because, given a pathline point p at time step, e.g., $t = 1.3$, we need to consider its spatially neighboring points at $t = 1.3$ in order to "propagate" its vector information to the same point p at the two neighboring time steps ($t = 1$ and $t = 2$) for estimating their corresponding vectors. Even considering that, these estimated vectors are not accurate, which negatively impacts loss computation and network training.

Although using streamlines to reconstruct steady flow data has been investigated [6], [7], using streamlines to recover unsteady flow data still poses three crucial questions. First, *how to take into account temporal coherence*? Unlike pathlines, streamlines only encode spatial information. Directly using available steady flow data reconstruction algorithms will not guarantee temporal coherence. Second, *how to ensure the recovered vector field can preserve both local and global information*? Conventional solutions only apply linear methods in the reconstruction, which is difficult to achieve the desired quality as vector field data often exhibit nonlocal and nonlinear behaviors. Third, *how to recover high-quality vector fields using sparsely sampled streamlines*? Since streamlines only cover a small set of voxels, accurately filling uncovered voxels is the key to achieve high-quality reconstruction.

To answer the above questions, we propose VFR-UFD, a new deep learning framework based on convolutional operations for reconstructing UFD. Initially, we derive low-quality vector fields from representative streamlines via diffusion. The input to VFR-UFD is these low-quality vector fields from consecutive time steps with inaccurate and noised information, and the output is the vector field at the middle time step. We interpolate the velocities from the generated vector field and compare them against the ones in representative streamlines. We minimize the errors between the interpolated velocities from the synthesized vector field and the ones from streamlines through iteratively refining the vector field globally and locally. To demonstrate the effectiveness of VFR-UFD, we show quantitative and qualitative results on several unsteady vector field data sets. We compare VFR-UFD against two VFR methods and a state-of-the-art compression algorithm. We show that VFR-UFD can achieve better quantitative results in terms of peak signal-to-noise ratio, average angle difference, pathline distance, and streamline distance and qualitative results in terms of pathline and streamline rendering.

RELATED WORK

Vector field reconstruction

VFR has been investigated for nearly three decades. Mussa-Ivaldi [8] reconstructed 2D vector fields from the sampled points through a least-square error measurement. Gouesbet and Letellier [9] presented a global VFR algorithm for time-varying systems through a polynomial approximation function. Xu and Prince [6] proposed gradient vector flow (GVF) that reconstructs a vector field from streamlines by minimizing the Laplacian error over the entire vector field. Chen et al. [10] utilized streamlines to estimate a partition of velocities and then linearly interpolated the missing velocities based on triangulation. Xu et al. [3] applied an entropy-based solution to guide streamline placement by estimating every voxel's information content. They used a method similar to GVF for VFR. Han et al. [7] presented a two-stage machine learning solution to reconstruct a steady vector field from a set of representative streamlines. Unlike previous works where they aim to reconstruct steady vector fields from streamlines, our goal is to reconstruct unsteady vector fields based on representative streamlines.

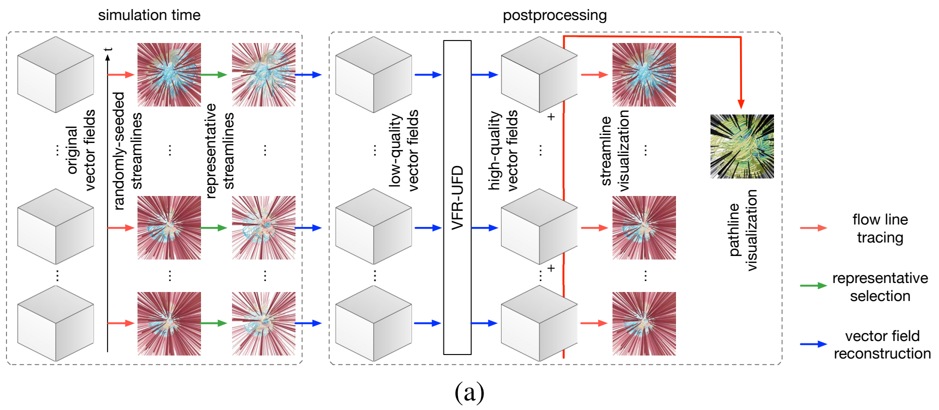
Deep learning for flow visualization

With the success of deep learning in computer vision and natural language processing, researchers have applied deep learning techniques to solve flow visualization problems. Hong et al. [11] utilized long short-term memory to estimate data patterns in different vector regions in particle tracing tasks in order to reduce I/O latency and accelerate the tracing. Xie et al. [12] proposed tempoGAN that generates spatial high-resolution flow sequences based on one generator, one spatial discriminator, and one temporal discriminator. Han et al. [5] designed an autoencoder to group streamlines or stream surfaces through representation learning and dimensionality reduction. They provided an interface for interactive exploration and understanding. Wiewel et al. [13] designed a recurrent neural network to predict 4D functions by learning the temporal evolution of the fluid flow. Kim and Günther [14] leveraged a convolutional neural network (CNN) to extract reference frames from unsteady 2D vector fields. Guo et al. [15] proposed a CNN that spatially upscales vector fields 4 or 8 times along each dimension. Jakob et al. [16] constructed a large 2D fluid flow field and proposed a sub-pixel CNN for effective flow map interpolation. Although we also utilize a CNN, our work is different in that it aims to leverage representative streamlines to recover UFD via denoising rather than upscaling.

VFR-UFD

We denote $\mathbf{F} = \{\mathbf{F}_1, \mathbf{F}_2, \dots, \mathbf{F}_m\}$ as an unsteady vector field where $\mathbf{F}_i$ is the vector field at time step i and m is the number of time steps. $\mathbf{s}_i = \{(\mathbf{p}_1, \mathbf{v}_1), (\mathbf{p}_2, \mathbf{v}_2), \dots, (\mathbf{p}_j, \mathbf{v}_j), \dots\}$ is all streamlines traced from $\mathbf{F}_i$ (i.e., we "concatenate" all streamlines and do not delineate which points correspond to which streamline), where $\mathbf{p}_j$ is the position at the j-th point and $\mathbf{v}_j$ is the velocity at $\mathbf{p}_j$. We also denote $\mathbf{F}^I = \{\mathbf{F}^I_1, \mathbf{F}^I_2, \dots, \mathbf{F}^I_m\}$ as a set of vector fields, which is the input to VFR-UFD.

As sketched in Figure 1 (a), at simulation time, scientists generate UFD and trace streamlines for each time step. They only save representative streamlines for each time step to reduce data storage. After that, we design VFR-UFD to recover UFD from the saved representative streamlines, and streamlines and pathlines are traced for analysis and visualization during post-processing. In particular, we use three variants of Tao et al. [4] (i.e., $p(s)$, $I(s; V)$, and REP) to select representative streamlines. Then, we reconstruct the corresponding vector fields based on a two-stage approach given the representative streamline set. At the first stage, we convert the streamlines into an inaccurate vector field with noises via diffusion using GVF [6]. At the second stage, we leverage VFR-UFD as a denoising model to refine the noisy vector field through



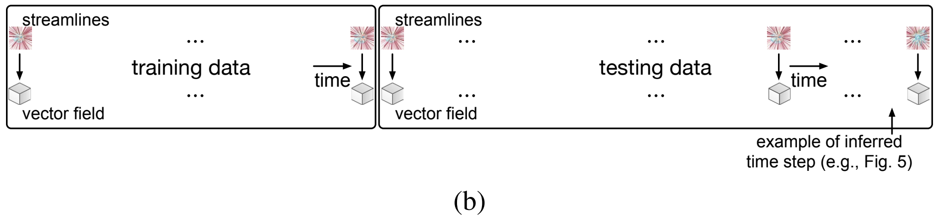


Figure 1. (a) Overview of VFR-UFD. During simulation time, we only save representative streamlines for storage saving. During postprocessing, VFR-UFD can reconstruct UFD from these integral lines, and streamlines and pathlines can be generated from the recovered UFD. (b) Training and inferring time steps from UFD.

interpolating velocities and calculating the difference between the interpolated ones and those obtained from the representative streamlines.

At the first diffusion stage, following GVF [6], we estimate the velocities at the voxels where streamlines pass by calculating the weighted average of velocities from the neighboring streamlines. Then, these velocities are diffused to these voxels where no streamlines pass by minimizing the Laplacian over the whole vector field. In this way, we convert a non-grid data (i.e., streamlines) into a grid data (i.e., vector field) with inaccurate velocities. The converted grid data are the input to VFR-UFD.

At the second denoising stage, we approximate a denoising function $\mathcal{D}$ that aims to achieve two goals: (1) the vector fields generated at the first stage are refined and denoised, (2) temporal coherence is taken into consideration. To achieve the first goal, we utilize several techniques, such as *recurrent residual block* (RRB) and *batch normalization* (BN). For the second goal, we apply sequential input and skip connection. VFR-UFD accepts three consecutive vector fields, from time step $t-1$ to time step $t+1$, as input, and outputs the vector field at time step t. (Note that it is straightforward to consider more time steps, e.g., five or seven, but it would require significantly more training time, more memory cost, and lead to much larger model size.) After that, we evaluate velocities in the synthesized vector field by comparing their differences with those from the representative streamlines. Following the suggestion of Guo et al. [15], VFR-UFD decomposes each vector field into three components (i.e., u, v, and w components) and processes them independently.

Figure 1 (b) shows that we use the early time steps of vector fields and representative streamlines for training. For inference, we use representative streamlines from later time steps as input, and the network infers the corresponding vector fields as output.

Network design and loss function

Design consideration. The key design of each neural network for the u, v, w components includes: (1) applying BN after each convolution (Conv) layer to accelerate training and avoid gradient vanishing, (2) using skip connection to

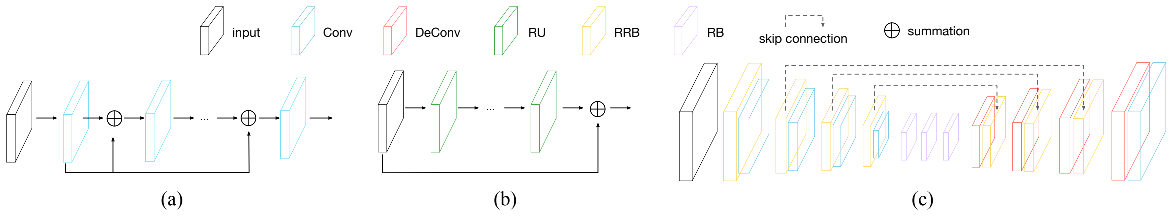


Figure 2. Network architecture of (a) recurrent unit (RU), (b) recurrent residual block (RRB), and (c) VFR-UFD.

merge feature maps with the same scale from different Conv layers, and (3) leveraging RRB to refine and denoise features at different levels. BN is a popular technique used to accelerate network training as it tackles the internal covariate shift problem by normalizing the input layers. In particular, each Conv layer is followed by BN and *rectified linear unit* (ReLU) layers to speed up training and improve model performance. Skip connection is a standard approach to fuse the features, which provides an alternative path for the gradients with backpropagation. In addition, it also helps smoothly minimize the loss and prevent the network from trapped in chaotic states. RRB is a recurrent block that reuses a set of Conv layers to refine and denoise features at different scales.

RU and RRB. The core of RRB is a *recurrent unit* (RU). As illustrated in Figure 2 (a), RU includes two Conv layers. The first Conv filters the input features, and the second one is applied r times for further refinement and denoising. In the k-th ($k \leq r$) refinement, the input is the output from the $k-1$-th refinement and the first Conv layer. These two outputs are bridged by summation. By recurrently applying Conv layers and summation, RU can effectively filter the features and reduce the learned parameters. Each RRB consists of r RUs. As shown in Figure 2 (b), in an RRB, the input is convoluted with r RUs without changing the dimensions, and an identity skip connection is applied to the input. Two outputs are merged by summation to produce the final output. The design of RU and RRB helps develop a deeper model efficiently and ensures better discriminative representations. Due to the accumulation feature denoising, RRB can refine and denoise the features effectively and enhance the quality of the synthesized vector fields.

Network architecture. The architecture of VFR-UFD is shown in Figure 2 (c). VFR-UFD has three individual nets, and each net consists of a *contracting/encoding* path, a *refining* path, and a successive *expanding/decoding* path. In the contracting/encoding path, there are four RRBs followed by the Conv layer whose stride is set to two to reduce the input dimension by half to refine and denoise features at different scales. In the refining path, three *residual blocks* (RBs) are applied to refine the features received from the contracting/encoding path. In the expanding/decoding path, three deconvolutional (DeConv) layers are followed by RRBs. The DeConv layers are used to upscale the input, and RRBs are employed for better feature representation and denoising. Moreover, skip connection is utilized to fuse the features from contracting and expanding paths with the same scale. Finally, a DeConv layer is applied to upscale the feature to the original dimension, and a Conv layer is employed to refine the final output. We do not follow the last Conv layer by BN and ReLU because a vector field's data range could be unbounded. The parameter details of VFR-UFD are listed in Table 1.

Table 1. VFR-UFD architecture parameter details.

type	kernel size	stride	output channels
input	N/A	N/A	3
RRB	3	1	16
Conv+BN+ReLU	4	2	16
RRB	3	1	32
Conv+BN+ReLU	4	2	32
RRB	3	1	64
Conv+BN+ReLU	4	2	64
RRB	3	1	128
Conv+BN+ReLU	4	2	128
3 × RB	3	1	128
DeConv+BN+ReLU	4	2	64
RRB	3	1	64
DeConv+BN+ReLU	4	2	32
RRB	3	1	32
DeConv+BN+ReLU	4	2	16
RRB	3	1	16
DeConv+BN+ReLU	4	2	8
Conv	3	1	1

Loss function. Given the streamlines, we use the mean squared error (MSE) to evaluate the quality of the reconstructed vector field. Specifically, we first calculate the MSE loss between the original velocity $\mathbf{v}_\mathbf{p}$ from streamlines and predicted velocity $\mathbf{v}'_\mathbf{p}$ interpolated from the reconstructed vector field at sample point $\mathbf{p}$, and then sum the MSE losses calculated for all the sample points. The overall training MSE loss is defined as

$$\mathcal{L} = \sum_{j=1}^{n} ||\mathbf{v}_{\mathbf{p}_j} - \mathbf{v}'_{\mathbf{p}_j}||_2, \quad (1)$$

where $||\cdot||_2$ denotes the L^2 norm, n is the number of points, and $\mathbf{v}'_{\mathbf{p}_j}$ is the interpolated velocity at $\mathbf{p}_j$ based on a trilinear function that interpolates the velocities at the eight neighboring voxels of $\mathbf{p}_j$.

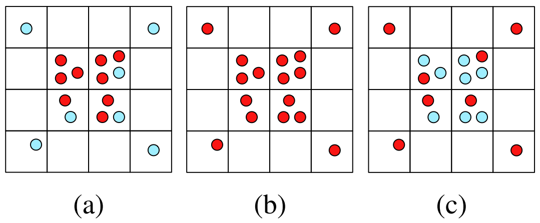


Figure 3. An illustration of different sampling schemes: (a) random sampling, (b) full sampling, and (c) coverage-driven sampling. Red points are sampled during training, while cyan points are not.

Optimization

Sampling. Intuitively, we can either use random sampling (i.e., randomly sampling a subset of all sample points to compute the MSE loss at every epoch) or full sampling (i.e., using all sample points to evaluate the MSE loss) to train VFR-UFD. However, both sampling schemes have their limitations. As sketched in Figure 3, random sampling may fail to sample voxels where only a few streamlines pass. This could lead to poor performance at these voxels since VFR-UFD has almost no supervision there. As for full sampling, it can guarantee to achieve good performance at each voxel. However, it requires a high computational cost as the number of sample points is large. To respond, we propose a *coverage-driven* sampling algorithm that can significantly reduce the training time while achieving high-quality performance. The algorithm is shown in Algorithm 1. We first initialize an empty list and shuffle the points in the training point set. This shuffle operation can guarantee that at every epoch, different points can be sampled. Then, we loop each point in the point set. If the voxels covered by the point are not observed in the list, we sample the current point as our training sample point.

Algorithm 1 Coverage-driven sampling algorithm.

```
Require: A set of points $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$
  Initialize an empty list $L$
  Shuffle points in $\mathbf{P}$
  for each point $\mathbf{p}$ in $\mathbf{P}$ do
    Compute the eight neighboring voxels $\mathbf{v}$ at $\mathbf{p}$
    if $\mathbf{v}$ is not covered by $\mathbf{p}'$, $\forall \mathbf{p}' \in L$ then
      Add $\mathbf{p}$ into $L$
    end if
  end for
  Return $L$
```

Optimization. The optimization of VFR-UFD is as follows. It accepts three consecutive vector fields $\mathbf{F}^I_{t-1}$, $\mathbf{F}^I_t$, $\mathbf{F}^I_{t+1}$ as input, and initializes network parameters θ_u, θ_v, and θ_w. VFR-UFD is iteratively updated using stochastic gradient descent until a certain number of epochs is reached. During each epoch, the coverage-driven sampling algorithm is applied to produce training sample points. VFR-UFD goes through the sample points and produces the vector field at time step t. The gradients $\nabla_{\theta_u}\mathcal{L}$, $\nabla_{\theta_v}\mathcal{L}$, and $\nabla_{\theta_w}\mathcal{L}$ is computed according to Equation 1, and the parameters θ_u, θ_v, and θ_w are automatically updated through the optimizer using the predefined learning rate and computed gradients. During inference, we run VFR-UFD in the same way as training except that the gradients may not be computed.

RESULTS AND DISCUSSION

In this section, we present VFR-UFD results, including data sets and network training, baseline methods for comparison, evaluation metrics, and quantitative and qualitative analysis. We also compare the results generated from different representative streamline selection criteria and discuss our work's limitations. In the Appendix, we study several parameter settings, including architecture design, number of representative streamlines, feature fusion schemes, temporal coherence, and sampling schemes, to evaluate VFR-UFD.

Table 2. The dimensions and training epochs of each data set.

data set	dimension ($x \times y \times z \times t$)	data size (GB)	# rep. lines	line size (GB)	reduction rate	comp. rate	training epochs
supercurrent	$256 \times 128 \times 32 \times 100$	1.172	300	0.136	8.62	11.90	50
supernova	$128 \times 128 \times 128 \times 100$	2.344	500	0.079	29.67	42.43	100
tornado	$128 \times 128 \times 128 \times 48$	1.125	500	0.040	28.13	51.12	100

Table 3. Average PSNR (dB), AAD, PD, and SD values, training time per epoch (in minutes), and inference time (in seconds) with different methods. The SD values are for the time steps reported in Figure 5. The best ones are highlighted in bold (same for the rest of tables in the paper).

data set	method	PSNR	AAD	PD	SD	train	infer
supercurrent	GVF	27.07	0.08	3.57	4.65	—	—
	Han et al.	22.08	0.16	4.98	11.84	101.04	0.12
	LC	31.52	0.04	2.78	3.34	—	—
	VFR-UFD	**36.15**	**0.02**	**1.22**	**2.99**	108.67	0.49
supernova	GVF	21.70	0.11	1.25	3.06	—	—
	Han et al.	19.97	0.18	2.14	6.20	60.53	0.18
	LC	25.88	0.18	5.34	9.57	—	—
	VFR-UFD	**31.40**	**0.03**	**0.99**	**2.10**	69.38	0.88
tornado	GVF	29.11	0.08	0.54	2.58	—	—
	Han et al.	20.82	0.31	7.19	9.99	33.63	0.11
	LC	24.91	0.15	3.01	2.16	—	—
	VFR-UFD	**34.03**	**0.05**	**0.53**	**0.65**	40.03	0.87

Data sets and network training

We show the simulation data sets experimented in Table 2. Representative streamlines are chosen from randomly-traced streamlines using the three variants of Tao et al. [4]: $p(s)$, $I(s;V)$, and REP. The number reported for representative streamlines is for each step, while the data and streamline sizes reported are for all time steps. The reduction rate is the ratio between the data and streamline sizes. We implemented VFR-UFD with PyTorch and used a single Nvidia Tesla V100 graphics card with 32 GB GPU memory for training and inference. For optimization, the network parameters were initialized using normal initialization, and the Adam optimizer was used to update the parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$). We set one training sample per mini-batch and the learning rate to 10^{-4}. The learning rate is decayed by $10\times$ after every 40 iterations. All these parameters were empirically determined based on experiments.

Results

Baselines. We compare VFR-UFD against the following three baseline methods (GVF and Han et al. reconstruct the vector field of each time step based on the representative streamlines given without considering temporal coherence, while LC directly compresses vector fields without using representative streamlines):

- GVF [6] is a standard and widely used method to reconstruct vector fields from a set of streamlines.
- Han et al. [7] is a two-stage deep learning method for reconstructing a steady vector field. At the first stage, the low-resolution vector field is generated from representative streamlines. The low-resolution vector field is then fed to a CNN to generate the high-resolution (i.e., original resolution) one.
- Lossy compression (LC) [17] is a common data compress approach, which can adaptively select either the improved Lorenzo predictor or the predictor based on optimized linear regression in different regions of the data set for compression.

In this paper, the selected representative streamlines are compressed for data reduction and later used to reconstruct VFR. Therefore, we also consider a compression algorithm (i.e., LC) as a baseline method for comparison. For a fair comparison, GVF, Han et al., and VFR-UFD all use the same set of representative streamlines for VFR. The compression rate for LC is set the same as that for VFR-UFD as reported in Table 2. For VFR-UFD, we also use a lossless floating-point compression solution [18] to compress the representative streamlines to further reduce the data storage. The compression rate reported in Table 2 is the ratio between the size

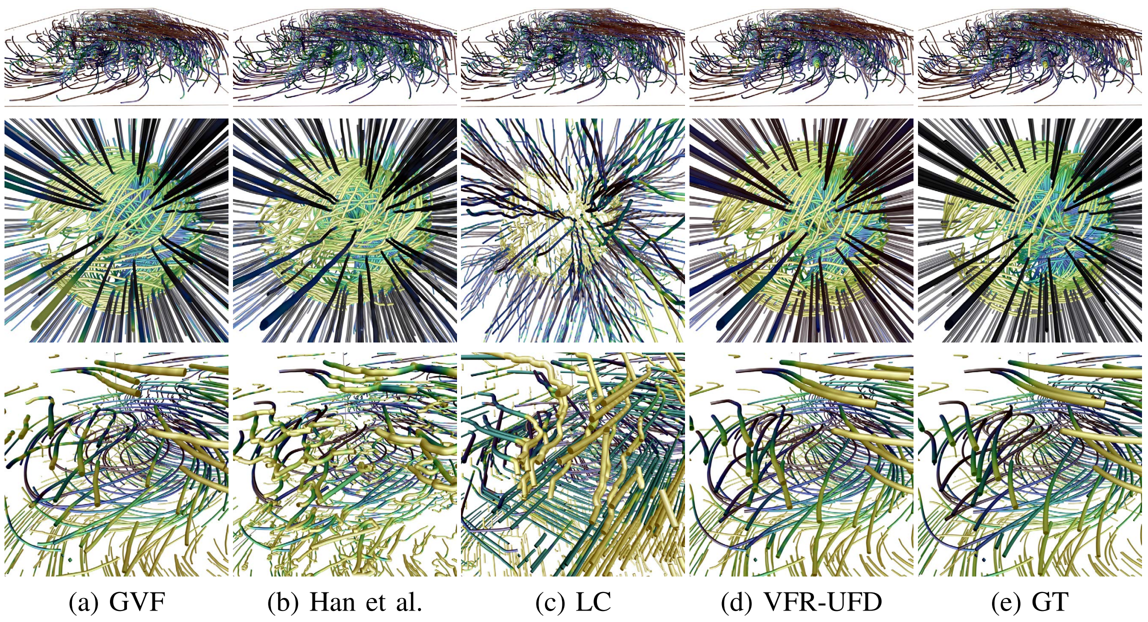


Figure 4. Pathline rendering results which are the inferred results (i.e., the networks do not see these vector fields during training, same for the rest of figures in the paper). Top to bottom: supercurrent, supernova, and tornado. The numbers of pathlines traced are $1,000$, 500, and $2,000$, respectively.

of compressed streamlines and the size of original vector fields. For Han et al. and VFR-UFD, the same training settings are used. All pathline and streamline visualization results by VFR-UFD are generated from synthesized vector fields, which are the inferred results (i.e., these vector fields are not used for training the model). All rendering results for the same data set use the same setting, including the set of randomly-placed seeds and the viewing parameters. We compare pathline and streamline visualization results generated by VFR-UFD and three baselines (i.e., GVF, Han et al., and LC) in reference to the GT.

Evaluation metrics. Following Guo et al. [15], we leverage peak signal-to-noise ratio (PSNR) and average angle difference (AAD) [15] to evaluate the quality of the synthesized vector fields. Besides, we use pathline distance (PD), which is defined as

$$PD(\mathbf{P}, \hat{\mathbf{P}}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_i}\sum_{j=1}^{n_i}\min_{\mathbf{p}\in\mathbf{P}_i}||\mathbf{p}-\hat{\mathbf{p}}_{ij}||_2, \tag{2}$$

where $\mathbf{P}$ and $\hat{\mathbf{P}}$ are the pathlines traced from GT and synthesized vector fields, N is the number of pathlines, n_i is the number of points at the i-th pathline, $\hat{\mathbf{p}}_{ij}$ is the j-th point at the i-th pathline traced from the synthesized vector field, and $\mathbf{P}_i$ is the i-th pathline traced from the GT vector field. Finally, we use streamline distance (SD), which for each time step, follows the same way as we compute pathline distance.

Quantitative and qualitative analysis. In Table 3, we report the average PSNR, AAD, PD values over the entire vector field sequence, and SD values on certain time steps, using GVF, Han et al., LC, and VFR-UFD methods. We can see that VFR-UFD outperforms GVF, Han et al., and LC for all three data sets across all the metrics (i.e., with the highest PSNR, lowest AAD, lowest PD, and lowest SD). In addition, we report the average training time per epoch (in minutes) and average inference time (in seconds) using the three data sets. In terms of training time, we see that Han et al. takes the shortest time compared to VFR-UFD. This is because the CNN used in Han et al. is a shallow network. However, we observe that VFR-UFD only increases a few minutes (i.e., 7.63, 8.85, and 6.40 minutes for the supercurrent, supernova, and tornado data sets, respectively) in training compared to Han et al. In terms of inference time, there is not a significant difference between Han et al. and VFR-UFD. Overall, the

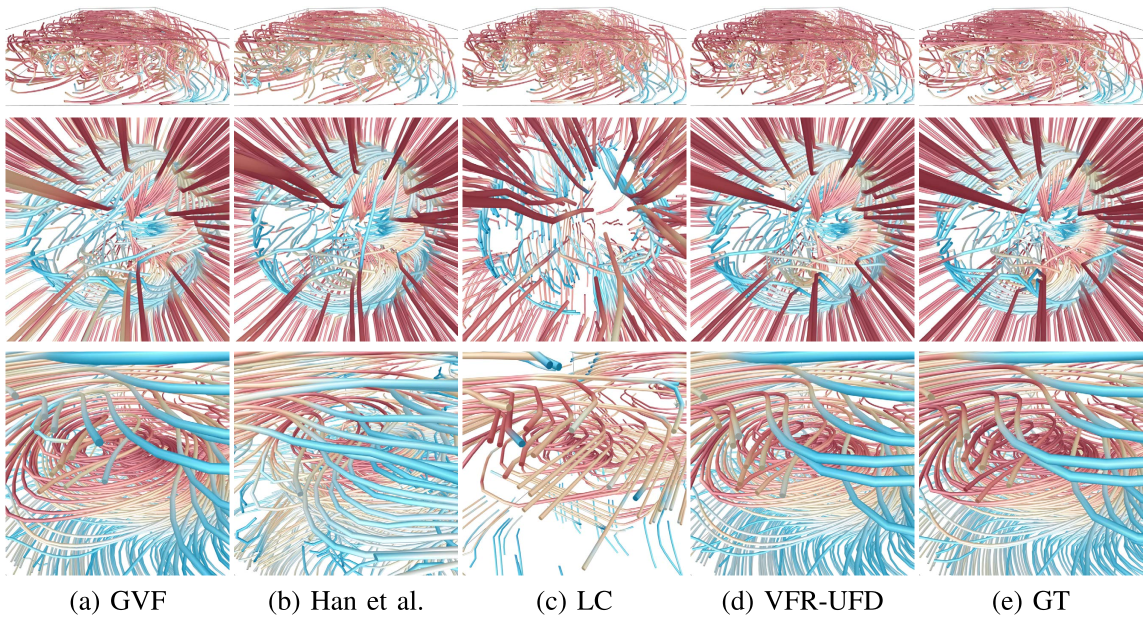


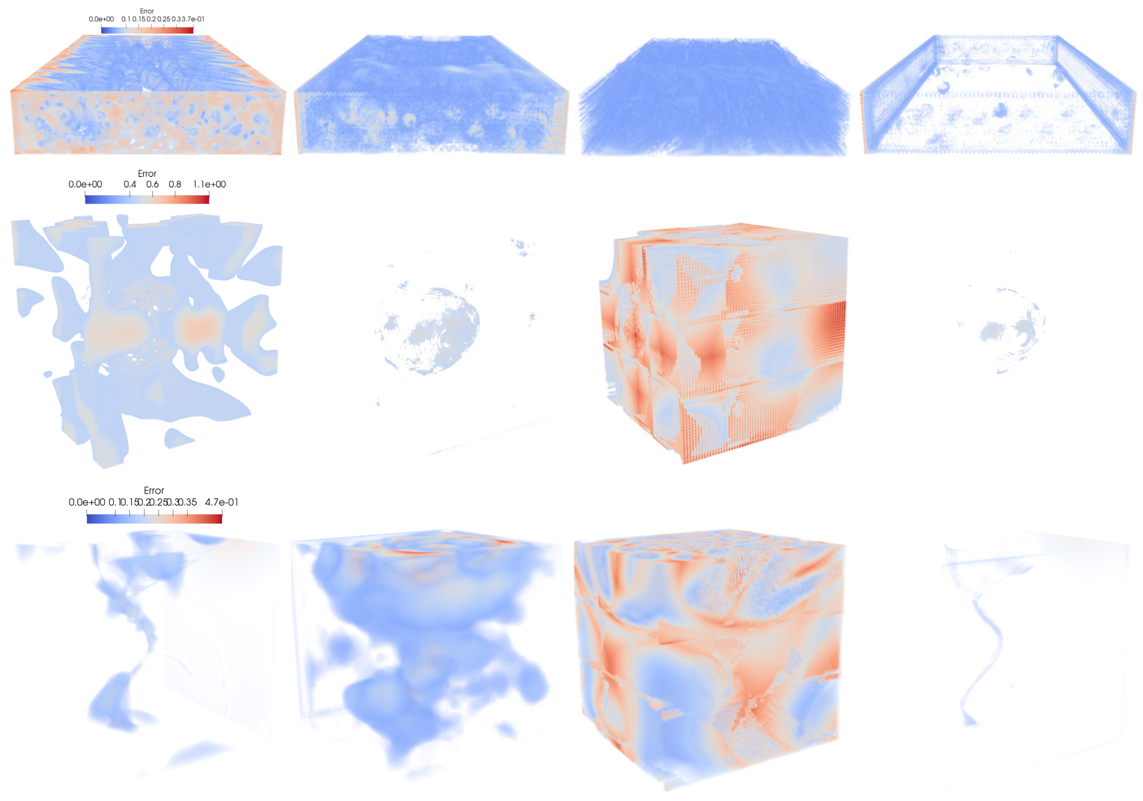
Figure 5. Streamline rendering results which are the inferred results. Top to bottom: supercurrent, supernova, and tornado. 500 streamlines are traced for each data set at time steps 45, 36, and 33, respectively.

quantitative results show that VFR-UFD is an effective and efficient method, as it achieves a good trade-off between performance gain and the increased training time.

In Figure 4, we present pathline rendering results of the synthesized vector fields generated by GVF, Han et al., LC, and VFR-UFD, with GT results provided for reference. Although subtle, GVF and Han et al. do not capture well the swirling patterns for the supercurrent data set. There is also a clear shift of velocity magnitude for Han et al. as judged by the pathline colors. LC does not generate finer details of the swirling patterns compared with GT, while VFR-UFD can capture these patterns well. For the supernova data set, compared with GVF and Han et al., VFR-UFD captures more details around the supernova's center, while LC leads to the worst visual result. For the tornado data set, VFR-UFD produces smoother pathlines than GVF and Han et al., confirming that VFR-UFD can capture finer details. Again, LC leads to the worst result.

In Figure 5, we present streamline rendering results of the synthesized vector fields generated by GVF, Han et al., LC, and VFR-UFD, with GT results provided for reference. For the supercurrent data set, LC and VFR-UFD produce better visual quality than GVF and Han et al. However, LC produces more errors than VFR-UFD at the bottom-left corner, and VFR-UFD captures the swirling patterns better than LC. For the supernova data set, VFR-UFD produces the best visual quality, while GVF leads to multiple errors at the bottom-left region, Han et al. yields errors at the top-left region, and LC clearly generates the worst overall result. For the tornado data set, VFR-UFD captures more details than GVF, Han et al., and LC. For example, GVF does not capture the details at the tornado's center, Han et al. yields low visual quality at the central and bottom-right regions, and LC leads to the worst result.

In Figure 6, we compare volume rendering results of errors introduced by the reconstructed vector fields generated by GVF, Han et al., LC, and VFR-UFD. These volumetric errors are computed following Han et al. [7]. For the supercurrent data set, we can see that GVF, Han et al., and LC generate more errors than VFR-UFD. For the supernova data set, it is obvious that VFR-UFD produces the fewest errors at the boundary and the center of the supernova, followed by Han et al., GVF, and LC. For the tornado data set, it is clear that VFR-UFD produces the fewest errors at the boundary and the core of the tornado, followed by GVF, Han et al., and LC.


Error
0.0e+00 0.1 0.15 0.2 0.25 0.3 3.7e-01
Error
0.0e+00 0.4 0.6 0.8 1.1e+00
Error
0.0e+00 0.1 0.15 0.2 0.25 0.3 0.35 4.7e-01

(a) GVF (b) Han et al. (c) LC (d) VFR-UFD

Figure 6. Comparison of volume rendering results of errors introduced by the reconstructed vector fields. Top to bottom: supercurrent, supernova, and tornado. The error volumes shown are for time steps 45, 36, and 33, respectively.

Why are baselines bad? As we observe from Figures 4, 5, and 6, VFR-UFD always achieves the best quality in terms of pathline and streamline renderings compared with GVF, Han et al., and LC. The possible explanations are as follows. GVF only applies linear interpolation to recover a vector field through aggregating the velocities from streamlines, which does not capture nonlinear flow patterns (e.g., the core of the supernova and the swirls of the supercurrent). Han et al. is a nonlinear approach for recovering a vector field from streamlines. However, due to the random initialization of the low-resolution vector field, the employed CNN is sensitive to the perturbation and cannot generate high-quality vector fields. As for LC, it cannot guarantee the quality of rendering results since a small change in a vector field could lead to poor quality of pathline and streamline renderings.

Discussion

Representative streamline selection criteria. To select representative streamlines for network training, we use three variants of Tao et al. [4]: $p(s)$, $I(s;V)$, and REP. To investigate the differences of these selection criteria, we experiment on the supercurrent data set. Figure 7 shows streamline and pathline rendering results under these three different streamline selection criteria. From the highlighted regions, we can see that, with VFR-UFD, using REP to select representative streamlines yields both streamline and pathline rendering results close to GT compared with $p(s)$ and $I(s;V)$. This is because REP focuses on domain coverage, and the selected streamlines well cover the entire domain, which helps VFR-UFD validate voxels throughout the volume during training. Table 4 reports the average PSNR and AAD values. We can see that when using $p(s)$, VFR-UFD produces the highest

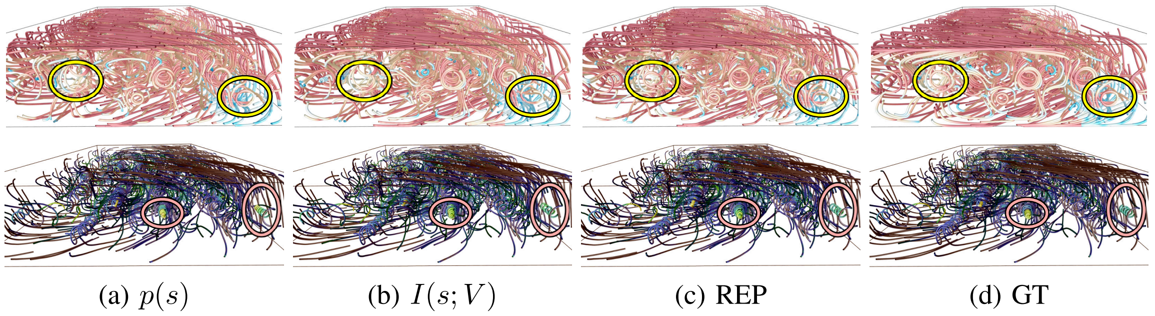


Figure 7. Comparison of streamline (top row) and pathline (bottom row) rendering results under different streamline selection criteria for generating the streamline set for training. $1,000$ streamlines at time step 56 and $1,000$ pathlines are traced for the supercurrent data set.

PSNR and lowest AAD values. However, the value differences between using $p(s)$ and REP are fairly small (i.e., 0.34 for PSNR and 0.002 for AAD). If we only consider feature regions with high entropy values, the PSNR gap between $p(s)$ and REP increases to 0.66, while both achieve the same AAD value of 0.012. Considering both quantitative and qualitative results, we recommend using REP to select representative streamlines for all data sets.

Table 4. Average PSNR (dB) and AAD values and only the feature regions (FR) for the entire vector field sequence using different streamline selection criteria for generating the streamline set for training.

data set	criterion	PSNR	AAD	PSNR-FR	AAD-FR
	$p(s)$	**36.49**	**0.018**	**42.02**	**0.012**
supercurrent	$I(s;V)$	32.76	0.038	38.94	0.015
	REP	36.15	0.020	41.36	0.012

Limitations. Our current work has the following limitations. First, although simple, assuming tracing streamlines independently from each time step does not incorporate temporal coherence in the first place. Considering representative streamline selection at simulation time requires additional integration work. Using a set of well-selected pathlines of the same size could lead to better UFD reconstruction quality, while representative pathline selection remains an open question. If the simulation uses the smoothed particle hydrodynamics (SPH) method, it is more convenient to directly consider the Lagrangian description of the flow field for VFR. Second, the network needs to be retrained for different data sets. A neural network model trained on various types of images could effectively infer unseen images from multiple categories. This is not the case for scientific data since the training data is limited (i.e., not diverse enough), and different scientific data sets may not follow a single distribution. For example, Guo et al. [15] showed that the results generated from joint training using two data sets are worse than those generated from separate training using a single data set. Still, it is possible to train VFR-UFD on a certain type of data sets and later apply it to reconstruct a different data set of the same type. For ensemble data sets, we can train the network on an ensemble data set and use it to directly infer another ensemble data set without retraining.

CONCLUSIONS AND FUTURE WORK

We have presented VFR-UFD, a new deep learning-based VFR solution for UFD. Our solution takes representative streamlines from each time step as input to generate low-quality vector fields, which are refined to yield high-quality vector fields through the designed neural nets. Compared to GVF and Han et al., VFR-UFD generates synthesized vector fields of higher quality, both qualitatively and quantitatively. We also compared VFR-UFD against a state-of-the-art lossy compression scheme. In the future, we will consider using physics-informed CNN that utilizes the physical properties of UFD for guiding neural network training. We will also tackle the more challenging scenario of VFR-UFD using pathlines instead of streamlines. Furthermore, we will generalize our work by exploring pretraining methods to save retraining time for different data sets.

ACKNOWLEDGMENT

This research was supported in part by the U.S. National Science Foundation through grants IIS-1455886, CCF-1617735, CNS-1629914, DUE-1833129, and IIS-1955395, and the NVIDIA GPU Grant Program. The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

1. B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. *Computer Graphics Forum*, 19(3):31–39, 2000.
2. T. Wischgoll, G. Scheuermann, and H. Hagen. Tracking closed streamlines in time dependent planar flows. In *Proceedings of International Symposium on Vision, Modeling and Visualization*, pages 447–454, 2001.
3. L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216–1224, 2010.
4. J. Tao, J. Ma, C. Wang, and C.-K. Shene. A unified approach to streamline selection and viewpoint selection for 3D flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2013.
5. J. Han, J. Tao, and C. Wang. FlowNet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 26(4):1732–1744, 2020.
6. C. Xu and J. L. Prince. Gradient vector flow: A new external force for snakes. In *Proceedings of IEEE International Conference on Computer Vision*, pages 66–71, 1997.
7. J. Han, J. Tao, H. Zheng, H. Guo, D. Z. Chen, and C. Wang. Flow field reduction via reconstructing vector data from 3D streamlines using deep learning. *IEEE Computer Graphics and Applications*, 39(4):54–67, 2019.
8. F. A. Mussa-Ivaldi. From basis functions to basis fields: Vector field approximation from sparse data. *Biological Cybernetics*, 67(6):479–489, 1992.
9. G. Gouesbet and C. Letellier. Global vector-field reconstruction by using a multivariate polynomial L2 approximation on nets. *Physical Review E*, 49(6):4955–4972, 1994.
10. Y. Chen, J. Cohen, and J. Krolik. Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1448–1455, 2007.
11. F. Hong, J. Zhang, and X. Yuan. Access pattern learning with long short-term memory for parallel particle tracing. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 76–85, 2018.
12. 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.
13. S. Wiewel, M. Becher, and N. Thuerey. Latent-space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum*, 38(2):71–82, 2019.
14. B. Kim and T. Günther. Robust reference frame extraction from unsteady 2D vector fields with convolutional neural networks. *Computer Graphics Forum*, 38(3):285–295, 2019.
15. L. Guo, S. Ye, J. Han, H. Zheng, H. Gao, D. Z. Chen, J.-X. Wang, and C. Wang. SSR-VFD: Spatial super-resolution for vector field data analysis and visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 71–80, 2020.
16. 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), 2021.
17. 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 *Proceedings of IEEE International Conference on Big Data*, pages 438–447, 2018.
18. P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

Pengfei Gu, is a Ph.D. student in the Department of Computer Science & Engineering at the University of Notre Dame. Contact him at pgu@nd.edu.

Jun Han, is a Ph.D. candidate in the Department of Computer Science & Engineering at the University of Notre Dame. Contact him at jhan5@nd.edu.

Danny Z. Chen, is a professor in the Department of Computer Science & Engineering at the University of Notre Dame. Contact him at dchen@nd.edu.

Chaoli Wang, is an associate professor in the Department of Computer Science & Engineering at the University of Notre Dame. Contact him at chaoli.wang@nd.edu.

APPENDIX

Parameter study

To evaluate VFR-UFD, we study the following parameter settings: architecture design, number of representative streamlines, feature fusion schemes, temporal coherence, and sampling schemes. Note that all parameter study results shown for VFR-UFD are the inferred results. That is, pathline and streamline visualization results and PSNR and AAD values reported are from the synthesized vector fields.

Table 1. Average PSNR (dB) and AAD values, and training time per epoch (in minutes) with different architecture designs.

data set	architecture	PSNR	AAD	train
supernova	No RRB	23.72	0.082	41.77
	1-RRB	26.07	0.060	52.25
	2-RRB	**31.40**	**0.029**	69.38
	3-RRB	30.72	0.039	84.84
tornado	No RRB	31.14	0.062	31.22
	1-RRB	**34.28**	**0.043**	36.58
	2-RRB	34.03	0.047	40.03
	3-RRB	33.86	0.049	50.80

Architecture design. To investigate the effectiveness of the number of RUs for refining and denoising inaccurate vector fields, we conduct experiments on the supernova and tornado data sets by setting different values (0 to 3) for r. We denote different architectures as r-RRB. In Figure 1, we show the streamline rendering results from the synthesized vector fields generated by different architectures (i.e., r-RRB). The streamline rendering results from the synthesized vector fields generated by 2-RRB are much closer to GT than those generated by other architectures. This indicates that, when a suitable number of RUs is set, RRB can refine and denoise the features to enhance the synthesized vector fields' quality. As shown in Table 1, we compute the average PSNR and AAD values, and the training time per epoch. For the supernova data set, 2-RRB performs the best (i.e., with the highest PSNR and lowest AAD). For the tornado data set, 1-RRB performs the best; however, the performance difference between 1-RRB and 2-RRB is small (i.e., 0.25 for PSNR and 0.004 for AAD). Considering that the streamline rendering results from the synthesized vector fields generated by 2-RRB are much closer to GT than those by 1-RRB and small r cannot completely clean the noises introduced by GVF and large r will result in gradient vanishing, we recommend using 2-RRB for both data sets.

Table 2. Average PSNR (dB) and AAD values, and training time per epoch (in minutes) using different numbers of representative streamlines selected by REP for training.

data set	# rep. lines	reduction rate	PSNR	AAD	train
supernova	100	146.50	25.34	0.078	17.73
	300	49.87	29.00	0.048	55.48
	500	29.67	**31.40**	**0.029**	69.38
tornado	100	140.63	26.52	0.120	9.85
	300	46.88	31.41	0.064	29.20
	500	28.13	**34.03**	**0.047**	40.03

Number of representative streamlines. To explore how the quality of the synthesized vector fields using VFR-UFD improves with the increase of representative streamlines used for training, we conduct experiments on the supernova and tornado data sets using different numbers of representative streamlines. As illustrated in Figure 2, we see that the quality of the synthesized vector fields using VFR-UFD improves gradually when the number of representative streamlines increases from 100 to 300 and 500. This is because the more representative streamlines used, the more information the training data can provide. Table 2 reports the average PSNR and AAD values, and the training time per epoch. The performance is the best when using 500 representative streamlines for both data sets, which is consistent with the pathline rendering results shown in Figure 2. However, with the increase of representative streamlines, the training takes a longer time. Considering the trade-off between the number of representative streamlines and the training time, we recommend using 500 representative streamlines for both data sets.

Feature fusion schemes. There are two standard feature fusion schemes: *concatenation* and *summation*. To investigate VFR-UFD's effectiveness using different feature fusion schemes, we conduct experiments on the supernova and tornado data sets. Specifically, we make use of concatenation and summation in RU to fuse the features. Figure 3 presents the pathline rendering results from the synthesized vector fields generated by VFR-UFD using different feature fusion schemes. For both data sets, the synthesized vector fields generated using summation are closer to GT than those generated using concatenation. For a clearer comparison, Table 3 reports the

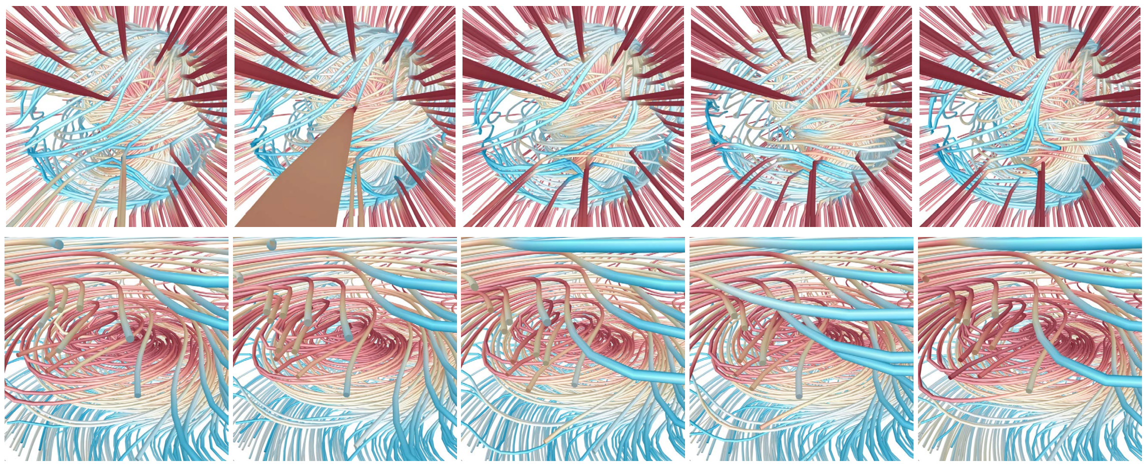


Figure 1. Comparison of streamline rendering results with different RRB blocks using the supernova (top) and tornado (bottom) data sets. 500 streamlines are traced for each data set.

average PSNR and AAD values. We can see that the performance of summation is better than that of concatenation for both data sets (especially for the supernova data set), which is consistent with the pathline rendering results. This is because in concatenation, Conv processes the noisy features without refinement every time. In contrast, in summation, the noisy features are refined after the addition operation and fed into the following Convs. Considering both the quantitative and qualitative results, we recommend using summation for all data sets.

Table 3. Average PSNR (dB) and AAD values using different feature fusion schemes.

data set	scheme	PSNR	AAD
supernova	concatenation	24.31	0.076
	summation	**31.40**	**0.029**
tornado	concatenation	33.26	0.048
	summation	**34.03**	**0.047**

Table 4. Average PSNR (dB) and AAD values without and with considering temporal coherence (TC).

data set	TC	PSNR	AAD
supercurrent	without	35.82	0.023
	with	**36.15**	**0.020**
supernova	without	29.07	0.045
	with	**31.40**	**0.029**

Temporal coherence. We take into account temporal coherence when using streamlines to recover UFD. To study the importance of temporal coherence on performance improvement, we perform experiments on the supernova and supercurrent data sets. In particular, instead of inputting three consecutive vector fields into VFR-UFD (i.e., temporal coherence is considered), we input a single vector field into VFR-UFD (i.e., temporal coherence is not considered). Figure 4 shows the pathline rendering results from the synthesized vector fields generated by VFR-UFD without and with considering temporal coherence. The synthesized vector fields generated by VFR-UFD when considering temporal coherence are closer to GT than those generated without considering temporal coherence. Table 4 reports the average PSNR and AAD values. We can see that the performance when considering temporal coherence is better than that without considering temporal coherence, even though the performance difference on the supercurrent data set is small (i.e., 0.33 for PSNR and 0.003 for AAD), which is consistent with the pathline rendering results. Therefore, incorporating temporal coherence can improve the quantitative and qualitative results of UFD.

Table 5. Average PSNR (dB) and AAD values, and training time per epoch (in minutes) using different sampling schemes.

data set	scheme	PSNR	AAD	train
supernova	random	29.03	0.049	52.25
	full	**31.80**	**0.026**	102.17
	coverage-driven	31.40	0.029	69.38

Sampling schemes. There are two standard sampling schemes, random sampling and full

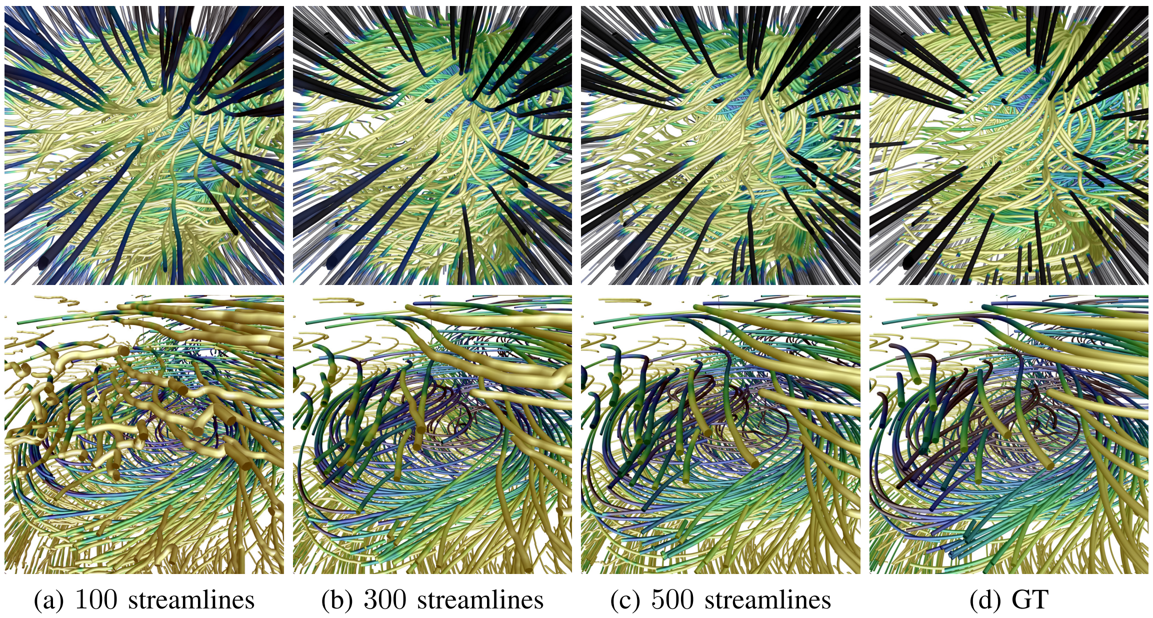


Figure 2. Comparison of pathline rendering results with different numbers of streamlines used in training using the supernova (top) and tornado (bottom) data sets. 500 and $2,000$ pathlines are traced for the supernova and tornado data sets, respectively.

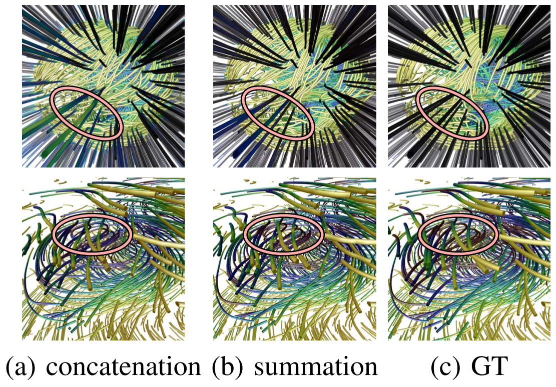


Figure 3. Comparison of pathline rendering results with different feature fusion methods using the supernova (top row) and tornado (bottom row) data sets. $1,000$ and $2,000$ pathlines are traced for the supernova and tornado data sets, respectively.

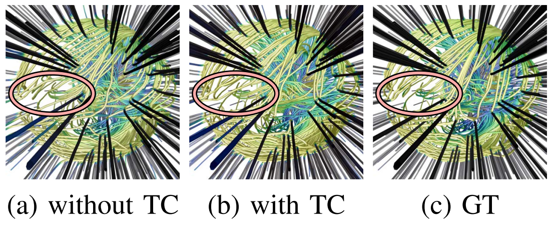
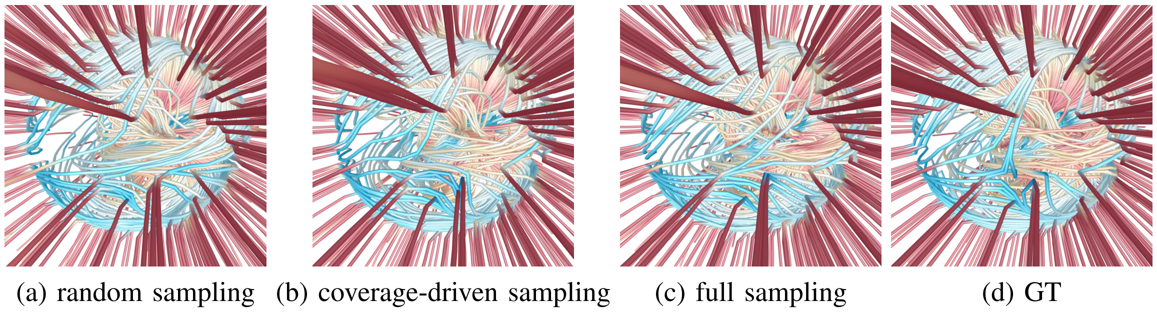


Figure 4. Comparison of pathline rendering results without and with temporal coherence (TC) using the supernova data set. 500 pathlines are traced.

sampling. Random sampling reduces the training time, but it cannot guarantee high performance. Instead, full sampling achieves high performance, but the training time is prolonged. Considering the trade-off between training time and quality performance, we propose the coverage-driven sampling scheme, which can achieve high performance while saving much training time. We conduct experiments using random sampling, full sampling, and coverage-driven sampling on the supernova data set to show our proposed coverage-driven sampling scheme's efficiency and efficacy. In the experiment with the supernova data set, random sampling and coverage-driven sampling use 59.75% and 77.26%, respectively, of the full samples. As shown in Figure 5, the synthesized vector fields generated by VFR-UFD using coverage-driven sampling and full sampling are much closer to GT than those using random sampling. Table 5 reports the average PSNR and AAD values along with training time per epoch. We can see that the performance using coverage-driven sampling is better than that using



(a) random sampling (b) coverage-driven sampling (c) full sampling (d) GT

Figure 5. Comparison of streamline rendering results with different sampling schemes using the supernova data set. 500 streamlines are traced.

random sampling, and is comparable to that using full sampling while saving 2.28 days in training time (running 100 epochs). As a consequence, we recommend using coverage-driven sampling.